Resolution-Based Methods for Linear-Time Temporal Logics

with Applications to Formal Verification

Thesis submitted in accordance with the requirements of the University of Liverpool
for the degree of Doctor in Philosophy by

Michel Ludwig

April 2010

Department of Computer Science
University of Liverpool

Primary Supervisor:	**Dr. Ullrich Hustadt** Department of Computer Science University of Liverpool
Secondary Supervisor:	**Dr. Boris Konev** Department of Computer Science University of Liverpool
Adviser:	**Dr. Clare Dixon** Department of Computer Science University of Liverpool
Internal Examiner:	**Dr. Alexei Lisitsa** Department of Computer Science University of Liverpool
External Examiner:	**Dr. Anatoli Degtyarev** Department of Computer Science King's College London

Abstract

Originally designed to represent tense in natural language, *temporal logics* have been used successfully in numerous application areas. In this thesis we focus on formal verification through resolution-based proof methods for two linear-time temporal logics: propositional linear-time temporal logic (PLTL) and monodic first-order temporal logic (FOTL). Both of these logics are interpreted over a model of time that is isomorphic to the natural numbers.

A machine-oriented resolution-based calculus called *fine-grained temporal resolution* has been previously introduced for PLTL and monodic FOTL. The calculus operates on a clausal normal form for PLTL and monodic FOTL formulae, the so-called *clausified temporal problems*. In this thesis, we consider its refinement, the *ordered fine-grained temporal resolution with selection* calculus, $\mathfrak{I}^{S,\succ}_{FG}$. It extends fine-grained temporal resolution with ordering restrictions and selection functions. A full proof of refutational completeness for ordered fine-grained temporal resolution with selection is given in this thesis.

Besides refutational completeness, another aspect of ordered fine-grained temporal resolution with selection that we analyse in this thesis concerns the elimination of redundant clauses. The question arises whether tautological and subsumed clauses can be eliminated without losing refutational completeness in the context of the $\mathfrak{I}^{S,\succ}_{FG}$-calculus. To that end, we identify syntactic criteria for identifying temporal clauses as tautologies. Moreover, we define a subsumption relation on temporal clauses and we illustrate how the calculus has to be extended to remain compatible with the removal of subsumed clauses and of tautologies.

However, some issues regarding the applicability of the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus in practice remain as it contains inference rules whose applicability conditions are only semi-decidable in general. Consequently, fair derivations, i.e. derivations in which every non-redundant clause that is derivable from a given clause set is eventually derived, cannot be guaranteed in practice as the applicability check for some inference rules might not terminate. We therefore present an inference procedure based on the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus that can construct fair derivations for reasoning in monodic first-order temporal logic, and we prove its refutational completeness. We also show that the new inference mechanism can be used as a decision procedure for some specific classes of temporal problems.

The fair inference procedure has been implemented in the monodic first-order temporal logic prover TSPASS, which is based on the first-order theorem prover SPASS. We describe the implementation and we analyse the effectiveness of redundancy elimination and the proof search performance of TSPASS on several examples.

In the final part of the thesis we are trying to fill a gap in functionality that resolution-based techniques admit over tableaux-based reasoning for formal verification purposes based on PLTL. In the case of a failure to prove the validity of a specific property by a tableaux-based method, a counter example demonstrating the erroneous behaviour has already been constructed. On the other hand, only the knowledge that the specification does not satisfy the required property is generally available for resolution-based verification attempts. We therefore present an algorithm that allows us to automatically construct a model for a satisfiable PLTL formula based on the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus and on the regular resolution-based model construction for sets of propositional clauses. We also briefly analyse the implementation of the model construction procedure as an extension of TSPASS, and we provide some experimental results on the practical performance of the model construction mechanism applied on benchmark classes.

Acknowledgements

Foremost, I would like to thank Ullrich Hustadt for his invaluable support during the three years of my PhD studies. Not only his interesting research ideas, but also his constructive criticism and generous assistance regarding problems of any nature greatly contributed to making this thesis possible.

I would also like to extend my gratitude to Boris Konev for giving helpful suggestions and to Clare Dixon for additionally being a thorough tester of TSPASS. Consequently, numerous bugs have been discovered and corrected. Moreover, I am grateful to my fellow PhD colleagues at the Department of Computer Science, who contributed to creating an enjoyable working atmosphere, and special thanks go to my parents and sisters.

Finally, I am thankful for the opportunities to present most of the results contained in this thesis at various international workshops and conferences.

The results described in Chapter 4 were published in [66] and presented at the International Workshop on First-Order Theorem Proving (FTP 2009) held in Oslo, Norway, during 6-7 July, 2009.

Chapter 5 is based on the material published in [65] and presented at the 22nd International Conference on Automated Deduction (CADE-22) held in Montreal, Canada, during 2-7 August, 2009.

The description of the theorem prover TSPASS given in Chapter 6 is derived from the material published in [68].

Lastly, the results described in Chapter 7 were published in [67] and presented at the 16th International Symposium on Temporal Representation and Reasoning (TIME 2009) held in Brixen-Bressanone, Italy, during 23-25 July, 2009.

Contents

Chapter 1

Introduction

In this introductory chapter we give a brief overview of the scientific field of formal verification, to which the methods developed in this thesis are meant to contribute. We begin by motivating the usefulness of formal verification before we introduce the two main constituents that different approaches towards formal verification have in common: the specification languages and the reasoning methods for establishing the correctness of a specified system. We conclude this chapter by briefly describing the novel contributions and we give a general outline of the thesis.

1.1 Formal Verification

The field of *formal verification* is concerned with verifying that a hardware or software system behaves correctly in all situations, which is an essential prerequisite in the design and construction of safety critical systems. In addition to knowing that a verified system is correct, the quest for formal verification is motivated by the high costs incurred in case of failure and when corrections of errors are required. For example, the following four case studies illustrate the dramatic consequences which errors in complex hardware and software systems can have.

- In order to improve the performance of the floating point unit (FPU) in the Pentium CPU, a new algorithm was used for computing the results of floating-point divisions [43]. This new algorithm used look-up tables with 1066 non-empty entries in intermediate computation steps. However, due to a mistake in how this table was initially constructed, 5 of those entries were not copied into the processor design, and as a result, the FPU returned wrong values on certain floating point computations. This became known as the Pentium FDIV bug in 1994, and the manufacturer Intel was forced to replace the defective chips.
- In the case of the maiden flight of the Ariane 5 launcher [61], flight 501, on 4th June, 1996, the launcher started to disintegrate 39 seconds after lift-off and was destroyed by an automatic safety mechanism, resulting in an associated financial loss of 1.9 billion

French Francs. The erroneous flight behaviour was caused by wrong input given to the on-board computer by the Inertial Reference System. The input data provided was in fact an error message but was interpreted as flight data instead. The error message resulted from a software exception which occurred due to the conversion of a 64-bit floating point number to a 16-bit signed integer number, which yielded an overflow. The overflow occurred during computations that were in fact intended for the Ariane 4 launcher but were no longer required for Ariane 5, resulting in computations with higher numerical values than the software expected. The software showing faulty behaviour on the Ariane 5 was initially designed for the Ariane 4 and was intended for a different flight path. According to [61] system engineering faults were the root cause of flight 501's failure.

- In 1999 the Mars Climate Orbiter spacecraft reached Mars and was supposed to enter a low orbit around Mars. However, during the design and construction phase of the spacecraft one error remained unnoticed [34]: the software used to control the thrusters on the ground was working with Imperial measurement units, whereas the spacecraft itself expected metric units. As a result the spacecraft descended to an altitude of 57 km above Mars, which was too low for its survival.

- Finally, the Mars Polar Lander was subject to a similar fate in 1999 [34]. Its mission involved a landing on the surface of Mars, but its last transmission was received when it prepared for entering Mars' orbit. The real cause of its disappearance still remains unknown. However, a logic error leading to a premature shutdown of the descent engines was discovered later on, together with problems related to the touchdown sensors. One possible explanation for the loss of the spacecraft would hence be a free fall onto the surface of Mars.

For formal verification to be applicable, precise yet understandable ways are necessary for specifying complex systems and what constitutes correct behaviour. In addition, exhaustive methods for determining that a system behaves correctly for all input patterns are required.

In Section 1.1.1 we describe the former aspect of specifying systems in a formal way in greater detail. For the latter point concerning methods to establish that systems show correct behaviour, two different approaches can be identified: model checking-based and direct proof-based verification methods, which are discussed in the Sections 1.1.2 and 1.1.3, respectively.

1.1.1 Formal Languages

The specification of hardware and software systems for formal verification is based on abstracting the behaviour of the considered system using a "language" for which the meaning of its "sentences" is defined in a precise way. The definition of the specification language is typically given on two levels.

The definition of its grammar, i.e. the way how "sentences", or more correctly *formulae*, are built in this language, is given on the first level together with the definition of the vocabulary that can be used, i.e. the basic building blocks, which are called *constants*, *functional symbols*, *terms*, *propositional symbols*, or *predicates* in our case, and the operators connecting the basic building blocks, like $\wedge$, $\vee$, $\Rightarrow$. Formulae can be seen as tree-like structures with basic building blocks appearing on the leaves and operators in the inner nodes.

On the second level, then, a precise way of interpreting the different building blocks and the operators connecting them together is provided. The interpretation assigns the value "true" or "false" to a formula according to the values assigned to the basic building blocks and according to the operators connecting them. If there is an assignment to the basic building blocks of a formula $\mathcal{F}$ such that its interpretation w.r.t. to the assigned values is "true", we say that the formula $\mathcal{F}$ is *satisfiable*. If every assignment leads to a interpretation that is "true", the formula $\mathcal{F}$ is said to be valid. An assignment of values to the basic building blocks such that the formula $\mathcal{F}$ is interpreted as "true" is called a *model* for the formula $\mathcal{F}$.

In this thesis we focus on *temporal logics* as specification languages, which were originally designed to represent tense in natural language [76]. Temporal logics can be used to express conditions that should hold in different moments in time and are as such naturally suited for expressing the specification and properties of hardware and software systems whose state changes over time.

A wide variety of different temporal logics exist. Beside having different vocabulary, they also differ in the considered model of time. Generally speaking, the class of languages with a branching-time model [33] represent several possible choices over time in a tree-like structure as interpretations for formulae, whereas the class of languages with a linear model of time [73,74] considers a single time line for which only one choice is possible at any moment in the time line. In addition to differences in representing *time flow*, temporal logics also vary in their notion of *instants in time*. Moments in time can be dense [16], i.e. isomorphic to the real numbers, or rather considered as discrete time instants [74], i.e. isomorphic to the natural numbers or integers. Moreover, temporal logics that do not consider time instants but rather time intervals also exist [71,82].

Important concepts in the verification of reactive and concurrent system can also be specified in temporal logics. For example, the notions of

- *safety*: "nothing bad happens",
- *liveness*: "something good eventually happens",
- *fairness*, i.e. constraints that exclude unwanted behaviour of the specified system,

represent important verification properties and can be expressed in temporal logics [32,74]. For instance, reactive systems are an important class of systems in the field of formal verification [45]. Their behaviour is characterized by a continuous interaction with the environment they are in. Verifying properties of reactive systems has been successfully achieved using temporal (and modal) logics [29, 32, 39, 73, 86].

Besides formal verification, other application areas of temporal logics include the specification of programs [74], temporal databases [89], knowledge representation through temporal description logics [5], logic programming in the form of executable temporal logics [11], and the analysis of natural language [85].

In this thesis we focus on propositional linear-time temporal logic (PLTL) and first-order temporal logic (FOTL) as specification languages. For both languages we assume a linear time flow model composed of discrete time instants with finite past and infinite future, i.e. a model of time isomorphic to the natural numbers. PLTL can be seen as an extension of propositional logic with temporal operators, whereas FOTL is a similar extension of first-order logic.

In the following two sections, we describe the two main approaches for establishing that a system behaves according to some specified behaviour in temporal logics.

1.1.2 Model Checking

Model checking is a prominent method for verifying that a system specification exhibits certain properties. It has been applied successfully both in the verification of hardware and software systems [19].

Verification via model checking involves validating that a specific temporal formula, which represents the property that is to be verified, is fulfilled in every possible state that the system can enter. Obviously, the number of possible states that can occur in hardware and software systems can grow very large, which is especially a problem for software verification via model checking. Tremendous research effort has gone into combating this space explosion problem, and impressive results have been achieved. Moreover, automated model checking tools like SPIN [48] and NuSMV [18] have also been developed. In the case where a specification does not fulfil a property, a counter-example, i.e. the error path in the state space, is automatically constructed by the model checking tool.

However, the model checking approach to formal verification presents problems in the case of infinite-state systems, which frequently occur with timed and hybrid systems [20], for example. Obviously, determining whether a property holds in *every* state is not possible directly in that case, and as such model checking can, for instance, only be applied to finite reductions of these infinite-state systems [21]. Other ways of handling infinite-state model checking have been developed, see e.g. [15], but the fundamental problem remains that it is impossible to check in finite time whether a property holds in infinite-state systems simply by visiting all the possible states of the systems.

1.1.3 Direct Proof Methods

In contrast to model checking, verification based on *direct proof* has the advantage that it can also be easily applied on infinite state systems. Instead of verifying that a property holds in every possible state that a specified system can be in, the proof-based verification methods try to perform "reasoning" with the system specification formula φ and the property ψ that is to be verified. More specifically, as proving the validity of the formula $\varphi \Rightarrow \psi$ is equivalent to establishing that the formula $\neg(\varphi \Rightarrow \psi) \equiv \varphi \wedge \neg\psi$ is unsatisfiable, the proof-based verification methods try to determine that the formula $\varphi \wedge \neg\psi$ is unsatisfiable.

In the following we present three proof methods that are of particular importance for temporal logics.

1.1.3.1 Tableaux-Based Proof Methods

Tableaux-based proof methods try to construct structures, so-called *tableaux*, for formulae from which models can be built easily. If such a structure has been exhaustively constructed and when it becomes apparent that no model can be extracted from it, one can conclude that the considered formula is unsatisfiable, and its negation therefore valid. As such the proof of unsatisfiability for a formula can become rather involved as the full structure representing all the potential models has to be explored before one can conclude that no model can exist. One has to note, though, the tableaux-based algorithms typically do not suffer from the state space explosion that is present in model checking approaches. However, tableaux algorithms share the aspect of model checking approaches that a model can be easily obtained once a formula is determined to be satisfiable, which is equivalent to finding a counter-example in the case of validity checking.

Tableaux algorithms for PLTL are given, for example, in [93], and for a fragment of FOTL (and decidable subclasses) in [60]. Implementations of tableaux algorithms for PLTL exist, for instance, in the Logics Workbench [46], the Tableaux Work Bench [4] and LoTREC [42].

1.1.3.2 Automata-Based Proof Methods

Proofs based on automata are mainly available for PLTL. The general idea behind this proof method lies in the structure of PLTL models: they can be seen as infinite strings over the language consisting of all the subsets of propositional symbols occurring in a specific formula. Büchi automata, i.e. automata that manipulate infinite strings, can be constructed in such a way that they only accept infinite strings which represent models of a given formula φ. Then, if such an automaton for the formula $\neg\varphi$ does not accept any infinite words, we can conclude that the formula $\neg\varphi$ is unsatisfiable, and therefore φ is valid. More details can be found in [75].

1.1.3.3 Resolution-Based Proof Methods

The main goal of resolution-based proof methods consists in deriving a contradiction from a given set of formulae. Broadly speaking, two different classes of resolution inference systems exist: non-clausal and clausal resolution. They mainly differ in the types of formulae on which the inference rules can be applied.

Non-clausal resolution methods can be applied on arbitrary formulae. Such an inference system has been developed for PLTL in [2], an extension to FOTL was given in [3]. The practical drawback of the inference systems presented in [2,3] lies in the large number of inference rules, which makes an implementation difficult.

In this thesis we therefore focus on clausal resolution. The inference rules of clausal resolution can only be applied on formulae that are in a normal form, the so-called *clausal normal form*. Formulae in clausal normal form are also called *clauses*. Essentially, in the non-temporal propositional case, clauses are mutisets of literals, where a literal is either p or $\neg p$ for a propositional symbol p. The empty clause, denoted by $\perp$, plays an important role in clausal resolution calculi as we will see later. Note that the empty clause is unsatisfiable. In the first-order non-temporal case, the notion of literal is extended to include arbitrary positive or negative predicates, i.e. $p(t_1, \ldots, t_n)$ or $\neg p(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are arbitrary terms. Efficient algorithms for transforming sets of arbitrary formulae into "small" clausal normal forms have also been developed.

The resolution calculus was initially introduced by J.A. Robinson in 1965 [79]. It consists of the following two inference rules:

- Resolution

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma}$$

  where C, D are disjunctions of literals and σ is a most general unifier of the atoms A and B

- Factoring

$$\frac{C \vee A \vee B}{(C \vee A)\sigma}$$

  where C is a disjunction of literals and σ is a most general unifier of the atoms A and B

A unifier σ of two atoms A and B is a function that substitutes terms for variables such that $A\sigma = B\sigma$.

The resolution rule states that if two clauses $C \vee A$ and $D \vee \neg B$ contain atoms A and B, respectively, which can be unified by a (most general) unifier σ, then one can derive a clause $(C \vee D)\sigma$, where the variables in the disjunctions of literals C and D are replaced by the terms assigned to them in σ. In the propositional case the intuition behind the resolution inference rules is the following: if one assumes that the two propositional clauses $C \vee A$, $D \vee \neg A$ are true, it is clear that $C \vee D$ must hold as A and $\neg A$ cannot both be true. The intuition behind the resolution rule in the first-order case is similar if one takes into account that first-order clauses are implicitly universally quantified, i.e. the fact that a first-order clause is true implies that it is still true if variables are replaced consistently by arbitrary terms.

The factoring rule, then, states that two occurrences of positive literals which can be unified by a most general unifier can be replaced by a single occurrence of them after the computed unifier has been applied on all the literals. In the propositional case the factoring rule simply removes duplicate literals from clauses, whereas the factoring rule in the first-order case removes literals that can be unified, i.e. made equal through the application of a unifier, from a given clause.

In an attempt to determine whether a set of clauses is unsatisfiable, the inference rules of the resolution calculus are exhaustively applied on the clauses contained in the set. New clauses are obtained from the applications of these inference rules, which are again considered for further inference computations. If the empty clause $\perp$ is obtained at some point, one can conclude that the initial set of clauses is unsatisfiable. Otherwise, if the resolution and factoring inference rules have been exhaustively applied and no new clauses can be derived, one can infer that the initial set of clauses is satisfiable.

Despite having been developed to increase the efficiency of automated theorem proving, a straight-forward implementation of the resolution inference rules presented above would not lead to an efficient theorem prover as the number of possible inferences would be far too large. To tackle this problem, numerous strategies have been developed that can help reduce the number of possible inferences and prune the search space in that way. These strategies include, for example, ordering restrictions and selection functions [10], which lead to modified inference rules, or, for instance, the set of support strategy [63], which enforces that only specific combinations of clauses are considered for computing inferences.

Other optimisations include the elimination of clauses that do not contribute or rather prolong the derivation of the empty clause. Techniques such as redundancy elimination can be applied here. Two important classes of redundant classes are *tautologies* and *subsumed* clauses. Tautologies are clauses that are true in every interpretation and they can (usually) be safely discarded as they cannot contribute to derivations of the empty clause. The notion of clause subsumption deals with "smaller" clauses that subsume "larger" clauses, i.e. whenever the "smaller" clause is satisfied in a model, the "larger" clause is also fulfilled. Additionally, the "larger" clause would potentially lead to longer derivations of the empty clause. Subsumed clauses can therefore be removed from the clause sets used for inference computations.

On the practical side then, the notion of *fair derivations* plays an important role in the design of automated theorem provers. Theorem provers must ensure that fair derivations are constructed, i.e. every (derived) clause must be considered for inference computations at some point in time. If one clause would never be chosen for computing inferences, it could happen that a clause essential for deriving the empty clause from an unsatisfiable clause set is not used and the empty clause could not be obtained. A variety of selection strategies for choosing clauses to compute inferences with whilst maintaining the fairness of derivations have been developed, see e.g. [91].

A large number of automated theorem provers for first-order logic based on resolution have been developed over time. The prover Otter [70] can be seen as the ancestor of all modern resolution-based theorem provers as its design principles laid the foundation for the architectures of modern provers. Other notable automated theorem provers for first-order logic (with equality) include E [81], which is based on superposition calculus for equational clausal logic [9], SPASS [92] and Vampire [78]. There also exist theorem provers for first-order logic (with equality) that are not based on resolution (or the superposition calculus) as such. One example of such a prover is Waldmeister [64] (for unit equational logic), whose inference procedure uses unfailing Knuth-Bendix completion [8].

When developing new inference calculi based on resolution, there are two theoretical concepts that are of particular importance: *soundness* and *refutational completeness.*

The concept of soundness relates to the correctness of the inference rules: one has to show that if a models fulfils all the premises of an inference rule, then the conclusion of that inference rule is also satisfied by the model. The property of correctness can be used to show the unsatisfiability of a clause set once the empty clause has been derived. If one assumes that the originally considered clause set has a model, it would follow from soundness of the inference rules that the empty clause would be satisfied in that model, which is then a contradiction as the empty clause is equivalent to **false** and consequently, it cannot be fulfilled by any model.

The concept of refutational completeness can be essential for the practical usefulness of a calculus. It characterises the fact that if a clause set is unsatisfiable, then the empty clause can be derived by an exhaustive application of the inference rules of the calculus on the clauses contained in the considered set.

If a calculus is both *sound* and *refutationally complete*, one can conclude that a clause set is unsatisfiable if and only if an exhaustive application of the calculus' inference rules derives the empty clause. It is common that proving the refutational completeness of a calculus is a harder task than showing the correctness of its inference rules.

For PLTL clausal resolution-based calculi were, for example, introduced in [17] and [90]. In this thesis we focus on the approach introduced in [37]. There, temporal formulae are first transformed into a normal form called *Separated Normal Form* (SNF), which is characterized by separating the different temporal aspects of temporal formulae. The transformation of a PLTL formula into SNF results in (possibly) several new PLTL formulae that are of three different types: formulae which have to hold at the initial moment in time, formulae that link consecutive time points and formulae that express conditional eventualities. Resolution inference rules are then devised that can operate on the first-two SNF formulae types, whereas a special inference rules is required to handle conditional eventualities.

For first-order temporal logic we only consider the monodic fragment in this thesis. In the monodic fragment formulae that appear under a temporal operator are only allowed to have at most one free variable. A first resolution-based calculus, called *monodic temporal resolution*, was introduced in [23]. A normal form called *Divided Separated Normal Form* (DSNF) has been developed for that calculus, which can be seen as a minor extension of SNF. A type for formulae that should hold at every moment in time has been added, and additionally, only unconditional eventualities occur in DSNF. As in the case for PLTL, monodic temporal resolution uses special inference rules to handle unconditional eventualities, the so-called eventuality resolution rules, and resolution-like inference rules for the remaining

DSNF formulae types.

Despite representing a big step towards efficient automated theorem proving in monodic FOTL, most inference rules of monodic temporal resolution have applicability conditions that are computationally too complex to be implemented efficiently. As a remedy, a more machine-oriented resolution-based calculus, the fine-grained (temporal) resolution calculus, has been introduced in [57, 58], which operates on problems in DSNF that have been clausified. The computationally complex inference rules of monodic temporal resolution have been replaced by resolution-based inference rules that operate on the different temporal clause types of clausified DSNF. In this regard fine-grained resolution can be seen as being more fine-grained than monodic temporal resolution as it performs "smaller" inference steps by manipulating temporal clauses. Moreover, special algorithms have been developed that aid the search for the premises of the eventuality resolution rules. As a refinement, the *ordered fine-grained (temporal) resolution with selection calculus* has been presented in [51]. It extends fine-grained resolution with ordering restrictions and selection functions. Another advantage of using clausal resolution and the normal form DSNF is that the temporal clauses in clausified DSNF can be translated into first-order logic. State-of-the-art theorem provers for first-order logic can then be used for temporal reasoning as most temporal inference rules can be mapped onto first-order resolution and factoring rules. A special treatment of eventualities remains necessary, though.

As a consequence of these theoretical advances, automated theorem provers based on resolution have been developed for PLTL and monodic FOTL. TRP [52,53] and TRP++ [49] are theorem provers for PLTL that implement the resolution-based calculus introduced in [37]. TRP is written in Prolog, whereas a C++ implementation of the temporal resolution calculus is available in TRP++. For monodic FOTL then, the theorem prover TeMP has been developed [50]. It is based on ordered fine-grained resolution with selection, and uses the first-order prover Vampire as inference kernel.

We continue now by stating the novel contributions of this thesis to the field of formal verification.

1.2 Novel Contributions

The novel contributions that can be found in this thesis consist in the following.

- A full proof of refutational completeness for ordered fine-grained temporal resolution with selection is given. The proof also includes the full details of the required lifting theorem.

- The compatibility of redundancy elimination, i.e. the removal of tautologies and subsumed clauses, in combination with ordered fine-grained temporal resolution has been studied formally. In order to show the refutational completeness of ordered fine-grained temporal resolution with selection in the presence of redundancy elimination, we have extended the calculus with two new inference rules. The proof of refutational completeness for the extended calculus uses proof-theoretical means. i.e. we show that for a given refutation that involves subsumed or tautological clauses, there also exists a refutation from the subsuming clauses which does not contain tautologies.

- We have developed an inference procedure for monodic FOTL based on ordered fine-grained temporal resolution with selection which can ensure fair derivations. Some inference rules of ordered-fine grained temporal resolution with selection have applicability conditions that are only semi-decidable. As a consequence, fair derivations cannot be guaranteed in practice. We have also proved the refutational completeness of the fair inference procedure, and we have shown that the fair inference algorithm can be used as a decision procedure for certain classes of temporal problems.
- The inference procedure has been implemented in the theorem prover TSPASS. We provide a detailed description of its implementation, which is based on the first-order theorem prover SPASS. We have also analysed the effectiveness of redundancy elimination and the proof search performance of TSPASS on several examples. Binaries and the source code for TSPASS are available at

  `http://www.csc.liv.ac.uk/~michel/software/tspass/`
- We have developed a model construction algorithm for satisfiable PLTL formula that uses saturations under ordered fine-grained temporal resolution with selection, for which we also provide a proof of correctness. The model construction procedure has been implemented as an extension of TSPASS, and we have analysed its practical performance on numerous examples.

We conclude this chapter by giving an organisational overview of this thesis.

1.3 Thesis Outline

The thesis is organised as follows.

In Chapter 2 we define the syntax and semantics of propositional linear-time temporal logic and first-order temporal logic. We also introduce the normal form that we consider for formulae of these logics, and describe how the formulae that are in normal form can be clausified for the resolution-based calculus that is introduced in Chapter 3. We also define some notions that are important for the subsequent parts of this thesis.

The main aim of Chapter 3 is to define the ordered fine-grained resolution with selection calculus and to prove its refutational completeness. The inference rules of ordered fine-grained resolution with selection can be classified into two categories: the fine-grained step resolution rules and the eventuality resolution rules in combination with the loop search algorithm FG-BFS. In order to show the refutational completeness of the calculus we briefly recall the monodic temporal resolution calculus before we define a refined version of monodic temporal resolution that is used in the proof of refutational completeness of the ordered fine-grained resolution with selection calculus. Subsequently, we then prove the lifting theorem for ordered fine-grained resolution with selection, which is required for the proof of refutational completeness in the final part of the chapter.

The focus of Chapter 4 lies on analysing the compatibility of ordered fine-grained resolution with selection with redundancy elimination methods such as the deletion of tautologies and subsumed clauses. We show how the calculus has to be extended in order to remain compatible with redundancy elimination rules. We also prove the refutational completeness of subsumption-compatible ordered fine-grained resolution with selection.

In Chapter 5 we present a new inference procedure for monodic FOTL that can ensure fair derivations. The inference procedure is based on ordered fine-grained resolution with selection and its main design principle consists in integrating the loop search procedure of ordered fine-grained resolution with selection into the main saturation under fine-grained step resolution. In this way the sequential execution of these two categories of inference rules, which can potentially lead to unfairness, can be avoided and fair derivations can be guaranteed.

Chapter 6 then describes the implementation of the fair inference procedure for monodic FOTL in the automated theorem prover TSPASS. We give a detailed description of how the first-order theorem prover SPASS 3.0 has been modified in order to accommodate the fair inference procedure for monodic FOTL. The chapter concludes with experimental results demonstrating the effectiveness of TSPASS on various problems mainly in comparison with the automated theorem prover TeMP. We also illustrate how the combinations of different redundancy elimination rules such as forward or backward subsumption and tautology deletion can influence the execution time required for finding proofs with TSPASS.

Finally, in Chapter 7 a method for automatically constructing models of satisfiable PLTL formulae is introduced. The construction is based on computing saturations under the PLTL version of ordered fine-grained resolution with selection. Regular model construction for propositional clause sets is then used to construct propositional models at the different time points of the temporal model. We present the construction procedure and prove its correctness. Additionally, we describe some practical considerations regarding the implementation of the model construction algorithm in the theorem prover TSPASS. Subsequently, we analyse some experimental results obtained by comparing the temporal model construction of TSPASS with a tableaux-based reasoning procedure implemented in the Logics Workbench on series of PLTL benchmark formulae.

Chapter 2

Linear-Time Temporal Logics

2.1 Introduction

Originally designed to represent tense in natural language [76], *temporal logics* have been used successfully in numerous application areas.

For example, important concepts in the verification of reactive and concurrent systems. such as safety, fairness and liveness, can be specified in temporal logics [32,74]. Verifying properties of reactive systems has been successfully achieved using temporal (and modal) logics [32,73,86].

Besides formal verification, other application areas of temporal logics include the specification of programs [74], temporal databases [89], knowledge representation through temporal description logics [5], logic programming in the form of executable temporal logics [11], and the analysis of natural language [85].

The aim of this chapter is to give a formal definition of the syntax and semantics for the two languages that we consider in this thesis: propositional linear-time temporal logic (PLTL) and monodic first-order temporal logic (FOTL). Both of these logics are interpreted over a model of time that is isomorphic to the natural numbers.

This chapter is organised as follows. First of all, we describe the syntax and semantics of propositional linear-time temporal logic. We then define the syntax and semantics of first-order temporal logic, and we give a formal definition of the monodic fragment. We also introduce some notions that are essential for the following parts of this thesis. In the subsequent section we present the normal form that we consider for PLTL and monodic FOTL formulae, and we conclude the chapter with a description of how the formulae that are in normal form can be clausified for the resolution-based calculus that is defined in Chapter 3.

2.2 Propositional Linear-Time Temporal Logics

The language of Propositional Linear Time Temporal Logic, PLTL, is an extension of classical propositional logic by temporal operators for a discrete linear model of time (i.e. isomorphic

$\mathfrak{M}_n \models \top$	
$\mathfrak{M}_n \not\models \bot$	
$\mathfrak{M}_n \models p$	iff $p \in D_n$
$\mathfrak{M}_n \models \neg\varphi$	iff not $\mathfrak{M}_n \models \varphi$
$\mathfrak{M}_n \models \varphi \vee \psi$	iff $\mathfrak{M}_n \models \varphi$ or $\mathfrak{M}_n \models \psi$
$\mathfrak{M}_n \models \varphi \wedge \psi$	iff $\mathfrak{M}_n \models \varphi$ and $\mathfrak{M}_n \models \psi$
$\mathfrak{M}_n \models \varphi \Rightarrow \psi$	iff $\mathfrak{M}_n \models \neg\varphi$ or $\mathfrak{M}_n \models \psi$
$\mathfrak{M}_n \models \varphi \Leftrightarrow \psi$	iff $\mathfrak{M}_n \models \varphi \Rightarrow \psi$ and $\mathfrak{M}_n \models \psi \Rightarrow \varphi$
$\mathfrak{M}_n \models \bigcirc\varphi$	iff $\mathfrak{M}_{n+1} \models \varphi$
$\mathfrak{M}_n \models \Diamond\varphi$	iff there exists $m \geq n$ such that $\mathfrak{M}_m \models \varphi$
$\mathfrak{M}_n \models \Box\varphi$	iff for all $m \geq n$, $\mathfrak{M}_m \models \varphi$
$\mathfrak{M}_n \models \varphi \mathsf{U} \psi$	iff there exists a $m \geq n$ such that $\mathfrak{M}_m \models \psi$ and $\mathfrak{M}_i \models \varphi$ for every $i, n \leq i < m$
$\mathfrak{M}_n \models \varphi \mathsf{W} \psi$	iff $\mathfrak{M}_n \models \varphi \mathsf{U} \psi$ or $\mathfrak{M}_n \models \Box\varphi$

Figure 2.1: Truth-Relation for Propositional Linear-Time Temporal Logic

to N). The vocabulary of PLTL is composed of a countably infinite set of *propositional symbols* p, q, p_0, p_1,p_2, ..., the *propositional operators* $\top$ (**true**), $\bot$ (**false**), $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$ and the *temporal operators* $\Box$ ('always in the future'), $\Diamond$ ('eventually in the future'), $\bigcirc$ ('at the next moment'), U ('until') and W ('weak until').

The set of PLTL formulae is defined as follows: $\top$ and $\bot$ are PLTL formula; any propositional symbol p is an *atomic* PLTL formula or *atom*; if φ and ψ are PLTL formulae, then so are

$$\neg\varphi, \varphi \vee \psi, \varphi \wedge \psi, \varphi \Rightarrow \psi, \varphi \Leftrightarrow \psi, \Box\varphi, \Diamond\varphi, \bigcirc\varphi, \varphi \mathsf{U} \psi, \text{ and } \varphi \mathsf{W} \psi.$$

Formulae of this logic are interpreted over temporal structures $\mathfrak{M} = (D_n)_{n \in \mathbb{N}}$ that associate with each element n of N, representing a moment in time, a propositional model (or valuation) D_n given by a set of propositional symbols. The definition of the *truth-relation* $\mathfrak{M}_n \models \varphi$ is given in Figure 2.1.

For a set $\mathcal{N}$ of PLTL formulae and $n \in \mathbb{N}$, we write $\mathfrak{M}_n \models \mathcal{N}$ if and only if $\mathfrak{M}_n \models \varphi$ holds for every formula $\varphi \in \mathcal{N}$.

A temporal structure $\mathfrak{M} = (D_n)_{n \in \mathbb{N}}$ is said to be a *model* for a formula φ if and only if it holds that $\mathfrak{M}_0 \models \varphi$. A formula is *satisfiable* if and only there exists a model for φ. A formula φ is *valid* if and only if every temporal structure $\mathfrak{M} = (D_n)_{n \in \mathbb{N}}$ is a model for φ.

We say that a set of PLTL formulae $\mathcal{N}$ *entails* a formula ψ, written $\mathcal{N} \models \psi$, if and only if every temporal structure $\mathfrak{M}$ that is a model for every formula $\varphi \in \mathcal{N}$ is a model for ψ.

We now the define the syntax and semantics of first-order temporal logic.

2.3 First-Order Temporal Logics

First of all, let $X = \{x, y, z, x_0, x_1, \ldots\}$ be a countably infinite set of *variables*, $CS = \{c, d, c_0, c_1, \ldots\}$ be a countably infinite set of *constants*, $FS = \{f, g, h, f_0, \ldots\}$ be a countably infinite set of *function symbols*, each with a fixed arity ≥ 1, and $PS = \{P, P_0, \ldots\}$ be a non-empty set of *predicate symbols*, each with a fixed arity ≥ 0. We write f/n or P/n to indicate that a function symbol or predicate is of arity n, respectively. A *proposition* is a predicate of arity 0.

We now introduce the notion of *terms*.

Definition 2.3.1. *Let $\Sigma \subseteq CS \cup FS$ and $Y \subseteq X$. Then the set of terms $T_\Sigma(Y)$ built over the set Σ and the set of variables Y is the smallest set inductively defined as follows:*

(i) Every variable $y \in Y$ is contained in $T_\Sigma(Y)$.

(ii) Every constant $c \in \Sigma \cap CS$ is contained in $T_\Sigma(Y)$.

(iii) If $t_1, \ldots, t_n \in T_\Sigma(Y)$ and $f/n \in \Sigma \cap FS$ is a function symbol, then $f(t_1, \ldots, t_n)$ is contained in $T_\Sigma(Y)$.

Next we recall the syntax and semantics of first-order logic, FOL, without equality. The set of FOL formulae (without equality) is defined as follows: $\top$ (**true**) and $\bot$ (**false**) are FOL formulae; if P is an n-ary predicate symbol and $t_1, \ldots, t_n$ are variables or constants, then $P(t_1, \ldots, t_n)$ is an *atomic* FOL formula; if φ and ψ are FOL formulae, then so are

$$\neg\varphi, \varphi \vee \psi, \varphi \wedge \psi, \varphi \Rightarrow \psi, \varphi \Leftrightarrow \psi, \exists x\varphi, \text{ and } \forall x\varphi.$$

FOL formulae are interpreted over first-order structures $\mathfrak{M} = (D, I)$, where D is a non-empty set, the *domain*, and I is an *interpretation*. The interpretation I maps every constant $c \in CS$ into an element $I(c) \in D$, every function symbol $f/n \in FS$ into a function $I(f)\colon D^n \rightarrow D$ and every predicate $P/n \in PS$ into a set $I(P) \subseteq D^n$. An *assignment* $\mathfrak{a}$ is a function from the set of variables X to D. For an assignment $\mathfrak{a}$ the interpretation I induces a interpretation of terms $I^\mathfrak{a}\colon T_{CS \cup FS}(X) \rightarrow D$ which is inductively defined as follows:

- For a variable $x \in X$:

$$I^\mathfrak{a}(x) = \mathfrak{a}(x)$$

- For a constant $c \in CS$:

$$I^\mathfrak{a}(c) = I(c)$$

- For terms $t_1, \ldots, t_n \in T_{CS \cup FS}(X)$ and a function symbol $f/n \in FS$:

$$I^\mathfrak{a}(f(t_1, \ldots, t_n)) = I(f)(I^\mathfrak{a}(t_1), \ldots, I^\mathfrak{a}(t_n))$$

The definition of the *truth-relation* $\mathfrak{M} \models^\mathfrak{a} \varphi$ for FOL is given in Figure 2.2.

Now, the language of First-Order (Linear Time) Temporal Logic, FOTL, is an extension of classical first-order logic by temporal operators for a discrete linear model of time (i.e. isomorphic to $\mathbb{N}$). The vocabulary of FOTL (without equality and function symbols) is composed of the set of variables X, the set of constants CS, the set of predicate symbols PS, the *propositional operators* $\top$, $\bot$, $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$, the *quantifiers* $\exists x_i$ and $\forall x_i$, and the

$\mathfrak{M} \models^{\mathfrak{a}} \top$	
$\mathfrak{M} \not\models^{\mathfrak{a}} \bot$	
$\mathfrak{M} \models^{\mathfrak{a}} P(t_1, \dots, t_m)$	iff $(I^{\mathfrak{a}}(t_1), \dots, I^{\mathfrak{a}}(t_m)) \in I(P)$
$\mathfrak{M} \models^{\mathfrak{a}} \neg\varphi$	iff not $\mathfrak{M} \models^{\mathfrak{a}} \varphi$
$\mathfrak{M} \models^{\mathfrak{a}} \varphi \vee \psi$	iff $\mathfrak{M} \models^{\mathfrak{a}} \varphi$ or $\mathfrak{M} \models^{\mathfrak{a}} \psi$
$\mathfrak{M} \models^{\mathfrak{a}} \varphi \wedge \psi$	iff $\mathfrak{M} \models^{\mathfrak{a}} \varphi$ and $\mathfrak{M} \models^{\mathfrak{a}} \psi$
$\mathfrak{M} \models^{\mathfrak{a}} \varphi \Rightarrow \psi$	iff $\mathfrak{M} \models^{\mathfrak{a}} \neg\varphi$ or $\mathfrak{M} \models^{\mathfrak{a}} \psi$
$\mathfrak{M} \models^{\mathfrak{a}} \varphi \Leftrightarrow \psi$	iff $\mathfrak{M} \models^{\mathfrak{a}} \varphi \Rightarrow \psi$ and $\mathfrak{M} \models^{\mathfrak{a}} \psi \Rightarrow \varphi$
$\mathfrak{M} \models^{\mathfrak{a}} \exists x\varphi$	iff $\mathfrak{M} \models^{\mathfrak{b}} \varphi$ for some assignment $\mathfrak{b}$ that may differ from $\mathfrak{a}$ only in x
$\mathfrak{M} \models^{\mathfrak{a}} \forall x\varphi$	iff $\mathfrak{M} \models^{\mathfrak{b}} \varphi$ for every assignment $\mathfrak{b}$ that may differ from $\mathfrak{a}$ only in x

Figure 2.2: Truth-Relation for First-Order Logic

temporal operators $\Box$ ('always in the future'), $\Diamond$ ('eventually in the future'), $\bigcirc$ ('at the next moment'), U ('until') and W ('weak until').

The set of FOTL formulae is defined as follows: $\top$ and $\bot$ are FOTL formulae; if P is an n-ary predicate symbol and $t_1, \dots, t_n$ are variables or constants, then $P(t_1, \dots, t_n)$ is an *atomic* FOTL formula; if φ and ψ are FOTL formulae, then so are

$$\neg\varphi, \varphi \vee \psi, \varphi \wedge \psi, \varphi \Rightarrow \psi, \varphi \Leftrightarrow \psi, \exists x\varphi, \forall x\varphi, \Box\varphi, \Diamond\varphi, \bigcirc\varphi, \varphi \,\mathsf{U}\, \psi, \text{ and } \varphi \,\mathsf{W}\, \psi.$$

An occurrence of a variable x in a FOTL formula φ is *bound* if and only if it occurs within a subformula ψ of φ such that $\exists x\psi$ or $\forall x\psi$ is also a subformula of φ; otherwise, the occurrence is *free*. A FOTL formula φ is said to be *open* if and only if it contains at least one free variable, and it is *closed* otherwise. For a given formula φ, we write $\varphi(x_1, \dots, x_n)$ to indicate that all the free variables of φ are among $x_1, \dots, x_n$. The set of variables contained in a term $t \in T_{CS \cup FS}(X)$ or FOTL formula φ (i.e. bound and free variables) is denoted by $var(t)$ and $var(\varphi)$, respectively; for the set of constants contained in the term t or FOTL formula φ, we write $const(t)$ and $const(\varphi)$, respectively. A term $t \in T_{CS \cup FS}(X)$ or FOTL formula φ is said to be *ground* if and only if $var(t) = \emptyset$ or $var(\varphi) = \emptyset$ holds, respectively.

Formulae of this logic are interpreted over structures $\mathfrak{M} = (D_n, I_n)_{n \in \mathbb{N}}$ that associate with each element n of $\mathbb{N}$, representing a moment in time, a first-order structure $\mathfrak{M}_n = (D_n, I_n)$ with its own non-empty domain D_n and interpretation I_n. An *assignment* $\mathfrak{a}$ is a function from the set of variables X to $\bigcup_{n \in \mathbb{N}} D_n$. The interpretation of terms $I_n^{\mathfrak{a}}$ induced by an interpretation I_n for an assignment $\mathfrak{a}$ is defined analogously to the first-order case.

The definition of the *truth-relation* $\mathfrak{M}_n \models^{\mathfrak{a}} \varphi$ (only for those $\mathfrak{a}$ such that $\mathfrak{a}(x) \in D_n$ for every variable x) is given in Figure 2.3.

In this thesis we make the *expanding domain assumption*, that is, $D_n \subseteq D_m$ if $n < m$, and we assume that the interpretation of constants is *rigid*, that is, $I_n(c) = I_m(c)$ for all $n, m \in \mathbb{N}$ and constants $c \in CS$.

A structure $\mathfrak{M} = (D_n, I_n)_{n \in \mathbb{N}}$ is said to be a *model* for a formula φ if and only if for every assignment $\mathfrak{a}$ with $\mathfrak{a}(x) \in D_0$ for every variable x it holds that $\mathfrak{M}_0 \models^{\mathfrak{a}} \varphi$. A formula

$\mathfrak{M}_n \models^{\mathfrak{a}} \top$	
$\mathfrak{M}_n \not\models^{\mathfrak{a}} \bot$	
$\mathfrak{M}_n \models^{\mathfrak{a}} P(t_1,\ldots,t_m)$	iff $(I_n^{\mathfrak{a}}(t_1),\ldots,I_n^{\mathfrak{a}}(t_m)) \in I_n(P)$
$\mathfrak{M}_n \models^{\mathfrak{a}} \neg\varphi$	iff not $\mathfrak{M}_n \models^{\mathfrak{a}} \varphi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \varphi \vee \psi$	iff $\mathfrak{M}_n \models^{\mathfrak{a}} \varphi$ or $\mathfrak{M}_n \models^{\mathfrak{a}} \psi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \varphi \wedge \psi$	iff $\mathfrak{M}_n \models^{\mathfrak{a}} \varphi$ and $\mathfrak{M}_n \models^{\mathfrak{a}} \psi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \varphi \Rightarrow \psi$	iff $\mathfrak{M}_n \models^{\mathfrak{a}} \neg\varphi$ or $\mathfrak{M}_n \models^{\mathfrak{a}} \psi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \varphi \Leftrightarrow \psi$	iff $\mathfrak{M}_n \models^{\mathfrak{a}} \varphi \Rightarrow \psi$ and $\mathfrak{M}_n \models^{\mathfrak{a}} \psi \Rightarrow \varphi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \exists x\varphi$	iff $\mathfrak{M}_n \models^{\mathfrak{b}} \varphi$ for some assignment $\mathfrak{b}$ that may differ from $\mathfrak{a}$ only in x and such that $\mathfrak{b}(x) \in D_n$
$\mathfrak{M}_n \models^{\mathfrak{a}} \forall x\varphi$	iff $\mathfrak{M}_n \models^{\mathfrak{b}} \varphi$ for every assignment $\mathfrak{b}$ that may differ from $\mathfrak{a}$ only in x and such that $\mathfrak{b}(x) \in D_n$
$\mathfrak{M}_n \models^{\mathfrak{a}} \bigcirc\varphi$	iff $\mathfrak{M}_{n+1} \models^{\mathfrak{a}} \varphi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \Diamond\varphi$	iff there exists $m \geq n$ such that $\mathfrak{M}_m \models^{\mathfrak{a}} \varphi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \Box\varphi$	iff for all $m \geq n$, $\mathfrak{M}_m \models^{\mathfrak{a}} \varphi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \varphi \,\mathsf{U}\, \psi$	iff there exists $m \geq n$ such that $\mathfrak{M}_m \models^{\mathfrak{a}} \psi$ and $\mathfrak{M}_i \models^{\mathfrak{a}} \varphi$ for every $i, n \leq i < m$
$\mathfrak{M}_n \models^{\mathfrak{a}} \varphi \mathsf{W} \psi$	iff $\mathfrak{M}_n \models^{\mathfrak{a}} \varphi \,\mathsf{U}\, \psi$ or $\mathfrak{M}_n \models^{\mathfrak{a}} \Box\varphi$

Figure 2.3: Truth-Relation for First-Order Temporal Logic

is *satisfiable* if and only there exists a model for φ. A formula φ is *valid* if and only if every temporal structure $\mathfrak{M} = (D_n, I_n)_{n \in \mathbb{N}}$ is a model for φ.

2.3.1 The Monodic Fragment

We can now define the notion of *monodic* FOTL formulae.

Definition 2.3.2. *A formula φ of* FOTL *is called* monodic *if and only if any subformula of φ that is of the form $\bigcirc\psi$, $\Box\psi$, $\Diamond\psi$, $\psi_1 \,\mathsf{U}\, \psi_2$, or $\psi_1 \,\mathsf{W}\, \psi_2$ contains at most one free variable.*

For example, the formulae $\exists x \Box \forall y P(x,y)$ and $\forall x \Box P(c,x)$ are monodic, whereas the formula $\forall x \exists y (Q(x,y) \Rightarrow \Box Q(x,y))$ is not monodic.

The set of valid formulae of FOTL is not recursively enumerable [87,88]. However, the set of valid *monodic* formulae is known to be finitely axiomatisable [94].

2.4 Additional Notions

We now introduce some important notions that are used in the subsequent sections and chapters. First of all, we define the concept of substitutions and some associated terminology like the domain and co-domain of substitutions.

Definition 2.4.1. *Let $\Sigma \subseteq CS \cup FS$. A* substitution *$\sigma\colon X \to T_\Sigma(X)$ is a function from variables to terms such that there are only finitely many variables $x \in X$ with $\sigma(x) \neq x$. By id we denote the identity substitution, which is defined as $id(x) = x$ for every variable $x \in X$.*

The set of all the substitutions over the variable set X and signature Σ is denoted by $Subst_{\Sigma}(X)$.

Definition 2.4.2. *Let $\Sigma \subseteq CS \cup FS$ and let $\sigma \colon X \rightarrow T_{\Sigma}(X)$ be a substitution.*

(i) The domain *$dom(\sigma)$ of the substitution σ is defined as follows:*

$$dom(\sigma) = \{ x \in X \mid \sigma(x) \neq x \}$$

If $dom(\sigma) = \{x_1, \dots, x_n\}$, then the substitution σ will also be represented as

$$\sigma = [x_1 \mapsto \sigma(x_1), \dots, x_n \mapsto \sigma(x_n)] .$$

(ii) The co-domain *$codom(\sigma)$ of the substitution σ is defined in the following way:*

$$codom(\sigma) = \{ \sigma(x) \mid x \in dom(\sigma) \}$$

Then, the application of substitutions on terms, atoms, literals and formulae is defined as follows.

Definition 2.4.3 (Application of Substitutions on Terms). *Let $\Sigma \subseteq CS \cup FS$, let $\sigma \colon X \rightarrow T_{\Sigma}(X)$ be a substitution and let $t \in T_{\Sigma}(X)$ be a term. Then, we inductively define a term $t\sigma \in T_{\Sigma}(X)$, which is the result of the* application of the substitution *σ on the term t, in the following way:*

- *For $t = x \in X$:*

$$t\sigma = x\sigma = \sigma(x)$$

- *If $t = f(t_1, \dots, t_n)$ for $t_1, \dots, t_n \in T_{\Sigma}(X)$:*

$$t\sigma = f(t_1\sigma, \dots, t_n\sigma)$$

Two substitutions can be composed in the following way.

Definition 2.4.4 (Composition of Substitutions). *Let $\sigma, \tau \colon X \rightarrow T_{\Sigma}(X)$ be substitutions. Then the composition $\sigma\tau \colon X \rightarrow T_{\Sigma}(X)$ of the two substitutions σ and τ is a substitution defined as follows. For $x \in X$, let*

$$(\sigma\tau)(x) = (x\sigma)\tau.$$

Then, the application of substitutions to atoms, literals and clauses is defined as follows.

Definition 2.4.5 (Application of Substitutions on Formulae). *Let $\Sigma \subseteq CS \cup FS$, let $\sigma \colon X \rightarrow T_{\Sigma}(X)$ be a substitution and let F be a* FOTL *formula built over Σ and PS. Then, we inductively define a* FOTL *formula $F\sigma$ built over Σ and PS, which is the result of the* application of the substitution *σ on the formula F, in the following way:*

- ***true** σ = **true***

- $\textit{false}\,\sigma = \textit{false}$
- *For* $P \in PS$ *and terms* $t_1, \ldots, t_n \in T_\Sigma(X)$:

$$P(t_1, \ldots, t_n)\sigma = P(t_1\sigma, \ldots, t_n\sigma)$$

- *For* FOTL *formulae* G *and* H *built over* Σ *and* PS *such that:*
  - $F\sigma = (\neg G)\sigma = \neg(G\sigma)$
  - $F\sigma = (G \wedge H)\sigma = G\sigma \wedge H\sigma$
  - $F\sigma = (G \vee H)\sigma = G\sigma \vee H\sigma$
  - $F\sigma = (G \Rightarrow H)\sigma = G\sigma \Rightarrow H\sigma$
  - $F\sigma = (\exists x G)\sigma = \exists y(G([x \mapsto y]\sigma))$
  - $F\sigma = (\forall x G)\sigma = \forall y(G([x \mapsto y]\sigma))$
  - $F\sigma = (\bigcirc G)\sigma = \bigcirc(G\sigma)$
  - $F\sigma = (\Box G)\sigma = \Box(G\sigma)$
  - $F\sigma = (\Diamond G)\sigma = \Diamond(G\sigma)$
  - $F\sigma = (G \,\mathsf{U}\, H)\sigma = (G\sigma)\,\mathsf{U}\,(H\sigma)$
  - $F\sigma = (G \,\mathsf{W}\, H)\sigma = (G\sigma)\,\mathsf{W}\,(H\sigma)$

  where y *is a fresh variable such that* $y \notin var(G)$, $y \notin dom(\sigma)$ *and* $y \notin var(codom(\sigma))$.

As a first simple observation, we note that the composition of substitutions is associative.

Lemma 2.4.6. *Let* $\Sigma \subseteq CS \cup FS$, *let* $\sigma, \tau, \upsilon \colon X \to T_\Sigma(X)$ *be substitutions, let* $t \in T_\Sigma(X)$ *be a term and let* F *be a* FOTL *formula built over* Σ *and* PS.
Then it holds that $(\sigma\tau)\upsilon = \sigma(\tau\upsilon)$, $t((\sigma\tau)\upsilon) = t(\sigma(\tau\upsilon))$ *and* $F((\sigma\tau)\upsilon) = F(\sigma(\tau\upsilon))$.

Finally, we define certain properties of substitutions.

Definition 2.4.7. *We say that a substitution* $\sigma \colon X \to T_\Sigma(X)$ *is*

(i) *a* variable renaming *if and only* $dom(\sigma) = codom(\sigma)$;

(ii) invertible *if and only if there exists a substitution* $\sigma^{-1} \colon X \to T_\Sigma(X)$ *such that* $\sigma\sigma^{-1} = id$;

(iii) idempotent *if and only if* $\sigma\sigma = \sigma$;

(iv) ground *if and only if for every variable* $x \in dom(\sigma)$: $\sigma(x) \in T_\Sigma(\emptyset)$;

(v) grounding *for a set* $\mathcal{N}$ *of terms or* FOTL *formulae if and only if for every term or* FOTL *formula* $\varphi \in \mathcal{N}$, *the term or* FOTL *formula* $\varphi\sigma$ *is ground.*

Definition 2.4.8. *Let σ and τ be two substitutions. We say that the substitution σ is* more general *than the substitution τ, written $\sigma \leq \tau$, if and only if there is a substitution ϱ with $\sigma\varrho = \tau$.*

We now introduce the normal form that we are considering for PLTL and monodic FOTL formulae.

2.5 Divided Separated Normal Form

Every monodic temporal formula can be transformed into an equi-satisfiable normal form, called *divided separated normal form (DSNF)* [58].

Definition 2.5.1. *A* monodic temporal problem P in divided separated normal form (DSNF) *is a quadruple $\langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$, where*

- *the universal part $\mathcal{U}$ and the initial part $\mathcal{I}$ are finite sets of first-order formulae;*
- *the step part $\mathcal{S}$ is a finite set of formulae of the form $p \Rightarrow \bigcirc q$, where p and q are propositions, and $P(x) \Rightarrow \bigcirc Q(x)$, where P and Q are unary predicate symbols and x is a variable;*
- *the eventuality part $\mathcal{E}$ is a finite set of formulae of the form $\Diamond L(x)$ (a* non-ground *eventuality clause) and $\Diamond l$ (a* ground eventuality *clause), where $L(x)$ is a unary non-ground literal with the variable x as its only argument, and l is either a proposition or a unary ground literal.*

We associate with each monodic temporal problem $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ the monodic FOTL formula $\mathcal{I} \wedge \Box\mathcal{U} \wedge \Box\forall x\mathcal{S} \wedge \Box\forall x\mathcal{E}$. When we talk about particular properties of a temporal problem (e.g., satisfiability, validity, logical consequences, etc.) we refer to properties of this associated formula.

The transformation to DSNF is based on a renaming and unwinding technique which substitutes non-atomic subformulae by atomic formulae with new predicate symbols and replaces temporal operators by their fixed-point definitions as described, for example, in [37].

In a first step, the formula for which the divided separated normal form is to be computed is transformed into negation normal form (preserving formulae equivalence), i.e. negations are pushed "inwards" as far as possible until they only precede atomic formulae. Double negations are removed. For example, negations applied on temporal operators are transformed as follows:

$$\begin{aligned}
\neg\Box\varphi(x) &\rightarrow \Diamond\neg\varphi(x)\\
\neg\Diamond\varphi(x) &\rightarrow \Box\neg\varphi(x)\\
\neg\bigcirc\varphi(x) &\rightarrow \bigcirc\neg\varphi(x)\\
\neg(\varphi(x)\,\mathsf{U}\,\psi(x)) &\rightarrow \neg\psi(x)\,\mathsf{W}\,(\neg\varphi(x)\wedge\neg\psi(x))\\
\neg(\varphi(x)\,\mathsf{W}\,\psi(x)) &\rightarrow \neg\psi(x)\,\mathsf{U}\,(\neg\varphi(x)\wedge\neg\psi(x))
\end{aligned}$$

In order to avoid an exponential increase in the formula size, it can be necessary to rename the formula $\neg\psi(x)$ after having transformed the formulae $\varphi(x)\,\mathsf{U}\,\psi(x)$ and $\varphi(x)\,\mathsf{W}\,\psi(x)$ into negation normal form (also see e.g. [72]).

After the transformation into negation normal form, each innermost open subformula $\xi(x)$, whose main connective is a temporal operator, is recursively renamed by $P_\xi(x)$, where $P_{\xi(x)}$ is a new unary predicate, and each innermost closed subformula ζ, whose main connective is a temporal operator, is renamed by p_ζ, where p_ζ is a new propositional symbol. In the terminology of [47], $P_\xi(x)$ and p_ζ are called the *surrogates* of $\xi(x)$ and ζ, respectively. Renaming introduces formulae defining $P_\xi(x)$ and p_ζ of the following form (since we are only interested in satisfiability, we use implications instead of equivalences for renaming positive occurrences of subformulae, see also [72]):

(a) $\Box\forall x(P_\xi(x) \Rightarrow \bigcirc\phi(x))$

(b) $\Box\forall x(P_\xi(x) \Rightarrow \Box\phi(x))$

(c) $\Box\forall x(P_\xi(x) \Rightarrow \phi(x)\,\mathsf{U}\,\psi(x))$

(d) $\Box\forall x(P_\xi(x) \Rightarrow \phi(x)\,\mathsf{W}\,\psi(x))$

(e) $\Box\forall x(P_\xi(x) \Rightarrow \Diamond\phi(x))$

The renamings introduced for closed subformulae ζ are of a similar form.

A formula of type (a) is already in the required form after the potentially complex formula occurring under the $\bigcirc$ operator has been renamed by a fresh symbol U that is defined by the formula $\Box\forall x(U(x) \Rightarrow \phi(x))$, which is then added to the set of universal formulae.

Formulae of type (b) are reduced in a satisfiability preserving way as follows:

$$\Box\forall x(P_\xi(x) \Rightarrow R(x)) \wedge \Box\forall x(R(x) \Rightarrow \bigcirc R(x)) \wedge \Box\forall x(R(x) \Rightarrow \varphi(x))$$

where R is a fresh unary predicate. Then, formulae of type (c) are satisfiability equivalent to:

$$\begin{aligned}\Box\forall x(P_\xi(x) \Rightarrow \Diamond\psi(x)) \wedge \Box\forall x(P_\xi(x) \Rightarrow (\phi(x) \vee \psi(x))) \wedge \Box\forall x(P_\xi(x) \Rightarrow (S(x) \vee \psi(x)))\\ \wedge \Box\forall x(S(x) \Rightarrow \bigcirc(\phi(x) \vee \psi(x)))\\ \wedge \Box\forall x(S(x) \Rightarrow \bigcirc(S(x) \vee \psi(x)))\end{aligned}$$

where S is a fresh unary predicate. Finally, formulae of type (d) can be transformed in a satisfiability equivalent way into:

$$\begin{aligned}\Box\forall x(P_\xi(x) \Rightarrow (\phi(x) \vee \psi(x))) \wedge \Box\forall x(P_\xi(x) \Rightarrow (T(x) \vee \psi(x)))\\ \wedge \Box\forall x(T(x) \Rightarrow \bigcirc(\phi(x) \vee \psi(x)))\\ \wedge \Box\forall x(T(x) \Rightarrow \bigcirc(T(x) \vee \psi(x)))\end{aligned}$$

where T is a fresh unary predicate. One has to note that new formulae of type (e) are introduced during the transformation of type (c) formulae. In a final step then, formulae of type (e) are first simplified to $\Box\forall x(P_\xi(x) \Rightarrow \Diamond L(x))$, by renaming complex formulae under the $\Diamond$ operator through fresh unary predicates L, and are then transformed in a satisfiability preserving way as follows:

$$\begin{aligned}&\Box\forall x((P_\xi(x) \wedge \neg L(x)) \Rightarrow \mathit{waitForL}(x))\\ \wedge\ &\Box\forall x(\mathit{waitForL}(x) \Rightarrow \bigcirc(\mathit{waitForL}(x) \vee L(x)))\\ \wedge\ &\Box\forall x\Diamond\neg\mathit{waitForL}(x)\end{aligned}$$

where *waitForL* is a fresh unary predicate. After renaming complex subformulae under $\bigcirc$ operators for newly introduced formulae of type (a), the transformation is complete. We hence obtain the following theorem:

Theorem 2.5.2 (see [23], Theorem 3.4). *Any monodic formula in first-order temporal logic can be transformed into an equi-satisfiable monodic temporal problem in DSNF with at most a linear increase in the size of the problem.*

We now illustrate how the following monodic FOTL formula $\mathcal{F}$ can be transformed into DSNF. Let

$$\begin{aligned}\mathcal{F} =\ &\Box\forall x\exists y\forall v\forall w(Q(y,x) \wedge (P(x,v) \Rightarrow P(y,w)))\\ \wedge\ &\Box\forall x(F(x) \Rightarrow \neg\neg\bigcirc\Box(G(x) \vee H(x)))\\ \wedge\ &\Box\forall x(F(x) \Rightarrow \neg(K(x)\,\mathsf{W}\,F(x)))\\ \wedge\ &F(c).\end{aligned}$$

We can observe that the formulae $\Box\forall x\exists y\forall v\forall w(Q(y,x) \wedge (P(x,v) \Rightarrow P(y,w)))$ and $F(c)$ are already in DSNF, being universal and initial formulae, respectively. The computation of the negation normal form for the formula $\Box\forall x(F(x) \Rightarrow \neg\neg\bigcirc\Box(G(x) \vee H(x)))$ yields

$$\Box\forall x(F(x) \Rightarrow \bigcirc\Box(G(x) \vee H(x))),$$

and the negation normal form of the formula $\Box\forall x(F(x) \Rightarrow \neg(K(x)\,\mathsf{W}\,F(x)))$ is

$$\Box\forall x(F(x) \Rightarrow (\neg F(x)\,\mathsf{U}\,(\neg K(x) \wedge \neg F(x)))).$$

We then rename the subformula $\Box(G(x) \vee H(x))$ of the formula $\Box\forall x(F(x) \Rightarrow \bigcirc\Box(G(x) \vee H(x)))$ by a fresh unary predicate U_1. Consequently, we obtain the formulae

$$\Box\forall x(F(x) \Rightarrow \bigcirc U_1(x))$$

and

$$\Box\forall x(U_1(x) \Rightarrow \Box(G(x) \vee H(x))).$$

We can thus observe that the former formula is already in DSNF, whereas the latter formula is of type (b).

As all the considered formulae are now of the types (a) - (e), we can apply the reductions described above. The formula $\Box\forall x(U_1(x) \Rightarrow \Box(G(x) \vee H(x)))$ is transformed into

$$\Box\forall x(U_1(x) \Rightarrow R(x)) \wedge \Box\forall x(R(x) \Rightarrow \bigcirc R(x)) \wedge \Box\forall x(R(x) \Rightarrow (G(x) \vee H(x))).$$

And the formula $\Box\forall x(F(x) \Rightarrow (\neg F(x) \,\mathsf{U}\, (\neg K(x) \wedge \neg F(x))))$ becomes

$$\begin{aligned} &\Box\forall x(F(x) \Rightarrow \Diamond(\neg K(x) \wedge \neg F(x))) \\ \wedge\, &\Box\forall x(F(x) \Rightarrow (\neg F(x) \vee (\neg K(x) \wedge \neg F(x)))) \\ \wedge\, &\Box\forall x(F(x) \Rightarrow (S(x) \vee (\neg K(x) \wedge \neg F(x)))) \\ \wedge\, &\Box\forall x(S(x) \Rightarrow \bigcirc(\neg F(x) \vee (\neg K(x) \wedge \neg F(x)))) \\ \wedge\, &\Box\forall x(S(x) \Rightarrow \bigcirc(S(x) \vee (\neg K(x) \wedge \neg F(x)))). \end{aligned}$$

In a final step formulae of type (e) have to be transformed, which only concerns the formula $\Box\forall x(F(x) \Rightarrow \Diamond(\neg K(x) \wedge \neg F(x)))$ in our example. The transformation results in the formula:

$$\Box\forall x(F(x) \Rightarrow \Diamond L(x)) \wedge \Box(\forall x(L(x) \Rightarrow (\neg K(x) \wedge \neg F(x))))$$

The first subformulae under the conjunction is then transformed in the following way:

$$\begin{aligned} &\Box\forall x((F(x) \wedge \neg L(x)) \Rightarrow \mathit{waitForL}(x)) \\ \wedge\, &\Box\forall x(\mathit{waitForL}(x) \Rightarrow \bigcirc(\mathit{waitForL}(x) \vee L(x))) \\ \wedge\, &\Box\forall x\Diamond\neg\mathit{waitForL}(x) \end{aligned}$$

Now, complex subformulae under $\bigcirc$ operators remain to be renamed. We obtain the following formulae:

$$\begin{aligned} &\Box\forall x(S(x) \Rightarrow \bigcirc U_2(x)) \\ \wedge\, &\Box\forall x(U_2(x) \Rightarrow (\neg F(x) \vee (\neg K(x) \wedge \neg F(x)))), \\ &\Box\forall x(S(x) \Rightarrow \bigcirc U_3(x)) \\ \wedge\, &\Box\forall x(U_3(x) \Rightarrow (S(x) \vee (\neg K(x) \wedge \neg F(x)))), \\ &\Box\forall x(\mathit{waitForL}(x) \Rightarrow \bigcirc U_4(x)) \\ \wedge\, &\Box\forall x(U_4(x) \Rightarrow (\mathit{waitForL}(x) \vee L(x))) \end{aligned}$$

Thus, the DSNF obtained for the formula $\mathcal{F}$ is the monodic temporal problem

$$\begin{aligned} \mathsf{P}_{\mathcal{F}} = \langle\{&\forall x\exists y\forall v\forall w(Q(y,x) \wedge (P(x,v) \Rightarrow P(y,w))), \\ &\forall x(U_1(x) \Rightarrow R(x)), \\ &\forall x(R(x) \Rightarrow (G(x) \vee H(x))), \\ &\forall x(F(x) \Rightarrow (\neg F(x) \vee (\neg K(x) \wedge \neg F(x)))), \\ &\forall x(F(x) \Rightarrow (S(x) \vee (\neg K(x) \wedge \neg F(x)))), \\ &\forall x(L(x) \Rightarrow (\neg K(x) \wedge \neg F(x))), \end{aligned}$$

$$
\begin{aligned}
&\forall x((F(x) \land \neg L(x)) \Rightarrow \mathit{waitForL}(x)),\\
&\forall x(U_2(x) \Rightarrow (\neg F(x) \lor (\neg K(x) \land \neg F(x)))),\\
&\forall x(U_3(x) \Rightarrow (S(x) \lor (\neg K(x) \land \neg F(x)))),\\
&\forall x(U_4(x) \Rightarrow (\mathit{waitForL}(x) \lor L(x)))\},\\
&\{F(c)\},\\
&\{F(x) \Rightarrow \bigcirc U_1(x),\\
&\ R(x) \Rightarrow \bigcirc R(x),\\
&\ S(x) \Rightarrow \bigcirc U_2(x),\\
&\ S(x) \Rightarrow \bigcirc U_3(x),\\
&\ \mathit{waitForL}(x) \Rightarrow \bigcirc U_4(x)\},\\
&\{\Diamond\neg \mathit{waitForL}(x)\}\rangle
\end{aligned}
$$

The main purpose of the divided separated normal form is to cleanly separate different temporal aspects of a FOTL formula from each other. Sometimes we additionally require the left-hand sides of step clauses in a temporal problem to be unique.

Definition 2.5.3. *A temporal problem $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ is said to have* unique left-hand sides in step clauses *if and only if there are no two step clauses $F \Rightarrow \bigcirc G, F' \Rightarrow \bigcirc G' \in \mathcal{S}$ with $(F \Rightarrow \bigcirc G) \neq (F' \Rightarrow \bigcirc G')$ and $F = F'$.*

Remark 2.5.4. *For example, if a temporal problem $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ contains step clauses $P(x) \Rightarrow \bigcirc Q(x), P(x) \Rightarrow \bigcirc Q'(x) \in \mathcal{S}$ with $Q(x) \neq Q'(x)$, then their left-hand sides can be rendered unique by replacing the two step clauses by a new step clause $P(x) \Rightarrow \bigcirc U(x)$ and a universal formulae $\Box\forall x(U(x) \Rightarrow (Q(x) \land Q'(x)))$ for a fresh unary predicate symbol U.*

For the resolution calculi in this thesis we will need to go one step further by transforming the universal and initial part of a monodic temporal problem into clause normal form. In the next section we show how monodic temporal problems in DSNF can be clausified.

2.6 Clausification of Temporal Problems

For first-order formulae the clausification of a closed formula $\mathcal{G}$ can be briefly described as follows (see e.g. [63] for more details).

First of all, the prenex normal form of $\mathcal{G}$, *Prenex*$(\mathcal{G})$, has to be computed, i.e. equivalences are replaced by conjunctions of implications, and quantifiers are moved outside of the formula. This transformation is performed by applying the following rewrite rules on innermost subformulae until the required form is obtained:

$$
\begin{aligned}
F \Leftrightarrow G &\rightarrow (F \Rightarrow G) \land (G \Rightarrow F)\\
\neg QxF &\rightarrow \bar{Q}x\neg F\\
(QxF) \land G &\rightarrow Qy(F[x \mapsto y] \land G)
\end{aligned}
$$

$$\begin{aligned}
(QxF) \vee G &\rightarrow Qy(F[x \mapsto y] \vee G) \\
(QxF) \Rightarrow G &\rightarrow \bar{Q}y(F[x \mapsto y] \Rightarrow G) \\
F \wedge QxG &\rightarrow Qy(F \wedge G[x \mapsto y]) \\
F \vee QxG &\rightarrow Qy(F \vee G[x \mapsto y]) \\
F \Rightarrow QxG &\rightarrow Qy(F \Rightarrow G[x \mapsto y]),
\end{aligned}$$

where F, G are arbitrary first-order formulae, $Q \in \{\forall, \exists\}$, $\bar{\exists} = \forall$, $\bar{\forall} = \exists$, and y is a fresh variable. The transformation results in a formula

$$\mathcal{G} \equiv Q_1 x_1 \ldots Q_m x_m : G,$$

where G is quantifier-free and $Q_i \in \{\forall, \exists\}$ for every i, $1 \leq i \leq m$. In the second step, also called *Skolemization* step, the existential quantifiers $\exists$ are removed, starting from the left-most quantifier in the prenex normal form. The variables bound by existential quantifiers are replaced by fresh constants or function symbols. The arity of the newly created constants or function symbols is determined by the number of universal quantifiers that precede the considered existential quantifier in the prenex normal form. The arguments of the fresh function symbols will be exactly the variables bound by the preceding universal quantifiers in the prenex normal form. After the Skolemization step, we have obtained the form

$$\mathcal{G} \leadsto \forall x_1 \ldots \forall x_n : H.$$

One has to observe that Skolemization usually does not preserve formulae equivalence but only satisfiability, which is sufficient for our purposes. We still have to note that there exist other variants of Skolemization, which can be found, for example, in [72].

Finally, the formula H has to be brought into conjunctive normal form (CNF). The transformation can be performed by applying the following rewrite rules recursively on the formula H.

$$\begin{aligned}
F \Leftrightarrow G &\rightarrow (F \Rightarrow G) \wedge (G \Rightarrow F) \\
F \Rightarrow G &\rightarrow \neg F \vee G \\
\neg(F \wedge G) &\rightarrow \neg F \vee \neg G \\
\neg(F \vee G) &\rightarrow \neg F \wedge \neg G \\
\neg\neg F &\rightarrow F \\
(F \wedge G) \vee H &\rightarrow (F \vee G) \wedge (F \vee H)
\end{aligned}$$

where F, G and H are quantifier-free first-order formulae. H is then brought in the form

$$H \leadsto \bigwedge_{i=1}^{m} \bigvee_{j=1}^{n_i} L_j^i$$

where L_{i_j} is a positive or negative atom, or *literal*, for every i_j, $1 \leq i \leq m$, $1 \leq j \leq n_i$. In order to avoid an exponential increase in the size of the formula obtained by transformation to CNF, subformulae might have to be renamed during the CNF transformation process [72]. The set

$$\{ L^i_j \mid 1 \leq i \leq m, 1 \leq j \leq n_i \}$$

will be called the *clause normal form* of the first-order formula $\mathcal{G}$.

Definition 2.6.1. *We define that*

- *a* literal *is either a positive or negative atom, i.e.* $P(t_1, \ldots, t_n)$ *or* $\neg P(t_1, \ldots, t_n)$, *respectively, where* $P/n \in PS$ *and* $t_1, \ldots, t_n \in T_\Sigma(X)$*;*
- *a* clause *is a multiset of literals, written as* $L_1 \vee \ldots \vee L_n$*, where* $L_1, \ldots, L_n$ *are literals. The* empty clause *is denoted by* $\perp$*.*
- *a clause is said to be* positive/negative *if and only if it only contains* positive/negative *literals.*

Now, the clausification of a monodic temporal problem P is defined as follows. Note that the usual first-order clausification process is performed for initial and universal formulae.

Definition 2.6.2. *Let* $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ *be a monodic temporal problem. The* clausification $\mathsf{Cls(P)}$ *of* P *is a quadruple* $\langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}' \rangle$ *such that*

- $\mathcal{U}'$ *is a set of clauses, called* universal clauses, *obtained by clausification of* $\mathcal{U}$*;*
- $\mathcal{I}'$ *is a set of clauses, called* initial clauses, *obtained by clausification of* $\mathcal{I}$*;*
- $\mathcal{S}'$ *is the smallest set of step clauses such that all step clauses from* $\mathcal{S}$ *are in* $\mathcal{S}'$ *and for every non-ground step clause* $P(x) \Rightarrow \bigcirc L(x)$ *in* $\mathcal{S}$ *and every constant* c *occurring in* P*, the clause* $P(c) \Rightarrow \bigcirc L(c)$ *is in* $\mathcal{S}'$*;*
- $\mathcal{E}'$ *is the smallest set of eventuality clauses such that all eventuality clauses from* $\mathcal{E}$ *are in* $\mathcal{E}'$ *and for every non-ground eventuality clause* $\Diamond L(x)$ *in* $\mathcal{E}$ *and every constant* c *occurring in* P*, the eventuality clause* $\Diamond L(c)$ *is in* $\mathcal{E}'$*.*

One has to note that new constants and, especially, function symbols of an arbitrary arity can be introduced during the Skolemization process[1]. As a consequence it is not generally possible to instantiate every variable that occurs in the original problem with all the constants and function symbols. On the other hand, the variables occurring in the step and eventuality clauses have to be instantiated with the constants that are present in the *original* problem (before Skolemization) in order to ensure the completeness of the resolution-based calculus presented in Chapter 3.

The notion of step clause can be generalised as follows.

[1] The definition of term interpretations and first-order interpretations has to be extended in the usual way for FOL in order to be able to handle function symbols.

Definition 2.6.3 (Arbitrary Step Clause). *A (arbitrary) step clause is a formula of the form*

$$\bigwedge_{i=1}^{m} A_i \Rightarrow \bigcirc \bigvee_{j=1}^{n} L_j$$

*where $A_1, \ldots, A_m$ are at most unary predicates and $L_1, \ldots, L_n$ are literals. For $m = 0$ we define $\bigwedge_{i=1}^{m} A_i =$ **true**, and for $n = 0$ we set $\bigvee_{j=1}^{n} L_j =$ **false**.*

Step clauses of the latter form can be derived by the calculus defined in Chapter 3. The clausification of the monodic temporal problem $\mathsf{P}_{\mathcal{F}}$ of Section 2.5 yields the following clausified temporal problem

$$\begin{aligned}
\mathsf{Cls}(\mathsf{P}_{\mathcal{F}}) = \langle\{ & Q(f(x), x), \\
& \neg P(x, v) \vee P(f(x), w), \\
& \neg U_1(x) \vee R(x), \\
& \neg R(x) \vee G(x) \vee H(x), \\
& \neg F(x) \vee \neg K(x), \neg F(x), \\
& \neg F(x) \vee S(x) \vee \neg K(x), \\
& \neg F(x) \vee S(x), \\
& \neg L(x) \vee \neg K(x), \\
& \neg L(x) \vee \neg F(x), \\
& \neg F(x) \vee L(x) \vee \mathit{waitForL}(x), \\
& \neg U_2(x) \vee \neg F(x) \vee \neg K(x), \\
& \neg U_2(x) \vee \neg F(x), \\
& \neg U_3(x) \vee S(x) \vee \neg K(x), \\
& \neg U_3(x) \vee S(x) \vee \neg F(x), \\
& \neg U_4(x) \vee \mathit{waitForL}(x) \vee L(x)\}, \\
& \{F(c)\}, \\
& \{F(x) \Rightarrow \bigcirc U_1(x), \\
& \ R(x) \Rightarrow \bigcirc R(x), \\
& \ S(x) \Rightarrow \bigcirc U_2(x), \\
& \ S(x) \Rightarrow \bigcirc U_3(x), \\
& \ \mathit{waitForL}(x) \Rightarrow \bigcirc U_4(x), \\
& \ F(c) \Rightarrow \bigcirc U_1(c), \\
& \ R(c) \Rightarrow \bigcirc R(c), \\
& \ S(c) \Rightarrow \bigcirc U_2(c), \\
& \ S(c) \Rightarrow \bigcirc U_3(c), \\
& \ \mathit{waitForL}(c) \Rightarrow \bigcirc U_4(c)\},
\end{aligned}$$

$$\{\Diamond\neg \mathit{waitForL}(x)\}\rangle,$$

where f is a fresh unary function symbol.

Finally, we introduce the notions of clause equality, clause set inclusion, and clause set equality *up to variable renaming.*

Definition 2.6.4. *We say that two clauses $\mathcal{C}$ and $\mathcal{D}$ are* equal up to renaming, *written $\mathcal{C} =_X \mathcal{D}$, if and only if there exists a variable renaming σ such that $\mathcal{C}\sigma = \mathcal{D}$.*

Definition 2.6.5. *Let $\mathcal{C}$ be a clause and $\mathcal{N}$ and $\mathcal{M}$ be sets of clauses. Then we say*

- *that the clause $\mathcal{C}$ is contained in the set $\mathcal{M}$, written $\mathcal{C} \in_X \mathcal{M}$, if and only if there exists a variable renaming σ such that $\mathcal{C}\sigma \in \mathcal{M}$, and*
- *that the set $\mathcal{M}$ is included in the set $\mathcal{N}$* up to variable renaming, *written $\mathcal{M} \subseteq_X \mathcal{N}$, if and only if for every clause $\mathcal{C} \in \mathcal{M}$ it holds that $\mathcal{C} \in_X \mathcal{N}$, and*
- *that the sets $\mathcal{M}$ and $\mathcal{N}$ are* equal up to variable renaming, *written $\mathcal{M} =_X \mathcal{N}$, if and only if $\mathcal{M} \subseteq_X \mathcal{N}$ and $\mathcal{N} \subseteq_X \mathcal{M}$ holds.*

2.7 Summary

In this chapter we gave a formal definition of the syntax and semantics for the two formal languages we are considering in this thesis: propositional linear-time temporal logic and monodic first-order temporal logic. Both of these logics are interpreted over a model of time that is isomorphic to the natural numbers.

First, we presented the syntax and semantics of propositional linear-time temporal logic. We then defined the syntax and semantics of first-order temporal logic, and we gave a formal definition of the monodic fragment. We also introduced some notions that are essential for the following parts of this thesis. Subsequently, we described the normal form for PLTL and monodic FOTL formulae that we consider in this thesis. Finally, we illustrated how the formulae that are in normal form can be clausified for the resolution-based calculus that is introduced in Chapter 3.

Chapter 3

Ordered Fine-Grained Resolution with Selection

3.1 Introduction

For monodic first-order temporal logic a first resolution-based calculus, called *monodic temporal resolution*, was introduced in [23]. The calculus operates on temporal problems in DSNF and uses special inference rules to handle eventuality formulae and resolution-like inference rules for the remaining DSNF formulae types.

However, as most inference rules of monodic temporal resolution have applicability conditions that are computationally too complex to be implemented efficiently, a more machine-oriented resolution-based calculus, the fine-grained (temporal) resolution calculus, has been introduced in [58], which accepts problems in DSNF that have been clausified. The computationally complex inference rules of monodic temporal resolution have been replaced by resolution-based inference rules that operate on the different temporal clause types of clausified DSNF. In this regard fine-grained resolution can be seen as being more fine-grained than monodic temporal resolution as it performs "smaller" inference steps by manipulating temporal clauses. Moreover, a special algorithm called FG-BFS has been developed which allows to exhaustively search for the premises of the eventuality resolution rules. As a refinement, the *ordered fine-grained (temporal) resolution with selection calculus* has been presented in [51]. It extends fine-grained resolution with ordering restrictions and selection functions. In this chapter we prove the refutational completeness of ordered fine-grained resolution with selection.

This chapter is structured as follows. We first recall the inference rules of monodic temporal resolution. We then introduce the ordered fine-grained temporal resolution with selection calculus before we focus on its proof of refutational completeness. For the completeness proof we define a refined version of monodic temporal resolution, for which we also prove that it is refutationally complete. Finally, we show that derivations of refined monodic temporal resolution can be simulated by ordered fine-grained resolution with selection, which obviously implies its refutational completeness.

3.2 Monodic Temporal Resolution

We begin by defining some notions that are necessary for presenting the inference rules of monodic temporal resolution. More specifically, we introduce the concepts of *constant flooding* and of *derived*, *merged derived* and *full merged* step clauses.

Definition 3.2.1. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem. Then, we say that the temporal problem $P^c = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}^c \rangle$ where*

$$\mathcal{E}^c = \mathcal{E} \cup \{ \Diamond L(c) \mid \Diamond L(x) \in \mathcal{E}, c \textit{ is a constant in } P \}$$

is the constant flooded form *of P. Evidently, P^c is satisfiability equivalent to P.*

Definition 3.2.2. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem.*

(i) Let

$$P_{i_1}(x) \Rightarrow \bigcirc M_{i_1}(x), \ldots, P_{i_k}(x) \Rightarrow \bigcirc M_{i_k}(x) \tag{3.1}$$

with $k \geq 0$ be a subset of step clauses contained in the set $\mathcal{S}$. Then formulae of the form

$$P_{i_j}(c) \Rightarrow \bigcirc M_{i_j}(c) \qquad \textit{and} \qquad \exists x \bigwedge_{j=1}^{k} P_{i_j}(x) \Rightarrow \bigcirc \exists x \bigwedge_{j=1}^{k} M_{i_j}(x), \tag{3.2}$$

where c is a constant in P and $j = 1, \ldots, k$, are called derived *step clauses*[1] *built from the temporal problem P.*

(ii) Let $\{\Phi_1 \Rightarrow \bigcirc \Psi_1, \ldots, \Phi_n \Rightarrow \bigcirc \Psi_n\}$ be a set of derived step clauses or ground *step clauses in P. Then $(\bigwedge_{i=1}^{n} \Phi_i) \Rightarrow \bigcirc(\bigwedge_{i=1}^{n} \Psi_i)$ is called a* merged derived step clause *built from the temporal problem P.*

(iii) Let $\mathcal{A} \Rightarrow \bigcirc \mathcal{B}$ be a merged derived step clause, let $P_1(x) \Rightarrow \bigcirc M_1(x)$, …, $P_k(x) \Rightarrow \bigcirc M_k(x)$ with $k \geq 0$ be a subset of the original step clauses in P, and let

$$\mathcal{A}(x) \overset{\text{def}}{=} \mathcal{A} \wedge \bigwedge_{i=1}^{k} P_i(x) \textit{ and } \mathcal{B}(x) \overset{\text{def}}{=} \mathcal{B} \wedge \bigwedge_{i=1}^{k} M_i(x).$$

Then $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc \mathcal{B}(x))$ is called a full merged step clause *built from the temporal problem P.*

Note that formulae of the form (3.2) are logical consequences of (3.1). In what follows, $\mathcal{A} \Rightarrow \bigcirc \mathcal{B}$ and $\mathcal{A}_i \Rightarrow \bigcirc \mathcal{B}_i$ denote merged derived step clauses, $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc \mathcal{B}(x))$ and $\forall x(\mathcal{A}_i(x) \Rightarrow \bigcirc \mathcal{B}_i(x))$ denote full merged step clauses.

We also define relations $\sqsubset$ and $\sqsubseteq$ on full merged and merged derived step clauses as follows.

[1] In [23] derived step clauses are called e-derived step clauses.

Definition 3.2.3. *Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem. We then define relations $\sqsubset$ and $\sqsubseteq$ on the right-hand sides of the merged derived and full merged step clauses built from the temporal problem P as follows.*

*(i) For every full merged or merged derived step clause $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc \mathcal{B}(x))$ with $\mathcal{B}(x) \neq$ **true**, we set **true** $\sqsubset \mathcal{B}(x)$ and **true** $\sqsubseteq \mathcal{B}(x)$.*

(ii) Additionally, for two full merged or merged derived step clauses $\forall x_0(\mathcal{A}(x_0) \Rightarrow \bigcirc \mathcal{B}(x_0))$, $\forall x_0'(\mathcal{A}'(x_0') \Rightarrow \bigcirc \mathcal{B}'(x_0'))$ and variable renamings η_1, η_2 such that

- *$Prenex(\mathcal{B}(x_0)) = \exists x_1, \ldots, x_p \bigwedge_{i=1}^n L_i$, $Prenex(\mathcal{B}'(x_0')) = \exists x_1', \ldots, x_q' \bigwedge_{j=1}^m L_j'$,*
- *$\{x_0\eta_1, x_1\eta_1, \ldots, x_p\eta_1\} \cap \{x_0'\eta_2, x_1'\eta_2 \ldots, x_q'\eta_2\} = \emptyset$, and*
- *$\{L_1\eta_1\sigma, \ldots, L_n\eta_1\sigma\} \subsetneq \{L_1'\eta_2, \ldots, L_m'\eta_2\}$ as multisets, where σ is a variable renaming,*

we define that $\mathcal{B}(x_0) \sqsubset \mathcal{B}'(x_0')$.

(iii) Finally, for two full merged or merged derived step clauses $\forall x_0(\mathcal{A}(x_0) \Rightarrow \bigcirc \mathcal{B}(x_0))$ and $\forall x_0'(\mathcal{A}'(x_0') \Rightarrow \bigcirc \mathcal{B}'(x_0'))$ such that $\mathcal{B}(x_0) \sqsubset \mathcal{B}'(x_0')$ or such that there exist variable renamings η_1, η_2 with

- *$Prenex(\mathcal{B}(x_0)) = \exists x_1, \ldots, x_p \bigwedge_{i=1}^n L_i$, $Prenex(\mathcal{B}'(x_0')) = \exists x_1', \ldots, x_q' \bigwedge_{j=1}^m L_j'$,*
- *$\{x_0\eta_1, x_1\eta_1, \ldots, x_p\eta_1\} \cap \{x_0'\eta_2, x_1'\eta_2 \ldots, x_q'\eta_2\} = \emptyset$, and*
- *$\{L_1\eta_1\sigma, \ldots, L_n\eta_1\sigma\} = \{L_1'\eta_2, \ldots, L_m'\eta_2\}$ as multisets, where σ is a variable renaming,*

we set $\mathcal{B}(x_0) \sqsubseteq \mathcal{B}'(x_0')$.

Remark 3.2.4. *We extend the previous definition on formulae that are of the same form as the right-hand sides of full merged (or merged derived) step clauses but do not necessarily result from full merged (or merged derived) step clauses.*

Finally, for every temporal problem P we define a set $\mathcal{M}(\mathsf{P})$ of full merged step clauses which do not contain duplicate subformulae.

Definition 3.2.5. *Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem. Then we denote by $\mathcal{M}(\mathsf{P})$ the set of all merged derived step clauses $\mathcal{A} \Rightarrow \bigcirc \mathcal{B}$ and full merged step clauses $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc \mathcal{B}(x))$ built from the temporal problem P such that*

(i) every step clause from the set $\mathcal{S}$ is used at most once in the construction of a derived step clause, and

(ii) every derived step clause and every ground step clause from the set $\mathcal{S}$ is used at most once in the construction of a merged derived step clause, and

(iii) no merged derived step clause contains multiple occurrences of the same ground step clause, and

(iv) every non-ground step clause from the set S is used at most once in the construction of a full-merged step clause, and

(v) for each two derived step clauses $\exists x\, F_1(x) \Rightarrow \bigcirc \exists x\, G_1(x)$ and $\exists x\, F_2(x) \Rightarrow \bigcirc \exists x\, G_2(x)$ that occur in a full-merged step clause $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc \mathcal{B}(x))$ or in a merged derived step clause $\mathcal{A} \Rightarrow \bigcirc \mathcal{B}$, it does not hold that

- *$F_1(x) \subseteq F_2(x)$ and $G_1(x) \subseteq G_2(x)$, or*
- *$F_2(x) \subseteq F_1(x)$ and $G_2(x) \subseteq G_1(x)$.*

Note that case *(iii)* ensures that derived step clauses $P(c) \Rightarrow \bigcirc Q(c)$ which are already contained in the set S do not occur twice in merged derived step clauses.

Remark 3.2.6. *As every temporal problem P only contains finitely many step clauses, it is easy to see that every set of full merged and merged derived clauses $\mathcal{M}(P)$ is finite.*

For example, the full merged step clause

$$\forall x([P(x) \land \exists y R(y) \land \exists z(R(z) \land T(z))] \Rightarrow \bigcirc[Q(x) \land \exists y S(y) \land \exists z(S(z) \land V(z))])$$

does not satisfy the condition (v) of Definition 3.2.5 as $\{R(x)\} \subseteq \{R(x), T(x)\}$ and $\{S(x)\} \subseteq \{S(x), V(x)\}$, but condition (v) is satisfied for the full merged step clause

$$\forall x([P(x) \land \exists y R(y) \land \exists z R(z)] \Rightarrow \bigcirc[Q(x) \land \exists y S(y) \land \exists z V(z)]).$$

We can now define the monodic temporal resolution calculus, $\mathfrak{I}_e$, for the expanding domain case. The inference rules of $\mathfrak{I}_e$ are the following.

- *Step resolution rule w.r.t. $\mathcal{U}$:*

$$\frac{\mathcal{A} \Rightarrow \bigcirc \mathcal{B}}{\neg \mathcal{A}}\ (\bigcirc^{\mathcal{U}}_{res}), \quad \text{if } \mathcal{U} \cup \{\mathcal{B}\} \models \bot.$$

- *Termination rule w.r.t. $\mathcal{U}$ and $\mathcal{I}$:*

$$\frac{}{\bot}\ (\bot^{\mathcal{U}}_{res}), \quad \text{if } \mathcal{U} \cup \mathcal{I} \models \bot.$$

- *Eventuality resolution rule w.r.t. $\mathcal{U}$:*

$$\frac{\forall x(\mathcal{A}_1(x) \Rightarrow \bigcirc \mathcal{B}_1(x)) \quad \cdots \quad \forall x(\mathcal{A}_n(x) \Rightarrow \bigcirc \mathcal{B}_n(x)) \qquad \Diamond L(x)}{\forall x \bigwedge_{i=1}^{n} \neg \mathcal{A}_i(x)}\ (\Diamond^{\mathcal{U}}_{res}),$$

where $\forall x(\mathcal{A}_i(x) \Rightarrow \bigcirc \mathcal{B}_i(x))$ are full merged step clauses such that for every i, $1 \leq i \leq n$, the *loop* side conditions $\forall x(\mathcal{U} \land \mathcal{B}_i(x) \Rightarrow \neg L(x))$ and $\forall x(\mathcal{U} \land \mathcal{B}_i(x) \Rightarrow \bigvee_{j=1}^{n}(\mathcal{A}_j(x)))$ are valid.[2]

The set of full merged step clauses, satisfying the loop side conditions, is called a *loop in* $\Diamond L(x)$ and the formula $\bigvee_{j=1}^{n} \mathcal{A}_j(x)$ is called a *loop formula*.

[2] In the case $\mathcal{U} \models \forall x \neg L(x)$, the *degenerate clause*, **true** $\Rightarrow \bigcirc$**true**, can be considered as a premise of this rule; the conclusion of the rule is then $\neg$**true** $\equiv$ **false**.

- *Ground eventuality resolution rule w.r.t. $\mathcal{U}$:*

$$\frac{\mathcal{A}_1 \Rightarrow \bigcirc\mathcal{B}_1 \quad \cdots \quad \mathcal{A}_n \Rightarrow \bigcirc\mathcal{B}_n \qquad \Diamond l}{\bigwedge_{i=1}^{n} \neg\mathcal{A}_i} \ (\Diamond^{\mathcal{U}}_{res}),$$

  where $\mathcal{A}_i \Rightarrow \bigcirc\mathcal{B}_i$ are merged derived step clauses such that for every i, $1 \leq i \leq n$, the *loop* side conditions $\mathcal{U} \wedge \mathcal{B}_i \models \neg l$ and $\mathcal{U} \wedge \mathcal{B}_i \models \bigvee_{j=1}^{n} \mathcal{A}_j$ are valid. The notions of *ground loop* and *ground loop formula* are defined similarly to the case above.

The notion of a derivation in the calculus $\mathfrak{I}_e$ is defined next.

Definition 3.2.7 (Derivation). *Let $\mathsf{P} = \langle \mathcal{U}_0, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ be a monodic temporal problem. A* derivation *from P is a sequence of universal parts, $\Delta = \mathcal{U}_0 \subsetneq \mathcal{U}_1 \subsetneq \mathcal{U}_2 \subsetneq \cdots$, such that $\mathcal{U}_{i+1}$ is obtained from $\mathcal{U}_i$ by applying an inference rule to $\langle \mathcal{U}_i, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ and adding its conclusion to $\mathcal{U}_i$. The initial, step and eventuality part of the temporal problem are not changed during a derivation.*

A derivation terminates *if and only if either a contradiction is derived, in which case we say that the derivation* terminates successfully, *or if no new formulae can be derived by further inference steps.*

A derivation $\Delta = \mathcal{U}_0 \subsetneq \mathcal{U}_1 \subsetneq \mathcal{U}_2 \subsetneq \cdots \subsetneq \mathcal{U}_n$ from $\langle \mathcal{U}_0, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ is called fair *(we adopt terminology from [10]) if and only if for every $i \geq 0$ and for every formula φ which can be derived by the inference rules of $\mathfrak{I}_e$ from premises in $\langle \mathcal{U}_i, \mathcal{I}, \mathcal{S}, \mathcal{E}^c\rangle$, there exists an index $j \geq i$ such that $\varphi \in \mathcal{U}_j$.*

Note that any derivation can be continued, yielding a terminating derivation. Note further that since there exist only finitely many non-equivalent merged derived or full merged step clauses, the number of non-equivalent conclusions of the inference rules of monodic temporal resolution is finite. Therefore, every derivation is finite. However, it is important to note that this does not imply the decidability of monodic first-order temporal logic as the applicability of the inference rules is not decidable.

Soundness and completeness of $\mathfrak{I}_e$ is stated in the following theorem.

Theorem 3.2.8 (see [23, Theorem 10.5]). *The rules of $\mathfrak{I}_e$ preserve satisfiability over expanding domains. A monodic temporal problem P with unique left-hand sides in step clauses is unsatisfiable over expanding domains if and only if any* fair *derivation in $\mathfrak{I}_e$ from P^c terminates successfully.*

We now give an example refutation under $\mathfrak{I}_e$. Let P be the following temporal problem

$$\begin{aligned}\mathsf{P} = \langle & \{\forall x(T(x) \Rightarrow (Q(x) \wedge \neg L(x))), \forall x(S(x) \Rightarrow Q(x))\},\\ & \{\exists x P(x)\},\\ & \{P(x) \Rightarrow \bigcirc S(x), Q(x) \Rightarrow \bigcirc T(x)\},\\ & \{\Diamond L(x)\}\rangle\end{aligned}$$

```
Function BFS
Input:   A temporal problem $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ and an eventuality clause $\Diamond L(x) \in \mathcal{E}$.
Output:  A formula $H(x)$ with at most one free variable.
Method:  (1) Let $H_0(x) = \mathbf{true}$; $\mathcal{N}_0 = \emptyset$; $i = 0$
         (2) Let $\mathcal{N}_{i+1} = \{\forall x(\mathcal{A}_j^{(i+1)}(x) \Rightarrow \bigcirc \mathcal{B}_j^{(i+1)}(x))\}_{j=1}^{k}$ be the set of
             all full merged step clauses such that for every $j \in \{1, \ldots, k\}$,
             $\forall x(\mathcal{U} \wedge \mathcal{B}_j^{(i+1)}(x) \Rightarrow (\neg L(x) \wedge H_i(x)))$ is valid. (The set $\mathcal{N}_{i+1}$ possibly
             includes the degenerated step clause $\mathbf{true} \Rightarrow \bigcirc\mathbf{true}$ in the case
             $\mathcal{U} \models \forall x(\neg L(x) \wedge H_i(x))$.)
         (3) If $\mathcal{N}_{i+1} = \emptyset$, return false; else let $H_{i+1}(x) = \bigvee_{j=1}^{k} \mathcal{A}_j^{(i+1)}(x)$
         (4) If $\forall x(H_i(x) \Rightarrow H_{i+1}(x))$, return $H_{i+1}(x)$.
         (5) i = i + 1; goto 2.
```

Figure 3.1: Breadth-First Search Algorithm

First of all, we can observe that the temporal problem P is already constant-flooded, i.e. $\mathsf{P}^c = \mathsf{P}$. Then, for the full merged step clause $\forall x(Q(x) \Rightarrow \bigcirc T(x))$ it holds that the loop side conditions $\forall x[((\forall y(T(y) \Rightarrow (Q(y) \wedge \neg L(y)))) \wedge T(x)) \Rightarrow (Q(x) \wedge \neg L(x))]$ are valid, i.e. we can apply the eventuality resolution rule for the eventuality $\Diamond L(x)$ and derive the universal formula $\forall x \neg Q(x)$. Now, for the (merged) derived step clause $\exists x(P(x) \Rightarrow \bigcirc S(x))$ it holds that $\{\forall x(S(x) \Rightarrow Q(x)), \forall x \neg Q(x), \exists x S(x)\} \models \bot$. We can therefore apply the step resolution rule and obtain the universal formula $\forall x \neg P(x)$. Finally, as $\{\exists x P(x), \forall x \neg P(x)\} \models \bot$, an application of the termination rule allows us to derive $\bot$.

The task of finding suitable full merged step clauses (or merged derived clauses) for applications of the eventuality resolution rules can be delegated to the Breadth-First Search (BFS) algorithm depicted in Figure 3.1. It returns all the possible loop formulae for a given eventuality and sets of universal and step clauses as input.

Monodic temporal resolution remains sound and refutationally complete if the eventuality resolution rules are restricted to loops found by the BFS algorithm.

Theorem 3.2.9 (see [23, Theorem 9.5 and 10.5]). *A monodic temporal problem P with unique left-hand sides in step clauses is unsatisfiable over expanding domains if and only if any* fair *derivation in $\mathfrak{I}_e$ from P^c with applications of the eventuality resolution rules restricted to loops found by the BFS algorithm terminates successfully.*

This concludes the description of monodic temporal resolution. In the next section we present the ordered fine-grained resolution with selection calculus.

3.3 Inference Rules

In this section we introduce the inference rules of *ordered fine-grained resolution with selection*. First, we present the core deduction rules, and subsequently, we introduce the FG-BFS algorithm, which can be used to find premises for the eventuality resolution rules.

The FG-BFS algorithm can be seen as a resolution-based variant of the BFS algorithm depicted in Figure 3.1.

3.3.1 Fine-Grained Step Resolution and Eventuality Resolution

Before we can give the definitions of the inference rules, we still have to introduce some auxiliary concepts. We begin by introducing the notion of *most general unifier* briefly.

Definition 3.3.1. *Let $\Sigma \subseteq CS \cup FS$. Furthermore, let $E = \{s_1 \overset{?}{=} t_1, \ldots, s_n \overset{?}{=} t_n\}$ be a multiset of terms equations over $T_\Sigma(X)$.*

(i) A substitution $\sigma\colon X \to T_\Sigma(X)$ is a unifier *of the equation multiset E if and only if*

$$\forall i, 1 \leq i \leq n\colon s_i\sigma = t_i\sigma$$

The set of all the unifiers of the equation multiset E is denoted by $Unif(E)$.

(ii) A substitution σ is called a most general unifier *of the equation multiset E if and only if*

$$\forall \tau \in Subst_\Sigma(X)\colon \tau \in Unif(E) \Rightarrow \sigma \leq \tau$$

(iii) The equation multiset $E = \{s_1 \overset{?}{=} t_1, \ldots, s_n \overset{?}{=} t_n\}$ is said to be in solved form *if and only if*

- $\forall i, 1 \leq i \leq n\colon s_i \in X$, *and*
- $\forall i, 1 \leq i \leq n\, \forall j, 1 \leq j \leq n\colon s_i \notin var(t_j) \wedge (i \neq j \Rightarrow s_i \neq s_j)$.

(iv) Let $A = p(s_1, \ldots, s_n)$, $B = p(t_1, \ldots, t_n)$ be two atoms. A substitution σ is said to be a most general unifier of the atoms A and B if and only if the substitution σ is a most general unifier of the equation multiset $\{s_1 \overset{?}{=} t_1, \ldots, s_n \overset{?}{=} t_n\}$.

Then, we assume that we are given an *admissible atom ordering* $\succ$, that is, a strict total ordering on ground atoms which is well-founded, and a *selection function S* which maps any first-order clause C to a (possibly empty) subset of its negative literals. For proving the refutational completeness of ordered fine-grained resolution with selection, we require selection functions to be instance compatible:

Definition 3.3.2. *We say that a selection function S is* instance compatible *if and only if for every clause C, for every substitution σ and for every literal $l \in C\sigma$ it holds that*

$$l \in S(C\sigma) \iff \exists l' \in S(C)\colon l'\sigma = l.$$

The atom ordering $\succ$ is extended to ground literals by $\neg A \succ A$ and $(\neg)A \succ (\neg)B$ if and only if $A \succ B$. The ordering is extended on the non-ground level as follows: for two arbitrary literals L and L', $L \succ L'$ if and only if $L\sigma \succ L'\sigma$ for every grounding substitution σ. A literal L is called (strictly) maximal w.r.t. a clause C if and only if there is no literal $L' \in C$

with $L' \succ L$ ($L' \succeq L$). A literal L is *eligible* in a clause $L \vee C$ for a substitution σ if either it is selected in $L \vee C$, or otherwise no literal is selected in $L \vee C$ and $L\sigma$ is maximal w.r.t. $C\sigma$.

The atom ordering $\succ$ and the instance compatible selection function S are used to restrict the applicability of the deduction rules of fine-grained resolution as follows. We also assume that the clauses used as premises for the different resolution-based inference rules are made variable disjoint beforehand.

(1) *First-order ordered resolution with selection between two universal clauses*

$$\frac{C_1 \vee A \quad \neg B \vee C_2}{(C_1 \vee C_2)\sigma}$$

if σ is a most general unifier of A and B, *A is eligible in $(C_1 \vee A)$ for σ, and $\neg B$ is eligible in $(\neg B \vee C_2)$ for σ.* The result is a universal clause.

(2) *First-order ordered positive factoring with selection*

$$\frac{C_1 \vee A \vee B}{(C_1 \vee A)\sigma}$$

if σ is a most general unifier of A and B, and *A is eligible in $(C_1 \vee A \vee B)$ for σ.* The result is again a universal clause.

(3) *First-order ordered resolution with selection between an initial and a universal clause, between two initial clauses, and ordered positive factoring with selection on an initial clause.* These are defined in analogy to the two deduction rules above with the only difference that the result is an initial clause.

(4) *Ordered fine-grained step resolution with selection.*

$$\frac{C_1 \Rightarrow \bigcirc(D_1 \vee A) \quad C_2 \Rightarrow \bigcirc(D_2 \vee \neg B)}{(C_1 \wedge C_2)\sigma \Rightarrow \bigcirc(D_1 \vee D_2)\sigma}$$

where $C_1 \Rightarrow \bigcirc(D_1 \vee A)$ and $C_2 \Rightarrow \bigcirc(D_2 \vee \neg B)$ are step clauses, σ is a most general unifier of the atoms A and B such that σ does not map variables from C_1 or C_2 into a constant or a functional term, *A is eligible in $(D_1 \vee A)$ for σ, and $\neg B$ is eligible in $(D_2 \vee \neg B)$ for σ.*

$$\frac{C_1 \Rightarrow \bigcirc(D_1 \vee A) \quad D_2 \vee \neg B}{C_1\sigma \Rightarrow \bigcirc(D_1 \vee D_2)\sigma}$$

where $C_1 \Rightarrow \bigcirc(D_1 \vee A)$ is a step clause, $D_2 \vee \neg B$ is a universal clause, and σ is a most general unifier of the atoms A and B such that σ does not map variables from C_1 into a constant or a functional term, *A is eligible in $(D_1 \vee A)$ for σ, and $\neg B$ is eligible in $(D_2 \vee \neg B)$ for σ.* There also exists a similar rule where the positive literal A is contained in a universal clause and the negative literal $\neg B$ in a step clause.

(5) *Ordered fine-grained positive step factoring with selection.*

$$\frac{C \Rightarrow \bigcirc(D \vee A \vee B)}{C\sigma \Rightarrow \bigcirc(D \vee A)\sigma}$$

where σ is a most general unifier of the atoms A and B such that σ does not map variables from C into a constant or a functional term, and *A is eligible in $(D \vee A \vee B)$ for σ.*

(6) *Clause conversion.* A step clause of the form $C \Rightarrow \bigcirc\bot$ is rewritten to the universal clause $\neg C$.

Step clauses of the form $C \Rightarrow \bigcirc\bot$ will also be called *terminating* or *final* step clauses.

(7) *Duplicate literal elimination in left-hand sides of terminating step clauses.* A clause of the form $(C \wedge A \wedge A) \Rightarrow \bigcirc\bot$ yields the clause $(C \wedge A) \Rightarrow \bigcirc\bot$.

(8) *Eventuality resolution rule w.r.t. $\mathcal{U}$:*

$$\frac{\forall x(\mathcal{A}_1(x) \Rightarrow \bigcirc\mathcal{B}_1(x)) \quad \cdots \quad \forall x(\mathcal{A}_n(x) \Rightarrow \bigcirc\mathcal{B}_n(x)) \qquad \Diamond L(x)}{\forall x \bigwedge_{i=1}^{n} \neg\mathcal{A}_i(x)} \; (\Diamond_{res}^{\mathcal{U}}),$$

where $\forall x(\mathcal{A}_i(x) \Rightarrow \bigcirc\mathcal{B}_i(x))$ are formulae computed from the set of step clauses such that for every i, $1 \leq i \leq n$, the *loop* side conditions $\forall x(\mathcal{U} \wedge \mathcal{B}_i(x) \Rightarrow \neg L(x))$ and $\forall x(\mathcal{U} \wedge \mathcal{B}_i(x) \Rightarrow \bigvee_{j=1}^{n} \mathcal{A}_j(x))$ are valid.[3]

(9) *Ground eventuality resolution rule w.r.t. $\mathcal{U}$:*

$$\frac{\mathcal{A}_1 \Rightarrow \bigcirc\mathcal{B}_1 \quad \cdots \quad \mathcal{A}_n \Rightarrow \bigcirc\mathcal{B}_n \qquad \Diamond l}{\bigwedge_{i=1}^{n} \neg\mathcal{A}_i} \; (\Diamond_{res}^{\mathcal{U}}),$$

where $\mathcal{A}_i \Rightarrow \bigcirc\mathcal{B}_i$ are ground formulae computed from the set of step clauses such that for every i, $1 \leq i \leq n$, the *loop* side conditions $\mathcal{U} \wedge \mathcal{B}_i \models \neg l$ and $\mathcal{U} \wedge \mathcal{B}_i \models \bigvee_{j=1}^{n} \mathcal{A}_j$ are valid. The notions of *ground loop* and *ground loop formula* are defined similarly to the case above.

Rules (1) to (7), also called rules of *fine-grained step resolution*, are either identical or closely related to the deduction rules of ordered first-order resolution with selection; a fact that we exploit in our implementation of the calculus.

Let *ordered fine-grained resolution with selection* be the calculus consisting of the rules (1) to (7) above, together with the ground and non-ground eventuality resolution rules described above, i.e. rules (8) and (9). We denote this calculus by $\mathfrak{I}_{FG}^{S,\succ}$.

Just as in the regular first-order case, selection functions are used as a control mechanism for handling the non-determinism that arises from inferences on negative literals. The proof of refutational completeness for regular first-order ordered resolution with selection

[3] In the case $\mathcal{U} \models \forall x \neg L(x)$, the *degenerate clause*, **true** $\Rightarrow \bigcirc$**true**, can be considered as a premise of this rule; the conclusion of the rule is then $\neg$**true** $\equiv$ **false**.

given in [10] does not put restrictions on which negative literals have to be chosen for inferences. They can hence be selected in a "don't-care", non-deterministic way. In practical implementations different strategies for selecting negative literals are employed. For example, the theorem prover SPASS 3.0 allows to define a list of predicates that become candidates for selection. Additionally, SPASS offers, for instance, the following selection strategies [92]: the selection of one negative literal in any clause with more than one maximal literal, or the selection of one negative literal in a clause if it contains at least one negative literal. In these cases if no negative literal from the given list of selection predicates is present in a clause, a negative literal of maximal weight is selected. Other theorem provers like Vampire also allow a multitude of different selection strategies together with the selection of both positive and negative literals [78]. The available selection strategies are also often goal-oriented.

Next we define the concept of derivations in the context of ordered fine-grained resolution with selection.

Definition 3.3.3 (Derivation). *A (linear)* derivation Δ *(in $\mathfrak{I}^{S,\succ}_{FG}$) from the clausification* $\mathsf{Cls}(P)$ *of a monodic temporal problem P is a sequence of tuples*

$$\Delta = \langle \mathcal{U}_0, \mathcal{I}_0, \mathcal{S}_0, \mathcal{E} \rangle, \langle \mathcal{U}_1, \mathcal{I}_1, \mathcal{S}_1, \mathcal{E} \rangle, \langle \mathcal{U}_2, \mathcal{I}_2, \mathcal{S}_2, \mathcal{E} \rangle, \ldots$$

such that each tuple at an index $i+1$ is obtained from the tuple at the index i by adding the conclusion of an application of one of the inference rules of $\mathfrak{I}^{S,\succ}_{FG}$ to premises from the sets $\mathcal{U}_i$, $\mathcal{I}_i$, $\mathcal{S}_i$ to that set, with the other sets as well as $\mathcal{E}$ remaining unchanged[4]*. The derivation Δ will also be denoted by a sequence of clauses $C_1, C_2, \ldots$ where each clause C_i is either contained in the problem $\langle \mathcal{U}_0, \mathcal{I}_0, \mathcal{S}_0, \mathcal{E} \rangle$ or is newly obtained in the inference step that derived the problem $\langle \mathcal{U}_i, \mathcal{I}_i, \mathcal{S}_i, \mathcal{E} \rangle$.*

*A derivation Δ such that the empty clause is an element of a $\mathcal{U}_i \cup \mathcal{I}_i$ is called a ($\mathfrak{I}^{S,\succ}_{FG}$-)*refutation *of $\langle \mathcal{U}_0, \mathcal{I}_0, \mathcal{S}_0, \mathcal{E} \rangle$.*

A derivation Δ is fair *if and only if for every $i \geq 0$ and for every clause C which can be derived by the inference rules of $\mathfrak{I}^{S,\succ}_{FG}$ from premises in $\langle \mathcal{U}_i, \mathcal{I}_i, \mathcal{S}_i, \mathcal{E} \rangle$, there exists an index $j \geq i$ such that C occurs in $\langle \mathcal{U}_j, \mathcal{I}_j, \mathcal{S}_j, \mathcal{E} \rangle$.*

A set of temporal clauses $\mathcal{N}$ is said to be saturated *under ordered fine-grained resolution with selection if and only if the resulting clauses from all the possible inferences under the rules of ordered fine-grained resolution with selection are already contained in the set $\mathcal{N}$.*

3.3.2 Resolution-Based Loop Search Algorithm

Loop formulae, which are required for applications of the eventuality resolution rules (8) and (9), can be computed by the fine-grained breadth-first search algorithm (FG-BFS), depicted in Figure 3.2. The process of running the FG-BFS algorithm is called *loop search.*

The algorithm takes as input a set of universal clauses $\mathcal{U}$ and a set of step clauses $\mathcal{S}$ saturated by ordered fine-grained resolution with selection, and an eventuality clause

[4] In an application of ground eventuality or eventuality resolution rule, the set $\mathcal{U}$ in the definition of the rule refers to $\mathcal{U}_i$.

```
Function FG-BFS
Input:   A set of universal clauses $\mathcal{U}$ and a set of step clauses $\mathcal{S}$, saturated under
         the fine-grained step resolution inference rules of ordered fine-grained
         resolution with selection, and an eventuality clause $\Diamond L(x) \in \mathcal{E}$, where
         $L(x)$ is unary literal.
Output:  A formula $H(x)$ with at most one free variable.
Method:  (1) Let $H_0(x) = \mathbf{true}$; $\mathcal{M}_0 = \emptyset$; $i = 0$
         (2) Let $\mathcal{N}_{i+1} = \mathcal{U} \cup \{ P(c^l) \Rightarrow \bigcirc M(c^l) \mid \text{original } P(x) \Rightarrow \bigcirc M(x) \in \mathcal{S} \} \cup$
             $\{\mathbf{true} \Rightarrow \bigcirc(\neg H_i(c^l) \vee L(c^l))\}$. Apply the fine-grained step resolution
             rules of ordered fine-grained resolution with selection *except the clause
             conversion rule* to $\mathcal{N}_{i+1}$. If we obtain a contradiction, then return the
             loop true (in this case $\forall x \neg L(x)$ is implied by the universal part).
             Otherwise let $\mathcal{M}_{i+1} = \{C_j \Rightarrow \bigcirc\bot\}_{j=1}^{n}$ be the set of all new *terminating*
             step clauses in the saturation of $\mathcal{N}_{i+1}$.
         (3) If $\mathcal{M}_{i+1} = \emptyset$, return false; else let $H_{i+1}(x) = \bigvee_{j=1}^{n}(\tilde{\exists} C_j)\{c^l \to x\}$
         (4) If $\forall x(H_i(x) \Rightarrow H_{i+1}(x))$, return $H_{i+1}(x)$.
         (5) $i = i + 1$; goto 2.
Note: The constant $c^l$ is a fresh constant used for loop search only
```

Figure 3.2: Breadth-first Search Algorithm Using Fine-grained Step Resolution.

$\Diamond L(x) \in \mathcal{E}$. It computes sequences of disjunctions H_0, H_1, H_2, ... such that for every $i > 0$ the formula $\forall x(H_i(x) \Rightarrow (H_{i-1}(x) \wedge \neg L(x)))$ is valid. A loop in $\Diamond L(x)$ has been found whenever the formula $\forall x(H_i(x) \Rightarrow H_{i+1}(x))$ is valid as well (step 4).

For the proof of refutational completeness we do not consider the FG-BFS algorithm, but we use its refined variant Restricted-FG-BFS, shown in Figure 3.3, instead. The operation $\mathrm{LT}(\mathcal{S})$ performs constant-flooding with the loop search constant c^l in original step clauses contained in the set $\mathcal{S}$, i.e.

$$\mathrm{LT}(\mathcal{S}) = \{ P(c^l) \Rightarrow \bigcirc M(c^l) \mid \text{original } P(x) \Rightarrow \bigcirc M(x) \in \mathcal{S} \}.$$

The refined algorithm imposes an additional filtering step on the computed terminating step clauses. The set of terminating step clauses is not allowed to contain clauses $C \Rightarrow \bigcirc\bot$ and $D \Rightarrow \bigcirc\bot$ such that $(\tilde{\exists} C)\{c^l \to x\} \sqsubseteq (\tilde{\exists} D)\{c^l \to x\}$.

We show that ordered fine-grained resolution with selection remains refutationally complete if applications of the eventuality resolution rules are restricted to loops found by the Restricted-FG-BFS algorithm.

We conclude this section by providing an example refutation under $\mathfrak{I}_{FG}^{S,\succ}$ of the temporal problem P that was introduced in Section 3.2. The clausification $\mathrm{Cls}(\mathsf{P}^c)$ of P^c yields the following

$$\begin{aligned}\mathrm{Cls}(\mathsf{P}^c) = (\{&\neg T(x) \vee Q(x),\\ &\neg T(x) \vee \neg L(x),\\ &\neg S(x) \vee Q(x)\},\\ \{&P(c)\},\end{aligned}$$

$$\begin{aligned}&\{P(x) \Rightarrow \bigcirc S(x),\\ &\ Q(x) \Rightarrow \bigcirc T(x)\},\\ &\{\Diamond L(x)\}\rangle\end{aligned}$$

where c is a fresh constant introduced during Skolemization. We consider the atom ordering $\succ$ defined by

$$L(c) \succ P(c) \succ Q(c) \succ S(c) \succ T(c)$$

and a selection function S which does not select any literals. We can first of all apply the FG-BFS algorithm on $\mathsf{Cls}(\mathsf{P}^c)$. For the first iteration we have $H_0(x) = \textbf{true}$ and we obtain the terminating step clauses $Q(x) \Rightarrow \bigcirc\bot$, i.e. $H_1(x) = Q(x)$. As the formula $\forall x(H_0(x) \Rightarrow H_1(x))$ is not valid, we have to continue with the next iteration, where we can derive the terminating step clauses $Q(x) \Rightarrow \bigcirc\bot$ and $P(x) \wedge Q(x) \Rightarrow \bigcirc\bot$, i.e. $H_2(x) = Q(x) \vee (P(x) \wedge Q(x))$. Now, as the formula $\forall x(H_1(x) \Rightarrow H_2(x))$ is valid, the FG-BFS algorithm returns the loop formula $H_2(x)$, which results in the two universal clauses $\neg Q(x)$ and $\neg P(x) \vee \neg Q(x)$ after applying the eventuality resolution rule.

By resolving the universal clause $\neg Q(x)$ with the universal clause $\neg S(x) \vee Q(x)$, we obtain the universal clause $\neg S(x)$, which can be resolved with the step clause $P(x) \Rightarrow \bigcirc S(x)$, resulting in the terminating step clause $P(x) \Rightarrow \bigcirc\bot$. An application of the clause conversion rule yields the universal clause $\neg P(x)$. Finally, the universal clause $\neg P(x)$ can be resolved with the initial clause $P(c)$ and we obtain the empty clause.

In the following section we provide the full proof of refutational completeness for ordered fine-grained resolution with selection.

3.4 Refutational Completeness

The proof of refutational completeness for ordered fine-grained resolution with selection is organised as follows. First, we define a refined version of the monodic temporal resolution calculus and show its refutational completeness. Then, we prove that for every refutation of a monodic temporal problem under refined monodic temporal resolution there exists a "similar" refutation under ordered fine-grained resolution with selection. In order to be able to show this result we also have to prove the lifting theorem for ordered fine-grained resolution with selection (without the eventuality resolution and the duplicate literal elimination in terminating step clauses rules).

3.4.1 Refined Monodic Temporal Resolution

As the step resolution rule of monodic temporal resolution can be applied on arbitrary merged derived clauses and as the BFS algorithm returns all possible longer combinations of full merged step clauses once a shorter loop formula has been detected, one can see that not all the full merged step clauses and merged derived clauses that occur in a $\mathfrak{I}_e$-derivation can also be derived by ordered fine-grained resolution with selection.

```
Function Restricted-FG-BFS
Input:   A set of universal clauses $\mathcal{U}$ and a set of step clauses $\mathcal{S}$, saturated under the fine-grained step resolution inference rules of ordered fine-grained resolution with selection, and an eventuality clause $\Diamond L(x) \in \mathcal{E}$.
Output:  A formula $R(x)$ with at most one free variable.
Method:  (1) Let $R_0(x) = \mathbf{true}$; $\mathcal{M}_0 = \emptyset$; $i = 0$
         (2) Let $\mathcal{N}_{i+1} = \mathcal{U} \cup \mathrm{LT}(\mathcal{S}) \cup \{\mathbf{true} \Rightarrow \bigcirc(\neg R_i(c^l) \vee L(c^l))\}$. Apply the fine-grained step resolution rules of ordered fine-grained resolution with selection *except the clause conversion rule* to $\mathcal{N}_{i+1}$. If we obtain a contradiction, then return the loop true (in this case $\forall x \neg L(x)$ is implied by the universal part).
             Otherwise let $\mathcal{T}_{i+1} = \{C_j \Rightarrow \bigcirc\bot\}_{j=1}^{m}$ be the set of all new *terminating* step clauses in the saturation of $\mathcal{N}_{i+1}$ which is free of step clauses $C \Rightarrow \bigcirc\bot$ and $D \Rightarrow \bigcirc\bot$ with $C\sigma = D$, where $\sigma$ is a variable renaming. Additionally, let $\mathcal{M}_{i+1} = \{D_j \Rightarrow \bigcirc\bot\}_{j=1}^{n} \subseteq \mathcal{T}_{i+1}$ be the set of all the minimal terminating step clauses w.r.t. the relation $\sqsubseteq$ and the set $(\tilde{\exists}\mathcal{T}_{i+1})\{c^l \to x\}$.
         (3) If $\mathcal{M}_{i+1} = \emptyset$, return false; else let $R_{i+1}(x) = \bigvee_{j=1}^{n}(\tilde{\exists} D_j)\{c^l \to x\}$
         (4) If $\forall x(R_i(x) \Rightarrow R_{i+1}(x))$, return $R_{i+1}(x)$.
         (5) i = i + 1; goto 2.
```

Figure 3.3: Restricted Breadth-First Search Using Ordered Fine-Grained Step Resolution with Selection

Hence, in order for our completeness proof for ordered fine-grained resolution with selection to succeed, the premises of some inference rules of monodic temporal resolution have to be restricted.

First of all, we have to restrict the step resolution rule by imposing an additional constraint on the merged derived step clauses that are used as premises. The refined inference rule is defined as follows.

Refined Step Resolution Rule w.r.t. $\mathcal{U}$:

$$\frac{\mathcal{A} \Rightarrow \bigcirc\mathcal{B}}{\neg\mathcal{A}} \quad (\bigcirc^{\mathcal{U}}_{res}), \qquad \begin{array}{l}\text{if } \mathcal{B} \text{ is minimal (w.r.t. the relation } \sqsubset\text{) such that}\\ \mathcal{U} \cup \{\mathcal{B}\} \models \bot\end{array}$$

Furthermore, we do not run the BFS algorithm (Figure 3.1) to compute premises for the eventuality resolution rules, but instead we use the refined breadth-first search (Ref-BFS) algorithm depicted in Figure 3.4.

The Ref-BFS algorithm can be seen as a refined version of the BFS algorithm. A comparison with the BFS algorithm shows that the selection constraints on full merged clauses in the Ref-BFS algorithm have been refined in a similar way as the refined step resolution rule mentioned above. These additional constraints in the selection process of full merged step clauses ensure that exactly the same full merged step clauses can also obtained during $\mathfrak{I}^{S,\succ}_{FG}$-derivations.

```
Function Ref-BFS
Input:   A temporal problem $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ and an eventuality clause $\Diamond L(x) \in \mathcal{E}$.
Output:  A formula $H'(x)$ with at most one free variable.
Method:  (1) Let $H'_0(x) = \mathbf{true}$; $\mathcal{N}'_0 = \emptyset$; $i = 0$
         (2) Let $\mathcal{N}'_{i+1} = \{\forall x(\mathcal{A}^{(i+1)}_j(x) \Rightarrow \bigcirc \mathcal{B}^{(i+1)}_j(x))\}^k_{j=1} \subseteq \mathcal{M}(\mathsf{P})$ be the set
             of all full merged step clauses such that for every $j \in \{1, \ldots, k\}$, the
             formula $\forall x(\mathcal{U} \wedge \mathcal{B}^{(i+1)}_j(x) \Rightarrow (\neg L(x) \wedge H_i(x)))$ is valid and such that
             for every $j \in \{1, \ldots, k\}$ and every $\mathcal{B}'(x) \sqsubset \mathcal{B}^{(i+1)}_j(x)$, the formula
             $\exists x(\mathcal{U} \wedge \mathcal{B}'(x) \wedge \neg(\neg L(x) \wedge H_i(x)))$ is satisfiable. (The set $\mathcal{N}'_{i+1}$ possibly
             includes the degenerated clause $\mathbf{true} \Rightarrow \bigcirc\mathbf{true}$ in the case $\mathcal{U} \models$
             $\forall x(\neg L(x) \wedge H_i(x))$.)
         (3) If $\mathcal{N}'_{i+1} = \emptyset$, return false; else let $H'_{i+1}(x) = \bigvee^k_{j=1}(\mathcal{A}^{(i+1)}_j(x))$
         (4) If $\forall x(H'_i(x) \Rightarrow H'_{i+1}(x))$, return $H'_{i+1}(x)$.
         (5) i = i + 1; goto 2.
```

Figure 3.4: Refined Breadth-First Search Algorithm

The refutational completeness of the calculus derived as described above from monodic temporal resolution will be proved in the subsequent section. The refined calculus will be called *refined monodic temporal resolution* in the following, and denoted by $\mathfrak{I}^{\sqsubseteq}_e$.

3.4.2 Properties of the Ref-BFS algorithm

Before we can show the refutational completeness of refined monodic temporal resolution, we have to prove some properties of the Ref-BFS algorithm.

First, we introduce the notion of upward closure under the relation $\sqsubseteq$ of a set of full merged (and merged derived) step clauses.

Definition 3.4.1. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem and let $\mathcal{N} \subseteq \mathcal{M}(P)$ be a set of full merged or merged derived step clauses. Then we define a set of full merged or merged derived step clauses $\mathcal{M}_{|\mathcal{N}}(P)$ as follows:*

$$\mathcal{M}_{|\mathcal{N}}(P) = \{\, \forall x(\mathcal{A}'(x) \Rightarrow \bigcirc \mathcal{B}'(x)) \in \mathcal{M}(P) \mid \exists \forall x(\mathcal{A}(x) \Rightarrow \bigcirc \mathcal{B}(x)) \in \mathcal{N} \colon \mathcal{B}(x) \sqsubseteq \mathcal{B}'(x) \,\}$$

We now show a couple of properties related to sets of full-merged step clauses $\mathcal{N}'_i$ that have been constructed in a run of the Ref-BFS algorithm.

Lemma 3.4.2. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality clause. Furthermore, let $\mathcal{N}'_0, \mathcal{N}'_1, \ldots$ be the sequence of sets and let $H'_0(x), H'_1(x), \ldots$ be the sequence of formulae constructed by a run of the Ref-BFS algorithm applied on the temporal problem P for the eventuality $\Diamond L(x)$.*

Then it holds for all $i \geq 1$ that $\mathcal{N}'_{i+1} \subseteq \mathcal{M}_{|\mathcal{N}'_i}(P)$ and that the formula $\forall x(H'_{i+1}(x) \Rightarrow H'_i(x))$ is valid.

Proof. By induction on i. We first of all observe that the formula $\forall x(H'_1(x) \Rightarrow \mathbf{true})$ is valid.

Now, let $i \geq 1$. We may assume that $\mathcal{N}'_{i+1} \neq \emptyset$ as otherwise $\mathcal{N}'_{i+1} = \emptyset$ and $H'_{i+1}(x) =$ **false**. Then, $\emptyset \subseteq \mathcal{N}'_i$ would clearly hold and the formula $\forall x(\mathbf{false} \Rightarrow H'_i(x))$ would be valid. Without loss of generality, let $\forall x(\mathcal{A}^{i+1}(x) \Rightarrow \bigcirc\mathcal{B}^{i+1}(x)) \in \mathcal{N}'_{i+1}$ (the case for $\mathbf{true} \Rightarrow \bigcirc\mathbf{true} \in \mathcal{N}'_{i+1}$ is similar). Then it holds that the formula $\forall x(\mathcal{U} \wedge \mathcal{B}^{(i+1)}(x) \Rightarrow (\neg L(x) \wedge H'_i(x)))$ is valid and that for every $\mathcal{B}'(x) \sqsubset \mathcal{B}^{(i+1)}(x)$, the formula $\exists x(\mathcal{U} \wedge \mathcal{B}'(x) \wedge \neg(\neg L(x) \wedge H'_i(x)))$ is satisfiable. From the induction hypothesis (and the validity of $\forall x(H'_1(x) \Rightarrow \mathbf{true})$) it follows that the formula $\forall x(H'_i(x) \Rightarrow H'_{i-1}(x))$ is valid, which implies that the formula $\forall x(\mathcal{U} \wedge \mathcal{B}^{(i+1)}(x) \Rightarrow (\neg L(x) \wedge H'_{i-1}(x)))$ is valid as well. Hence, there exists a formula $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}'_i$ such that $\mathcal{A}(x) \sqsubseteq \mathcal{A}^{i+1}(x)$ and $\mathcal{B}(x) \sqsubseteq \mathcal{B}^{i+1}(x)$ (and thus, the formula $\forall x(\mathcal{A}^{i+1}(x) \Rightarrow \mathcal{A}(x))$ is valid). We can conclude that $\mathcal{N}'_{i+1} \subseteq \mathcal{M}_{|\mathcal{N}'_i}(\mathsf{P})$, which implies that the formula $\forall x(H'_{i+1}(x) \Rightarrow H'_i(x))$ is valid. □

Corollary 3.4.3. *Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality clause. Furthermore, let $\mathcal{N}'_0, \mathcal{N}'_1, \ldots$ be the sequence of sets constructed by a run of the Ref-BFS algorithm applied on the temporal problem P for the eventuality $\Diamond L(x)$.*

Then it holds for all $i \geq 1$ that $\mathcal{M}_{|\mathcal{N}'_{i+1}}(\mathsf{P}) \subseteq \mathcal{M}_{|\mathcal{N}'_i}(\mathsf{P})$.

Proof. Let $i \geq 1$ and let $\forall x(\mathcal{A}''(x) \Rightarrow \bigcirc\mathcal{B}''(x)) \in \mathcal{M}_{|\mathcal{N}'_{i+1}}(\mathsf{P})$, that is, there exists a full merged step clause $\forall x(\mathcal{A}'(x) \Rightarrow \bigcirc\mathcal{B}'(x)) \in \mathcal{N}'_{i+1}$ with $\mathcal{B}'(x) \sqsubseteq \mathcal{B}''(x)$. By Lemma 3.4.2 there exists a full merged step clause $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}'_i$ with $\mathcal{B}(x) \sqsubseteq \mathcal{B}'(x)$. Therefore, it is easy to see that $\mathcal{B}(x) \sqsubseteq \mathcal{B}''(x)$ holds. □

Lemma 3.4.4. *Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality clause. Then the Ref-BFS algorithm applied on the problem P for the eventuality $\Diamond L(x)$ will require only finitely many iterations.*

Proof. Let $\mathcal{N}'_0, \mathcal{N}'_1, \ldots$ be the sequence of sets and $H'_0(x), H'_1(x), \ldots$ be the sequence of formulae constructed by the run of the Ref-BFS algorithm. By Corollary 3.4.3 it holds for all $i \geq 1$ that $\mathcal{M}_{|\mathcal{N}'_{i+1}}(\mathsf{P}) \subseteq \mathcal{M}_{|\mathcal{N}'_i}(\mathsf{P})$. As every set $\mathcal{M}_{|\mathcal{N}'_{i+1}}(\mathsf{P})$ is finite, it either holds that there exists an index $j \geq 1$ with $\mathcal{M}_{|\mathcal{N}'_{j+1}}(\mathsf{P}) = \mathcal{M}_{|\mathcal{N}'_j}(\mathsf{P})$ or there exists an index $k \geq 1$ with $\mathcal{N}'_k \subseteq \mathcal{M}_{|\mathcal{N}'_k}(\mathsf{P}) = \emptyset$, which immediately implies that the algorithm Ref-BFS stops at the k-th iteration. If $\mathcal{N}'_j \subseteq \mathcal{M}_{|\mathcal{N}'_j}(\mathsf{P}) = \mathcal{M}_{|\mathcal{N}'_{j+1}}(\mathsf{P}) \supseteq \mathcal{N}'_{j+1}$, then it is easy to see that the formula $\forall x(H'_j(x) \Rightarrow H'_{j+1}(x))$ is valid and the algorithm Ref-BFS stops at the $(j+1)$-th iteration. □

Lemma 3.4.5. *Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality clause. Furthermore, let $\mathcal{N}'_0, \mathcal{N}'_1, \ldots, \mathcal{N}'_i, \mathcal{N}'_{i+1}, \mathcal{N}'_{i+2}$ be a sequence of sets and let $H'_0(x), H'_1(x), \ldots, H'_i(x), H'_{i+1}(x), H'_{i+2}(x)$ be a sequence of formulae constructed by a run of the Ref-BFS algorithm applied on the temporal problem P for the eventuality $\Diamond L(x)$. Additionally, we assume that the formula $\forall x(H'_i(x) \Leftrightarrow H'_{i+1}(x))$ is valid.*

Then it holds that $\mathcal{N}'_{i+1} = \mathcal{N}'_{i+2}$.

Proof. First of all, we can observe that if $H'_i(x) = \mathbf{false}$ or $H'_{i+1}(x) = \mathbf{false}$, it must hold that $H'_i(x) = \mathbf{false} = H'_{i+1}(x)$. It then follows from Lemma 3.4.2 that $\mathcal{N}'_{i+1} = \emptyset = \mathcal{N}'_{i+2}$. Hence, we may now assume that $H'_i(x) \neq \mathbf{false} \neq H'_{i+1}(x)$.

Next, we show that $\mathcal{N}'_{i+1} \subseteq \mathcal{N}'_{i+2}$. The inclusion $\mathcal{N}'_{i+2} \subseteq \mathcal{N}'_{i+1}$ can be shown analogously. Without loss of generality, let $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}'_{i+1}$ (the remaining case for $\mathbf{true} \Rightarrow \bigcirc\mathbf{true} \in \mathcal{N}'_{i+1}$ is similar). Then it holds that the formula $\forall x(\mathcal{U} \land \mathcal{B}(x) \Rightarrow (\neg L(x) \land H'_i(x)))$ is valid and that for every $\mathcal{B}'(x) \sqsubset \mathcal{B}(x)$, the formula $\exists x(\mathcal{U} \land \mathcal{B}' \land \neg(\neg L(x) \land H'_i(x)))$ is satisfiable. Hence, as the formula $\forall x(H'_i(x) \Leftrightarrow H'_{i+1}(x))$ is valid, it follows that the formula $\forall x(\mathcal{U} \land \mathcal{B}(x) \Rightarrow (\neg L(x) \land H'_{i+1}(x)))$ is valid and that for every $\mathcal{B}'(x) \sqsubset \mathcal{B}(x)$, the formula $\exists x(\mathcal{U} \land \mathcal{B}' \land \neg(\neg L(x) \land H'_{i+1}(x)))$ is satisfiable. We can conclude that $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}'_{i+2}$. □

Corollary 3.4.6. *Let $P = \langle\mathcal{U},\mathcal{I},\mathcal{S},\mathcal{E}\rangle$ be a monodic temporal problem and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality clause. Furthermore, let $\mathcal{N}'_0, \mathcal{N}'_1, \ldots, \mathcal{N}'_i, \mathcal{N}'_{i+1}, \mathcal{N}'_{i+2}, \ldots$ be a sequence of sets and let $H'_0(x), H'_1(x), \ldots, H'_i(x), H'_{i+1}(x), H'_{i+2}(x), \ldots$ be a sequence of formulae constructed by a run of the Ref-BFS algorithm applied on the temporal problem P for the eventuality $\Diamond L(x)$. Additionally, we assume that the formula $\forall x(H'_i(x) \Leftrightarrow H'_{i+1}(x))$ is valid.*

Then it holds for all $j \geq 1$ that $\mathcal{N}'_{i+j} = \mathcal{N}'_{i+j+1}$ and that the formula $\forall x(H'_{i+j}(x) \Leftrightarrow H'_{i+j+1}(x))$ is valid.

Proof. Follows from Lemma 3.4.5 and from the fact that for all $j \in \mathbb{N}$ the set equality $\mathcal{N}'_{i+j} = \mathcal{N}'_{i+j+1}$ implies that the formula $\forall x(H'_{i+j}(x) \Leftrightarrow H'_{i+j+1}(x))$ is valid. □

The next lemma establishes a correctness result for the Ref-BFS algorithm.

Lemma 3.4.7. *Let $P = \langle\mathcal{U},\mathcal{I},\mathcal{S},\mathcal{E}\rangle$ be a monodic temporal problem and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality clause. Additionally, let $H(x)$ denote the formula computed by the Ref-BFS algorithm applied on the problem P for the eventuality $\Diamond L(x)$.*

Then the formula $\forall x(\mathcal{U} \land H(x) \Rightarrow \bigcirc\Box\neg L(x))$ is valid.

Proof. Similarly to the proof given in [23]. □

The subsequent lemma shows that if a monodic temporal problem admits a loop formula $\bigvee_{j=1}^{n} \mathcal{A}_j(x)$ for an eventuality $\Diamond L(x)$, then the Ref-BFS algorithm computes a loop formula $\bigvee_{k=1}^{m} \mathcal{A}'_k(x)$ such that the formula $\forall x(\bigvee_{j=1}^{n} \mathcal{A}_j(x) \Rightarrow \bigvee_{k=1}^{m} \mathcal{A}'_k(x))$ is valid.

Lemma 3.4.8. *Let $P = \langle\mathcal{U},\mathcal{I},\mathcal{S},\mathcal{E}\rangle$ be a monodic temporal problem such that the set $\mathcal{U}$ is satisfiable, let $\forall x(\mathcal{A}_j(x) \Rightarrow \bigcirc\mathcal{B}_j(x))$ for $j \in \{1,\ldots,n\}$ be full merged step clauses and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality clause such that the loop side conditions $\forall x(\mathcal{U} \land \mathcal{B}_j(x) \Rightarrow \neg L(x))$ and $\forall x(\mathcal{U} \land \mathcal{B}_j(x) \Rightarrow \bigvee_{j=1}^{n} \mathcal{A}_j(x))$ are valid for all j, $1 \leq j \leq n$.*

Then the Ref-BFS algorithm applied on the temporal problem P for the eventuality $\Diamond L(x)$ returns a formula $H(x) = \bigvee_{k=1}^{m} \mathcal{A}'_k(x)$ such that for all $\forall x(\mathcal{A}_j(x) \Rightarrow \bigcirc\mathcal{B}_j(x))$, $j \in \{1,\ldots,n\}$, there exists a full merged step clause $\forall x(\mathcal{A}'_k(x) \Rightarrow \bigcirc\mathcal{B}'_k(x)) \in \mathcal{M}(P)$

with $1 \leq k \leq m$, $\mathcal{A}'_k(x) \sqsubseteq \mathcal{A}_k(x)$ *and* $\mathcal{B}'_k(x) \sqsubseteq \mathcal{B}_k(x)$, *which implies that the formula* $\forall x(\bigvee_{j=1}^n \mathcal{A}_j(x) \Rightarrow \bigvee_{k=1}^m \mathcal{A}'_k(x))$ *is valid.*

Proof. Let $\mathcal{N}'_0, \mathcal{N}'_1, \ldots, \mathcal{N}'_i, \ldots$ be the sequence of sets and $H'_0(x), H'_1(x), \ldots, H'_i(x), \ldots$ be the sequence of formulae constructed by a run of the algorithm Ref-BFS applied on the temporal problem P for the eventuality $\Diamond L(x)$. We show by induction on i with $i \geq 1$ that for all $\forall x(\mathcal{A}_j(x) \Rightarrow \bigcirc\mathcal{B}_j(x))$, $j \in \{1, \ldots, n\}$, there exists a full merged step clause $\forall x(\mathcal{A}'_j(x) \Rightarrow \bigcirc\mathcal{B}'_j(x)) \in \mathcal{N}'_i$ such that the formula $\forall x(\mathcal{A}_j(x) \Rightarrow \mathcal{A}'_j(x))$ is valid. By Lemma 3.4.4 the algorithm Ref-BFS then returns a formula $H(x)$ with the required properties.

For $i = 1$, we first of all observe for all j, $1 \leq j \leq n$, that the formulae $\forall x(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow \neg L(x)) \equiv \forall x(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow \neg L(x) \wedge \mathbf{true})$ are valid. It follows for every j, $1 \leq j \leq n$, that there exists a full merged step clause $\forall x(\mathcal{A}'_j(x) \Rightarrow \bigcirc\mathcal{B}'_j(x)) \in \mathcal{N}'_1$ such that $\mathcal{A}'_j(x) \sqsubseteq \mathcal{A}_j(x)$ and $\mathcal{B}'_j(x) \sqsubseteq \mathcal{B}_j(x)$, which implies that the formula $\forall x(\mathcal{A}_j(x) \Rightarrow \mathcal{A}'_j(x))$ is valid for all j, $1 \leq j \leq n$.

If $i > 1$, then it follows from the induction hypothesis for all j, $1 \leq j \leq n$, that there exists a full merged step clause $\forall x(\mathcal{A}''_j(x) \Rightarrow \bigcirc\mathcal{B}''_j(x)) \in \mathcal{N}'_{i-1}$ such that the formula $\forall x(\mathcal{A}_j(x) \Rightarrow \mathcal{A}''_j(x))$ is valid. Hence, $H'_{i-1}(x) \equiv H''_{i-1}(x) \vee \bigvee_{j=1}^n \mathcal{A}''_j(x)$ for a formula $H''_{i-1}(x)$. We obtain for all j, $1 \leq j \leq n$, from the formula $\forall x(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow \bigvee_{j=1}^n \mathcal{A}_j(x))$ that the formula $\forall x(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow \bigvee_{j=1}^n \mathcal{A}''_j(x))$ is valid and thus, we can conclude for all j, $1 \leq j \leq n$, that the following formulae are valid:

$$\forall x\Big(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow \neg L(x) \wedge (H''_{i-1}(x) \vee \bigvee_{j=1}^{n} \mathcal{A}''_j(x))\Big) \equiv \forall x\Big(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow \neg L(x) \wedge H'_{i-1}(x)\Big)$$

Thus, it is easy to see that for every j, $1 \leq j \leq n$, there exists a full merged step clause $\forall x(\mathcal{A}'_j(x) \Rightarrow \bigcirc\mathcal{B}'_j(x)) \in \mathcal{N}'_i$ such that $\mathcal{A}'_j(x) \sqsubseteq \mathcal{A}_j(x)$ and $\mathcal{B}'_j(x) \sqsubseteq \mathcal{B}_j(x)$. Clearly, the formula $\forall x(\mathcal{A}_j(x) \Rightarrow \mathcal{A}'_j(x))$ is valid for all j, $1 \leq j \leq n$. □

Finally, we show that the Ref-BFS algorithm computes the same loop formulae if it is applied for the same eventuality $\Diamond L(x)$ on two sets $\mathcal{U}, \mathcal{U}'$ with $\mathcal{U} \models \mathcal{U}'$ and $\mathcal{U}' \models \mathcal{U}$.

Lemma 3.4.9. *Let* $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ *and* $P' = \langle \mathcal{U}', \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ *be monodic temporal problems such that* $\mathcal{U} \models \mathcal{U}'$ *and* $\mathcal{U}' \models \mathcal{U}$, *and let* $\Diamond L(x) \in \mathcal{E}$ *be an eventuality clause. Additionally, let* $\mathcal{N}_0, \mathcal{N}_1, \ldots$ *be a sequence of sets and let* $H_0(x), H_1(x), \ldots$ *be a sequence of formulae constructed by a run of the Ref-BFS algorithm applied on the temporal problem* P *for the eventuality* $\Diamond L(x)$. *Finally, let* $\mathcal{N}'_0, \mathcal{N}'_1, \ldots$ *be a sequence of sets and let* $H'_0(x), H'_1(x), \ldots$ *be a sequence of formulae constructed by a run of the Ref-BFS algorithm applied for the eventuality* $\Diamond L(x) \in \mathcal{E}$ *on the temporal problem* P'.

Then it holds for all $j \in \mathbb{N}$ *that* $\mathcal{N}_j = \mathcal{N}'_j$ *and* $H_j(x) = H'_j(x)$.

Proof. By induction on j. For $j = 0$ we have $\mathcal{N}_0 = \emptyset = \mathcal{N}'_0$ and $H_0(x) = \mathbf{true} = H'_0(x)$.

If $j > 0$, then it follows from the induction hypothesis that $\mathcal{N}_{j-1} = \mathcal{N}'_{j-1}$ and $H_{j-1}(x) = H'_{j-1}(x)$. We now show that $\mathcal{N}_j \subseteq \mathcal{N}'_j$. Let $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}_j$. Then, it follows that the formula $\forall x(\mathcal{U} \wedge \mathcal{B}(x) \Rightarrow (\neg L(x) \wedge H_i(x)))$ is valid and that for every $\mathcal{B}'(x) \sqsubset \mathcal{B}(x)$, the formula $\exists x(\mathcal{U} \wedge \mathcal{B}'(x) \wedge \neg(\neg L(x) \wedge H_i(x)))$ is satisfiable. Hence, as $\mathcal{U} \models \mathcal{U}'$ and $\mathcal{U}' \models \mathcal{U}$, the formula $\forall x(\mathcal{U}' \wedge \mathcal{B}(x) \Rightarrow (\neg L(x) \wedge H_i(x)))$ is valid and for every $\mathcal{B}'(x) \sqsubset \mathcal{B}(x)$, the formula $\exists x(\mathcal{U}' \wedge \mathcal{B}'(x) \wedge \neg(\neg L(x) \wedge H_i(x)))$ is satisfiable. We can conclude that $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}_j'$. Analogously, one can show that $\mathcal{N}_j' \subseteq \mathcal{N}_j$. Finally, we also observe that $H_j(x) = H_j'(x)$ as $\mathcal{N}_j = \mathcal{N}_j'$. □

We have now established all the required results to prove the refutational completeness of refined monodic temporal resolution.

3.4.2.1 Proof of Refutational Completeness

The proof of completeness for refined monodic temporal resolution proceeds by simulating a derivation of (regular) monodic temporal resolution $\mathfrak{I}_e$ by refined monodic temporal resolution $\mathfrak{I}_e^{\sqsubseteq}$. For every deduction step $\tilde{\mathcal{U}} \cup \{\tilde{\mathcal{F}}\}$, where $\tilde{\mathcal{F}}$ is the newly derived formula by (regular) monodic temporal resolution, and every set of universal clauses $\mathcal{U}$ with $\mathcal{U} \models \tilde{\mathcal{U}}$, we show that there exists a derivation step $\mathcal{U} \cup \{\mathcal{F}\}$ by refined monodic temporal resolution such that $\mathcal{U} \cup \{\mathcal{F}\} \models \tilde{\mathcal{U}} \cup \{\tilde{\mathcal{F}}\}$. Thus, if the set $\tilde{\mathcal{U}} \cup \{\tilde{\mathcal{F}}\}$ is unsatisfiable, it must also hold that the set $\mathcal{U} \cup \{\mathcal{F}\}$ is unsatisfiable.

We begin by simulating the step resolution rule.

Lemma 3.4.10. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem and let $\tilde{\mathcal{U}}$ be a set of universal formulae such that $\mathcal{U} \models \tilde{\mathcal{U}}$. Additionally, let $\tilde{\mathcal{A}} \Rightarrow \bigcirc\tilde{\mathcal{B}}$ be a merged derived clause such that $\tilde{\mathcal{U}} \cup \{\tilde{\mathcal{B}}\} \models \bot$.*

Then there exists a merged derived step clause $\mathcal{A} \Rightarrow \bigcirc\mathcal{B}$ such that $\mathcal{B} \sqsubseteq \tilde{\mathcal{B}}$, $\mathcal{U} \cup \{\mathcal{B}\} \models \bot$ and $\mathcal{B}$ is minimal with respect to $\mathcal{U}$. Furthermore, it holds that:

$$\mathcal{U} \cup \{\neg\mathcal{A}\} \models \tilde{\mathcal{U}} \cup \{\neg\tilde{\mathcal{A}}\}$$

Proof. It is clear that a merged derived clause $\mathcal{A} \Rightarrow \bigcirc\mathcal{B}$ exists such that $\mathcal{A} \sqsubseteq \tilde{\mathcal{A}}$, $\mathcal{B} \sqsubseteq \tilde{\mathcal{B}}$, $\mathcal{U} \cup \{\mathcal{B}\} \models \bot$ and $\mathcal{B}$ is minimal with respect to $\mathcal{U}$ exists. Thus, $\neg\mathcal{A} \models \neg\tilde{\mathcal{A}}$ and we obtain $\mathcal{U} \cup \{\neg\mathcal{A}\} \models \tilde{\mathcal{U}} \cup \{\neg\tilde{\mathcal{A}}\}$. □

Now, we show that the eventuality resolution rule of (regular) monodic temporal resolution can be simulated in the refined calculus.

Lemma 3.4.11. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem such that the set $\mathcal{U}$ is satisfiable and let $\tilde{\mathcal{U}}$ be a set of universal formulae such that $\mathcal{U} \models \tilde{\mathcal{U}}$. Additionally, let $\forall x(\mathcal{A}_j(x) \Rightarrow \bigcirc\mathcal{B}_j(x))$ for $j \in \{1, \ldots, n\}$ be full merged step clauses and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality clause such that the loop side conditions $\forall x(\tilde{\mathcal{U}} \wedge \mathcal{B}_j(x) \Rightarrow \neg L(x))$ and $\forall x(\tilde{\mathcal{U}} \wedge \mathcal{B}_j(x) \Rightarrow \bigvee_{j=1}^{n} \mathcal{A}_j(x))$ are valid for all j, $1 \leq j \leq n$.*

Then there exists a loop formula $H(x) = \bigvee_{k=1}^{m} \mathcal{A}_k'(x)$ computed by the Ref-BFS algorithm applied on the temporal problem $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ for the eventuality $\Diamond L(x)$ such that the *formula* $\forall x(\bigvee_{j=1}^{n} A_j(x) \Rightarrow \bigvee_{k=1}^{m} A'_k(x))$ *is valid. Moreover, it holds that:*

$$\mathcal{U} \cup \{\forall x(\bigwedge_{k=1}^{m} \neg A'_k(x))\} \models \tilde{\mathcal{U}} \cup \{\forall x(\bigwedge_{j=1}^{n} \neg A_j(x))\}$$

Proof. First of all, as $\mathcal{U} \models \tilde{\mathcal{U}}$ holds, it follows that the formulae $\forall x(\mathcal{U} \wedge B_j(x) \Rightarrow \neg L(x))$ and $\forall x(\mathcal{U} \wedge B_j(x) \Rightarrow \bigvee_{j=1}^{n} A_j(x))$ are valid for all j, $1 \leq j \leq n$. Thus, by Lemma 3.4.8 the Ref-BFS algorithm applied for the eventuality $\Diamond L(x)$ on the temporal problem $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ computes a formula $H(x) = \bigvee_{k=1}^{m} A'_k(x)$ such that $\forall x(\bigvee_{j=1}^{n} A_j(x) \Rightarrow \bigvee_{k=1}^{m} A'_k(x))$ is valid.

Furthermore, we obtain that the formula $\forall x(\bigwedge_{k=1}^{m} \neg A'_k(x) \Rightarrow \bigwedge_{j=1}^{n} \neg A_j(x))$ is valid as well. Hence, we can infer that $\{\forall x(\bigwedge_{k=1}^{m} \neg A'_k(x))\} \models \{\forall x(\bigwedge_{j=1}^{n} \neg A_j(x))\}$, and finally $\mathcal{U} \cup \{\forall x(\bigwedge_{k=1}^{m} \neg A'_k(x))\} \models \tilde{\mathcal{U}} \cup \{\forall x(\bigwedge_{j=1}^{n} \neg A_j(x))\}$ as $\mathcal{U} \models \tilde{\mathcal{U}}$. □

The next proposition then regroups the previous results.

Lemma 3.4.12. *Let* $\mathsf{P} = \langle \tilde{\mathcal{U}}_0, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ *be a monodic temporal problem and let* $\tilde{\Delta} = \tilde{\mathcal{U}}_0 \subsetneq \tilde{\mathcal{U}}_1 \subsetneq \cdots \subsetneq \tilde{\mathcal{U}}_m$ *be a derivation in* $\mathfrak{I}_e$. *Then there exists a derivation* $\Delta = \mathcal{U}_0 \subsetneq \mathcal{U}_1 \subsetneq \cdots \subsetneq \mathcal{U}_n$ *(*$m \geq n$*) in* $\mathfrak{I}_e^{\sqsubseteq}$ *from* P *where* $\mathcal{U}_0 = \tilde{\mathcal{U}}_0$ *and such that* $\mathcal{U}_i \models \tilde{\mathcal{U}}_i$ *for all* i, $1 \leq i \leq n$, *and for all* i, $1 \leq i \leq n$, *either the set* $\mathcal{U}_i$ *is unsatisfiable or it is obtained through an application of the same inference rule as the one used to derive the set* $\tilde{\mathcal{U}}_i$, *except that*

- *every application of the (regular) step resolution rule is replaced by an application of the refined step resolution rule on a step clause* $A' \Rightarrow \bigcirc B'$ *with* $B' \sqsubseteq B$, *where* $A \Rightarrow \bigcirc B$ *is the premise of the (regular) step resolution rule,*
- *the application of the eventuality resolution rules is only performed on satisfiable sets of universal formulae and it is restricted to loops found by the Ref-BFS algorithm.*

Proof. By induction on the length n of the derivation employing Lemmata 3.4.10 and 3.4.11 and by using the observation that for every two sets of universal clauses $\mathcal{U}, \tilde{\mathcal{U}}$ with $\mathcal{U} \models \tilde{\mathcal{U}}$ and a set of initial clauses $\mathcal{I}$, $\mathcal{U} \cup \mathcal{I} \models \tilde{\mathcal{U}} \cup \mathcal{I}$ holds. □

Finally, we can establish the refutational completeness of refined monodic temporal resolution for monodic temporal problems with unique left-hand sides in step clauses.

Lemma 3.4.13. *Let* $\mathsf{P} = \langle \mathcal{U}_0, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ *be an unsatisfiable monodic temporal problem with unique left-hand sides in step clauses and let* P^c *be its constant-flooded form. Then there exists a successfully terminating derivation* Δ *in* $\mathfrak{I}_e^{\sqsubseteq}$ *from* P^c *which restricts the application of the eventuality resolution rules to loops found by the Ref-BFS algorithm. Furthermore, the refined step resolution rule and the eventuality resolution rules are only applied on satisfiable sets of universal formulae in the derivation* Δ.

Proof. By Theorem 3.2.8 there exists a successfully terminating derivation $\Delta' = \tilde{\mathcal{U}}_0 \subsetneq \tilde{\mathcal{U}}_1 \subsetneq \ldots \subsetneq \tilde{\mathcal{U}}_m$ in $\mathfrak{I}_e$ from P^c such that $\perp$ in $\tilde{\mathcal{U}}_m$ and $\perp \notin \mathcal{U}_j$ for all $1 \leq j < m$.

Then, by Lemma 3.4.12 there exists a $\mathfrak{I}_e^{\subseteq}$-derivation $\Delta = \mathcal{U}_0 \subsetneq \mathcal{U}_1 \subsetneq \ldots \subsetneq \mathcal{U}_n$ $(m \geq n)$ such that either $\bot \in \mathcal{U}_n$ or the set $\mathcal{U}_n$ is unsatisfiable. Furthermore, the eventuality resolution rules are only applied on satisfiable sets of universal formulae in the derivation Δ. Moreover, if the set $\mathcal{U}_n$ is unsatisfiable but $\bot \notin \mathcal{U}_n$, we can apply the termination rule and extend the derivation Δ with the set $\mathcal{U}_{n+1} = \mathcal{U}_n \cup \{\bot\}$.

Finally, if in the derivation Δ the refined step resolution rule is applied on an unsatisfiable set of universal formulae, then the first such application can be replaced by an application of the termination rule in order to obtain the desired result. □

Note that the condition which states that the step resolution rule and the Ref-BFS algorithm are only applied on satisfiable sets of universal formulae is again necessary for ensuring the correspondence between specific full merged or merged derived clauses and those step clauses found by resolution-based methods.

3.4.3 Proof of the Lifting Theorem

Before we can show the refutational completeness of ordered fine-grained resolution with selection, we have to prove the lifting theorem for $\mathfrak{I}_{FG}^{S,\succ}$ without considering the eventuality resolution and the duplicate literal elimination in terminating step clauses rules.

But as the inference rules of fine-grained step resolution that involve step clauses require special restrictions on most general unifiers, we need to analyse the notions of substitutions and most general unifiers in greater detail before we can prove the lifting theorem. For example, for the instantiation $p(c) \Rightarrow \bigcirc q(c)$ of the step clause $p(x) \Rightarrow \bigcirc q(x)$ using the substitution $\sigma = [x \mapsto c]$, there exists a resolution inference between the step clause $p(c) \Rightarrow \bigcirc q(c)$ and the universal clause $\neg q(c)$. But due to the restrictions imposed on most general unifiers for variables occurring in left-hand sides of step clauses, no inference is possible between the uninstantiated step clause $p(x) \Rightarrow \bigcirc q(x)$ and the universal clause $\neg q(c)$. Consequently, in order to obtain a result that is similar to the "traditional" lifting theorem (see, e.g., [10]), one can, for instance, allow specific instances of clauses only. A "traditional" result can be shown for ordered fine-grained resolution with selection if the substitutions that link the non-instantiation to the instantiation level only map variables into constants that do not occur on the uninstantiated level, which is also sufficient for proving the refutational completeness of the $\mathfrak{I}_{FG}^{S,\succ}$-calculus. Additionally, as arbitrary function symbols can be introduced during the Skolemization process, we have to prove some results involving arbitrary terms and not only constants, which complicates the required proof steps considerably.

First of all, we show some properties of substitutions and most general unifiers. Unless noted otherwise, we assume in this section that Σ is a set of constants or function symbols, i.e. $\Sigma \subseteq CS \cup FS$, and that atoms are built over the term set $T_\Sigma(X)$.

Lemma 3.4.14. *Let $\sigma \colon X \to T_\Sigma(X)$ be a substitution. Then σ is idempotent if and only if it holds that $dom(\sigma) \cap var(codom(\sigma)) = \emptyset$.*

Proof. Can be found in [62]. □

Lemma 3.4.15. *Let $\sigma: X \to T_\Sigma(X)$ be a substitution. Then σ is a variable renaming if and only if σ is invertible.*

Proof. Can be found in [62]. □

Lemma 3.4.16. *Let $\sigma: X \to T_\Sigma(X)$ be a variable renaming. Then σ is injective.*

Proof. By Lemma 3.4.15 the substitution σ is invertible with inverse σ^{-1} such that $\sigma\sigma^{-1} = id$. If we assume that there are variables $x, y \in X$ with $x \neq y$ and $\sigma(x) = \sigma(y)$, then it follows that $x = x\sigma\sigma^{-1} = y\sigma\sigma^{-1} = y$, which contradicts the fact that $x \neq y$. □

Lemma 3.4.17. *Let $\sigma: X \to T_\Sigma(X)$ be an invertible substitution and let $\sigma^{-1}: X \to T_\Sigma(X)$ be a substitution with $\sigma\sigma^{-1} = id$. Then σ^{-1} is invertible.*

Proof. First of all, as the substitution σ is invertible, we can infer that $dom(\sigma) = codom(\sigma)$ by Lemma 3.4.15. Additionally, $codom(\sigma) \subseteq X$ and $codom(\sigma^{-1}) \subseteq X$ must hold. Furthermore, it is easy to verify that $dom(\sigma^{-1}) = codom(\sigma)$ and $codom(\sigma^{-1}) = dom(\sigma)$, which implies that $dom(\sigma^{-1}) = codom(\sigma^{-1})$. Hence, we can conclude that σ^{-1} is invertible by applying Lemma 3.4.15 again. □

Lemma 3.4.18. *Let $\sigma, \tau: X \to T_\Sigma(X)$ be variable renamings. Then it holds that $\sigma\tau$ is a variable renaming.*

Proof. It follows from Lemma 3.4.15 that there are substitutions σ^{-1}, τ^{-1} with $\sigma\sigma^{-1} = id = \tau\tau^{-1}$. Thus, $(\sigma\tau)(\tau^{-1}\sigma^{-1}) = id$ and $\sigma\tau$ is invertible, which implies that $\sigma\tau$ is a variable renaming by Lemma 3.4.15. □

Lemma 3.4.19. *Let $\sigma, \tau: X \to T_\Sigma(X)$ be most general unifiers of two atoms A and B such that σ is idempotent. Then it holds that $\tau = \sigma\tau$.*

Proof. As the substitution τ is a unifier of the atoms A and B, there exists a substitution β with $\tau = \sigma\beta$. Additionally, as σ is idempotent, it holds that $\tau = (\sigma\sigma)\beta = \sigma(\sigma\beta) = \sigma\tau$. □

The following proposition can also be found in [62].

Lemma 3.4.20. *Let $\sigma, \tau: X \to T_\Sigma(X)$ be most general unifiers of two atoms A and B such that σ is idempotent. Then there exists a variable renaming φ with $\sigma\varphi = \tau$.*

Proof. First of all, we define a substitution α as follows. For $x \in X$, let

$$\alpha(x) = \begin{cases} \tau(x) & \text{if } x \in dom(\tau) \text{ and } x \notin dom(\sigma) \\ x & \text{otherwise} \end{cases}$$

As σ is idempotent, it holds by Lemmata 3.4.14 and 3.4.19 that $\tau = \sigma\tau = \sigma\alpha$. Moreover, as the substitution σ is a unifier of the two atoms A and B, there exists a substitution β with

$\sigma = \tau\beta$. Thus, we obtain $\sigma = \tau\beta = \sigma\alpha\beta$. Now, for $x \in dom(\alpha)$ it holds that $x \notin dom(\sigma)$, which implies that $x = x\alpha\beta$. As $\alpha(x) \neq x$, it follows that $codom(\alpha) \subseteq dom(\beta) \subseteq X$ and that $\alpha|_{dom(\alpha)}$ is injective as for any two variables $x, y \in dom(\alpha)$ with $x\alpha = y\alpha$ it holds that $x = x\alpha\beta = y\alpha\beta = y$.

Let $\gamma\colon codom(\alpha) \setminus dom(\alpha) \longrightarrow dom(\alpha) \setminus codom(\alpha)$ now be a bijection. We then define a substitution $\alpha' = \alpha \cup \gamma$. The substitution α' is well-defined and it is a variable renaming as

$$dom(\alpha') = dom(\alpha) \cup (codom(\alpha) \setminus dom(\alpha)) = codom(\alpha) \cup (dom(\alpha) \setminus codom(\alpha)) = codom(\alpha')$$

Furthermore, we show that $codom(\alpha) \setminus dom(\alpha) \subseteq dom(\sigma)$. Let therefore $x \in codom(\alpha) \setminus dom(\alpha)$. If we assume that $x \notin dom(\sigma)$, then it follows that $x = x\beta$ as $\sigma = \sigma\alpha\beta$. Thus, $x \notin dom(\beta)$, which contradicts the fact that $codom(\alpha) \subseteq dom(\beta)$.

Finally, we can conclude that $\tau = \sigma\alpha = \sigma\alpha'$ as $dom(\sigma) \cap var(codom(\sigma)) = \emptyset$. □

Lemma 3.4.21. *Let $\sigma\colon X \longrightarrow T_\Sigma(X)$ be a most general unifier of two atoms A and B. Additionally, let ζ be a variable renaming. Then it holds that $\sigma\zeta$ is a most general unifier of A and B.*

Proof. Let τ be a unifier of A and B. We need to show that there is a substitution φ with $\sigma\zeta\varphi = \tau$.

As σ is a most general unifier of A and B, it follows that there is a substitution ψ such that $\sigma\psi = \tau$. Moreover, by Lemma 3.4.15 there exists a substitution ζ^{-1} such that $\zeta\zeta^{-1} = id$, which implies that

$$(\sigma\zeta)(\zeta^{-1}\psi) = (\sigma(\zeta\zeta^{-1}))\psi = \sigma\psi = \tau$$

Hence, by choosing $\varphi = \zeta^{-1}\psi$, we obtain the desired result. □

The next major result in this section will be established by Lemma 3.4.32, which can be seen as a lifting lemma for most general unifiers. It states that for a set of constants $\mathbf{C} \subseteq CS$ and atoms $A, \tilde{A}, B, \tilde{B}$ such that the substitution $\sigma\colon X \longrightarrow T_\Sigma(X)$ is a most general unifier of $\tilde{A}$, $\tilde{B}$ and such that there is a substitution $\eta\colon X \longrightarrow T_\Sigma(X)$ with $A\eta = \tilde{A}$, $B\eta = \tilde{B}$, and $codom(\eta) \subseteq \mathbf{C}$, there exists a most general unifier $\tau\colon X \longrightarrow T_\Sigma(X)$ of A and B, and a substitution $\varphi\colon X \longrightarrow T_\Sigma(X)$ with $\tau\varphi = \eta\sigma$ and $codom(\varphi) \subseteq \mathbf{C}$.

The proof of the following lemmata relies on derivations produced by the rule-based unification algorithm $\mathcal{U}$ shown in Figure 3.5, which computes a most general unifier. The algorithm has been introduced in [7]. We prove the existence of most general unifiers with certain properties by analysing derivations produced by the algorithm $\mathcal{U}$.

We first of all define the following notions that are related to the algorithm $\mathcal{U}$.

Definition 3.4.22. *In the context of the rule-based unification algorithm $\mathcal{U}$ we define that:*

(i) A system *A is either $\bot$ (the* failure system*) or a pair $P; S$ consisting of a multiset P of unification problems and a set of equations S in solved form.*

Trivial:

$$\{s \overset{?}{=} s\} \cup P'; S \Rightarrow P'; S$$

Decomposition:

$$\{f(s_1, \ldots, s_n) \overset{?}{=} f(t_1, \ldots, t_n)\} \cup P'; S \Rightarrow \{s_1 \overset{?}{=} t_1, \ldots, s_n \overset{?}{=} t_n\} \cup P'; S$$

for $n \geq 0$

Symbol Clash:

$$\{f(s_1, \ldots, s_n) \overset{?}{=} g(t_1, \ldots, t_m)\} \cup P'; S \Rightarrow \bot$$

if $f \neq g$

Orient:

$$\{t \overset{?}{=} x\} \cup P'; S \Rightarrow \{x \overset{?}{=} t\} \cup P'; S$$

if t is not a variable

Occurrence Check:

$$\{x \overset{?}{=} t\} \cup P'; S \Rightarrow \bot$$

if $x \in var(t)$ and $x \neq t$

Variable Elimination:

$$\{x \overset{?}{=} t\} \cup P'; S \Rightarrow P'\{x \mapsto t\}; S\{x \mapsto t\} \cup \{x \approx t\}$$

if $x \notin var(t)$

Figure 3.5: Rule-Based Unification Algorithm $\mathcal{U}$

(ii) *A derivation $\mathcal{D}$ produced by the algorithm $\mathcal{U}$ is a sequence of systems*

$$A_1, A_2, \ldots, A_n$$

where $n > 1$ and for every i with $1 < i \leq n$, the system A_i results from the system A_{i-1} through an application of one of the inference rules depicted in Figure 3.5. The length of the derivation $\mathcal{D}$ is denoted by $|\mathcal{D}|$, i.e. $|\mathcal{D}| = n$.

(iii) *A derivation $\mathcal{D} = A_1, \ldots, A_n$ is said to be* maximal *if and only if $A_n = \bot$ or $A_n = \emptyset; S$ for a set of equations S in solved form.*

(iv) *For a set $S = \{x_1 \overset{?}{=} t_1, \ldots, x_n \overset{?}{=} t_n\}$ of equations in solved form, we write σ_S to denote the substitution $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ that is induced by the set of equations S.*

Thus, the algorithm $\mathcal{U}$ derives new systems from given systems through its deduction rules. When started on a system $\{A \overset{?}{=} B\}; \emptyset$ with two unifiable atoms A and B, it eventually computes a most general unifier of the atoms A and B as stated in the following correctness result for the algorithm $\mathcal{U}$.

Theorem 3.4.23 (see [7]). *Let A and B be two unifiable atoms.*

Then any maximal derivation computed by the algorithm $\mathcal{U}$ shown in Figure 3.5 starting from the system $P; S = \{A \overset{?}{=} B\}; \emptyset$ results in a system $\emptyset; S$ such that σ_S is an idempotent *most general unifier of A and B, $(dom(\sigma_S) \cup (codom(\sigma_S) \cap X)) \subseteq var(A) \cup var(B)$ and $const(codom(\sigma_S)) \subseteq const(A) \cup const(B)$.*

The proofs of the next two lemmata rely on Theorem 3.4.23.

Lemma 3.4.24. *Let A, B be two unifiable atoms and $\sigma \colon X \longrightarrow T_\Sigma(X)$ be a most general unifier of A and B. Then it holds that $const(codom(\sigma)) \subseteq const(A) \cup const(B)$.*

Proof. First of all, it follows from Theorem 3.4.23 that the unification algorithm shown in Figure 3.5 computes a most general unifier λ of A and B such that $const(codom(\lambda)) \subseteq const(A) \cup const(B)$. Moreover, as σ is a most general unifier of A and B, there is a substitution φ such that $\sigma\varphi = \lambda$. If we now assume that there is a variable x and a constant $c \in const(\sigma(x))$ such that $c \notin const(A) \cup const(B)$, then it follows that $c \in const((\sigma(x))\varphi)$ as $c\varphi = c$. Thus, $c \in const(codom(\lambda)) \subseteq const(A) \cup const(B)$, which is obviously a contradiction. □

Lemma 3.4.25. *Let $\sigma, \tau \colon X \longrightarrow T_\Sigma(X)$ be most general unifiers of two atoms A and B. Then there exists a variable renaming φ with $\sigma\varphi = \tau$.*

Proof. By using, for instance, the algorithm $\mathcal{U}$ depicted in Figure 3.5, we obtain from Theorem 3.4.23 that there exists an idempotent most general unifier θ of the atoms A and B. Moreover, it follows from Lemma 3.4.20 that there are variable renamings φ,ψ with $\theta\varphi = \sigma$ and $\theta\psi = \tau$. Lemma 3.4.15 implies that there is a substitution φ^{-1} with $\varphi\varphi^{-1} = id$. Thus, $\theta = \sigma\varphi^{-1}$ and $\sigma(\varphi^{-1}\psi) = (\sigma\varphi^{-1})\psi = \theta\psi = \tau$. Finally, it follows from Lemmata 3.4.15, 3.4.17 and 3.4.18 that $\varphi^{-1}\psi$ is a variable renaming. □

The next major result in this section will be established by Lemma 3.4.32. For a set $\mathbf{C} \subseteq CS$ of constants and atoms $A, \tilde{A}, B, \tilde{B}$ such that the substitution $\sigma \colon X \longrightarrow T_\Sigma(X)$ is a most general unifier of $\tilde{A}$ and $\tilde{B}$ and such that there is a substitution $\eta \colon X \longrightarrow T_\Sigma(X)$ with $A\eta = \tilde{A}$, $B\eta = \tilde{B}$, and $codom(\eta) \subseteq \mathbf{C}$, the existence of the most general unifier $\tau \colon X \longrightarrow T_\Sigma(X)$ of A and B, and of the substitution $\varphi \colon X \longrightarrow T_\Sigma(X)$ with $\tau\varphi = \eta\sigma$ and $codom(\varphi) \subseteq \mathbf{C}$ will be shown by analysing a special kind of derivations of the algorithm $\mathcal{U}$.

We will only consider *liftable* derivations produced by the unification algorithm $\mathcal{U}$ depicted in Figure 3.5:

Definition 3.4.26. *Let $\mathcal{D} = P_1; S_1 \Rightarrow P_2; S_2 \Rightarrow \ldots \Rightarrow P_n; S_n$ be a derivation produced by the algorithm $\mathcal{U}$. We say that the derivation $\mathcal{D}$ is* liftable *if and only if the following three conditions are satisfied:*

(i) the trivial *rule is only applied to pairs of variables $x \stackrel{?}{=} x$ or to pairs of constants $c \stackrel{?}{=} c$, and*

(ii) the decomposition *rule is only applied to pairs of terms $f(s_1, \ldots, s_n) \stackrel{?}{=} f(t_1, \ldots, t_n)$ with $n > 0$, and finally*

(iii) every application of the orient *rule is immediately followed by an application of the* occurrence check *or* variable elimination *rule.*

The next lemma then proves some simple properties of derivations produced by the algorithm $\mathcal{U}$.

Lemma 3.4.27. *Let $\mathcal{D} = P_1; S_1 \Rightarrow \ldots \Rightarrow P_n; S_n$ $(n \geq 2)$ be a derivation produced by the algorithm $\mathcal{U}$ such that $S_1 = \emptyset$.*

Then it holds that for every i with $1 \leq i \leq n$ that

- $dom(\sigma_{S_i}) \subseteq dom(\sigma_{S_{i+1}})$ *for* $i < n$,
- $\{x \overset{?}{=} t\} \in P_i$ *implies that* $x \notin dom(\sigma_{S_i})$,
- $const(codom(\sigma_{S_i})) \subseteq const(P_{i-1}) \cup const(S_{i-1})$ *for* $i > 1$, *and*
- $dom(\sigma_{S_i}) \subseteq var(P_1)$.

Proof. Follows straightforwardly from the definition of the algorithm $\mathcal{U}$ shown in Figure 3.5. □

The following two lemmata prove some results that are needed for Lemma 3.4.30.

Lemma 3.4.28. *Let $\mathbf{C} \subseteq CS$ be a set of constants. Additionally, let $\tilde{P}; \tilde{S} \Rightarrow \tilde{P}'; \tilde{S}'$ be a transformation step performed by the algorithm $\mathcal{U}$ in a liftable derivation that has been initially started on an empty set of solved equations such that the* orient *rule is not applied and such that there is a substitution $\lambda\colon X \longrightarrow T_\Sigma(X)$, a multiset of equations P and a set of equations in solved form S with*

(a) $codom(\lambda) \subseteq \mathbf{C}$,

(b) $P\lambda = \tilde{P}$,

(c) $\forall x \in dom(\sigma_{\tilde{S}})\colon x\sigma_S\lambda = x\sigma_{\tilde{S}}$, *and*

(d) $\forall x \in dom(\sigma_S)\colon x \notin dom(\sigma_{\tilde{S}}) \Rightarrow x\sigma_S\lambda = x\lambda$.

Then there exists a multiset of equations P'', a set of equations S'', a derivation $P; S \Rightarrow^* P''; S''$ produced by the algorithm $\mathcal{U}$ and a substitution $\varphi\colon X \longrightarrow T_\Sigma(X)$ such that

(i) $dom(\lambda) \subseteq dom(\varphi)$,

(ii) $dom(\varphi) \setminus dom(\lambda) \subseteq dom(\sigma_{\tilde{S}'})$,

(iii) $\forall x \in dom(\lambda)\colon \varphi(x) = \lambda(x)$,

(iv) $codom(\varphi) \subseteq \mathbf{C}$,

(v) $P'\varphi = \tilde{P}'$,

(vi) $\forall x \in dom(\sigma_{\tilde{S}'})\colon x\sigma_{S'}\varphi = x\sigma_{\tilde{S}'}$,

(vii) $\forall x \in dom(\sigma_{S'})\colon x \notin dom(\sigma_{\tilde{S}'}) \Rightarrow x\sigma_{S'}\varphi = x\varphi$, *and*

(viii) $\forall x \in dom(\sigma_{S'}) \setminus dom(\sigma_S)\colon x \notin dom(\sigma_{\tilde{S}'}) \Rightarrow x \in dom(\lambda)$.

Proof. We have to distinguish between the different rules that could have been used to obtain the derivation step $\tilde{P};\tilde{S} \Rightarrow \tilde{P}';\tilde{S}'$. Since $\tilde{P}';\tilde{S}'$ is not the failure system $\perp$, neither the *occurrence check*, nor the *symbol clash* rule could have been applied in the derivation of $\tilde{P}';\tilde{S}'$.

For the *trivial* rule, we obtain $\tilde{P} = \{\tilde{s} \overset{?}{=} \tilde{s}\} \cup \tilde{Q}$, $\tilde{P}' = \tilde{Q}$ and $\tilde{S} = \tilde{S}'$. As the considered derivation is liftable, we can conclude that $\tilde{s}$ is either a variable or a constant. Let now $s \overset{?}{=} t \in P$ such that $s\lambda = \tilde{s}$ and $t\lambda = \tilde{s}$. If $s = t$, then the *trivial* rule can be applied on $P;S = \{s \overset{?}{=} s\} \cup Q;S$. The result of the application is the system $P'';S'' = Q;S$, i.e. $S'' = S$, and as $Q\lambda = \tilde{Q}$, the properties (i) to (viii) are obviously satisfied for the substitution λ. Otherwise, $s \neq t$ holds, and as $codom(\lambda) \subseteq \mathbf{C}$, there are the following possibilities: $s \in \mathbf{C}$ and $t \in X$, or $s \in X$ and $t \in \mathbf{C}$, or $s \in X$ and $t \in X$.

If $s \in X$ and $t = c \in \mathbf{C}$, we can infer that $\lambda(s) = c = t = \tilde{s}$. Thus, the *variable elimination* rule can be applied on the system $P;S = \{s \overset{?}{=} c\} \cup Q;S$, which results in the system $P'';S'' = Q\{s \mapsto c\};S\{s \mapsto c\} \cup \{s \approx c\}$. It is then easy to see that $\sigma_{S''} = \sigma_S\{s \mapsto c\}$. Now, let $\varphi := \lambda$. The properties(i), (ii), (iii) and (iv) are then trivially satisfied. As $Q\lambda = \tilde{Q}$, $\lambda(s) = c$, and therefore, $\{s \mapsto c\}\lambda = \lambda$, it holds that $P''\varphi = P''\lambda = Q\{s \mapsto c\}\lambda = \tilde{Q} = \tilde{P}'$, i.e. property (v) is satisfied. For property (vi), let $z \in dom(\sigma_{\tilde{S}''}) = dom(\sigma_{\tilde{S}})$. Hence, $z\sigma_S\varphi = z\sigma_S\lambda = z\sigma_{\tilde{S}} = z\sigma_{\tilde{S}'}$ and as again $\{s \mapsto c\}\varphi = \varphi$, we obtain $z\sigma_{S''}\varphi = z\sigma_{\tilde{S}';}$, i.e. property (vi) holds. Now for property (vii), it first of all holds that $s\sigma_{S''}\varphi = c = s\varphi$. We still have to consider the case when $z \in dom(\sigma_{S''})$ such that $z \notin dom(\sigma_{\tilde{S}'})$ and $z \neq x$. It follows that $z \in dom(\sigma_S)$ and $z \notin dom(\sigma_{\tilde{S}})$, which implies $z\sigma_S\varphi = z\sigma_S\lambda = z\lambda = z\varphi$. And therefore, as $\{s \mapsto c\}\varphi = \varphi$, it holds that $z\sigma_{S''}\varphi = z\varphi$. Finally, we note that $s \in dom(\lambda)$ and thus, property (viii) is satisfied.

If $s = d \in \mathbf{C}$ and $t \in X$ with $\lambda(t) = d = s$, we can apply the *orient rule* rule on the system $P;S = \{d \overset{?}{=} t\} \cup Q;S$. The result of its application is the system $P';S' = \{t \overset{?}{=} d\} \cup Q;S$. It is now possible to apply the *variable elimination* rule on the system $P';S'$, resulting in the system $P'';S'' = Q\{t \mapsto d\};S\{t \mapsto d\} \cup \{t \approx d\}$. Again, note that $\sigma_{S''} = \sigma_S\{t \mapsto d\}$, and for the substitution $\varphi := \lambda$ it holds that $\{t \mapsto d\}\lambda = \lambda$. Similarly to the previous case one can show that the properties (i) to (viii) are satisfied.

Finally, if $s = x \in X$ and $t = y \in X$, we can apply the *variable elimination* rule on the system $P;S = \{x \overset{?}{=} y\} \cup Q;S$, which derives the system $P'';S'' = Q\{x \mapsto y\};S\{x \mapsto y\} \cup \{x \approx y\}$. It is again easy to see that $\sigma_{S''} = \sigma_S\{x \mapsto y\}$. Let now $\varphi := \lambda$, for which it holds that $\{x \mapsto y\}\varphi = \varphi$ as $\varphi(x) = \varphi(y)$. Again, it can be shown similarly to the case of $s \in X$ and $t \in \mathbf{C}$ that the properties (i) to (viii) are satisfied.

For the *decomposition* rule we obtain $\tilde{P} = \{f(\tilde{s}_1,\ldots,\tilde{s}_n) \overset{?}{=} f(\tilde{t}_1,\ldots,\tilde{t}_n)\} \cup \tilde{Q}$, $\tilde{P}' = \{\tilde{s}_1 \overset{?}{=} \tilde{t}_1,\ldots,\tilde{s}_n \overset{?}{=} \tilde{t}_n\} \cup \tilde{Q}$ and $\tilde{S} = \tilde{S}'$. As the considered derivation is liftable, we have $n > 0$.

Additionally, as $codom(\lambda) \subseteq \mathbf{C}$, it follows that there is $f(s_1, \ldots, s_n) \overset{?}{=} f(t_1, \ldots, t_n) \in P$ such that $(f(s_1, \ldots, s_n))\lambda = f(\tilde{s}_1, \ldots, \tilde{s}_n)$ and also $(f(t_1, \ldots, t_n))\lambda = f(\tilde{t}_1, \ldots, \tilde{t}_n)$. Consequently, we can apply the *decomposition* rule to $P; S = \{f(s_1, \ldots, s_n) \overset{?}{=} f(t_1, \ldots, t_n)\} \cup Q; S$, which yields the system $P''; S'' = \{s_1 \overset{?}{=} t_1, \ldots, s_n \overset{?}{=} t_n\} \cup Q; S$. As $Q\lambda = \tilde{Q}$ and thus $P''\lambda = \tilde{P}'$, it obviously holds that the properties (i) to (viii) are satisfied for the substitution λ.

For the *variable elimination* rule, we have $\tilde{P} = \{\tilde{x} \overset{?}{=} \tilde{t}\} \cup \tilde{Q}$, $\tilde{P}' = \tilde{Q}\{\tilde{x} \mapsto \tilde{t}\}$, $\tilde{S}' = \tilde{S}\{\tilde{x} \mapsto \tilde{t}\} \cup \{\tilde{x} \overset{?}{=} \tilde{t}\}$ and $\tilde{x} \notin var(\tilde{t})$. Let $x \overset{?}{=} t \in P$ such that $x\lambda = \tilde{x}$ and $t\lambda = \tilde{t}$. Thus, as $\tilde{x} \in X$, it must hold that $\lambda(x) = x = \tilde{x}$. Furthermore, we obtain $x \notin var(t)$ as otherwise $\tilde{x} \in var(\tilde{t})$. So, we can apply the variable elimination rule on $P; S = \{x \overset{?}{=} t\} \cup Q; S$, and the result of its application is the system $P''; S'' = Q\{x \mapsto t\}; S\{x \mapsto t\} \cup \{x \overset{?}{=} t\}$. Let now $\varphi := \lambda$. The properties (i), (ii), (iii) and (iv) are trivially satisfied. Additionally, as $\lambda(x) = x$ and $codom(\varphi) = codom(\lambda) \subseteq \mathbf{C}$, we get:

$$\{x \mapsto t\}\varphi = \varphi\{x \mapsto t\varphi\} = \varphi\{\tilde{x} \mapsto \tilde{t}\}$$

And hence, as $Q\varphi = Q\lambda = \tilde{Q}$:

$$\begin{aligned} P''\varphi = (Q\{x \mapsto t\})\varphi = Q(\{x \mapsto t\}\varphi) = Q(\varphi\{\tilde{x} \mapsto \tilde{t}\}) &= (Q\varphi)\{\tilde{x} \mapsto \tilde{t}\} \\ &= \tilde{Q}\{\tilde{x} \mapsto \tilde{t}\} \\ &= \tilde{P}' \end{aligned}$$

Property (v) is thus satisfied. Moreover, it follows from the properties of the algorithm $\mathcal{U}$ (see Lemma 3.4.27) that $x \notin dom(\sigma_S)$ and that $x = \tilde{x} \notin dom(\sigma_{\tilde{S}})$. Thus, we obtain $\sigma_{S''} = \sigma_S\{x \mapsto t\}$ and $\sigma_{\tilde{S}'} = \sigma_{\tilde{S}}\{\tilde{x} \mapsto \tilde{t}\}$. For property (vi), it first of all holds that $x\sigma_{S''}\varphi = t\varphi = \tilde{t} = x\sigma_{\tilde{S}'}$. Then, let $z \in dom(\sigma_{\tilde{S}'})$ such that $z \neq \tilde{x}$, which implies that $z \in dom(\sigma_{\tilde{S}})$ and thus $z\sigma_S\varphi = z\sigma_S\lambda = z\sigma_{\tilde{S}}$. Applying the substitution $\{\tilde{x} \mapsto \tilde{t}\}$ on both sides of the equality yields the following result:

$$z\sigma_{S''}\varphi = z\sigma_S\{x \mapsto t\}\varphi = z\sigma_S\varphi\{\tilde{x} \mapsto \tilde{t}\} = z\sigma_{\tilde{S}}\{\tilde{x} \mapsto \tilde{t}\} = z\sigma_{\tilde{S}'}$$

Then, for property (vii), let $z \in dom(\sigma_{S''})$ such that $z \notin dom(\sigma_{\tilde{S}'})$, which implies that $z \neq x$. Thus, $z \in dom(\sigma_S)$ and as $dom(\sigma_{\tilde{S}}) \subseteq dom(\sigma_{\tilde{S}'})$ (see Lemma 3.4.27), it holds that $z \notin dom(\sigma_{\tilde{S}})$. We can conclude from the assumptions then that $z\sigma_S\varphi = z\sigma_S\lambda = z\lambda = z\varphi$. Thus, as $codom(\varphi) \subseteq \mathbf{C}$, it holds that either $z\sigma_S\varphi \in X$ or $z\sigma_S\varphi \in \mathbf{C}$. Consequently, we obtain that $z\sigma_S \in X$ or $z\sigma_S \in \mathbf{C}$. Furthermore, if we assume $z\sigma_S = x$, then it would follow that $x = x\varphi = z\sigma_S\varphi = z\varphi$, which contradicts property (iv). Hence, $z\sigma_S \neq x$ and:

$$z\sigma_{S''}\varphi = z\sigma_S\{x \mapsto t\}\varphi = z\sigma_S\varphi = z\varphi$$

Finally, it holds that $x = \tilde{x} \in dom(\sigma_{\tilde{S}'})$ and thus, property (viii) is satisfied. □

Lemma 3.4.29. *Let $\mathbf{C} \subseteq CS$ be a set of constants. Additionally, let $\tilde{P}; \tilde{S} \Rightarrow \tilde{P}'; \tilde{S}' \Rightarrow \tilde{P}''; \tilde{S}''$ be two consecutive transformation steps performed by the algorithm $\mathcal{U}$ in a liftable derivation that has been initially started on an empty set of solved equations such that the system $\tilde{P}'; \tilde{S}'$*

is obtained through the orient *rule and such that there is a substitution* $\lambda\colon X \longrightarrow T_\Sigma(X)$, *a multiset of equations* P *and a set of equations in solved form* S *with*

(a) $codom(\lambda) \subseteq \mathbf{C}$,

(b) $P\lambda = \tilde{P}$,

(c) $\forall x \in dom(\sigma_{\tilde{S}})\colon x\sigma_S\lambda = x\sigma_{\tilde{S}}$, *and*

(d) $\forall x \in dom(\sigma_S)\colon x \notin dom(\sigma_{\tilde{S}}) \Rightarrow x\sigma_S\lambda = x\lambda$.

Then there exists a multiset of equations P'', *a set of equations* S'', *a derivation* $P; S \Rightarrow^* P''; S''$ *produced by the algorithm* $\mathcal{U}$ *and a substitution* $\varphi\colon X \longrightarrow T_\Sigma(X)$ *such that*

(i) $dom(\lambda) \subseteq dom(\varphi)$,

(ii) $dom(\varphi) \setminus dom(\lambda) \subseteq dom(\sigma_{\tilde{S}'})$,

(iii) $\forall x \in dom(\lambda)\colon \varphi(x) = \lambda(x)$,

(iv) $codom(\varphi) \subseteq \mathbf{C}$,

(v) $P''\varphi = \tilde{P}''$,

(vi) $\forall x \in dom(\sigma_{\tilde{S}''})\colon x\sigma_{S''}\varphi = x\sigma_{\tilde{S}''}$,

(vii) $\forall x \in dom(\sigma_{S''})\colon x \notin dom(\sigma_{\tilde{S}''}) \Rightarrow x\sigma_{S''}\varphi = x\varphi$, *and*

(viii) $\forall x \in dom(\sigma_{S''}) \setminus dom(\sigma_S)\colon x \notin dom(\sigma_{\tilde{S}''}) \Rightarrow x \in dom(\lambda)$.

Proof. Let $\tilde{P} = \{\tilde{t} \overset{?}{=} \tilde{x}\} \cup \tilde{Q}$ and $\tilde{P}' = \{\tilde{x} \overset{?}{=} \tilde{t}\} \cup \tilde{Q}$ such that $\tilde{t}$ is not a variable and hence, $x \neq \tilde{t}$. Additionally, $\tilde{S} = \tilde{S}'$ holds and as the derivation $\tilde{P}'; \tilde{S}' \Rightarrow \tilde{P}''; \tilde{S}''$ is liftable but not failing, it must have been obtained through the *variable elimination* rule. Hence, $\tilde{P}'' = \tilde{Q}\{\tilde{x} \mapsto \tilde{t}\}$ and $\tilde{S}'' = \tilde{S}\{\tilde{x} \mapsto \tilde{t}\} \cup \{\tilde{x} \overset{?}{=} \tilde{t}\}$ such that $\tilde{x} \notin var(\tilde{t})$. Let $t \overset{?}{=} x \in P$ such that $t\lambda = \tilde{t}$ and $x\lambda = \tilde{x}$. Thus, as $\tilde{x} \in X$, it must hold that $\lambda(x) = x = \tilde{x}$. Furthermore, we obtain $x \notin var(t)$ as otherwise $\tilde{x} \in var(\tilde{t})$.

Then, if t is not a variable, we can apply the *orient* rule to the system $P; S = \{t \overset{?}{=} x\} \cup Q; S$ and we obtain the system $P'; S' = \{x \overset{?}{=} t\} \cup Q; S$. An application of the *variable elimination* rule yields the system $P''; S'' = Q\{x \mapsto t\}; S\{x \mapsto t\} \cup \{x \overset{?}{=} t\}$. Similar arguments to the ones used in the proof of Lemma 3.4.28 show that the properties (i) to (viii) hold for the substitution λ.

It remains to consider the case when t is a variable $y \in X$. It cannot be that $x = y$ as otherwise $x \in var(y) = var(t)$. So, $x \neq y$ and we can apply the *variable elimination* rule to the system $P; S = \{y \overset{?}{=} x\} \cup Q; S$ and we obtain the system $P''; S'' = Q\{y \mapsto x\}; S\{y \mapsto x\} \cup \{y \overset{?}{=} x\}$. We now define a substitution φ as follows. For $z \in X$, let

$$\varphi(z) = \begin{cases} \tilde{t} & \text{if } z = x \\ \lambda(z) & \text{otherwise} \end{cases}$$

φ is well-defined and as $x \notin dom(\lambda)$, we get $dom(\lambda) \subseteq dom(\varphi)$ and hence, property (i) holds. Moreover, property (ii) is satisfied as well as $dom(\varphi) \setminus dom(\lambda) = \{x\}$ and $x = \tilde{x} \in dom(\sigma_{\tilde{S}'})$. Then, for property (iii) we simply observe again that $x \notin dom(\lambda)$. Furthermore, as $\tilde{t} = \lambda(y) \in codom(\lambda)$ and $codom(\lambda) \subseteq \mathbf{C}$, it obviously follows that $codom(\varphi) \subseteq \mathbf{C}$, i.e. property (iv) is satisfied. Additionally, as $\lambda(x) = x$, we obtain $\varphi = \lambda\{x \mapsto \tilde{t}\}$, and it holds that:

$$\lambda\{\tilde{x} \mapsto \tilde{t}\} = \lambda\{x \mapsto \lambda(y)\} = \{y \mapsto x\}(\lambda\{x \mapsto \lambda(y)\}) = \{y \mapsto x\}\varphi$$

Thus, we get:

$$\begin{aligned} P''\varphi = (Q\{y \mapsto x\})\varphi = Q(\{y \mapsto x\}\varphi) = Q(\lambda\{\tilde{x} \mapsto \tilde{t}\}) &= (Q\lambda)\{\tilde{x} \mapsto \tilde{t}\} \\ &= \tilde{Q}\{\tilde{x} \mapsto \tilde{t}\} = \tilde{P}'' \end{aligned}$$

Hence, property (v) holds. Similar arguments to the ones used in the proof of Lemma 3.4.28 show that $\sigma_{S''} = \sigma_S\{y \mapsto x\}$ and $\sigma_{\tilde{S}''} = \sigma_{\tilde{S}}\{\tilde{x} \mapsto \tilde{t}\}$. For property (vi), we observe that $x \notin dom(\sigma_{S''}) \supseteq dom(\sigma_S)$ as otherwise the equation $y \overset{?}{=} x$ would not be an element of P, which implies that:

$$x\sigma_{S''}\varphi = x\varphi = \tilde{t} = \tilde{x}\sigma_{\tilde{S}''} = x\sigma_{\tilde{S}''}$$

Now, let $z \in dom(\sigma_{\tilde{S}''})$ such that $z \neq x$, from which it follows that $z \in dom(\sigma_{\tilde{S}})$. Hence, by using the assumptions we obtain $z\sigma_S\lambda = z\sigma_{\tilde{S}}$, which implies that:

$$\begin{aligned} z\sigma_{S''}\varphi = z(\sigma_S\{y \mapsto x\})\varphi = z\sigma_S(\{y \mapsto x\}\varphi) &= z\sigma_S(\lambda\{\tilde{x} \mapsto \tilde{t}\}) \\ &= (z\sigma_S\lambda)\{\tilde{x} \mapsto \tilde{t}\} \\ &= (z\sigma_{\tilde{S}})\{\tilde{x} \mapsto \tilde{t}\} \\ &= z(\sigma_{\tilde{S}}\{\tilde{x} \mapsto \tilde{t}\}) = z\sigma_{\tilde{S}''} \end{aligned}$$

Then, for property (vii) it first of all holds that $y\sigma_{S''}\varphi = x\varphi = \tilde{t} = y\lambda = y\varphi$. Now, let $z \in dom(\sigma_{S''})$ such that $z \notin dom(\sigma_{\tilde{S}''})$ and $z \neq y$, which implies that $z \in dom(\sigma_S)$ and $z \notin dom(\sigma_{\tilde{S}})$ as $dom(\sigma_{\tilde{S}}) \subseteq dom(\sigma_{\tilde{S}''})$. Hence, it follows from the assumptions that $z\sigma_S\lambda = z\lambda$ and therefore:

$$\begin{aligned} z\sigma_{S''}\varphi = z(\sigma_S\{y \mapsto x\})\varphi = z\sigma_S(\{y \mapsto x\}\varphi) &= z\sigma_S(\lambda\{\tilde{x} \mapsto \tilde{t}\}) \\ &= (z\sigma_S\lambda)\{\tilde{x} \mapsto \tilde{t}\} \\ &= (z\lambda)\{\tilde{x} \mapsto \tilde{t}\} \\ &= z(\lambda\{\tilde{x} \mapsto \tilde{t}\}) = z\varphi \end{aligned}$$

Finally, we note that $y \in dom(\lambda)$ as $\lambda(y) = \tilde{t} \notin X$ and thus, property (viii) is satisfied. □

The next lemma regroups the results established by the two previous propositions.

Lemma 3.4.30. *Let $\mathbf{C} \subseteq CS$ be a set of constants. Additionally, let $\tilde{\mathcal{D}} = \tilde{P}; \emptyset \Rightarrow^+ \tilde{P}'; \tilde{S}'$ be a liftable derivation of the algorithm $\mathcal{U}$ such that there is a substitution $\eta: X \longrightarrow T_\Sigma(X)$ and a multiset of equations P with* *(a)* $codom(\eta) \subseteq \mathbf{C}$, *and*

(b) $P\eta = \tilde{P}$.

Then there exists a multiset of equations P', *a set of equations* S', *a derivation* $P; \emptyset \Rightarrow^* P'; S'$ *produced by the algorithm* $\mathcal{U}$ *and a substitution* $\varphi\colon X \longrightarrow T_\Sigma(X)$ *such that*

(i) $dom(\eta) \subseteq dom(\varphi)$,

(ii) $codom(\varphi) \subseteq \mathbf{C}$,

(iii) $\forall x \in dom(\eta)\colon \varphi(x) = \eta(x)$,

(iv) $dom(\varphi) \setminus dom(\eta) \subseteq dom(\sigma_{\tilde{S}'})$,

(v) $P'\varphi = \tilde{P}'$,

(vi) $\forall x \in dom(\sigma_{\tilde{S}'})\colon x\sigma_{S'}\varphi = x\sigma_{\tilde{S}'}$, *and*

(vii) $\forall x \in dom(\sigma_{S'})\colon x \notin dom(\sigma_{\tilde{S}'}) \Rightarrow x\sigma_{S'}\varphi = x\eta \wedge x \in dom(\eta)$.

Proof. By induction on the length $|\tilde{\mathcal{D}}|$ of the derivation $\tilde{\mathcal{D}}$. If $|\tilde{\mathcal{D}}| = 1$ and the *orient* rule is not applied, or $|\tilde{\mathcal{D}}| = 2$ and the *orient* rule is applied as first inference step, then the properties (i) to (vii) follow immediately from Lemmata 3.4.28 and 3.4.29.

For the case where $|\tilde{\mathcal{D}}| > 1$ and which is not covered above, we can split the derivation $\tilde{\mathcal{D}}$ into a derivation $\tilde{\mathcal{E}} = \tilde{P}; \tilde{S} \Rightarrow^* \tilde{Q}; \tilde{R}$ and into a double rule application $\tilde{Q}; \tilde{R} \Rightarrow \tilde{P}_1; \tilde{S}_1 \Rightarrow \tilde{P}'; \tilde{S}'$ if the system has been obtained through the *orient* rule or into a single rule application $\tilde{Q}; \tilde{R} \Rightarrow \tilde{P}'; \tilde{S}'$, otherwise. By applying the induction hypothesis, we can conclude that there is a derivation $\mathcal{E} = P; S \Rightarrow^* Q; R$ and a substitution λ such that the properties (i) to (vii) hold for the derivation $\mathcal{E}$ and the substitution λ. It now follows from Lemmata 3.4.28 and 3.4.29 that there exists a derivation $Q; R \Rightarrow^+ P'; S'$ and a substitution φ such that

- $dom(\lambda) \subseteq dom(\varphi)$,
- $codom(\varphi) \subseteq \mathbf{C}$,
- $\forall x \in dom(\lambda)\colon \varphi(x) = \lambda(x)$,
- $dom(\varphi) \setminus dom(\lambda) \subseteq dom(\sigma_{\tilde{S}'})$,
- $P'\varphi = \tilde{P}'$,
- $\forall x \in dom(\sigma_{\tilde{S}'})\colon x\sigma_{S'}\varphi = x\sigma_{\tilde{S}'}$,
- $\forall x \in dom(\sigma_{S'})\colon x \notin dom(\sigma_{\tilde{S}'}) \Rightarrow x\sigma_{S'}\varphi = x\varphi$, and
- $\forall x \in dom(\sigma_{S'}) \setminus dom(\sigma_R)\colon x \notin dom(\sigma_{\tilde{S}'}) \Rightarrow x \in dom(\lambda)$.

Thus, as $dom(\eta) \subseteq dom(\lambda)$, we obtain $dom(\eta) \subseteq dom(\varphi)$, i.e. property (i) is satisfied. Moreover, the properties (ii), (v), and (vi) hold as well.

For statement (iii), let $x \in dom(\eta)$, which implies that $x \in dom(\lambda)$ and $\eta(x) = \lambda(x)$ (through the induction hypothesis). Finally, as hence $\lambda(x) = \varphi(x)$, it holds that $\eta(x) = \varphi(x)$.

In order to prove property (iv), we first of all note that $dom(\sigma_{\tilde{R}}) \subseteq dom(\sigma_{\tilde{S}'})$. Let now $x \in dom(\varphi) \setminus dom(\eta)$. If $x \in dom(\lambda)$, then it follows (from the induction hypothesis) that $x \in dom(\sigma_{\tilde{R}})$, from which we conclude that $x \in dom(\sigma_{\tilde{S}'})$. Otherwise, $x \notin dom(\lambda)$ and we immediately obtain $x \in dom(\sigma_{\tilde{S}'})$.

For the remaining property (vii), let $x \in dom(\sigma_{S'})$ such that $x \notin dom(\sigma_{\tilde{S}'})$. It follows first of all that $x\sigma_{S'}\varphi = x\varphi$. Moreover, as $dom(\sigma_{\tilde{R}}) \subseteq dom(\sigma_{\tilde{S}'})$, it holds that $x \notin dom(\sigma_{\tilde{R}})$. We need to distinguish between the following two cases now. If $x \in dom(\sigma_R)$, then we can conclude from the induction hypothesis that $x \in dom(\eta)$. Hence, by property (iii) we have $\eta(x) = \varphi(x)$ and thus, $x\sigma_{S'}\varphi = x\eta$. Finally, for the case where $x \notin dom(\sigma_R)$, we immediately obtain $x \in dom(\lambda) \subseteq dom(\varphi)$. Furthermore, by property (iv) $x \in dom(\eta)$ must hold as otherwise $x \in dom(\sigma_{\tilde{S}'})$. Hence, by property (iii) we have $\eta(x) = \varphi(x)$ again, which implies in conclusion that $x\sigma_{S'}\varphi = x\eta$. □

We still need to prove one property related to the derivation and the substitution obtained through the previous lemma.

Lemma 3.4.31. *Let* $\mathbf{C} \subseteq CS$ *be a set of constants. Additionally, let* $\tilde{\mathcal{D}} = \tilde{P}; \tilde{S} \Rightarrow^* \tilde{P}'; \tilde{S}'$ *be a liftable derivation of the algorithm* $\mathcal{U}$ *such that there is a substitution* $\eta\colon X \to T_\Sigma(X)$ *with* $P\eta = \tilde{P}$, $\sigma_S\eta = \sigma_{\tilde{S}}$ *and* $codom(\eta) \subseteq \mathbf{C}$ *for a multiset of equations* P *and a set of equations in solved form* S. *Additionally, let* $\mathcal{D} = P; S \Rightarrow^* P'; S'$ *be the derivation and* $\varphi\colon X \to T_\Sigma(X)$ *be the substitution obtained through Lemma 3.4.30. Then it holds that:*

$$\forall x \in dom(\sigma_{\tilde{S}'})\colon x \in dom(\sigma_{S'}) \lor x \in var(codom(\sigma_{S'}))$$

Proof. Let $x \in dom(\sigma_{\tilde{S}'})$ for which we assume that $x \notin dom(\sigma_{S'})$. As $x \in dom(\sigma_{\tilde{S}'})$, it follows from the properties of algorithm $\mathcal{U}$ that there is exactly one application of the *variable elimination* rule contained in the derivation $\tilde{\mathcal{D}}$ which has added the variable x to the domain of the substitution $\sigma_{\tilde{S}'}$. By considering how the derivation $\mathcal{D}$ has been obtained, one can see that there is a corresponding application of the *variable elimination* rule in the derivation $\mathcal{D}$, which adds a variable y to the domain of an intermediate substitution σ such that $x \in var(codom(\sigma))$ (as $x \notin dom(\sigma_{S'})$, none of the other possible cases for obtaining a corresponding application of the *variable elimination* rule can apply). Finally, as $x \notin dom(\sigma_{S'})$, the *variable elimination* rule is not applied on the variable x in the derivation $\mathcal{D}$ and hence, the variable x is not replaced by another term in the derivation $\mathcal{D}$. Thus, $x \in var(codom(\sigma_{S'}))$. □

Now we can state the result that links most general unifiers to substitutions η with $codom(\eta) \subseteq \mathbf{C}$.

Lemma 3.4.32. *Let* $\mathbf{C} \subseteq CS$ *be a set of constants. Additionally, let* $A, \tilde{A}, B, \tilde{B}$ *be atoms such that the substitution* $\sigma\colon X \longrightarrow T_\Sigma(X)$ *is a most general unifier of* $\tilde{A}$ *and* $\tilde{B}$ *and such that there is a substitution* $\eta\colon X \longrightarrow T_\Sigma(X)$ *with* $A\eta = \tilde{A}$, $B\eta = \tilde{B}$, *and* $codom(\eta) \subseteq \mathbf{C}$.

Then there exists a most general unifier $\tau\colon X \longrightarrow T_\Sigma(X)$ *of* A *and* B, *and a substitution* $\varphi\colon X \longrightarrow T_\Sigma(X)$ *with* $\tau\varphi = \eta\sigma$ *and* $codom(\varphi) \subseteq \mathbf{C}$.

Proof. Let $A = p(s_1, \ldots, s_n)$ and $B = p(t_1, \ldots, t_n)$ for a predicate symbol p of arity n and terms $s_1, \ldots, s_n, t_1, \ldots, t_n$. Additionally, let $P = \{s_1 \overset{?}{=} t_1, \ldots, s_n \overset{?}{=} t_n\}$ and $\tilde{P} = P\eta$. As the atoms $\tilde{A}$ and $\tilde{B}$ are unifiable, it follows from Theorem 3.4.23 that any maximal derivation by the algorithm $\mathcal{U}$ which starts from the system $\tilde{P}; \emptyset$ ends in a system $\emptyset; \tilde{S}$ such that the substitution $\sigma_{\tilde{S}}$ is an idempotent most general unifier of $\tilde{A}$ and $\tilde{B}$, and $dom(\sigma_{\tilde{S}'}) \subseteq var(\tilde{A}) \cup var(\tilde{B})$. Moreover, it is easy to see that a liftable derivation of $\mathcal{U}$ which starts from $\tilde{P}; \emptyset$ and ends in $\emptyset; \tilde{S}'$ exists. Let $\tilde{\tau} = \sigma_{\tilde{S}'}$ be the resulting most general unifier of $\tilde{A}$ and $\tilde{B}$. Then, as $\emptyset\eta = \emptyset$, it follows from Lemma 3.4.30 that there exists a derivation produced by the algorithm $\mathcal{U}$ which starts in the system $P; \emptyset$ and ends in the system $P'; S'$, together with a substitution φ such that

(i) $dom(\eta) \subseteq dom(\varphi)$,

(ii) $codom(\varphi) \subseteq \mathbf{C}$,

(iii) $\forall x \in dom(\eta)\colon \varphi(x) = \eta(x)$,

(iv) $dom(\varphi) \setminus dom(\eta) \subseteq dom(\sigma_{\tilde{S}'})$,

(v) $P'\varphi = \emptyset$,

(vi) $\forall x \in dom(\sigma_{\tilde{S}'})\colon x\sigma_{S'}\varphi = x\sigma_{\tilde{S}'}$, and

(vii) $\forall x \in dom(\sigma_{S'})\colon x \notin dom(\sigma_{\tilde{S}'}) \Rightarrow x\sigma_{S'}\varphi = x\eta \wedge x \in dom(\eta)$.

Obviously, $P' = \emptyset$ and the derivation is maximal, which implies that $\tau := \sigma_{S'}$ is an idempotent most general unifier of A and B by Theorem 3.4.23. We now define a new substitution ψ as follows. For $z \in X$, let

$$\psi(z) = \begin{cases} \eta(z) & \text{if } z \notin var(codom(\tau)) \\ \varphi(z) & \text{otherwise} \end{cases}$$

It is clear that $codom(\psi) \subseteq \mathbf{C}$. We still need to show that $\tau\psi = \eta\tilde{\tau}$. For that purpose, let $x \in X$ be a variable.

- Assume $x \in dom(\tilde{\tau})$. It follows then from property (vi) that $x\tau\varphi = x\tilde{\tau}$. Additionally, we have $x \notin dom(\eta)$ as otherwise $x \notin dom(\tilde{\tau}) \subseteq var(\tilde{P})$ would hold. Consequently, $\eta(x) = x$ and $x\tau\varphi = x\eta\tilde{\tau}$.
    - If $x \in dom(\tau)$, then $x\tau\psi = x\tau\varphi$ holds and therefore, $x\tau\psi = x\eta\tilde{\tau}$.

- If $x \notin dom(\tau)$, then we obtain $x \in var(codom(\tau))$ by Lemma 3.4.31 and thus, $x\tau\psi = x\psi = x\varphi = x\tau\varphi = x\eta\tilde{\tau}$.

- Assume $x \notin dom(\tilde{\tau})$.
  - If $x \in dom(\tau)$, then it follows first of all that $x\tau\psi = x\tau\varphi$ holds. Furthermore, by property (vii) we have $x\tau\varphi = x\eta$, which implies that $x\tau\psi = x\eta$.
    - If $x \notin dom(\eta)$, then $\eta(x) = x$ and thus, $x\tau\psi = x\eta = x = x\tilde{\tau} = x\eta\tilde{\tau}$.
    - If $x \in dom(\eta)$, then as $codom(\eta) \subseteq \mathbf{C}$, we obtain $x\eta = x\eta\tilde{\tau}$ and hence, $x\tau\psi = x\eta\tilde{\tau}$.
  - For $x \notin dom(\tau)$, we need to distinguish between the following cases:
    - If $x \in var(codom(\tau))$
      - If $x \notin dom(\eta)$, then it follows from property (iv) that $x \notin dom(\varphi)$, which implies that $x\varphi = x = x\eta$. Thus, we obtain:

$$x\tau\psi = x\psi = x\varphi = x\eta = x = x\tilde{\tau} = x\eta\tilde{\tau}$$

      - If $x \in dom(\eta)$, then by property (iii) $x\varphi = x\eta$ holds. Additionally, as $codom(\eta) \subseteq \mathbf{C}$, it follows that $x\eta = x\eta\tilde{\tau}$, and therefore:

$$x\tau\psi = x\psi = x\varphi = x\eta = x\eta\tilde{\tau}$$

    - If $x \notin var(codom(\tau))$, then it first of all holds that $x\tau\psi = x\psi = x\eta$.
      - If $x \notin dom(\eta)$, then $\eta(x) = x$ and thus, $x\tau\psi = x\eta = x = x\tilde{\tau} = x\eta\tilde{\tau}$.
      - If $x \in dom(\eta)$, then as $codom(\eta) \subseteq \mathbf{C}$, we obtain $x\eta = x\eta\tilde{\tau}$ and hence, $x\tau\psi = x\eta\tilde{\tau}$.

Finally, as $\tilde{\tau}$ and σ are most general unifiers of the atoms $\tilde{A}$ and $\tilde{B}$, it follows from Corollary 3.4.25 that there exists a variable renaming ζ with $\tilde{\tau}\zeta = \sigma$ and $codom(\zeta) \subseteq X$. We now define a substitution ν as follows. For $z \in X$, let

$$\nu(z) = \begin{cases} \psi(v) & \text{if } \exists v \in X : \zeta(v) = z \wedge v \in dom(\psi) \\ z & \text{otherwise} \end{cases}$$

The substitution ν is well defined as the substitution ζ is injective by Lemma 3.4.16, and we have $codom(\nu) \subseteq \mathbf{C}$. We now show that $\zeta\nu = \psi\zeta$. For that purpose, let $z \in X$. We need to distinguish between then following two cases then. If $z \in dom(\psi)$, then it holds that $z\zeta\nu = z\psi = z\psi\zeta$ as $codom(\psi) \subseteq \mathbf{C}$ (and as $codom(\zeta) \subseteq X$). For the remaining case in which $z \notin dom(\psi)$, we obtain $z\zeta\nu = z\zeta = z\psi\zeta$. Hence, it holds that:

$$(\tau\zeta)\nu = \tau(\zeta\nu) = \tau(\psi\zeta) = (\tau\psi)\zeta = (\eta\tilde{\tau})\zeta = \eta(\tilde{\tau}\zeta) = \eta\sigma$$

Finally, by using Lemma 3.4.21, we obtain that $\tau\zeta$ is a most general unifier of the atoms A and B. □

Before we can prove the lifting lemmata for the different inference rules, we still have to show one additional proposition.

Lemma 3.4.33. *Let $\mathbf{C} \subseteq CS$ be a set of constants. Additionally, let A, B, $\tilde{A}$, $\tilde{B}$ be atoms such there is a substitution $\eta\colon X \longrightarrow T_\Sigma(X)$ with $A\eta = \tilde{A}$, $B\eta = \tilde{B}$, $codom(\eta) \subseteq \mathbf{C}$ and such that $(const(A) \cap \mathbf{C}) \cup (const(B) \cap \mathbf{C}) = \emptyset$. Moreover, let $\sigma\colon X \longrightarrow T_\Sigma(X)$ be a most general unifier of $\tilde{A}$ and $\tilde{B}$ and let $Y \subseteq X$ be an arbitrary variable set with $\forall x \in var(Y\eta)\colon \sigma(x) \in X$. Finally, let $\tau\colon X \longrightarrow T_\Sigma(X)$ be a most general unifier of A and B and let φ be a substitution with $\tau\varphi = \eta\sigma$.*

Then it holds for all $x \in Y$ that $\tau(x) \in X$.

Proof. Let $x \in Y$. First of all, if follows from Lemma 3.4.24 that $const(codom(\tau)) \cap \mathbf{C} = \emptyset$. We now need to distinguish between the following two cases.

If $x \in dom(\eta)$, then it holds that $x\tau\varphi = x\eta\sigma = x\eta \in \mathbf{C}$. Thus, it must hold that $\tau(x)$ is either a variable or a constant. But if we assume that $\tau(x)$ is a constant, then it follows that $x\tau\varphi = x\tau \notin \mathbf{C}$ as $const(codom(\tau)) \cap \mathbf{C} = \emptyset$, which is obviously a contradiction.

Finally, for the case that $x \notin dom(\eta)$, we obtain $x \in var(Y\eta)$, and hence, $x\tau\varphi = x\eta\sigma = x\sigma \in X$. It is now easy to see that $\tau(x) \in X$ must hold. □

We can now prove the first lifting lemma for the step resolution rule between two step clauses.

Lemma 3.4.34 (Lifting Lemma 1). *Let $\mathbf{C} \subseteq CS$ be a set of constants. Additionally, let $\mathcal{C}$, $\mathcal{D}$, $\tilde{\mathcal{C}}$, $\tilde{\mathcal{D}}$ be step clauses such that there are substitutions $\lambda_1, \lambda_2\colon X \longrightarrow T_\Sigma(X)$ with $\mathcal{C}\lambda_1 = \tilde{\mathcal{C}}$, $\mathcal{D}\lambda_2 = \tilde{\mathcal{D}}$, $codom(\lambda_1) \cup codom(\lambda_2) \subseteq \mathbf{C}$, $(const(\mathcal{C}) \cap \mathbf{C}) \cup (const(\mathcal{D}) \cap \mathbf{C}) = \emptyset$ and such that the step clause $\tilde{\mathcal{E}}$ is a binary resolvent by fine-grained step-resolution from $\tilde{\mathcal{C}}$ and $\tilde{\mathcal{D}}$.*

Then there exists a binary resolvent $\mathcal{E}$ by fine-grained step resolution from $\mathcal{C}$ and $\mathcal{D}$ such that $const(\mathcal{E}) \cap \mathbf{C} = \emptyset$ and such that there exists a substitution $\varphi\colon X \longrightarrow T_\Sigma(X)$ with $\mathcal{E}\varphi = \tilde{\mathcal{E}}$ and $codom(\varphi) \subseteq \mathbf{C}$.

Proof. Let $\tilde{\mathcal{C}}$ be $\tilde{C}_1 \Rightarrow \bigcirc(\tilde{C}_2 \vee \tilde{A})$, $\tilde{\mathcal{D}}$ be $\tilde{D}_1 \Rightarrow \bigcirc(\tilde{D}_2 \vee \neg\tilde{B})$. Additionally, let $\tilde{\mu}$ and $\tilde{\nu}$ be the variable renamings used such that $\tilde{\mathcal{C}}\tilde{\mu}$ and $\tilde{\mathcal{D}}\tilde{\nu}$ are variable disjoint. Then, the resolvent $\tilde{\mathcal{E}}$ is of the following form:

$$(\tilde{C}_1\tilde{\mu} \wedge \tilde{D}_1\tilde{\nu})\sigma \Rightarrow \bigcirc(\tilde{C}_2\tilde{\mu} \vee \tilde{D}_2\tilde{\nu})\sigma$$

where σ is a most general unifier of the literals $\tilde{A}\tilde{\mu}$ and $\tilde{B}\tilde{\nu}$ such that σ does not map variables from $\tilde{C}_1\tilde{\mu}$ or $\tilde{D}_1\tilde{\nu}$ into a constant or a functional term, $\tilde{A}\tilde{\mu}$ is eligible in $\tilde{C}_2\tilde{\mu} \vee \tilde{A}\tilde{\mu}$ for σ and $\neg\tilde{B}\tilde{\nu}$ is eligible in $\tilde{D}_2\tilde{\nu} \vee \neg\tilde{B}\tilde{\nu}$ for σ. Additionally, let S denote the instance compatible selection function that has been used in the inference.

Now, let $\mathcal{C}$ be $C_1 \Rightarrow \bigcirc(C_2 \vee A)$ and $\mathcal{D}$ be $D_1 \Rightarrow \bigcirc(D_2 \vee \neg B)$ such that $C_1\lambda_1 = \tilde{C}_1$, $C_2\lambda_1 = \tilde{C}_2$, $A\lambda_1 = \tilde{A}$ and $D_1\lambda_2 = \tilde{D}_1$, $D_2\lambda_2 = \tilde{D}_2$, $B\lambda_2 = \tilde{B}$. Moreover, due to the instance compatibility of the selection function we can assume that the literal B is selected in the clause $D_2 \vee B$ if the literal $B\lambda_2\tilde{\nu}$ is selected in the clause $(D_2 \vee B)\lambda_2\tilde{\nu}$. As $codom(\lambda_1) \subseteq \mathbf{C}$ and $codom(\lambda_2) \subseteq \mathbf{C}$, there are substitutions μ and ν respectively such that the clauses $\mathcal{C}\mu$

and $\mathcal{D}\nu$ are variable disjoint. Hence, it is easy to see that there exists a substitution η with $codom(\eta) \subseteq \mathbf{C}$, $\mathcal{C}\mu\eta = \tilde{\mathcal{C}}\tilde{\mu}$ and $\mathcal{D}\nu\eta = \tilde{\mathcal{D}}\tilde{\nu}$, which implies that the substitution $\eta\sigma$ is a unifier of the two literals $A\mu$ and $B\nu$. Obviously, it holds that $const(\mathcal{C}\mu) = const(\mathcal{C})$ and $const(\mathcal{D}\nu) = const(\mathcal{D})$; thus, by Lemma 3.4.32 there exists a most general unifier τ of $A\mu$ and $B\nu$, and a substitution φ with $\tau\varphi = \eta\sigma$ and $codom(\varphi) \subseteq \mathbf{C}$. Additionally, by applying Lemma 3.4.33 on the set $var(C_1\nu) \cup var(D_1\mu)$, we can see that τ does not map variables of $C_1\nu$ or $D_1\mu$ into a constant or functional term.

Furthermore, if the literal $\neg\tilde{B}\tilde{\nu} = \neg B\lambda_2\tilde{\nu}$ has been selected in the clause $\tilde{D}_2\tilde{\nu} \vee \neg\tilde{B}\tilde{\nu}$, then it follows from the instance compatibility of the selection function that there exists a literal B such that the literal $\neg B\nu$ is selected in the clause $D_2\nu \vee \neg B\nu$. Otherwise, no literal is selected in $\tilde{D}_2\tilde{\nu}$ and the literal $\neg\tilde{B}\tilde{\nu}\sigma$ is maximal w.r.t. $\tilde{D}_2\tilde{\nu}\sigma$. Consequently, as $(D_2\nu \vee \neg B\nu)\eta = \tilde{D}_2\tilde{\nu} \vee \neg\tilde{B}\tilde{\nu}$, we obtain again from the instance compatibility of the selection function that no literal is selected in $D_2\nu$, and as $\neg\tilde{B}\tilde{\nu}\sigma = \neg B\nu\eta\sigma = \neg(B\nu\tau)\varphi$ and $\tilde{D}_2\tilde{\nu}\sigma = D_2\nu\eta\sigma = (D_2\nu\tau)\varphi$, it follows from properties of the ordering $\succ$ on literals that $\neg B\nu\tau$ is maximal w.r.t. $D_2\nu\tau$. Hence, $\neg B\nu$ is eligible in $D_2\nu$ for τ. Similarly, one can show that $A\mu$ is eligible in $C_2\mu$ for τ, i.e. there exists a resolvent $\mathcal{E}$ by fine-grained step resolution from $\mathcal{C}\mu$ and $\mathcal{D}\nu$, which is of the following form:

$$(C_1\mu \wedge D_1\nu)\tau \Rightarrow \bigcirc(C_2\mu \vee D_2\nu)\tau$$

Additionally, it holds that:

$$\begin{aligned}\mathcal{E}\varphi &= (C_1\mu \wedge D_1\nu)\tau\varphi \Rightarrow \bigcirc(C_2\mu \vee D_2\nu)\tau\varphi\\ &= (C_1\mu \wedge D_1\nu)\eta\sigma \Rightarrow \bigcirc(C_2\mu \vee D_2\nu)\eta\sigma\\ &= (\tilde{C}_1\tilde{\mu} \wedge \tilde{D}_1\tilde{\nu})\sigma \Rightarrow \bigcirc(\tilde{C}_2\tilde{\mu} \vee \tilde{D}_2\tilde{\nu})\sigma\\ &= \tilde{\mathcal{E}}\end{aligned}$$

Finally, Lemma 3.4.24 implies that $const(\mathcal{E}) \cap \mathbf{C} = \emptyset$. □

We now state the lifting lemma for the resolution inference rule between step and universal clauses.

Lemma 3.4.35 (Lifting Lemma 2). *Let $\mathbf{C} \subseteq CS$ be a set of constants. Additionally, let $\mathcal{C}$, $\tilde{\mathcal{C}}$ be step clauses and $\mathcal{D}$, $\tilde{\mathcal{D}}$ be universal clauses such that there are substitutions $\lambda_1, \lambda_2 \colon X \to T_\Sigma(X)$ with $\mathcal{C}\lambda_1 = \tilde{\mathcal{C}}$, $\mathcal{D}\lambda_2 = \tilde{\mathcal{D}}$, $codom(\lambda_1) \cup codom(\lambda_2) \subseteq \mathbf{C}$, $(const(\mathcal{C}) \cap \mathbf{C}) \cup (const(\mathcal{D}) \cap \mathbf{C}) = \emptyset$, and such that the step clause $\tilde{\mathcal{E}}$ is a binary resolvent by fine-grained step-resolution from $\tilde{\mathcal{C}}$ and $\tilde{\mathcal{D}}$.*

Then there exists a binary resolvent $\mathcal{E}$ by fine-grained step resolution from $\mathcal{C}$ and $\mathcal{D}$ such that $const(\mathcal{E}) \cap \mathbf{C} = \emptyset$ and such that there exists a substitution $\varphi \colon X \to T_\Sigma(X)$ with $\mathcal{E}\varphi = \tilde{\mathcal{E}}$ and $codom(\varphi) \subseteq \mathbf{C}$.

Proof. Analogously to the proof of the Lemma 3.4.34. □

Now we prove the lifting lemma for the factoring in right-hand sides of step clauses inference rule.

Lemma 3.4.36 (Lifting Lemma 3). *Let* $\mathbf{C} \subseteq CS$ *be a set of constants. Additionally, let* C, $\tilde{C}$ *be step clauses such that there is a substitution* λ: $X \longrightarrow T_\Sigma(X)$ *with* $C\lambda = \tilde{C}$, $codom(\lambda) \subseteq \mathbf{C}$, $const(C) \cap \mathbf{C} = \emptyset$, *and such that the clause* $\tilde{\mathcal{E}}$ *is the conclusion of the* ordered fine-grained positive step factoring with selection *rule applied to* $\tilde{C}$.

Then there exists a step clause $\mathcal{E}$ *which is the conclusion of the* ordered fine-grained positive step factoring with selection *rule applied to* C *such that* $const(\mathcal{E}) \cap \mathbf{C} = \emptyset$ *and such that there exists a substitution* φ: $X \longrightarrow T_\Sigma(X)$ *with* $\mathcal{E}\varphi = \tilde{\mathcal{E}}$ *and* $codom(\varphi) \subseteq \mathbf{C}$.

Proof. Let $\tilde{C}$ be $\tilde{C}_1 \Rightarrow \bigcirc(\tilde{C}_2 \vee \tilde{A} \vee \tilde{B})$. Then, the clause $\tilde{\mathcal{E}}$ is of they following form:

$$\tilde{C}_1\sigma \Rightarrow \bigcirc(\tilde{C}_2\sigma \vee \tilde{A}\sigma)$$

where σ is a most general unifier of the atoms $\tilde{A}$ and $\tilde{B}$ such that σ does not map variables from $\tilde{C}_1$ into a constant or a functional term and such that the literal $\tilde{A}$ is eligible in $\tilde{C}_2 \vee \tilde{A} \vee \tilde{B}$ for σ. Additionally, let S denote the instance compatible selection function that has been used in the inference.

Now, let C be $C_1 \Rightarrow \bigcirc(C_2 \vee A \vee B)$ such that $C_1\lambda = \tilde{C}_1$, $C_2\lambda = \tilde{C}_2$, $A\lambda = \tilde{A}$ and $B\lambda = \tilde{B}$. Then, by Lemma 3.4.32 there exists a most general unifier τ of A and B, and a substitution φ with $\tau\varphi = \lambda\sigma$ and $codom(\varphi) \subseteq \mathbf{C}$. Additionally, it follows from Lemma 3.4.33 that τ does not map variables from C_1 into a constant or functional term.

Since the literal $\tilde{A}$ is eligible in $\tilde{C}_2 \vee \tilde{A} \vee \tilde{B}$ for σ and $\tilde{A}$ is positive, it follows that no literal is selected in $\tilde{C} = \tilde{C}_2 \vee \tilde{A} \vee \tilde{B}$ and $\tilde{A}\sigma$ is maximal w.r.t. $\tilde{D}_1\sigma$. Thus, as $C\lambda = \tilde{C}$, we obtain from the instance compatibility of the selection function that no literal is selected in the clause $C_2 \vee A \vee B$. Additionally, as $(A\tau)\varphi = A\lambda\sigma = \tilde{A}\sigma$ and $(C_2\tau)\varphi = C_2\lambda\sigma = \tilde{C}_2\sigma$, we can infer from the properties of the literal ordering $\succ$ that the literal $A\tau$ is maximal w.r.t. $C_2\tau$. Thus, the literal A is eligible in the clause $C_2 \vee A \vee B$ for τ, i.e. we can apply the *ordered fine-grained positive step factoring with selection* rule on the step clause C together with the substitution τ. The result of its application is the following step clause $\mathcal{E}$:

$$C_1\tau \Rightarrow \bigcirc(C_2 \vee A)\tau$$

Moreover, it holds that:

$$\begin{aligned}\mathcal{E}\varphi &= C_1\tau\varphi \Rightarrow \bigcirc(C_2 \vee A)\tau\varphi \\ &= C_1\lambda\sigma \Rightarrow \bigcirc(C_2 \vee A)\lambda\sigma \\ &= \tilde{C}_1\sigma \Rightarrow \bigcirc(\tilde{C}_2 \vee \tilde{A})\sigma \\ &= \tilde{\mathcal{E}}\end{aligned}$$

Finally, Lemma 3.4.24 implies that $const(\mathcal{E}) \cap \mathbf{C} = \emptyset$. □

Next, we prove the lifting lemma for the clause conversion rule.

Lemma 3.4.37 (Lifting Lemma 4). *Let $\mathbf{C} \subseteq CS$ be a set of constants. Additionally, let $\mathcal{C}$, $\tilde{\mathcal{C}}$ be step clauses such that there is a substitution $\lambda\colon X \to T_\Sigma(X)$ with $\mathcal{C}\lambda = \tilde{\mathcal{C}}$, $codom(\lambda) \subseteq \mathbf{C}$, $const(\mathcal{C}) \cap \mathbf{C} = \emptyset$, and such that the universal clause $\tilde{\mathcal{E}}$ is the conclusion of the* clause conversion *rule applied to $\tilde{\mathcal{C}}$.*

Then there exists a universal clause $\mathcal{E}$ which is the conclusion of the clause conversion *rule applied to $\mathcal{C}$ such that $const(\mathcal{E}) \cap \mathbf{C} = \emptyset$ and $\mathcal{E}\lambda = \tilde{\mathcal{E}}$.*

Proof. Let $\tilde{\mathcal{C}}$ be $\tilde{C}_1 \Rightarrow \bigcirc\bot$ and $\mathcal{C}$ be $C_1 \Rightarrow \bigcirc\bot$. Then, we obtain $\tilde{\mathcal{E}} = \neg\tilde{C}_1$. By applying the *clause conversion* rule on the clause $\mathcal{C}$, we get a clause $\mathcal{E} = \neg C_1$. Clearly, $const(\mathcal{E}) \cap \mathbf{C} = \emptyset$, and as $\mathcal{C}\lambda = \tilde{\mathcal{C}}$, it holds that $\mathcal{E}\lambda = \tilde{\mathcal{E}}$. □

We conclude by stating the lifting theorem for ordered fine-grained temporal resolution with selection without the eventuality resolution and the duplicate literal elimination in terminating step clauses rules.

Theorem 3.4.38 (Lifting Theorem). *Let $\mathbf{C} \subseteq CS$ be a set of constants. Additionally, let $\mathcal{N}$ be a set of constant-flooded temporal clauses with $const(\mathcal{N}) \cap \mathbf{C} = \emptyset$, and let $\tilde{\mathcal{N}}$ be a set of instances of clauses in $\mathcal{N}$ such that there is a substitution $\lambda\colon X \to T_\Sigma(X)$ with $codom(\lambda) \subseteq \mathbf{C}$ and $\mathcal{N}\lambda = \tilde{\mathcal{N}}$. Additionally, let $\tilde{\Delta} = \tilde{\mathcal{C}}_1, \ldots, \tilde{\mathcal{C}}_n$ be a $\mathfrak{I}^{S,\succ}_{FG}$-derivation from $\tilde{\mathcal{N}}$ which does not contain an application of an eventuality resolution or duplicate literal elimination in terminating step clauses rule.*

Then there exists a $\mathfrak{I}^{S,\succ}_{FG}$-derivation $\Delta = \mathcal{C}_1, \ldots, \mathcal{C}_n$ from $\mathcal{N}$ such that for every i, $1 \le i \le n$, there exists a substitution $\lambda_i\colon X \to T_\Sigma(X)$ with $\mathcal{C}_i\lambda_i = \tilde{\mathcal{C}}_i$ and $codom(\lambda_i) \subseteq \mathbf{C}$, which is denoted by $\Delta \le_{s,\mathbf{C}} \tilde{\Delta}$. It also holds that $const(\Delta) \cap \mathbf{C} = \emptyset$.

Proof. By induction on the length $|\tilde{\Delta}|$ of the derivation $\tilde{\Delta}$ using Lemmata 3.4.34, 3.4.35, 3.4.36, 3.4.37 and the lifting lemmata for ordered first-order resolution with selection (see, e.g., [10]). □

3.4.4 Proof of Refutational Completeness

The proof of completeness for ordered fine-grained resolution with selection consists in a simulation of a derivation produced by refined monodic temporal resolution $\mathfrak{I}^{\subseteq}_{e}$ by ordered fine-grained resolution with selection. We show that in each derivation step both calculi can derive formulae which have the same clausification result.

The next three lemmata are the key propositions for the completeness proof. They establish that particular merged derived and full merged step clauses can also be obtained by resolution, and vice versa, that some refutations by step resolution correspond to merged derived and full merged step clauses. Similar (but more concisely written) proofs can be found in [58].

Unless noted otherwise, we assume in this section that we are given an admissible atom ordering $\succ$ and an instance compatible selection function S. Also, the equality relation on formulae is interpreted w.r.t. commutativity of the disjunctions and the conjunctions present in the formulae and w.r.t. consistent renamings of bound variables occurring in the formulae.

The next two lemmata show that merged derived and full merged step clauses which are of a special form can also be derived by ordered fine-grained resolution with selection.

Lemma 3.4.39. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem with unique left-hand sides in step clauses such that the set $\mathcal{U}$ is satisfiable and let $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E} \rangle$ be a temporal problem such that the set $\mathcal{I}'$ is the clausification of the set $\mathcal{I}$, the set $\mathcal{S}'$ is the clausification of the constant-flooded set $\mathcal{S}$ (with respect to the temporal problem P) and such that* $\mathrm{Cls}(\mathcal{U}) \subseteq \mathcal{U}'$. *Moreover, let $\mathcal{A} \Rightarrow \bigcirc \mathcal{B}$ be a merged derived step clause such that $\mathcal{U} \cup \{\mathcal{B}\} \models$ **false** and for every $\mathcal{B}' \sqsubset \mathcal{B}$ the set $\mathcal{U} \cup \{\mathcal{B}'\}$ is satisfiable.*

Then there exists a step clause $\mathcal{P} \Rightarrow \bigcirc \bot$ obtained by ordered fine-grained resolution with selection from P' such that $Prenex(\mathcal{A}) = \tilde{\exists} \mathcal{P}$.

Proof. As the set $\mathcal{U}$ is satisfiable and $\mathcal{U} \cup \{\mathcal{B}\} \models$ **false**, it holds that $\mathcal{A} \Rightarrow \bigcirc \mathcal{B}$ is different from **true** $\Rightarrow \bigcirc$**true**. The merged derived clause $\mathcal{A} \Rightarrow \bigcirc \mathcal{B}$ is thus built from some (derived) step clauses $p_i \Rightarrow \bigcirc q_i$, $1 \leq i \leq m_1$, $P_j(c_j) \Rightarrow \bigcirc Q_j(c_j)$, $1 \leq j \leq m_2$, and $M_l^k(x) \Rightarrow \bigcirc N_l^k(x)$, $1 \leq k \leq m_3$, $1 \leq l \leq n_k$ in P, i.e.

$$\mathcal{A} \equiv \bigwedge_{i=1}^{m_1} p_i \wedge \bigwedge_{j=1}^{m_2} P_j(c_j) \wedge \bigwedge_{k=1}^{m_3} \left(\exists x \bigwedge_{l=1}^{n_k} M_l^k(x) \right)$$

and

$$\mathcal{B} \equiv \bigwedge_{i=1}^{m_1} q_i \wedge \bigwedge_{j=1}^{m_2} Q_j(c_j) \wedge \bigwedge_{k=1}^{m_3} \left(\exists x \bigwedge_{l=1}^{n_k} N_l^k(x) \right)$$

Furthermore, by definition of the step resolution rule, it holds that $\mathcal{U} \cup \{\mathcal{B}\} \models$ **false**, which implies that

$$\mathcal{U} \cup \{ \bigwedge_{i=1}^{m_1} q_i \wedge \bigwedge_{j=1}^{m_2} Q_j(c_j) \wedge \bigwedge_{k=1}^{m_3} \bigwedge_{l=1}^{n_k} N_l^k(d_k) \} \models \textbf{false}$$

holds, where d_k, $1 \leq k \leq m_3$, are fresh Skolem constants. Let now

$$L^c = \{ q_i \mid 1 \leq i \leq m_1 \} \cup \{ Q_j(c_j) \mid 1 \leq j \leq m_2 \} \cup \{ N_l^k(d_k) \mid 1 \leq k \leq m_3, 1 \leq l \leq n_k \}$$

be the clausification of $\mathcal{B}$, consisting solely of positive unit clauses. As $\mathcal{U} \cup \{\mathcal{B}\}$, and hence, $L^c \cup \mathcal{U}$ is unsatisfiable, it follows from refutational completeness of ordered resolution with selection (without the duplicate literal elimination rule) that there is a refutation Δ of $L^c \cup \mathcal{U}$. Moreover, as $\mathcal{U}$ is satisfiable and $\mathcal{U} \cup \{\mathcal{B}\}$ is unsatisfiable, the set L^c cannot be empty. Additionally, it is important to note that every unit clause contained in the set L^c is involved in the refutation Δ as otherwise there would exist a formula $\mathcal{B}' \sqsubset \mathcal{B}$ such that Δ is a refutation of $\mathcal{U} \cup \{\mathcal{B}'\}$, which contradicts the fact that the set $\mathcal{U} \cup \{\mathcal{B}'\}$ is satisfiable. Similarly, it is easy to see that the literals p_i, $1 \leq i \leq m_1$, $P_j(c_j)$, $1 \leq j \leq m_2$, and $M_l(x)$, $1 \leq l \leq n_k$ are pairwise different.

Let $\mathcal{S}^c$ be the following set of step clauses that correspond to the literals in L^c,

$$\begin{aligned}\mathcal{S}^c = \{\, p_i \Rightarrow \bigcirc q_i \mid 1 \leq i \leq m_1 \,\} \cup \{\, P_j(c_j) \Rightarrow \bigcirc Q_j(c_j) \mid 1 \leq j \leq m_2 \,\} \\ \cup \{\, M_l^k(d_k) \Rightarrow \bigcirc N_l(d_k) \mid 1 \leq k \leq m_3, 1 \leq l \leq n_k \,\}\end{aligned}$$

As every step clause in $\mathcal{S}^c$ is ground, it is possible to redo the derivation Δ as a derivation Γ' on the level of universal and step clauses in the calculus $\mathfrak{I}^{S,\succ}_{FG}$. The result of the derivation Γ' is a step clause $\mathcal{P}^c \Rightarrow \bigcirc\bot$, where

$$\mathcal{P}^c \equiv \bigwedge_{i=1}^{m_1} p_i \wedge \bigwedge_{j=1}^{m_2} P_j(c_j) \wedge \bigwedge_{k=1}^{m_3} \bigwedge_{l=1}^{n_k} M_l^k(d_k).$$

By setting $\mathbf{C} = \{\, d_k \mid 1 \leq k \leq m_3 \,\}$, it follows from the Lifting Theorem (Theorem 3.4.38) that there exists a derivation Γ such that $\Gamma \leq_{s,\mathbf{C}} \Gamma'$ and such that Γ is a proof of the clause $\mathcal{P} \Rightarrow \bigcirc\bot$ with

$$\mathcal{P} \equiv \bigwedge_{i=1}^{m_1} p_i \wedge \bigwedge_{j=1}^{m_2} P_j(c_j) \wedge \bigwedge_{k=1}^{m_3} \bigwedge_{l=1}^{n_k} M_l^k(x_k).$$

As the set of clausified step clauses $\mathcal{S}'$ is constant-flooded, the derivation Γ only uses clauses from the temporal problem P'. Finally, through applications of the duplicate literal elimination rule on the step clause $\mathcal{P} \Rightarrow \bigcirc\bot$, we can obtain a step clause $\mathcal{P}' \Rightarrow \bigcirc\bot$ such that $\mathit{Prenex}(\mathcal{A}) = \tilde{\exists}\mathcal{P}'$. □

Lemma 3.4.40. *Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem with unique left-hand sides in step clauses such that the set $\mathcal{U}$ is satisfiable and let $\mathsf{P}' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}' \cup \mathrm{LT}(\mathcal{S}'), \mathcal{E} \rangle$ be a temporal problem such that the set $\mathcal{I}'$ is the clausification of the set $\mathcal{I}$, the set $\mathcal{S}'$ is the clausification of the constant-flooded set $\mathcal{S}$ (with respect to the temporal problem P) with* $\mathrm{Cls}(\mathbf{true} \Rightarrow \bigcirc\neg\Phi(c^l)) \subseteq \mathsf{P}'$ *and such that* $\mathrm{Cls}(\mathcal{U}) \subseteq \mathcal{U}'$*. Moreover, let $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x))$ be a full merged step such that the formula $\forall x(\mathcal{U} \wedge \mathcal{B}(x) \Rightarrow \Phi(x))$ is valid and such that for every $\mathcal{B}'(x) \sqsubset \mathcal{B}(x)$ the formula $\exists x(\mathcal{U} \wedge \mathcal{B}'(x) \wedge \neg\Phi(x))$ is satisfiable.*

Then there exists a step clause $\mathcal{P} \Rightarrow \bigcirc\bot$ obtained by ordered fine-grained resolution with selection without the clause conversion rule from P' such that $\mathit{Prenex}(\mathcal{A}(x)) = (\tilde{\exists}\mathcal{P})\{c^l \to x\}$.

Proof. The proof is similar to the proof of Lemma 3.4.39.

First of all, if the full merged step clause $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x))$ is equal to the step clause $\mathbf{true} \Rightarrow \bigcirc\mathbf{true}$, then it follows from the assumption that the formula $\mathcal{U} \wedge \neg\Phi(c^l)$ is unsatisfiable. Thus, there exists a refutation Δ of the set $\mathcal{U} \cup \{\neg\Phi(c^l)\}$ by ordered first-order resolution with selection. By replacing the clauses resulting from clausifying the formula $\neg\Phi(c^l)$ by step clauses resulting from the clausification of $\mathbf{true} \Rightarrow \bigcirc\neg\Phi(c^l)$, it is easy to see that the derivation Δ can in fact be identified with a derivation of the final clause $\mathbf{true} \Rightarrow \bigcirc\bot$ by ordered fine-grained step resolution without applying the clause conversion rule.

Otherwise, the full merged step clause $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x))$ is different from the clause $\mathbf{true} \Rightarrow \bigcirc\mathbf{true}$. We have that the formula $\mathcal{U} \wedge \mathcal{B}(c^l) \wedge \neg\Phi(c^l)$ is unsatisfiable and that for all $\mathcal{B}' \sqsubset \mathcal{B}$ the formula $\mathcal{U} \wedge \mathcal{B}'(c^l) \wedge \neg\Phi(c^l)$ is satisfiable. Analogously to the proof of Lemma 3.4.39, by lifting all the constants except c^l we obtain a step clause $\mathcal{P} \Rightarrow \bigcirc\bot$ obtained by ordered fine-grained resolution with selection without applying the the clause conversion rule from P' such that $\mathit{Prenex}(\mathcal{A}(c^l)) = (\tilde{\exists}\mathcal{P})$. We can thus conclude that $\mathit{Prenex}(\mathcal{A}(x)) = (\tilde{\exists}\mathcal{P})\{c^l \to x\}$. □

We now show the converse statement, i.e. we prove that some derivations of terminating step clauses correspond to full merged step clauses with some special properties.

First of all, we have to define the operations $\mathrm{Res}(\mathcal{U})$ and $\mathrm{Res}^{\infty}(\mathcal{U})$ which are applied on sets of first-order clauses $\mathcal{U}$.

Definition 3.4.41. *Let $\mathcal{U}$ be a set of first-order clauses. Then we denote by $\mathrm{Res}(\mathcal{U})$ the set of all the clauses resulting from applying the ordered resolution with selection and ordered positive factoring with selection rules on clauses from the set $\mathcal{U}$. We also define that $\mathrm{Res}^0(\mathcal{U}) = \mathcal{U}$, $\mathrm{Res}^i(\mathcal{U}) = \mathrm{Res}(\mathrm{Res}^{i-1}(\mathcal{U}))$ for $i > 0$ and*

$$\mathrm{Res}^{\infty}(\mathcal{U}) = \bigcup_{i=0}^{\infty}(\mathrm{Res}^i(\mathcal{U}))$$

The next proposition is a direct adaptation of an analogous proof found in [58].

Lemma 3.4.42. *Let $P = \langle\mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ be a monodic temporal problem with unique left-hand sides in step clauses such that the set $\mathcal{U}$ is satisfiable and let $P' = \langle\mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}\rangle$ be a temporal problem such that the set $\mathcal{I}'$ is the clausification of the set $\mathcal{I}$, the set $\mathcal{S}'$ is the clausification of the constant-flooded set $\mathcal{S}$ (with respect to the temporal problem P) and such that $\mathrm{Cls}(\mathcal{U}) \subseteq \mathcal{U}'$ and $\mathrm{Res}^{\infty}(\mathcal{U}') = \mathrm{Res}^{\infty}(\mathrm{Cls}(\mathcal{U}))$. Additionally, let Δ be a proof of a final clause $C \Rightarrow \bigcirc\bot$ by the step and universal clause deduction rules of ordered fine-grained resolution with selection except the clause conversion rule from the set of universal clauses $\mathcal{U}'$ and the set of step clauses $\mathrm{LT}(\mathcal{S}') \cup \mathrm{Cls}(\{\mathbf{true} \Rightarrow \bigcirc\neg\Phi(c^l)\})$. Moreover, we assume that at least one of the clauses that originate from the formula $\mathbf{true} \Rightarrow \bigcirc\neg\Phi(c^l)$ is involved in the proof of the terminating step clause $C \Rightarrow \bigcirc\bot$.*

Then there exists a full-merged clause $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x))$ built from P such that the formula $\forall x(\mathcal{U} \wedge \mathcal{B}(x) \Rightarrow \Phi(x))$ is valid and $\mathit{Prenex}(\mathcal{A}(x)) = (\tilde{\exists}C)\{c^l \to x\}$.

Proof. First of all, we assume that the derivation Δ is tree-like, i.e. it may contain multiple copies of some clauses. Let $p_i \Rightarrow \bigcirc q_i$, $1 \leq i \leq m_1$, and $M_j(x_j) \Rightarrow \bigcirc N_j(x_j)$, $1 \leq j \leq m_2$, be all the step clauses contained in the set $\mathrm{LT}(\mathcal{S}')$ that are involved in the derivation Δ, where $p_i \Rightarrow \bigcirc q_i$ either denotes an *original* ground step clause or a ground step clause that has been added by constant-flooding. If $m_1 = 0$ and $m_2 = 0$, then it follows that $C = \mathbf{true}$. We can also infer that the set of clauses $\mathcal{U} \cup \mathrm{Cls}(\neg\Phi(c^l))$ is unsatisfiable, which implies that the set of formulae $\mathcal{U} \cup \{\exists x \neg\Phi(x)\}$ is also unsatisfiable, and thus, we obtain that the formula $\mathcal{U} \Rightarrow \forall x\, \Phi(x)$ is valid. The full-merged step clause $\mathbf{true} \Rightarrow \bigcirc\mathbf{true}$ fulfils the required properties. We may now assume that $m_1 \neq 0$ or $m_2 \neq 0$.

By accumulating the variable renamings and most general unifiers that have been applied on the step clauses in the derivation Δ, it is possible to construct a derivation Δ' from a finite set of instances of the step clauses $p_i \Rightarrow \bigcirc q_i$ $(1 \leq i \leq m_1)$ and $M_j(x_j) \Rightarrow \bigcirc N_j(x_j)$ $(1 \leq j \leq m_2)$ (and some universal clauses) such that the most general unifiers are identity substitutions on the variables occurring in the left-hand sides of step clauses[5]. Thus, there exist substitutions $\sigma_{i,j}$ for $1 \leq i \leq m_2$ and $1 \leq j \leq s_i$ such that:

$$C = \bigwedge_{i=1}^{m_1} p_i \wedge \bigwedge_{j=1}^{m_2} \bigwedge_{k=1}^{s_j} M_j(x_j)\sigma_{j,k}$$

Due to the restrictions on most general unifiers for the step resolution rules, one can see (by induction) that every substitution $\sigma_{j,k}$ for $1 \leq j \leq m_2$ and $1 \leq k \leq s_j$ maps the variable x_j into a variable.

In order to construct the full-merged step clause, we need to classify the step clause instances according to the values of the substitutions $\sigma_{j,k}$. We define an equivalence relation Σ on the step clauses as follows. For every j, j', $1 \leq j, j' \leq m_2$, and every k, k' with $1 \leq k \leq s_j$ and $1 \leq k' \leq s_{j'}$ we have:

$$(M_j(x_j)\sigma_{j,k} \Rightarrow \bigcirc N_j(x_j)\sigma_{j,k}, M_{j'}(x_{j'})\sigma_{j',k'} \Rightarrow \bigcirc N_{j'}(x_{j'})\sigma_{j',k'}) \in \Sigma$$

if and only if $x_j\sigma_{j,k} = x_{j'}\sigma_{j',k'}$. Furthermore, let N be the number of equivalence classes of Σ and let $\mathcal{I}_l$ for $1 \leq l \leq N$ be the sets of index pairs corresponding to the step clauses contained in the l-th equivalence class. Hence, we get:

$$C \equiv \bigwedge_{i=1}^{m_1} p_i \wedge \bigwedge_{l=1}^{N} \bigwedge_{(j,k)\in\mathcal{I}_l} M_j(x_j)\sigma_{j,k}$$

Now, let $D = \bigwedge_{i=1}^{m_1} q_i \wedge \bigwedge_{l=1}^{N} \bigwedge_{(j,k)\in\mathcal{I}_l} N_j(x_j)\sigma_{j,k}$ be built of the right-hand sides of the considered step clauses. Then, it follows from the assumptions that there exists a refutation of the set $\{\tilde{\forall} D\} \cup \mathcal{U}' \cup \{\neg\Phi(c^l)\}$ by (regular) first-order resolution such that the most general unifiers used are just identity substitutions on the variables $x_j\sigma_{j,k}$ $(1 \leq j \leq m_2$, $1 \leq k \leq s_j)$. Consequently, as $\text{Res}^\infty(\mathcal{U}') = \text{Res}^\infty(\text{Cls}(\mathcal{U}))$, there exists a refutation of the set $\{\tilde{\forall} D\} \cup \text{Cls}(\mathcal{U}) \cup \{\neg\Phi(c^l)\}$, which implies that $\tilde{\forall} D \wedge \mathcal{U} \wedge \neg\Phi(c^l) \models \textbf{false}$. Furthermore, by replacing the variables $\{x_j\sigma_{j,k} \mid 1 \leq j \leq m_2, 1 \leq k \leq s_j\}$ in the first-order derivation by fresh constants $d_1, \ldots, d_N$, it is easy to see that:

$$\bigwedge_{i=1}^{m_1} q_i \wedge \bigwedge_{l=1}^{N} \bigwedge_{(j,k)\in\mathcal{I}_l} (N_j(x_j)\sigma_{j,k})\{x_j\sigma_{j,k} \rightarrow d_l\} \wedge \mathcal{U} \wedge \neg\Phi(c^l) \models \textbf{false}$$

Hence, it follows that:

$$\bigwedge_{i=1}^{m_1} q_i \wedge \bigwedge_{l=1}^{N} \exists y\Big(\bigwedge_{(j,k)\in\mathcal{I}_l} (N_j(x_j)\sigma_{j,k})\{x_j\sigma_{j,k} \rightarrow y\}\Big) \wedge \mathcal{U} \wedge \neg\Phi(c^l) \models \textbf{false}$$

[5] We assume that the non-ground binary resolution rule can be applied on premises that are not variable-disjoint, which maintains the soundness of the derivation.

Finally, as $c^l \notin \mathcal{U}$ and $c^l \notin \mathit{const}(N_j(x_j))$ for all j, $1 \leq j \leq m_2$, we obtain:

$$\exists x\Big(\bigwedge_{i=1}^{m_1} q_i\{c^l \rightarrow x\} \wedge \bigwedge_{l=1}^{N} \exists y\Big(\bigwedge_{(j,k)\in\mathcal{I}_l} (N_j(x_j)\sigma_{j,k})\{x_j\sigma_{j,k} \rightarrow y\}\Big) \wedge \mathcal{U} \wedge \neg\Phi(x)\Big) \models \textbf{false}$$

Thus, it holds that $\forall x(\mathcal{U} \wedge \mathcal{B}(x) \Rightarrow \Phi(x))$ is valid and $\mathit{Prenex}(\mathcal{A}(x)) = (\tilde{\exists} C)\{c^l \rightarrow x\}$, where the full-merged step clause $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x))$ results from the following full-merged step clause built from the temporal problem P, after potentially adjusting the number of duplicate occurrences of some predicates:

$$\forall x\left(\bigwedge_{i=1}^{m_1} p_i\{c^l \rightarrow x\} \wedge \bigwedge_{l=1}^{N} \exists y\Big(\bigwedge_{(j,k)\in\mathcal{I}_l} (M_j(x_j)\sigma_{j,k})\{x_j\sigma_{j,k} \rightarrow y\}\Big)\right.$$
$$\left.\Rightarrow \bigcirc \bigwedge_{i=1}^{m_1} q_i\{c^l \rightarrow x\} \wedge \bigwedge_{l=1}^{N} \exists y\Big(\bigwedge_{(j,k)\in\mathcal{I}_l} (N_j(x_j)\sigma_{j,k})\{x_j\sigma_{j,k} \rightarrow y\}\Big)\right)$$

□

We can now show the simulation invariant for the eventuality resolution rules, first on the level of the Ref-BFS and the Restricted-FG-BFS algorithms.

Lemma 3.4.43. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ be a monodic temporal problem with unique left-hand sides in step clauses such that the set $\mathcal{U}$ is satisfiable and let $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}\rangle$ be a temporal problem such that the set $\mathcal{I}'$ is the clausification of the set $\mathcal{I}$, the set $\mathcal{S}'$ is the clausification of the constant-flooded set $\mathcal{S}$ (with respect to the temporal problem P) and such that* $\mathrm{Cls}(\mathcal{U}) \subseteq \mathcal{U}'$ *and* $\mathrm{Res}^{\infty}(\mathcal{U}') = \mathrm{Res}^{\infty}(\mathrm{Cls}(\mathcal{U}))$. *Additionally, let $\mathcal{N}_0', \mathcal{N}_1', \ldots$ be the sets of merged derived clauses and $H_0'(x), H_1'(x), \ldots$ be the sequence of formulae constructed in a run of the Ref-BFS algorithm applied on the temporal problem P for an eventuality $\Diamond L(x) \in \mathcal{E}$. Finally, let $\mathcal{M}_0, \mathcal{M}_1, \ldots$ be the sets of terminating step clauses and $R_0(x), R_1(x), \ldots$ be the sequence of formulae constructed in a run of the Restricted-FG-BFS algorithm applied on the temporal problem P' for the same eventuality $\Diamond L(x) \in \mathcal{E}$.*

Then it holds for all $i \geq 0$ that $\mathit{Prenex}(H_i'(x)) = \mathit{Prenex}(R_i(x))$ and:

$$\{\, \mathit{Prenex}(\mathcal{A}(x)) \mid \forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}_i' \,\} = \{\, (\tilde{\exists} D)\{c^l \rightarrow x\} \mid D \Rightarrow \bigcirc\bot \in \mathcal{M}_i \,\}$$

Proof. By induction on i. In the case of $i = 0$ we have $H_0'(x) = \textbf{true} = R_0(x)$ and $\mathcal{N}_0' = \emptyset = \mathcal{M}_0$. For $i + 1$ with $i \geq 0$, we can first of all assume that $\mathcal{N}_i' \neq \emptyset \neq \mathcal{M}_i$ and it it follows from the induction hypothesis that $\mathit{Prenex}(H_i'(x)) = \mathit{Prenex}(R_i(x))$, which implies that $H_i'(x) \equiv R_i(x)$. Now let $\mathcal{N}_{i+1}' = \{\forall x(\mathcal{A}_j^{(i+1)}(x) \Rightarrow \bigcirc\mathcal{B}_j^{(i+1)}(x))\}_{j=1}^{k}$, $\mathcal{T}_{i+1} = \{C_j \Rightarrow \bigcirc\bot\}_{j=1}^{m}$ and $\mathcal{M}_{i+1} = \{D_j \Rightarrow \bigcirc\bot\}_{j=1}^{n}$ be the sets respectively computed by the Ref-BFS and Restricted-FG-BFS algorithms. We show that

$$\{\, \mathit{Prenex}(\mathcal{A}(x)) \mid \forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}_{i+1}' \,\} = \{\, (\tilde{\exists} D)\{c^l \rightarrow x\} \mid D \Rightarrow \bigcirc\bot \in \mathcal{M}_{i+1} \,\}.$$

For $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}_{i+1}'$, we obtain by Lemma 3.4.40 (for $\Phi(x) = \neg L(x) \wedge R_i(x)$) that there exists a step clause $\mathcal{P} \Rightarrow \bigcirc\bot$ obtained by ordered fine-grained resolution with selection without the clause conversion rule from $\langle \mathcal{U}', \mathcal{I}', \mathcal{S}' \cup \mathrm{Cls}(\mathbf{true} \Rightarrow \bigcirc\neg\Phi(c^l)), \mathcal{E}\rangle$ such that $Prenex(\mathcal{A}(x)) = (\tilde{\exists}\mathcal{P})\{c^l \rightarrow x\}$. If we assume that there is a $C_j \Rightarrow \bigcirc\bot \in \mathcal{T}_{i+1}$ for $1 \leq j \leq m$ such that $(\tilde{\exists}\mathcal{P})\{c^l \rightarrow x\} \sqsupset (\tilde{\exists}C_j)\{c^l \rightarrow x\}$, then it would follow from Lemma 3.4.42 that there exists a full merged clause $\forall x(\mathcal{A}'(x) \Rightarrow \bigcirc\mathcal{B}'(x))$ built from P such that the formula $\forall x(\mathcal{U} \wedge \mathcal{B}'(x) \Rightarrow \neg L_i(x) \wedge H_i'(x))$ is valid and $Prenex(\mathcal{A}'(x)) = (\tilde{\exists}C_j)\{c^l \rightarrow x\}$. Thus, it would hold that $\mathcal{B}'(x) \sqsubset \mathcal{B}(x)$ and that the formula $\exists x(\mathcal{U} \wedge \mathcal{B}'(x) \wedge \neg(\neg L(x) \wedge H_i(x)))$ is unsatisfiable, which contradicts the fact that $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}_{i+1}'$. We can hence conclude that $\mathcal{P} \Rightarrow \bigcirc\bot \in \mathcal{M}_{i+1}$.

Let now $D_j \Rightarrow \bigcirc\bot \in \mathcal{M}_{i+1}$ for $1 \leq j \leq n$. Then it follows from Lemma 3.4.42 that there exists a full merged clause $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x))$ built from P such that the formula $\forall x(\mathcal{U} \wedge \mathcal{B}(x) \Rightarrow \neg L_i(x) \wedge H_i'(x))$ is valid and $Prenex(\mathcal{A}(x)) = (\tilde{\exists}D_j)\{c^l \rightarrow x\}$. If we assume that there is a full merged step clause $\forall x(\mathcal{A}'(x) \Rightarrow \bigcirc\mathcal{B}'(x))$ built from P such that $\mathcal{B}'(x) \sqsubset \mathcal{B}(x)$ and the formula $\exists x(\mathcal{U} \wedge \mathcal{B}'(x) \wedge \neg(\neg L(x) \wedge H_i(x)))$ is unsatisfiable, then it would follow from Lemma 3.4.40 for the minimal such full merged step clause $\forall x(\mathcal{A}''(x) \Rightarrow \bigcirc\mathcal{B}''(x))$ that there exists a step clause $C \Rightarrow \bigcirc\bot$ obtained by ordered fine-grained resolution with selection without the clause conversion rule from $\langle \mathcal{U}', \mathcal{I}', \mathcal{S}' \cup \mathrm{Cls}(\mathbf{true} \Rightarrow \bigcirc\neg\Phi(c^l)), \mathcal{E}\rangle$ such that $Prenex(\mathcal{A}''(x)) = (\tilde{\exists}C)\{c^l \rightarrow x\}$. Hence, it would hold that $C \Rightarrow \bigcirc\bot \in \mathcal{T}_{i+1}$ (or potentially a variable-renamed version) and $(\tilde{\exists}C)\{c^l \rightarrow x\} \sqsubset (\tilde{\exists}D_j)\{c^l \rightarrow x\}$, which contradicts the fact $D_j \Rightarrow \bigcirc\bot \in \mathcal{M}_{i+1}$. Thus, we obtain $\forall x(\mathcal{A}(x) \Rightarrow \bigcirc\mathcal{B}(x)) \in \mathcal{N}_{i+1}'$.

Finally, we observe that $\bot \notin \mathrm{Res}^\infty(\mathcal{U}')$ as $\mathrm{Res}^\infty(\mathcal{U}') = \mathrm{Res}^\infty(\mathrm{Cls}(\mathcal{U}))$ and $\mathcal{U}$ is satisfiable. Hence, $Prenex(H_{i+1}'(x)) = Prenex(R_{i+1}(x))$ holds. □

In a second step we prove the simulation invariant for the eventuality resolution rules w.r.t. the formulae computed by Ref-BFS and the Restricted-FG-BFS algorithms.

Lemma 3.4.44. *Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ be a monodic temporal problem with unique left-hand sides in step clauses such that the set $\mathcal{U}$ is satisfiable and let $\mathsf{P}' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}\rangle$ be a temporal problem such that the set $\mathcal{I}'$ is the clausification of the set $\mathcal{I}$, the set $\mathcal{S}'$ is the clausification of the constant-flooded set $\mathcal{S}$ (with respect to the temporal problem P) and such that $\mathrm{Cls}(\mathcal{U}) \subseteq \mathcal{U}'$ and $\mathrm{Res}^\infty(\mathcal{U}') = \mathrm{Res}^\infty(\mathrm{Cls}(\mathcal{U}))$. Additionally, let $H'(x)$ be the result of applying the Ref-BFS algorithm on the temporal problem P and let $R(x)$ be the result of applying the Restricted-FG-BFS algorithm on the sets $\mathcal{U}'$ and $\mathcal{S}'$, both for an eventuality $\Diamond L(x) \in \mathcal{E}$.*

Then it holds that $Prenex(H'(x)) = Prenex(R(x))$.

Proof. By Lemma 3.4.43 we obtain $Prenex(H_i'(x)) = Prenex(R_i(x))$ for all $i \geq 0$. Thus, it holds that either $H'(x) = \mathbf{false}$ and $R(x) = \mathbf{false}$, or $Prenex(H'(x)) = Prenex(H_j'(x)) = Prenex(R_j(x)) = Prenex(R(x))$ for a $j \geq 1$. □

Finally, we can establish the refutational completeness of ordered fine-grained resolution with selection. First, we prove the result for monodic temporal problems with unique left-hand sides in step clauses.

Theorem 3.4.45. *Let $P = \langle \mathcal{U}_0, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be an unsatisfiable monodic temporal problem with unique left-hand sides in step clauses, let P^c be its constant-flooded form and let $P' = \mathrm{Cls}(P^c) = \langle \mathcal{U}'_0, \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$ be the clausification of P^c. Additionally, let $\succ$ be an admissible atom ordering and S be an instance compatible selection function.*

Then there exists a refutation of P' by ordered fine-grained resolution with selection ($\mathfrak{I}^{S,\succ}_{FG}$) such that the premises for the eventuality resolution rules are found by the Restricted-FG-BFS algorithm, which is only applied on the set of step clauses $\mathcal{S}'$.

Proof. The proof of completeness proceeds along the lines of the completeness proof of $\mathfrak{I}_{FG}$ presented in [58].

By Lemma 3.4.13 there exists a successfully terminating derivation $\Delta = \mathcal{U}_0, \ldots, \mathcal{U}_n$ from P^c in $\mathfrak{I}^{\subseteq}_{e}$ (i.e. $\bot \in \mathcal{U}_n$) which uses the refined step resolution rule instead of the regular step resolution rule and which restricts the application of the eventuality resolution rules to loops found by the Ref-BFS algorithm. Furthermore, the refined step resolution rule and the eventuality resolution rules are only applied on satisfiable sets of universal formulae in the derivation Δ. By induction on the length of the derivation we show that this derivation can be simulated by $\mathfrak{I}^{S,\succ}_{FG}$. We construct a refutation $\Delta'_n = C^1_0, \ldots, C^{m_0}_0, \ldots, C^1_n, \ldots, C^{m_n}_n$ of P' where each step in Δ will correspond to one or more steps in Δ'_n. At the start Δ just consists of $\mathcal{U}_0$ and the corresponding derivation Δ'_0 consists of all the clauses $C^1_0, \ldots, C^{n_0}_0$ in P'. Let $I(\Delta'_i)$, $U(\Delta'_i)$, $S(\Delta'_i)$ denote the set of all initial, universal and step clauses in Δ'_i $(1 \leq i \leq n)$, respectively. By the fact that clausification preserves satisfiability, $\mathcal{U}_0$ is satisfiable if and only if $U(\Delta'_0)$ is satisfiable and $\mathcal{U}_0 \cup \mathcal{I}$ is satisfiable if and only if $U(\Delta'_0) \cup I(\Delta'_0)$ is satisfiable. Furthermore, if $\mathcal{U}_0$ would contain $\bot$, then Δ'_0 would contain the empty clause. It is clear that $\mathrm{Cls}(\mathcal{U}_0) \subseteq U(\Delta'_0)$ and $\mathrm{Res}^{\infty}(U(\Delta'_0)) = \mathrm{Res}^{\infty}(\mathrm{Cls}(\mathcal{U}_0))$.

Now, in each step of Δ a first-order formula u_i, $1 \leq i \leq n$, is added to $\mathcal{U}_{i-1}$ to obtain $\mathcal{U}_i$, where u_i is the conclusion of one the deduction rules of $\mathfrak{I}^{\subseteq}_{e}$ applied to $\langle \mathcal{U}_{i-1}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$. We show that using $\mathfrak{I}^{S,\succ}_{FG}$ we can derive a formula $C^{m_i}_i$ from the clauses in the derivation Δ'_{i-1} constructed so far such that $\mathrm{Cls}(C^{m_i}_i) = \mathrm{Cls}(u_i)$. We then add $\mathrm{Cls}(C^{m_i}_i)$ and all intermediate clauses $C^1_i, \ldots, C^{m_i-1}_i$ used in its derivation to Δ'_{i-1}. It obviously holds then that $\mathrm{Cls}(\mathcal{U}_i) \subseteq U(\Delta'_i)$ and $\mathrm{Res}^{\infty}(U(\Delta'_i)) = \mathrm{Res}^{\infty}(\mathrm{Cls}(\mathcal{U}_i))$. To show the existence and derivability of $C^{m_i}_i$, we consider which deduction rule of $\mathfrak{I}^{\subseteq}_{e}$ has been used to derive u_i with $i \geq 1$. As $i > 0$, it follows from the induction hypothesis that there exists a derivation $\Delta'_{i-1} = C^1_0, \ldots, C^{m_0}_0, \ldots, C^1_{i-1}, \ldots, C^{m_i-1}_{i-1}$ such that $\mathrm{Cls}(\mathcal{U}_{i-1}) \subseteq U(\Delta'_{i-1})$ and $\mathrm{Res}^{\infty}(U(\Delta'_{i-1})) = \mathrm{Res}^{\infty}(\mathrm{Cls}(\mathcal{U}_{i-1}))$.

Suppose u_i has been derived by an application of the (initial) termination rule (which implies that u_i is $\bot$). Then the set $\mathcal{U}_{i-1} \cup \mathcal{I}$ of first-order formulae is unsatisfiable. By completeness of first-order ordered resolution with selection (see, e.g., [10]), we will be able to derive the empty clause in a derivation $C^1_i, \ldots, C^{m_i}_i$, i.e. $C^{m_i}_i = \bot$, from the clauses in $\mathrm{Cls}(\mathcal{U}_{i-1}) \cup \mathrm{Cls}(\mathcal{I})$. Thus, as $\mathrm{Cls}(\mathcal{U}_{i-1}) \subseteq U(\Delta'_{i-1})$ and $\mathrm{Cls}(\mathcal{I}) \subseteq I(\Delta'_{i-1})$, it is possible to derive the clauses $C^1_i, \ldots, C^{m_i}_i$ (identified as universal or initial clauses) also from $U(\Delta'_{i-1}) \cup I(\Delta'_{i-1})$ using the resolution and factoring rules of $\mathfrak{I}^{S,\succ}_{FG}$ for universal and initial clauses, that is, rules 1 to 3. We extend Δ'_{i-1} by $C_i^1, \ldots, C_i^{m_i}$.

Suppose u_i has been derived by an application of the refined step resolution rule on a satisfiable set of universal formulae $\mathcal{U}_{i-1}$. Then, we have $u_i = \neg\mathcal{A}$, where $\mathcal{A} \Rightarrow \bigcirc\mathcal{B}$ is a merged derived step clause built from $\langle \mathcal{U}_{i-1}, \mathcal{I}, \mathcal{S}, \mathcal{E}^c \rangle$. By Lemma 3.4.39 there exists a step clause $\mathcal{P} \Rightarrow \bigcirc\bot$ obtained by ordered fine-grained resolution with selection from $\langle U(\Delta'_{i-1}), \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$ such that $Prenex(\mathcal{A}) = \tilde{\exists}\mathcal{P}$. An application of rule 6 for clause conversion allows us to derive the universal clause $\neg\mathcal{P}$. It is easy to see that $\mathrm{Cls}(\neg\mathcal{A}) = \mathrm{Cls}(\tilde{\forall}\neg\mathcal{P})$. We add all the clauses in the derivation of $\mathcal{P} \Rightarrow \bigcirc\bot$ to Δ' as $C_i^1, \ldots, C_i^{m_i-1}$ for some m_i, and also add $\mathrm{Cls}(\tilde{\forall}\neg\mathcal{P})$ as $C_i^{m_i}$.

Suppose u_i has been derived by an application of the non-ground eventuality resolution rule on a loop formula $H'(x) = \bigvee_{j=1}^{k} \mathcal{A}_j(x)$ found by the Ref-BFS algorithm which has been applied on a satisfiable set of universal formulae $\mathcal{U}_{i-1}$ for an eventuality $\Diamond L(x) \in \mathcal{E}$ and where $\forall x(\mathcal{A}_1(x) \Rightarrow \bigcirc\mathcal{B}_1(x)), \ldots, \forall x(\mathcal{A}_k(x) \Rightarrow \bigcirc\mathcal{B}_k(x))$ are full merged step clauses built from $\langle \mathcal{U}_{i-1}, \mathcal{I}, \mathcal{S}, \mathcal{E}^c \rangle$. Then, it holds that $u_i = \forall x \neg H'(x) = \forall x \bigwedge_{j=1}^{k} \neg\mathcal{A}_j(x)$. It follows from Lemma 3.4.44 that the algorithm Ref-BFS applied on the sets $U(\Delta'_{i-1})$ and S' for the eventuality $\Diamond L(x)$ will compute a formula $R(x)$ such that $Prenex(H'(x)) = Prenex(R(x))$. It is easy to see that $\mathrm{Cls}(\forall x \neg H'(x)) = \mathrm{Cls}(\forall x \neg R(x))$. We add $\mathrm{Cls}(\forall x \neg H'(x))$ to Δ'_{i-1}.

Finally, the remaining case where the clause u_i has been derived by an application of the ground eventuality resolution rule follows in analogy to the non-ground eventuality resolution case. □

In a final step we prove the refutational completeness of ordered fine-grained resolution with selection for arbitrary monodic temporal problems.

Theorem 3.4.46 (Refutational Completeness). *Let $\mathsf{P} = \langle \mathcal{U}_0, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be an unsatisfiable monodic temporal problem, let P^c be its constant-flooded form and let $\mathsf{P}' = \mathrm{Cls}(\mathsf{P}^c) = \langle \mathcal{U}'_0, \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$ be the clausification of P^c. Additionally, let $\succ$ be an admissible atom ordering and S be an instance compatible selection function.*

Then there exists a refutation of P' by ordered fine-grained resolution with selection $\mathfrak{I}_{FG}^{S,\succ}$ such that the premises for the eventuality resolution rules are found by Restricted-FG-BFS algorithm, which is only applied on the set of step clauses $\mathcal{S}'$.

Proof. Let

$$\mathcal{S}' = \bigcup_{i=1}^{m_1} \{ P_i(x) \Rightarrow \bigcirc Q_j^i(x) \mid 1 \le j \le n_i^1 \} \cup \bigcup_{i=1}^{m_2} \{ p_i \Rightarrow \bigcirc q_j^i \mid 1 \le j \le n_i^2 \} \cup \mathcal{T},$$

where $\mathcal{T}$ only contains step clauses with unique left-hand sides and $\mathcal{T}$ is maximal with that property. Now, let $\tilde{\mathsf{P}}' = \langle \tilde{\mathcal{U}}'_0, \mathcal{I}', \tilde{\mathcal{S}}', \mathcal{E}^c \rangle$ be a temporal problem with

$$\tilde{\mathcal{U}}'_0 = \mathcal{U}'_0 \cup \bigcup_{i=1}^{m_1} \{ \neg R_i(x) \vee Q_j^i(x) \mid 1 \le j \le n_i^1 \} \cup \bigcup_{i=1}^{m_2} \{ \neg r_i \vee q_j^i \mid 1 \le j \le n_i^2 \}$$

and

$$\begin{aligned}\tilde{\mathcal{S}}' = \mathcal{T} \cup \bigcup_{c \in const(\mathsf{P})} &\{ P_i(c) \Rightarrow \bigcirc R_i(c) \mid 1 \leq i \leq m_1 \} \\ &\cup \{ P_i(x) \Rightarrow \bigcirc R_i(x) \mid 1 \leq i \leq m_1 \} \cup \{ p_i \Rightarrow \bigcirc r_i \mid 1 \leq i \leq m_2 \},\end{aligned}$$

where R_i, $1 \leq i \leq m_1$, and p_i, $1 \leq i \leq m_2$, are fresh unary predicate symbols and fresh propositions, respectively. It is easy to see that the temporal problem $\tilde{\mathsf{P}}'$ is satisfiable if and only if the temporal problem P' is satisfiable. Moreover, the step clauses in the temporal problem $\tilde{\mathsf{P}}'$ have unique left-hand sides. By Theorem 3.4.45 there hence exists a refutation $\tilde{\Delta}$ of the temporal problem $\tilde{\mathsf{P}}'$ by ordered fine-grained resolution with selection $\mathfrak{I}_{FG}^{S,\succ}$ such that the premises for the eventuality resolution rules are found by Restricted-FG-BFS algorithm, which is only applied on the set of step clauses $\tilde{\mathcal{S}}'$. We can assume without loss of generality that the derivation $\tilde{\Delta}$ does not contain any clauses that do not contribute to the derivation of the empty clause.

As the literals $R_i(x)$ $(1 \leq i \leq m_1)$ and r_i $(1 \leq i \leq m_2)$ occur positively only in the set $\tilde{\mathcal{S}}'$ and negatively only in the set $\tilde{\mathcal{U}}_0'$, one can infer that in the derivation $\tilde{\Delta}$ those positive occurrences of a literal $R_i(x)$ $(1 \leq i \leq m_1)$ or r_i $(1 \leq i \leq m_2)$ not removed by factoring inferences are eventually resolved with their corresponding negative occurrences, and vice versa. Additionally, the literals $R_i(x)$ $(1 \leq i \leq m_1)$ and r_i $(1 \leq i \leq m_2)$ are only directly involved in derivations of terminating step clauses $C \Rightarrow \bigcirc\bot$. Furthermore, as the positive occurrences of $R_i(x)$ $(1 \leq i \leq m_1)$ and r_i $(1 \leq i \leq m_2)$ are essentially unit clauses in the derivation $\tilde{\Delta}$, it is therefore easy to see that the literals $R_i(x)$ $(1 \leq i \leq m_1)$ and r_i $(1 \leq i \leq m_2)$ add an additional resolution inference in comparison to applying ordered fine-grained resolution with selection on the original temporal problem P' directly. We can thus infer that for every derivation of a terminating step clause $C \Rightarrow \bigcirc\bot$ from the temporal problem $\tilde{\mathsf{P}}'$ by $\mathfrak{I}_{FG}^{S,\succ}$ there exists a corresponding derivation by $\mathfrak{I}_{FG}^{S,\succ}$ from the original temporal problem P', which results in the same terminating step clause $C \Rightarrow \bigcirc\bot$ (up to variable renaming).

Thus, we can conclude that a refutation Δ can be obtained from the temporal problem P' by ordered fine-grained resolution with selection such that the premises for the eventuality resolution rules are found by Restricted-FG-BFS algorithm, which is only applied on the set of step clauses $\mathcal{S}'$. □

3.5 Summary

The aim of this chapter was to prove the refutational completeness of ordered fine-grained temporal resolution with selection.

After having briefly recalled the inference rules of monodic temporal resolution and ordered fine-grained temporal resolution with selection, we focused on the proof of refutational completeness for ordered fine-grained resolution with selection. First, we defined a refined version of monodic temporal resolution, for which we also proved that it is refutationally complete. We then showed the lifting theorem for ordered fine-grained resolution with selection without the eventuality resolution rules and the arbitrary factoring in left-hand sides of terminating step clauses rule. Subsequently, we proved that derivations of refined monodic temporal resolution can be simulated by ordered fine-grained resolution with selection. As refined monodic temporal resolution was shown to be refutationally complete for temporal problems that only contain step clauses with unique left-hand sides, we obtained a completeness result for ordered fine-grained resolution with selection restricted to those temporal problems. In a final step we then extended this completeness result to arbitrary temporal problems.

Chapter 4

Redundancy Elimination in Monodic Temporal Reasoning

4.1 Introduction

In this chapter we focus on another aspect of ordered fine-grained temporal resolution with selection, namely, redundancy elimination. In order to decrease the time required for finding proofs in practice, one would ideally like to identify clauses that do not contribute to refutations or lead to longer derivations. The use of an ordering and a selection function which restricts inferences to literals which are selected or, in the absence of selected literals, to (strictly) maximal literals, already reduces the number of possible inferences considerably. However, it cannot prevent the derivation of redundant clauses, e.g. tautological clauses or clauses which are subsumed by other, simpler, clauses. Redundancy elimination is therefore an important ingredient for practical resolution calculi and for theorem provers based on such calculi. In fact, it is common that state-of-the-art resolution-based theorem provers spend more time with reducing the clause sets used for inference computations than with deriving new clauses.

The question arises whether tautological and subsumed clauses can be eliminated without losing refutational completeness in the context of the $\mathfrak{I}^{S,\succ}_{FG}$-calculus. The answer is not as straightforward as it may appear. For example, lock resolution is not compatible with the removal of tautologies [10]. So, there is at least the theoretical possibility that our calculus could become incomplete if tautologies were to be removed from derivations. However, this will turn out not to be the case. Regarding the elimination of subsumed clauses, we will show in Section 4.2.2 that the notion of a subsumed clause needs to be defined carefully in our context otherwise refutational completeness will indeed be lost.

The chapter is organised as follows. Section 4.2 focuses on redundancy elimination in combination with the resolution-based inference rules of ordered fine-grained resolution with selection. We first show which temporal clauses are in fact tautologies. Then we define a subsumption relation on temporal clauses and we illustrate how the calculus has to be extended to remain compatible with the removal of subsumed clauses and of tautologies.

The extended calculus will be called *subsumption-compatible ordered fine-grained resolution with selection.* We also prove the subsumption lemmata for the subsumption-compatible calculus.

In Section 4.3 we analyse the problem of combining the loop search process with redundancy elimination: it is not clear at first sight whether the FG-BFS and the Restricted FG-BFS algorithms still compute loops correctly if, for example, subsumed clauses are removed. For this purpose, we introduce a resolution-based loop search algorithm called *Subsumption-Restricted-FG-BFS.* After proving some of its properties, we conclude with showing the refutational completeness of subsumption-compatible ordered fine-grained resolution with selection where applications of the eventuality resolution rules are restricted to loops found by the Subsumption-Restricted-FG-BFS algorithm.

4.2 Adding Redundancy Elimination

Given that our calculus uses an ordering refinement, it seems natural to establish that the calculus admits redundancy elimination by using the approach in [10, Section 4.2]. To do so, we would first need to define a model functor I that maps any (not necessarily satisfiable) temporal problem P not containing the empty clause to an interpretation $\mathfrak{M}_{\mathsf{P}}$ and then show that $\mathfrak{I}^{S,\succ}_{FG}$ has the reduction property for counterexamples with respect to the model functor I and an ordering $\succ$, that is, for every temporal problem P and minimal clause C in P which is false in $\mathfrak{M}_{\mathsf{P}}$, there exists an inference with (main) premise C and conclusion $\mathcal{D}$ that is also false in P but smaller than C w.r.t. $\succ$. We could then define a clause C to be redundant w.r.t. P if there exists clauses $C_1, \ldots, C_k$ in P such that $C_1, \ldots, C_k \models C$ and $C \succ C_i$ for all i, $1 \leq i \leq k$, and it would be straightforward to show that $\mathfrak{I}^{S,\succ}_{FG}$ remains complete if redundant clauses are eliminated from derivations.

First of all, we can observe that the notion of reducing counterexamples does not really fit well with the eventuality resolution rules and moreover, due to the presence of eventualities in temporal problems, defining an appropriate model functor is a non-trivial and open problem. For example, consider the satisfiable propositional temporal problem $\mathsf{P} = \langle \{p \vee q\}, \emptyset, \{p \Rightarrow \bigcirc \neg l\}, \{\Diamond l\} \rangle$ and an ordering $\succ$ such that $p \succ q$. Applying the standard model functor defined in [10] to the clause $p \vee q$ results in a model in which p is true. Given that $p \vee q$ is a universal clause, it would be natural to define $\mathfrak{M}_{\mathsf{P}}$ in such way that p is true at every moment of time. However, due to the step clause $p \Rightarrow \bigcirc \neg l$, $\Box \Diamond l$ is not true in $\mathfrak{M}_{\mathsf{P}}$ which means that $\mathfrak{M}_{\mathsf{P}}$ is not a model of P. Thus, this simplistic approach to defining a model functor is not correct for temporal problems containing eventualities.

In Chapter 7 we introduce a model functor I for propositional temporal problems, which is able to associate a model $\mathfrak{M}$ of P with every satisfiable temporal problem P, but $\mathfrak{I}^{S,\succ}_{FG}$ is not reductive w.r.t. I. Thus, this model functor is not suitable for establishing that $\mathfrak{I}^{S,\succ}_{FG}$ admits redundancy elimination.

Consequently, we have to follow a different approach in order to show that ordered fine-grained resolution with selection can be extended with redundancy elimination rules.

In the following we will define the notions of a tautological clause and of a subsumed clause. For proving that $\mathfrak{I}_{FG}^{S,\succ}$ is still complete if such clauses are eliminated during a derivation, we need to show that for every refutation without redundancy elimination there exists a refutation with redundancy elimination. It turns out that in order to be able to do so, we need to add two inference rules to our calculus.

4.2.1 Tautological Temporal Clauses

First of all, we consider tautological clauses. As a tautological clause is defined to be a clause that is true in every structure $\mathfrak{M} = (D_n, I_n)_{n \in \mathbb{N}}$, we obtain the following lemma:

Lemma 4.2.1. *Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a temporal problem, and let $\mathcal{C}$ be a initial, universal or step clause. Then:*

(i) If $\mathcal{C}$ is an initial or universal clause, then $\mathcal{C}$ is a tautology if and only if $\mathcal{C} = \neg L \vee L \vee C'$, for some possibly empty disjunction of literals C'.

(ii) If $\mathcal{C} = C_1 \Rightarrow \bigcirc C_2$ is a step clause, then $\mathcal{C}$ is a tautology if and only if $C_2 = \neg L \vee L \vee C_2'$, for some possibly empty disjunction of literals C_2'.

Proof. Follows straightforwardly from the fact that if a clause does not contain complementary literals, then one can construct an interpretation which falsifies it. □

Note that for point (ii) of the lemma above C_1 is assumed to be **true** or a non-empty conjunction of atoms.

Thus, just as in the non-temporal first-order case, there is again a syntactic criterion for characterising tautologies, namely the presence of complementary literals. For a set of clauses $\mathcal{N}$ or a temporal problem P, we denote by $\mathsf{taut}(\mathcal{N})$ or $\mathsf{taut}(\mathsf{P})$ the set of all the tautological clauses contained in the set $\mathcal{N}$ or the temporal problem P, respectively.

4.2.2 A Subsumption Relation on Temporal Clauses

The subsumption relation on initial, universal and step clauses is now defined as follows.

Definition 4.2.2. *Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a temporal problem. We define a subsumption relation $\leq_s$ on initial, universal and step clauses as follows:*

(i) For two initial clauses $\mathcal{C}$ and $\mathcal{D}$, two universal clauses $\mathcal{C}$ and $\mathcal{D}$, or a universal clause $\mathcal{C}$ and an initial clause $\mathcal{D}$ we define

$$\mathcal{C} \leq_s \mathcal{D} \textit{ if and only if there exists a substitution } \sigma \textit{ with } \mathcal{C}\sigma \subseteq \mathcal{D}.$$

(ii) For two step clauses $\mathcal{C} = C_1 \Rightarrow \bigcirc C_2$ and $\mathcal{D} = D_1 \Rightarrow \bigcirc D_2$ we define

$$\mathcal{C} \leq_s \mathcal{D} \textit{ if and only if there exists a substitution } \sigma \textit{ with } C_1\sigma \subseteq D_1, C_2\sigma \subseteq D_2 \textit{ and}$$
$$\textit{for every } x \in \mathit{var}(C_1) \cap \mathit{var}(C_2)\colon \sigma(x) \in X.$$

(iii) For a universal clause $\mathcal{C}$ and a step clause $\mathcal{D} = D_1 \Rightarrow \bigcirc D_2$ we define

$$\mathcal{C} \leq_s \mathcal{D} \text{ if and only if there exists a substitution } \sigma \text{ with } \mathcal{C}\sigma \subseteq \neg D_1 \text{ or } \mathcal{C}\sigma \subseteq D_2.$$

For two sets of clauses $\mathcal{N}$ and $\mathcal{N}'$ we denote by $\mathcal{N} \leq_s \mathcal{N}'$ that all clauses in $\mathcal{N}'$ are subsumed by clauses in $\mathcal{N}$.

Remark 4.2.3. *For two formulae C and D with*

$$C = \bigwedge_{i=1}^{m_1} A_i^1(x_{1,i}) \wedge \bigwedge_{j=1}^{n_1} A_j^2(c_j) \wedge \bigwedge_{k=1}^{l_1} p_k \text{ and } D = \bigwedge_{i=1}^{m_2} B_i^1(x_{2,i}) \wedge \bigwedge_{j=1}^{n_2} B_j^2(d_j) \wedge \bigwedge_{k=1}^{l_2} q_k,$$

where A_i^1, A_j^2, B_i^1, B_j^2 are monadic predicates, $x_{1,i}$, $x_{2,i}$ are variables, c_j, d_j, are constants and p_k, q_k, are propositions, we write $C \leq_s D$ if and only if there exists a substitution σ with $C\sigma \subseteq D$, i.e. $\neg C \leq_s \neg D$ holds, where $\neg C$ and $\neg D$ are considered to be negative universal clauses.

Thus, subsumption between two initial, two universal or an initial and a universal clause is defined analogously to the subsumption on regular first-order clauses. However, we can only allow a universal clause to subsume an initial clause, but not conversely, as an initial clause only holds in the initial moment of time while a universal clause is true at every moment of time. We also allow subsumption between a universal and a step clause if and only if the universal clause either subsumes the negated left-hand side or the right-hand side of the step clause.

For subsumption between two step clauses $C_1 \Rightarrow \bigcirc C_2$ and $D_1 \Rightarrow \bigcirc D_2$, we have to impose an additional constraint on the substitution that is used for the multiset inclusion: in analogy to the inference rules of the calculus that involve step clauses (rules 4 and 5 of ordered fine-grained resolution with selection), it has to be ensured that variables occurring both in the left-hand side C_1 and in the right-hand side C_2 are only mapped to variables. While for the inference rules themselves this restriction is imposed to ensure soundness, here the motivation is completeness.

To see that, consider a temporal problem P with universal clauses $P(x)$ and $\neg Q(c)$ and a step clause $P(x) \Rightarrow \bigcirc Q(x)$. The clausification of P will then also contain a step clause $P(c) \Rightarrow \bigcirc Q(c)$. This additional step clause can be resolved with $\neg Q(c)$ using rule 4 of ordered fine-grained resolution with selection together with the identity substitution as unifier to obtain $P(c) \Rightarrow \bigcirc\bot$, which, using the conversion rules, gives us a new universal clause $\neg P(c)$. Another inference step with $P(x)$ results in a contradiction. Now, without a restriction on the substitution that can be used in subsumption, $P(x) \Rightarrow \bigcirc Q(x)$ would subsume $P(c) \Rightarrow \bigcirc Q(c)$. We could then try to derive a contradiction by resolving $P(x) \Rightarrow \bigcirc Q(x)$ with $\neg Q(c)$. However, the unifier of $Q(x)$ and $Q(c)$ maps the variable x, which also occurs in the left-hand side of the step clause to the constant c. Thus, an inference by rule 4 using these two premises is not possible and a contradiction can no longer be derived.

Definition 4.2.4. *Let $\mathcal{C}$ and $\mathcal{D}$ be initial, step or universal clauses. Then we say that $\mathcal{C}$* properly *subsumes $\mathcal{D}$, written $\mathcal{C} <_s \mathcal{D}$, if and only if $\mathcal{C}$ subsumes $\mathcal{D}$ but not vice-versa, i.e. $\mathcal{C} <_s \mathcal{D}$ if and only if $\mathcal{C} \leq_s \mathcal{D}$ and $\mathcal{D} \not\leq_s \mathcal{C}$.*

Remark 4.2.5. *For two formulae C and D with*

$$C = \bigwedge_{i=1}^{m_1} A_i^1(x_{1,i}) \wedge \bigwedge_{j=1}^{n_1} A_j^2(c_j) \wedge \bigwedge_{k=1}^{l_1} p_k \text{ and } D = \bigwedge_{i=1}^{m_2} B_i^1(x_{2,i}) \wedge \bigwedge_{j=1}^{n_2} B_j^2(d_j) \wedge \bigwedge_{k=1}^{l_2} q_k,$$

where A_i^1, A_j^2, B_i^1, B_j^2 are monadic predicates, $x_{1,i}$, $x_{2,i}$ are variables, c_j, d_j, are constants and p_k, q_k, are propositions, we write $C <_s D$ if and only if $\neg C <_s \neg D$ holds, where $\neg C$ and $\neg D$ are considered to be negative universal clauses.

We can now show the following two lemmata.

Lemma 4.2.6. *The relation $<_s$ is well-founded.*

Proof. Follows from the fact that the inclusion relation on multisets is well-founded. □

Lemma 4.2.7. *Let $\mathcal{C}$ and $\mathcal{D}$ be initial, step or universal clauses such that $\mathcal{C} \leq_s \mathcal{D}$. Then it holds for an initial clause $\mathcal{D}$ that the formula $[(\Box)\tilde{\forall}\mathcal{C}] \Rightarrow [\tilde{\forall}\mathcal{D}]$ is valid, and for a step or universal clause $\mathcal{D}$ that the formula $[\Box\tilde{\forall}\mathcal{C}] \Rightarrow [\Box\tilde{\forall}\mathcal{D}]$ is valid, where $\tilde{\forall}\mathcal{C}$ denotes the universal closure of $\mathcal{C}$.*

Proof. Follows from the definition of the truth-relation given in Figure 2.3. □

Having defined criteria for identifying tautological and subsumed clauses, we could now try to prove that for every refutation without redundancy elimination there exists a refutation with redundancy elimination. However, it turns out that such a correspondence is difficult to establish if the refutation contains applications of the duplicate literal elimination rule whose premise is subsumed.

For example, consider the step clause $\mathcal{D}_1 = P(x) \wedge P(x) \Rightarrow \bigcirc\bot$ which is subsumed by $\mathcal{C}_1 = P(x) \wedge P(y) \Rightarrow \bigcirc\bot$. From $\mathcal{D}_1$ we can derive $\mathcal{D}_2 = P(x) \Rightarrow \bigcirc\bot$ using the duplicate literal elimination rule. But our calculus does not contain a rule which allows us to derive a clause $\mathcal{C}_2$ from $\mathcal{C}_1$ that subsumes $\mathcal{D}_2$ nor does $\mathcal{C}_1$ itself subsume $\mathcal{D}_2$. Similarly, the universal clause $\mathcal{C}_3 = \neg P(x) \vee \neg P(y)$ would also subsume $\mathcal{D}_1$. But again, $\mathcal{C}_3$ does not subsume $\mathcal{D}_2$ nor can we derive a clause from $\mathcal{C}_3$ which subsumes $\mathcal{D}_2$ using the rules of our calculus.

4.2.3 Subsumption-Compatible Ordered Fine-Grained Resolution with Selection

In order to deal with these two cases, we need additional factoring rules. One of these rules has be applied on (at most) monadic negative universal clauses:

Definition 4.2.8. *A clause C is said to be a* (at most) monadic negative universal clause *if and only if the clause C is a negative universal clause $\neg A_1 \vee \ldots \vee \neg A_n$ such that every atom A_i ($1 \leq i \leq n$) is of arity equal to or less than 1.*

For example, the universal clauses $\neg p \vee \neg q$ and $\neg P(x) \vee \neg Q(y) \vee p$ are (at most) monadic negative universal clauses. Next, we extend our calculus by the following two rules:

- (Arbitrary) Factoring in left-hand sides of terminating step clauses:

  $$\frac{C \wedge A \wedge B \Rightarrow \bigcirc\bot}{(C \wedge A)\sigma \Rightarrow \bigcirc\bot},$$

  where σ is a most general unifier of the atoms A and B.

- (Arbitrary) Factoring in (at most) monadic negative universal clauses:

  $$\frac{\neg A_1 \vee \cdots \vee \neg A_n \vee \neg A_{n+1}}{(\neg A_1 \vee \cdots \vee \neg A_n)\sigma},$$

  where $\neg A_1 \vee \cdots \vee \neg A_n \vee \neg A_{n+1}$ is a (at most) monadic negative universal clause and σ is a most general unifier of the atoms A_n and A_{n+1}.

The calculus ordered fine-grained resolution with selection extended by the two rules introduced above will be called *subsumption-compatible* ordered fine-grained resolution with selection and will be denoted by $\mathfrak{I}^{S,\succ}_{FG,Sub}$. The fine-grained step resolution inference rules of $\mathfrak{I}^{S,\succ}_{FG,Sub}$ encompass the fine-grained step resolution inference rules of ordered fine-grained resolution with selection and are extended by the rules of (arbitrary) factoring in left-hand sides of terminating step clauses and (arbitrary) factoring in (at most) monadic negative universal clauses.

As we will see, the arbitrary factoring in left-hand sides of terminating step clauses rule will also play an important role in our completeness proof for the fair inference procedure that will be introduced in Chapter 5.

We still have to note that in contrast to the propositions stated in [66] we no longer require additional restrictions on the selection functions.

4.2.4 Subsumption Lemmata

We now have everything in place to show that subsumption-compatible ordered fine-grained resolution with selection allows the elimination of tautological and subsumed clauses.

First of all, we prove that tautologies either do not contribute or only prolong derivations of the empty clause.

Lemma 4.2.9. *Let C_1, C_2 be initial, universal or step clauses such that C_1 is a tautology. Then it holds that every resolvent C of C_1 and C_2 is either a tautology or subsumed by C_2. If both C_1 and C_2 are tautologies, then C is also a tautology.*

Proof. Let $\mathcal{C}_1 = (C_1 \Rightarrow \bigcirc)D_1$ (i.e. $\mathcal{C}_1 = C_1 \Rightarrow \bigcirc D_1$ if $\mathcal{C}_1$ is a step clause, and $\mathcal{C}_1 = D_1$ otherwise), $\mathcal{C}_2 = (C_2 \Rightarrow \bigcirc)D_2$ and $D_1 = E_1 \vee L \vee \neg L$. Additionally, let ν_1, ν_2 be variable renamings such that $var(\mathcal{C}_1\nu_1) \cap var(\mathcal{C}_2\nu_2) = \emptyset$ and let σ be the most general unifier used in the resolution inference, which is such that variables from $var(\mathcal{C}_1\nu_1) \cup var(\mathcal{C}_2\nu_2)$ are mapped into variables if $\mathcal{C}_1$ or $\mathcal{C}_2$ is a step clause. It is easy to see that if the literal which is resolved upon is different from $L\nu_1$ and $\neg L\nu_1$, then the clause $\mathcal{C}$ is again a tautology as it contains the two complementary literals $L\nu_1\sigma$ and $\neg L\nu_1\sigma$. We may therefore assume without loss of generality that the literal $L\nu_1$ in the clause $\mathcal{C}_1\nu_1$ will be resolved with a literal $\neg L'\nu_2$ in the clause $\mathcal{C}_2\nu_2$ (the remaining other case is similar). Thus, let $D_2 = E_2 \vee \neg L'$ and $\mathcal{C} = ((C_1\nu_1 \wedge C_2\nu_2)\sigma \Rightarrow \bigcirc)(E_1\nu_1 \vee \neg L\nu_1 \vee E_2\nu_2)\sigma$. As σ is a most general unifier of the atoms $L\nu_1$ and $L'\nu_2$, it holds that $L\nu_1\sigma = L'\nu_2\sigma$ and hence, $D_2\nu_2\sigma = (E_2 \vee \neg L')\nu_2\sigma \subseteq (E_1\nu_1 \vee \neg L\nu_1 \vee E_2\nu_2)\sigma$.

If $\mathcal{C}_2$ is an initial clause, we can infer that $\mathcal{C}$ is also an initial clause, and consequently, $\mathcal{C}_2 \leq_s \mathcal{C}$ holds. Then, if $\mathcal{C}_2$ is a universal clause and the clause $\mathcal{C}_1$ is an initial or universal clause, the clause $\mathcal{C}$ is also an initial or universal clause, respectively. We can conclude that $\mathcal{C}_2 \leq_s \mathcal{C}$ holds again. In the case where $\mathcal{C}_2$ is an universal clause and the clause $\mathcal{C}_1$ is a step clause, we have that $\mathcal{C}$ is a step clause, and also that $\mathcal{C}_2 \leq_s \mathcal{C}$ holds. Finally, if $\mathcal{C}_2$ is a step clause, then it follows that the clause $\mathcal{C}$ is a step clause and we obtain for every variable $x \in var(C_2) \cap var(D_2)$ that $x\nu_2\sigma$ is a variable. We can thus infer that $\mathcal{C}_2 \leq_s \mathcal{C}$ holds as well.

We still have to analyse the case where $\mathcal{C}_1$ and $\mathcal{C}_2$ are tautologies. One can then infer that the clause $\mathcal{C}$ contains a pair of complementary literals, even if $\mathcal{C}_2$ is a universal clause and $\mathcal{C}$ a step clause as tautological universal clauses cannot subsume the left-hand sides of step clauses. □

Lemma 4.2.10. *Let $\mathcal{C}_1$ be a tautology. Then it holds that every factor $\mathcal{C}$ of $\mathcal{C}_1$ is a tautology.*

Proof. Let $\mathcal{C}_1 = (C_1 \Rightarrow \bigcirc)D_1$ and $D_1 = D \vee L \vee \neg L$. Additionally, let A, $B \in D_1$ be the atoms which are factored upon and let σ be a most general unifier of the atoms A and B. Additionally, if $A\sigma \neq L\sigma \neq B\sigma$ holds, we obtain again that $\mathcal{C}$ contains two complementary literals $L\sigma$, $\neg L\sigma$ and $\mathcal{C}$ is therefore a tautology. Finally, in the case where $A\sigma = L\sigma = B\sigma$, we can conclude that $\mathcal{C}$ contains a pair of complementary literals $A\sigma$ and $\neg A\sigma$ as factoring only removes positive occurrences of literals, which also implies that $\mathcal{C}$ is a tautology. □

The next propositions establish the subsumption lemmata for the step resolution inference rules of $\mathfrak{I}^{S,\succ}_{FG,Sub}$, i.e. for every inference rule of $\mathfrak{I}^{S,\succ}_{FG,Sub}$, except the eventuality resolution rules, and clauses $\tilde{\mathcal{C}}_1, \mathcal{C}_1, \tilde{\mathcal{C}}_2, \mathcal{C}_2$ with $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}_1$, $\mathcal{C}_2 \leq_s \tilde{\mathcal{C}}_2$ and clause $\tilde{\mathcal{C}}$ obtained through an inference between $\tilde{\mathcal{C}}_1$ and $\tilde{\mathcal{C}}_2$, we show that either

(i) $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$ or

(ii) $\mathcal{C}_2 \leq_s \tilde{\mathcal{C}}$ holds, or

(iii) there exists an inference between $\mathcal{C}_1$ and $\mathcal{C}_2$ resulting in a clause $\mathcal{C}$ with $\mathcal{C} \leq_s \tilde{\mathcal{C}}$.

Lemmata 4.2.22 and 4.2.23 will regroup the results established by the following propositions.

Lemma 4.2.11. *Let $\tilde{C}_1$, $\tilde{C}_2$, C_1, C_2 be universal clauses such that $C_1 \leq_s \tilde{C}_1$ and $C_2 \leq_s \tilde{C}_2$. Furthermore, let $\tilde{C}$ be a resolvent of $\tilde{C}_1$ and $\tilde{C}_2$ by ordered resolution with selection between two universal clauses. Then one of the following statements holds:*

(a) $C_1 \leq_s \tilde{C}$, or

(b) $C_2 \leq_s \tilde{C}$, or

(c) there exists a resolvent C of C_1 and C_2 by ordered resolution with selection such that $C \leq_s \tilde{C}$.

Proof. Similar to the first-order case. □

Lemma 4.2.12. *Let C and $\tilde{C}$ be universal clauses such that $C \leq_s \tilde{C}$. Furthermore, let $\tilde{\mathcal{D}}$ be the result of applying the ordered positive factoring with selection rule on $\tilde{C}$. Then one of the following statements holds:*

(a) $C \leq_s \tilde{\mathcal{D}}$, or

(b) there exists a universal clause $\mathcal{D}$ obtained through an application of the ordered positive factoring with selection rule on C such that $\mathcal{D} \leq_s \tilde{\mathcal{D}}$.

Proof. Similar to the first-order case. □

Lemma 4.2.13. *Let $\tilde{C}_1$, $\tilde{C}_2$ be initial clauses and C_1, C_2 be initial or universal clauses such that $C_1 \leq_s \tilde{C}_1$ and $C_2 \leq_s \tilde{C}_2$. Furthermore, let $\tilde{C}$ be a resolvent of $\tilde{C}_1$ and $\tilde{C}_2$ by ordered resolution with selection between two initial clauses. Then one of the following statements holds:*

(a) $C_1 \leq_s \tilde{C}$, or

(b) $C_2 \leq_s \tilde{C}$, or

(c) there exists a resolvent C of C_1 and C_2 by ordered resolution with selection such that $C \leq_s \tilde{C}$.

Proof. Similar to the first-order case by considering that initial and universal clauses can be identified with first-order clauses. The only difference is that if we resolve an initial clause with a universal clause or an initial clause, then we obtain an initial clause. A resolution inference between two universal clauses yields a universal clause again. □

Lemma 4.2.14. *Let $\tilde{C}_1$ be an initial, C_1 be an initial or universal and C_2, $\tilde{C}_2$ be universal clauses such that $C_1 \leq_s \tilde{C}_1$ and $C_2 \leq_s \tilde{C}_2$. Furthermore, let $\tilde{C}$ be a resolvent of $\tilde{C}_1$ and $\tilde{C}_2$ by ordered resolution with selection between an initial and a universal clause. Then one of the following statements holds:*

(a) $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$, *or*

(b) $\mathcal{C}_2 \leq_s \tilde{\mathcal{C}}$, *or*

(c) there exists a resolvent $\mathcal{C}$ *of* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *by ordered resolution with selection such that* $\mathcal{C} \leq_s \tilde{\mathcal{C}}$.

Proof. Similar to the first-order case. □

Lemma 4.2.15. *Let* $\mathcal{C}$ *and* $\tilde{\mathcal{C}}$ *be initial or universal clauses such that* $\mathcal{C} \leq_s \tilde{\mathcal{C}}$. *Furthermore, let* $\tilde{\mathcal{D}}$ *be the result of applying the ordered positive factoring with selection rule on* $\tilde{\mathcal{C}}$. *Then one of the following statements holds:*

(a) $\mathcal{C} \leq_s \tilde{\mathcal{D}}$, *or*

(b) there exists an initial or universal clause $\mathcal{D}$ *obtained through an application of the ordered positive factoring with selection rule on* $\mathcal{C}$ *such that* $\mathcal{D} \leq_s \tilde{\mathcal{D}}$.

Proof. Similar to the first-order case. □

Lemma 4.2.16. *Let* $\tilde{\mathcal{C}}_1$, $\tilde{\mathcal{C}}_2$ *be step clauses and let* $\mathcal{C}_1$, $\mathcal{C}_2$ *be universal or step clauses such that* $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}_1$ *and* $\mathcal{C}_2 \leq_s \tilde{\mathcal{C}}_2$. *Furthermore, let* $\tilde{\mathcal{C}}$ *be a resolvent of* $\tilde{\mathcal{C}}_1$ *and* $\tilde{\mathcal{C}}_2$ *by ordered fine-grained step resolution with selection. Then one of the following statements holds:*

(a) $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$, *or*

(b) $\mathcal{C}_2 \leq_s \tilde{\mathcal{C}}$, *or*

(c) there exists a resolvent $\mathcal{C}$ *of* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *by ordered fine-grained step resolution with selection such that* $\mathcal{C} \leq_s \tilde{\mathcal{C}}$.

Proof. First of all, we assume without loss of generality that $var(\tilde{\mathcal{C}}_1) \cap var(\tilde{\mathcal{C}}_2) = \emptyset$ (otherwise, variable renamings need to be applied accordingly). Let $\tilde{\mathcal{C}}_1 = \tilde{C}_1 \Rightarrow \bigcirc\tilde{D}_1$, $\tilde{\mathcal{C}}_2 = \tilde{C}_2 \Rightarrow \bigcirc\tilde{D}_2$; $\tilde{D}_1 = \tilde{E}_1 \vee \tilde{A}$, $\tilde{D}_2 = \tilde{E}_2 \vee \neg\tilde{B}$ and let $\mathcal{C} = (\tilde{C}_1 \wedge \tilde{C}_2)\tilde{\sigma} \Rightarrow \bigcirc(\tilde{E}_1 \vee \tilde{E}_2)\tilde{\sigma}$, where $\tilde{\sigma}$ is a most general unifier of the atoms $\tilde{A}$ and $\tilde{B}$ such that variables from $var(\tilde{C}_1) \cup var(\tilde{C}_2)$ are mapped into variables. We first of all observe that if $\mathcal{C}_1$ or $\mathcal{C}_2$ are universal clauses such that $\mathcal{C}_1 \leq_s \neg\tilde{C}_1$ or $\mathcal{C}_2 \leq_s \neg\tilde{C}_2$ holds, then we can infer that $\mathcal{C}_1 \leq_s \neg(\tilde{C}_1 \wedge \tilde{C}_2)\tilde{\sigma}$ or $\mathcal{C}_2 \leq_s \neg(\tilde{C}_1 \wedge \tilde{C}_2)\tilde{\sigma}$ holds, and thus $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$ or $\mathcal{C}_2 \leq_s \tilde{\mathcal{C}}$. Without loss of generality we may hence assume that $\mathcal{C}_1 \leq_s \tilde{D}_1$ holds if $\mathcal{C}_1$ is a universal clause and $\mathcal{C}_2 \leq_s \tilde{D}_2$ if $\mathcal{C}_2$ is a universal clause.

Now, let $\mathcal{C}_1 = (C_1 \Rightarrow \bigcirc)D_1$, $\mathcal{C}_2 = (C_2 \Rightarrow \bigcirc)D_2$ and η_1, η_2 be substitutions with $C_1\eta_1 \subseteq \tilde{C}_1$, $D_1\eta_1 \subseteq \tilde{D}_1$, $C_2\eta_2 \subseteq \tilde{C}_2$ and $D_2\eta_2 \subseteq \tilde{D}_2$ such that η_1 maps variables from $var(C_1) \cap var(D_1)$ into variables if $\mathcal{C}_1$ is a step clause and η_2 maps variables from $var(C_2) \cap var(D_2)$ into variables if $\mathcal{C}_2$ is a step clause. Then, if all the literals A with $A\eta_1 = \tilde{A}$ occur together less often in D_1 than the literal $\tilde{A}$ in $\tilde{D}_1$, we can infer that $D_1\eta_1 \subseteq \tilde{E}_1$ and thus, $C_1\eta_1\tilde{\sigma} \subseteq (\tilde{C}_1 \wedge \tilde{C}_2)\tilde{\sigma}$ and $D_1\eta_1\tilde{\sigma} \subseteq (\tilde{E}_1 \vee \tilde{E}_2)\tilde{\sigma}$. Additionally, if $\mathcal{C}_1$ is a step clause, then it holds for a variable $x \in var(C_1) \cap var(D_1)$ that $x\eta_1\tilde{\sigma} = y\tilde{\sigma} = z$ for variables $y, z \in X$ as $y \in var(\tilde{C}_1)$. We can consequently infer that $C_1 \leq_s \tilde{C}$ holds.

Similarly, if all the literals $\neg B$ with $\neg B\eta_2 = \tilde{B}$ occur together less often in D_2 than the literal $\neg\tilde{B}$ in $\tilde{D}_2$, we obtain $C_2 \leq_s \tilde{C}$.

Otherwise, we consider the case where $C_1\eta_1 \subseteq \tilde{C}_1$, $D_1 = E_1 \vee A$, $E_1\eta_1 \subseteq \tilde{E}_1$, $A\eta_1 = \tilde{A}$ and $C_2\eta_2 \subseteq \tilde{C}_2$, $D_2 = E_2 \vee \neg B$, $E_2\eta_2 \subseteq \tilde{E}_2$, $\neg B\eta_2 = \neg\tilde{B}$. Let ν_1 and ν_2 be variable renamings such that $var(C_1\nu_1) \cap var(C_2\nu_2) = \emptyset$. We need to show that $C_1\nu_1$ can be resolved with $C_2\nu_2$. Due to the instance compatibility of the selection function, we can assume that the literal $\neg B$ is selected in the clause D_2 if the literal $\neg\tilde{B}$ is selected in the clause $\tilde{D}_2$. It also follows that $A\nu_1\nu_1^{-1}\eta_1\tilde{\sigma} = A\eta_1\tilde{\sigma} = \tilde{A}\tilde{\sigma} = \tilde{B}\tilde{\sigma} = B\eta_2\tilde{\sigma} = B\nu_2\nu_2^{-1}\eta_2\tilde{\sigma}$, i.e. the atoms $A\nu_1$ and $B\nu_2$ are unifiable by a most general unifier σ with $dom(\sigma) \cup (codom(\sigma) \cap X) \subseteq var(A\nu_1) \cup var(B\nu_2)$ (see Theorem 3.4.23). For the substitution $\eta = \nu_1^{-1}\eta_1 \cup \nu_2^{-1}\eta_2$ we have $C_1\nu_1\eta \subseteq \tilde{C}_1$, $D_1\nu_1 = E_1\nu_1 \vee A\nu_1$, $E_1\nu_1\eta \subseteq \tilde{E}_1$, $A\nu_1\eta = \tilde{A}$ and $C_2\nu_2\eta \subseteq \tilde{C}_2$, $D_2\nu_2 = E_2\nu_2 \vee \neg B\nu_2$, $E_2\nu_2\eta \subseteq \tilde{E}_2$, $B\nu_2\eta = \tilde{B}$. Moreover, it holds that η maps variables from $var(C_1\nu_1) \cap var(D_1\nu_1)$ into variables if C_1 is a step clause, and that η maps variables from $var(C_2\nu_2) \cap var(D_2\nu_2)$ into variables if C_2 is a step clause (see Lemma 3.4.16).

Then, as $A\nu_1\eta\tilde{\sigma} = \tilde{A}\tilde{\sigma} = \tilde{B}\tilde{\sigma} = B\nu_2\eta\tilde{\sigma}$ holds, the substitution $\eta\tilde{\sigma}$ is a unifier of the atoms $A\nu_1$ and $B\nu_2$. Thus, let φ be a substitution such that $\sigma\varphi = \eta\tilde{\sigma}$. Now, if C_1 is a step clause, let $x \in var(C_1\nu_1)$. If $x \notin var(D_1\nu_1)$, then we have $x \notin dom(\sigma)$ as also $x \notin var(D_2\nu_2)$ holds, i.e. $\sigma(x) = x$. Otherwise, $x \in var(D_1\nu_1)$ and we obtain $\eta(x) \in var(\tilde{C}_1)$, which implies that $x\sigma\varphi = x\eta\tilde{\sigma} \in X$ and hence, $\sigma(x) \in X$. Analogously,.one can show that σ maps variables from $var(C_2\nu_2)$ into variables if C_2 is a step clause.

It follows from the properties of the selection function and of the atom ordering that the literal $A\nu_1$ is eligible in $D_1\nu_1$ for σ and the literal $\neg B\nu_2$ is eligible in $D_2\nu_2$ for σ. Consequently, there exists a resolvent $C = ((C_1\nu_1 \wedge C_2\nu_2)\sigma \Rightarrow \bigcirc)(E_1\nu_1 \vee E_2\nu_2)\sigma$ such that $(C_1\nu_1 \wedge C_2\nu_2)\sigma\varphi \subseteq (\tilde{C}_1 \wedge \tilde{C}_2)\tilde{\sigma}$ and $(E_1\nu_1 \vee E_2\nu_2)\sigma\varphi \subseteq (\tilde{E}_1 \vee \tilde{E}_2)\tilde{\sigma}$. We still have to show that $C \leq_s \tilde{C}$ holds. If C_1 and C_2 are universal clauses, we can immediately conclude that $C \leq_s \tilde{C}$ holds. In the case where C_1 or C_2 is a step clause, let $x \in var((C_1\nu_1 \wedge C_2\nu_2)\sigma) \cap var((E_1\nu_1 \vee E_2\nu_2)\sigma)$. Thus, there exists variables $y \in var(C_1\nu_1) \cup var(C_2\nu_2)$, $z \in var(E_1\nu_1) \cup var(E_2\nu_2)$ with $\sigma(y) = x = \sigma(z)$.

If $x \neq y$, we can infer that $y \in var(D_1\nu_1) \cup var(D_2\nu_2)$ as $y \in dom(\sigma)$. Hence, it holds that $y \in (var(C_1\nu_1) \cap var(D_1\nu_1)) \cup (var(C_2\nu_2) \cap var(D_2\nu_2))$ and $\varphi(x) = y\sigma\varphi = y\eta\tilde{\sigma}$. Furthermore, as $\eta(y) \in X$, we have $\eta(y) \in var(\tilde{C}_1) \cup var(\tilde{C}_2)$, which implies that $\varphi(x) = y\eta\tilde{\sigma} \in X$.

In the case where $x = y = z$, we obtain that $x \in (var(C_1\nu_1) \cap var(D_1\nu_1)) \cup (var(C_2\nu_2) \cap var(D_2\nu_2))$, and thus similarly, $\varphi(x) = x\sigma\varphi = x\eta\tilde{\sigma} \in X$. Finally, if $x = y$ and $x \neq z$, we get $x \in var(C_1\nu_1) \cup var(C_2\nu_2)$ and $x \in (codom(\sigma) \cap X) \subseteq var(D_1\nu_1) \cup var(D_2\nu_2)$. Again, it holds that $\varphi(x) = x\sigma\varphi = x\eta\tilde{\sigma} \in X$.

We can conclude that $C \leq_s \tilde{C}$ holds. □

Lemma 4.2.17. *Let $\tilde{C}_1 = \tilde{C}_1 \Rightarrow \bigcirc\tilde{D}_1$ be a step clause, let C_1 be a universal or a step clause and let C_2, $\tilde{C}_2$ be universal clauses such that $C_1 \leq_s \tilde{C}_1$ and $C_2 \leq_s \tilde{C}_2$. Furthermore,* *let $\tilde{C}$ be a resolvent of $\tilde{C}_1$ and $\tilde{C}_2$ by ordered fine-grained step resolution with selection. Then one of the following statements holds:*

(a) $C_1 \leq_s \tilde{C}$, or

(b) $C_2 \leq_s \tilde{C}$, or

(c) there exists a resolvent C of C_1 and C_2 by ordered fine-grained step resolution with selection such that $C \leq_s \tilde{C}$.

Additionally, if $C_2 = \tilde{C}_2$, then the statement (a) or (c) holds.

Proof. Similar to the proof of Lemma 4.2.16. If $C_2 = \tilde{C}_2$, then one can show that the statement (a) or (c) holds (in addition to statement (b) being potentially satisfied). □

Lemma 4.2.18. *Let $\tilde{C}_1 = \tilde{C}_1 \Rightarrow \bigcirc\tilde{D}_1$ be a step clause and C_1 be a universal or a step clause such that $C_1 \leq_s \tilde{C}_1$. Furthermore, let $\tilde{C}$ be the result of applying the ordered fine-grained positive step factoring rule on $\tilde{C}_1$. Then one of the following statements holds:*

(a) $C_1 \leq_s \tilde{C}$, or

(b) there exists a universal or step clause C obtained through an application of the ordered fine-grained positive factoring rule on C_1 such that $C \leq_s \tilde{C}$.

Proof. Let $C_1 = (C_1 \Rightarrow \bigcirc)D_1$, $\tilde{D}_1 = \tilde{D} \vee \tilde{A} \vee \tilde{B}$ and $\tilde{\sigma}$ be a most general unifier of the atoms $\tilde{A}$ and $\tilde{B}$ such that $\tilde{\sigma}$ does not map variables from $\tilde{C}_1$ into a constant or a functional term and such that $\tilde{A}$ is eligible in $\tilde{D} \vee \tilde{A} \vee \tilde{B}$ for $\tilde{\sigma}$. Hence, we obtain $\tilde{C} = \tilde{C}_1\tilde{\sigma} \Rightarrow \bigcirc(\tilde{D} \vee \tilde{A})\tilde{\sigma}$.

First of all, if C_1 is a universal clause and $C_1 \leq_s \tilde{C}_1$, then as $\tilde{C}_1 \leq_s \tilde{C}_1\tilde{\sigma}$ clearly holds, we obtain $C_1 \leq_s \tilde{C}_1\tilde{\sigma}$ and thus, $C_1 \leq_s \tilde{C}$. We may now assume that $C_1 \leq_s \tilde{D}_1$ holds if C_1 is a universal clause. Let η be the substitution such that $C_1\eta \subseteq \tilde{C}_1$, $D_1\eta \subseteq \tilde{D}_1$ and such that η maps variables from $var(C_1) \cap var(D_1)$ into variables if C_1 is a step clause.

Then, if the clause D_1 does not contain a pair of atoms A and B with $A\eta = \tilde{A}$ and $B\eta = \tilde{B}$, we obtain that $C_1\eta\tilde{\sigma} \subseteq \tilde{C}_1\tilde{\sigma}$ and $D_1\eta\tilde{\sigma} \subseteq (\tilde{D} \vee \tilde{A})\tilde{\sigma}$. If C_1 is a universal clause, we have $C_1 \leq_s \tilde{C}$. Otherwise, C_1 is a step clause, and let $x \in var(C_1) \cap var(D_1)$. We obtain $\eta(x) \in X$, i.e. $\eta(x) \in var(\tilde{C}_1)$ and thus, $x\eta\tilde{\sigma} \in X$, which implies that $C_1 \leq_s \tilde{C}$ holds.

Otherwise, let A and B be atoms in the clause D_1 with $A\eta = \tilde{A}$, $B\eta = \tilde{B}$ and $D_1 = D \vee A \vee B$. We need to show that a factoring step on C_1 is possible. We can first of all infer that $A\eta\tilde{\sigma} = \tilde{A}\tilde{\sigma} = \tilde{B}\tilde{\sigma} = B\eta\tilde{\sigma}$. Thus, there exists a most general unifier of the atoms A and B with $dom(\sigma) \cup (codom(\sigma) \cap X) \subseteq var(A) \cup var(B) \subseteq var(D_1)$ (see Theorem 3.4.23). Let φ be a substitution with $\sigma\varphi = \eta\tilde{\sigma}$, which implies that $C_1\sigma\varphi \subseteq \tilde{C}_1\tilde{\sigma}$ and $D_1\sigma\varphi \subseteq \tilde{D}_1\tilde{\sigma}$. It then follows from the properties of the atom ordering that the literal A is eligible in $D \vee A \vee B$ for σ. If C_1 is a universal clause, we can infer that there exists a universal clause $C = (D \vee A)\sigma$ obtained through an application of the ordered positive factoring with selection rule on C_1. As $D_1\sigma\varphi \subseteq \tilde{D}_1\tilde{\sigma}$ holds, we have $C \leq_s \tilde{C}$.

Then, in the case where $\mathcal{C}_1$ is a step clause, let $x \in var(C_1)$ such that $x \in dom(\sigma)$. It follows that $x \in var(D_1)$ and $x\sigma\varphi = x\eta\tilde{\sigma} \in X$, which implies that $\varphi(x) \in X$. We can infer there exists a step clause $\mathcal{C} = C_1\sigma \Rightarrow \bigcirc(D \vee A)\sigma$ obtained through an application of the ordered positive factoring with selection rule on $\mathcal{C}_1$.

Now, let $x \in var(C_1\sigma) \cap var((D \vee A)\sigma)$. Thus, there exists variables $y \in var(C_1)$ and $z \in var(D \vee A)$ with $\sigma(y) = x = \sigma(z)$. If $x \neq y$, we can infer that $y \in var(D \vee A \vee B) = var(D_1)$ as $y \in dom(\sigma)$. Furthermore, it holds that $\varphi(x) = y\sigma\varphi = y\eta\tilde{\sigma}$. Hence, as $\eta(y) \in X$, we have $\eta(y) \in var(\tilde{C}_1)$, which implies that $\varphi(x) = x\eta\tilde{\sigma} \in X$.

In the case where $x = y = z$, we obtain that $x \in var(C_1) \cap var(D_1)$, and thus similarly, $\varphi(x) = x\sigma\varphi = x\eta\tilde{\sigma} \in X$. Finally, if $x = y$ and $x \neq z$, we have that $x \in var(C_1)$ and $x \in (codom(\sigma) \cap X) \subseteq var(D_1)$. Again, it holds that $\varphi(x) = x\sigma\varphi = x\eta\tilde{\sigma} \in X$.

Finally, we can conclude that $\mathcal{C} \leq_s \tilde{\mathcal{C}}$ holds. □

Lemma 4.2.19. *Let $\tilde{\mathcal{C}}_1$ be a step clause and $\mathcal{C}_1$ be a universal or a step clause such that $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}_1$. Furthermore, let $\tilde{\mathcal{C}}$ be the result of applying the clause conversion rule on $\tilde{\mathcal{C}}_1$. Then one of the following statements holds:*

(a) $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$, or

(b) there exists a universal clause $\mathcal{C}$ obtained through an application of the clause conversion rule on $\mathcal{C}_1$ such that $\mathcal{C} \leq_s \tilde{\mathcal{C}}$.

Proof. Let $\tilde{\mathcal{C}} = \tilde{C} \Rightarrow \bigcirc\bot$, $\mathcal{C}_1 = (\neg)C(\Rightarrow \bigcirc\bot)$ and $\tilde{\mathcal{C}} \doteq \neg\tilde{C}$. If $\mathcal{C}_1$ is a universal clause and $\mathcal{C}_1 = \bot$, then $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$ clearly holds. Otherwise, we have $\mathcal{C}_1 = \neg C \leq_s \neg\tilde{C}$ and thus, $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$. Finally, in the case where $\mathcal{C}_1$ is a step clause, we also obtain $\neg C \leq_s \neg\tilde{C}$. An application of the clause conversion rule on the step clause $\mathcal{C}_1$ then yields the clause $\mathcal{C} = \neg C$, i.e. $\mathcal{C} \leq_s \tilde{\mathcal{C}}$ holds. □

Lemma 4.2.20. *Let $\tilde{\mathcal{C}}_1 = \tilde{C}_1 \Rightarrow \bigcirc\bot$ be a terminating step clause and $\mathcal{C}_1$ be a universal or a step clause such that $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}_1$. Furthermore, let $\tilde{\mathcal{C}}$ be the result of applying the factoring in left-hand sides of terminating step clauses rule on $\tilde{\mathcal{C}}_1$. Then one of the following statements holds:*

(a) $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$, or

(b) there exists a universal or a step clause $\mathcal{C}$ obtained through an application of the factoring in at most monadic negative universal clauses or in left-hand sides of terminating step clauses rule on $\mathcal{C}_1$ such that $\mathcal{C} \leq_s \tilde{\mathcal{C}}$.

Proof. Let $\tilde{C}_1 = \tilde{C} \wedge \tilde{A} \wedge \tilde{B}$, $\mathcal{C}_1 = (\neg)C_1(\Rightarrow \bigcirc\bot)$ and $\tilde{\mathcal{C}} = (\tilde{C} \wedge \tilde{A})\tilde{\sigma} \Rightarrow \bigcirc\bot$, where $\tilde{\sigma}$ is a most general unifier of the atoms $\tilde{A}$ and $\tilde{B}$. Moreover, let η be a substitution with $C_1\eta \subseteq (\neg)\tilde{C}_1$.

If C_1 does not contain a pair of literals A_1, A_2 with $A_1\eta = \tilde{A}$ and $A_2\eta = \tilde{B}$, we obtain $C_1\eta \subseteq (\neg)(\tilde{C} \wedge \tilde{A})$. Hence, $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$ holds.

Otherwise, C_1 contains a pair of literals A_1, A_2 with $A_1\eta = \tilde{A}$ and $A_2\eta = \tilde{B}$, i.e. $C_1 = C \wedge A_1 \wedge A_2$. Thus, as the atoms A_1 and A_2 are unifiable with $A_1\eta\tilde{\sigma} = A_2\eta\tilde{\sigma}$, there exists a most general unifier σ of A_1 and A_2, together with a substitution φ such that $\sigma\varphi = \eta\tilde{\sigma}$. If $\mathcal{C}_1$ is a universal clause, we observe that it only contains negative literals with at most one free variable as $\mathcal{C}_1 \subseteq \neg\tilde{C}_1$ holds. We can thus apply the factoring in at most monadic negative universal clause rule on $\mathcal{C}_1$ and obtain the universal clause $\mathcal{C} = \neg C \vee \neg A_1$. As $\mathcal{C}\sigma\varphi \subseteq \neg(\tilde{C} \wedge \tilde{A})\tilde{\sigma}$, we obtain $\mathcal{C} \leq_s \tilde{\mathcal{C}}$.

Finally, if $\mathcal{C}_1$ is a step clause, we can apply the factoring in left-hand sides of terminating step clauses rule and derive the terminating step clause $\mathcal{C} = (C \wedge A_1)\sigma \Rightarrow \bigcirc\bot$. We obtain $(C \wedge A_1)\sigma\varphi \subseteq (\tilde{C} \wedge \tilde{A})\tilde{\sigma}$ and we can infer that $\mathcal{C} \leq_s \tilde{\mathcal{C}}$ holds. □

Lemma 4.2.21. *Let $\tilde{\mathcal{C}}_1 = \neg A_1 \vee \ldots \vee \neg A_n \vee \neg A_{n+1}$ be an at most monadic negative universal clause and $\mathcal{C}_1$ be a universal clause such that $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}_1$. Furthermore, let $\tilde{\mathcal{C}}$ be the result of applying the factoring in at most monadic negative universal clauses rule on $\tilde{\mathcal{C}}_1$. Then one of the following statements holds:*

(a) $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$, or

(b) there exists a universal clause $\mathcal{C}$ obtained through an application of the factoring in at most monadic negative universal clauses rule on $\mathcal{C}_1$ such that $\mathcal{C} \leq_s \tilde{\mathcal{C}}$.

Proof. Let $\tilde{\mathcal{C}}_1 = \neg\tilde{A}_1 \vee \ldots \vee \neg\tilde{A}_n \vee \neg\tilde{A}_{n+1}$ and $\tilde{\mathcal{C}} = (\neg\tilde{A}_1 \vee \ldots \vee \neg\tilde{A}_n)\tilde{\sigma}$, where $\tilde{\sigma}$ is a most general unifier of the atoms $\tilde{A}_n$ and $\tilde{A}_{n+1}$. Additionally, let η be a substitution with $\mathcal{C}_1\eta \subseteq \tilde{\mathcal{C}}_1$. If $\mathcal{C}_1$ does not contain a pair of literals A_n, A_{n+1} with $A_n\eta = \tilde{A}_n$ and $A_{n+1}\eta = \tilde{A}_{n+1}$, we can infer that $\mathcal{C}_1\eta \subseteq \neg\tilde{A}_1 \vee \ldots \vee \neg\tilde{A}_n$. Thus, $\mathcal{C}_1\eta\tilde{\sigma} \subseteq \tilde{\mathcal{C}}$, i.e. $\mathcal{C}_1 \leq_s \tilde{\mathcal{C}}$ holds.

Otherwise, $\mathcal{C}_1$ contains a pair of literals A_n, A_{n+1} with $A_n\eta = \tilde{A}_n$ and $A_{n+1}\eta = \tilde{A}_{n+1}$, i.e. $\mathcal{C}_1 = C \vee \neg A_n \vee \neg A_{n+1}$ for an at most monadic negative universal clause C. Then, as $A_n\eta\tilde{\sigma} = \tilde{A}_n\tilde{\sigma} = \tilde{A}_{n+1}\tilde{\sigma} = A_{n+1}\eta\tilde{\sigma}$, there exists a most general unifier σ of the atoms A_n and A_{n+1}. We can thus apply the factoring in at most monadic negative universal clauses rule on $\mathcal{C}_1$ and obtain the universal clause $\mathcal{C} = (C \vee \neg A_n)\sigma$. Let φ be a substitution with $\sigma\varphi = \eta\tilde{\sigma}$. Then, it is easy to see that $\mathcal{C}\varphi \subseteq \tilde{\mathcal{C}}$ holds, i.e. we have $\mathcal{C} \leq_s \tilde{\mathcal{C}}$. □

The next two lemmata regroup the previous results. First, we show that if subsumption is limited to step clauses in the derivation of a step clause $\tilde{\mathcal{C}}$ which does not contain an application of the clause conversion rule, then one can derive a step clause $\mathcal{C}$ from the subsuming step clauses and the same universal clauses such that $\mathcal{C} \leq_s \tilde{\mathcal{C}}$.

Lemma 4.2.22. *Let $\mathcal{U}$ be a set of universal clauses and let $\mathcal{N}$, $\tilde{\mathcal{N}}$ be sets of step clauses such that $\mathcal{N} \leq_s \tilde{\mathcal{N}}$. Additionally, let $\tilde{\Delta}$ be a derivation of a step clause $\tilde{\mathcal{C}}$ from clauses in $\mathcal{U} \cup \tilde{\mathcal{N}}$ by the fine-grained step resolution inference rules of either ordered fine-grained resolution with selection or subsumption-compatible ordered fine-grained resolution with selection, but without applying the clause conversion rule.*

Then there exists a derivation Δ of a step clause $\mathcal{C}$ by subsumption-compatible ordered fine-grained resolution with selection from clauses in $\mathcal{U} \cup \mathcal{N}$ such that $\mathcal{C} \leq_s \tilde{\mathcal{C}}$.

Proof. Lemma 4.2.22 is shown by induction on the length of the derivation $\tilde{\Delta}$. For the base case we assume that $\tilde{C}$ is a step clause in $\tilde{\mathcal{N}}$. Then there exists a step clause C in $\mathcal{N}$ with $C \leq_s \tilde{C}$.

For the induction step we consider a step clause $\tilde{C}$ that is derived by one of the rules of (subsumption-compatible) ordered fine-grained step resolution excluding the clause conversion rule from premises $\tilde{C}_1$ and $\tilde{C}_2$, which are either elements of $\mathcal{U} \cup \tilde{\mathcal{N}}$ or previously derived clauses. It follows that either $\tilde{C}_1$ or $\tilde{C}_2$ (or both) are step clauses. By the induction hypothesis there are clauses C_1 and C_2 with $C_1 \leq_s \tilde{C}_1$ and $C_2 \leq_s \tilde{C}_2$ which are either elements of $\mathcal{U} \cup \mathcal{N}$ or previously derived such that C_1 is a step clause if $\tilde{C}_1$ is a step clause, and C_2 is a step clause if $\tilde{C}_2$ is a step clause. Thus, by using Lemmata 4.2.16, 4.2.17, 4.2.18, and 4.2.20, either $C_1 \leq_s \tilde{C}$, $C_2 \leq_s \tilde{C}$ or we can derive a step clause C with $C \leq_s \tilde{C}$ from C_1 and C_2. □

The next proposition is a more general version of Lemma 4.2.22. It allows for arbitrary subsumption and it considers all the fine-grained step resolution inference rules of $\mathfrak{I}^{S,\succ}_{FG,Sub}$.

Lemma 4.2.23. *Let $\mathcal{N}$ and $\tilde{\mathcal{N}}$ be sets of initial, universal clauses or step clauses such that $\mathcal{N} \leq_s \tilde{\mathcal{N}}$. Additionally, let $\tilde{\Delta}$ be a derivation of a clause $\tilde{C}$ from clauses in $\tilde{\mathcal{N}}$ by the fine-grained step resolution inference rules of either ordered fine-grained resolution with selection or subsumption-compatible ordered fine-grained resolution with selection.*

Then there exists a derivation Δ of a clause C by subsumption-compatible ordered fine-grained resolution with selection from clauses in $\mathcal{N}$ such that $C \leq_s \tilde{C}$. It also holds that the clause conversion rule is only applied in the derivation Δ if it has been applied in order to obtain the derivation $\tilde{\Delta}$.

The previous statement still holds if $\mathcal{N} \leq_s \tilde{\mathcal{N}} \setminus \mathrm{taut}(\tilde{\mathcal{N}})$ and $\tilde{C}$ is not a tautology.

Proof. Let $\tilde{\Delta} = \tilde{\mathcal{D}}_1, \ldots, \tilde{\mathcal{D}}_{n-1}, \tilde{C}(= \tilde{\mathcal{D}}_n)$. If $\mathcal{N} \leq_s \tilde{\mathcal{N}}$, then one can show the existence of the derivation Δ by induction on the length of the derivation $\tilde{\Delta}$ in analogy to Lemma 4.2.22 by using Lemmata 4.2.11, 4.2.12, 4.2.13, 4.2.14, 4.2.15, 4.2.16, 4.2.17, 4.2.18, 4.2.19, 4.2.20, and 4.2.21.

In the case where $\mathcal{N} \leq_s \tilde{\mathcal{N}} \setminus \mathrm{taut}(\tilde{\mathcal{N}})$ holds, it can be shown inductively for every clause $\tilde{\mathcal{D}}_i$ $(1 \leq i \leq n)$ which is not a tautology that there exists a derivation Δ of a clause $\mathcal{D}_i$ with $\mathcal{D}_i \leq_s \tilde{\mathcal{D}}_i$ by subsumption-compatible ordered fine-grained resolution with selection from clauses in $\mathcal{N} \setminus \mathrm{taut}(\tilde{\mathcal{N}})$ by using using Lemmata 4.2.9, 4.2.10, 4.2.11, 4.2.12, 4.2.13, 4.2.14, 4.2.15, 4.2.16, 4.2.17, 4.2.18, 4.2.19, 4.2.20, and 4.2.21. □

4.3 Subsumption and Loop Search

Lemma 4.2.23 shows that we can eliminate tautologies and subsumed clauses during the construction of a derivation at the level of inference rule applications of the $\mathfrak{I}^{S,\succ}_{FG,Sub}$ calculus. However, the rules of the calculus are also applied within the fine-grained breadth-first search algorithm FG-BFS (and its more restricted version Restricted-FG-BFS) which is

```
Function Subsumption-Restricted-FG-BFS
Input:   A set of universal clauses $\mathcal{U}$ and a set of step clauses $\mathcal{S}$, saturated under
         the fine-grained step resolution inference rules of subsumption-compatible
         ordered fine-grained resolution with selection, and an eventuality clause
         $\Diamond L(x) \in \mathcal{E}$.
Output:  A formula $R(x)$ with at most one free variable.
Method:  (1) Let $R_0(x) = \mathbf{true}$; $M_0 = \emptyset$; $i = 0$
         (2) Let $\mathcal{N}'_{i+1} = \mathcal{U} \cup \mathrm{LT}(\mathcal{S}) \cup \{\mathbf{true} \Rightarrow \bigcirc(\neg R_i(c^l) \vee L(c^l))\}$. Apply the
             fine-grained step resolution rules of subsumption-compatible ordered
             fine-grained resolution with selection except the clause conversion
             rule to $\mathcal{N}'_{i+1}$, together with the removal of tautological and subsumed
             clauses. If we obtain a contradiction, then return the loop true (in
             this case $\forall x \neg L(x)$ is implied by the universal part).
             Otherwise let $\mathcal{M}'_{i+1} = \{D_j \Rightarrow \bigcirc\bot\}_{j=1}^{n}$ be the set of all new termi-
             nating step clauses in the saturation of $\mathcal{N}'_{i+1}$.
         (3) If $\mathcal{M}'_{i+1} = \emptyset$, return false; else let $R_{i+1}(x) = \bigvee_{j=1}^{n}(\tilde{\exists} D_j)\{c^l \to x\}$
         (4) If $\forall x(R_i(x) \Rightarrow R_{i+1}(x))$, return $R_{i+1}(x)$.
         (5) i = i + 1; goto 2.
```

Figure 4.1: Breadth-First Search using Subsumption-Compatible Ordered Fine-Grained Step Resolution with Selection together with Redundancy Elimination

used to find loop formulae for the application of the eventuality resolution rules. Naturally, the question arises whether tautological and subsumed clauses can also be eliminated within FG-BFS and Restricted-FG-BFS.

4.3.1 Subsumption-Restricted Loop Search Algorithm

The answer to that is positive. Figure 4.1 shows the so-called *subsumption-restricted* breadth-first search algorithm using ordered fine-grained step resolution with selection, a modification of FG-BFS which removes tautological and subsumed clauses during the saturation process by subsumption-compatible ordered fine-grained resolution with selection in step (2) of the algorithm. In the way in which the algorithm shown in Figure 4.1 is defined the constructed sets $\mathcal{M}'_i$ will not contain terminating step clauses $C \Rightarrow \bigcirc\bot$ and $D \Rightarrow \bigcirc\bot$ such that $C \leq_s D$.

Due to the removal of tautological and subsumed clauses it is not clear at first sight that the Subsumption-Restriction-FG-BFS algorithm still finds all the possible loop formulae as the equivalence of two formulae $R_i(x)$ and $R_{i+1}(x)$ might no longer be achieved. We will show in the subsequent sections that the Subsumption-Restricted-FG-BFS algorithm indeed remains correct and computes all the possible loop formulae for a given temporal problem. We also prove the refutational completeness of $\mathfrak{I}^{S,\succ}_{FG,Sub}$ where applications of the eventuality resolution rules are restricted to loops found by the Subsumption-Restricted-FG-BFS algorithm.

4.3.2 Properties of the Loop Search Algorithm

We begin by showing some properties of the Restricted-FG-BFS algorithm. First of all, we introduce a couple of notions that are important for the remaining proofs.

For a set of clauses $\mathcal{N}$ we denote the set of all the clauses resulting from inferences by the resolution-based and factoring rules of subsumption-complete ordered fine-grained resolution with selection (i.e. without the clause conversion or the eventuality resolution rules) on clauses from the set $\mathcal{N}$ by $\mathrm{Res}_{\mathsf{Sub}}(\mathcal{N})$. Additionally, we define that $\mathrm{Res}^0_{\mathsf{Sub}}(\mathcal{N}) = \mathcal{N}$, $\mathrm{Res}^i_{\mathsf{Sub}}(\mathcal{N}) = \mathrm{Res}_{\mathsf{Sub}}(\mathrm{Res}^{i-1}_{\mathsf{Sub}}(\mathcal{N}))$ for $i > 0$ and

$$\mathrm{Res}^{\infty}_{\mathsf{Sub}}(\mathcal{N}) = \bigcup_{i=0}^{\infty} \mathrm{Res}^i_{\mathsf{Sub}}(\mathcal{N})$$

For a set of clauses $\mathcal{N}$ we denote the set of all the clauses resulting from inferences by the fine-grained step resolution inference rules of subsumption-compatible ordered fine-grained resolution with selection from clauses in the set $\mathcal{N}$ by $\mathrm{Res}_{\mathsf{Sub,Conv}}(\mathcal{N})$. The notions of $\mathrm{Res}^0_{\mathsf{Sub,Conv}}(\mathcal{N})$, $\mathrm{Res}^i_{\mathsf{Sub,Conv}}(\mathcal{N})$ for $i > 0$ and $\mathrm{Res}^{\infty}_{\mathsf{Sub,Conv}}(\mathcal{N})$ are defined analogously to the sets $\mathrm{Res}^0_{\mathsf{Sub}}(\mathcal{N})$, $\mathrm{Res}^i_{\mathsf{Sub}}(\mathcal{N})$ for $i > 0$ and $\mathrm{Res}^{\infty}_{\mathsf{Sub}}(\mathcal{N})$ introduced above.

The following two definitions are necessary for showing that the Subsumption-Restricted-FG-BFS algorithm only requires finitely many iterations on any clausified monodic temporal problem in order to compute all the possible loop formulae.

Definition 4.3.1. *Let $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E} \rangle$ be a clausified monodic temporal problem and let $\mathcal{O} = \{P_1(x), \ldots, P_n(x)\}$ be the set of all the non-ground atoms occurring in the left-hand sides of the step clauses contained in the set $\mathcal{S}'$. Then we denote by $\mathcal{T}(P')$ the set of all the terminating step clauses built from the symbols occurring in the left-hand sides of the step clauses contained in the set $\mathcal{S}'$ which are instantiated by constants from $const(P') \cup \{c^l\}$, free of duplicate atoms and such that each subset of the set $\mathcal{O}$ occurs at most once in a given terminating step clause (after renaming the variable used for the considered subset instance in the terminating step clause to x)*[1]*. We also assume that the set $\mathcal{T}(P')$ contains exactly one representative for each equivalence class under the relation $=_X$.*

Remark 4.3.2. *As the sets $\mathcal{S}'$ and $const(P')$ are finite, it follows that the set $\mathcal{T}(P')$ is finite as well.*

Definition 4.3.3. *Let $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E} \rangle$ be a clausified monodic temporal problem and let $\mathcal{N} \subseteq \mathcal{T}(P')$ be a set of terminating step clauses. Then we define a set of terminating step clauses $\mathcal{T}_{|\mathcal{N}}(P')$ as follows:*

$$\mathcal{T}_{|\mathcal{N}}(P') = \{ D \Rightarrow \bigcirc\bot \in \mathcal{T}(P') \mid \exists C \Rightarrow \bigcirc\bot \in \mathcal{N} \colon C \leq_s D \}$$

The following proposition establishes a basic result about the terminating step clauses that are constructed during runs of the Subsumption-Restricted-FG-BFS algorithm.

[1] For this definition free variables are not considered to be existentially quantified and the loop search constant c^l is not replaced by a fresh variable.

Lemma 4.3.4. *Let $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E} \rangle$ be a clausified monodic temporal problem. Additionally, let $\mathcal{N}'_0, \mathcal{N}'_1, \ldots$ be the initial sets of universal and step clauses in each iteration and $\mathcal{M}'_0, \mathcal{M}'_1, \ldots$ be the sets of terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem P for an eventuality $\Diamond L(x) \in \mathcal{E}$ (without considering potential derivations of the empty clause).*

Then it holds that:

(i) for every i and for every terminating step clause $C \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$ there exists a variable renaming σ such that $\mathbf{true} \Rightarrow \bigcirc(\neg C\sigma \vee L(c^l)) \in \mathcal{N}'_{i+1}$; and conversely,

(ii) for every i and for every newly added step clause $\mathbf{true} \Rightarrow \bigcirc(\neg C \vee L(c^l)) \in \mathcal{N}'_{i+1}$ there exists a variable renaming σ such that $C\sigma \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$.

Proof. Follows for every i from the construction of the set $\mathcal{N}'_{i+1}$. □

In the next lemma we prove that for every terminating step clause $C \Rightarrow \bigcirc\bot$ constructed in an iteration $i+1$ there exists a terminating step clause $D \Rightarrow \bigcirc\bot$ obtained in the iteration i such that $D \leq_s C$.

Lemma 4.3.5. *Let $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E} \rangle$ be a clausified monodic temporal problem. Additionally, let $\mathcal{M}'_0, \mathcal{M}'_1, \ldots$ be the sets of terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem P' for an eventuality $\Diamond L(x) \in \mathcal{E}$ (without considering potential derivations of the empty clause).*

Then it holds for all $i \geq 1$ and for every terminating step clause $C \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i+1}$ that there exists a terminating step clause $D \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$ with $D \leq_s C$.

Proof. Let $\mathcal{N}'_0, \mathcal{N}'_1, \ldots$ be the initial sets of universal and step clauses in each iteration. We now show the statement of the lemma by induction on i.

For $i = 1$, let $C \Rightarrow \bigcirc\bot \in \mathcal{M}'_2$ be a terminating step clause. Additionally, let $\mathbf{true} \Rightarrow \bigcirc(\neg C_1 \vee L(c^l)), \ldots, \mathbf{true} \Rightarrow \bigcirc(\neg C_n \vee L(c^l))$ for $n \geq 1$ be all the loop search step clauses from the set $\mathcal{N}'_2$ that occur in the derivation of the step clause $C \Rightarrow \bigcirc\bot$. Then, as $\mathcal{N}'_1 = \mathcal{U}' \cup \mathrm{LT}(\mathcal{S}') \cup \{\mathbf{true} \Rightarrow \bigcirc L(c^l)\}$ and $\mathbf{true} \Rightarrow \bigcirc L(c^l) \leq_s \mathbf{true} \Rightarrow \bigcirc(\neg C_i \vee L(c^l))$ for every i, $1 \leq i \leq n$, it follows from Lemma 4.2.22 that there exists a terminating step clause $D \Rightarrow \bigcirc\bot \in \mathcal{M}'_1$ with $D \leq_s C$. We still have to note that there cannot be a clause $\mathcal{C} \in \mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathcal{U}' \cup \mathrm{LT}(\mathcal{S}'))$ which subsumes one of the clauses that participates in the derivation of the terminating step clause $D \Rightarrow \bigcirc\bot$ and leads to the derivation of a clause $\mathcal{D}'$ with $\mathcal{D}' <_s D \Rightarrow \bigcirc\bot$ as this would imply that $C \Rightarrow \bigcirc\bot \notin \mathcal{M}'_{i+1}$.

If $i > 1$, let $C \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i+1}$ be a terminating step clause. Furthermore, let $\mathbf{true} \Rightarrow \bigcirc(\neg C_1 \vee L(c^l)), \ldots, \mathbf{true} \Rightarrow \bigcirc(\neg C_n \vee L(c^l))$ for $n \geq 1$ be all the loop search step clauses from the set $\mathcal{N}'_{i+1}$ that occur in the derivation of the step clause $C \Rightarrow \bigcirc\bot$. Then, by Lemma 4.3.4 there exist variable renamings $\sigma_1, \ldots, \sigma_n$ such that $C_1\sigma_1 \Rightarrow \bigcirc\bot, \ldots, C_n\sigma_n \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$. It follows from the induction hypothesis that there are terminating step clauses $D_1 \Rightarrow \bigcirc\bot, \ldots, D_n \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i-1}$ such that $D_i \leq_s C_i$ for every i, $1 \leq i \leq n$. By Lemma 4.3.4

there exist variable renamings $\sigma'_1, \ldots, \sigma'_n$ such that $\mathbf{true} \Rightarrow \bigcirc(\neg D_1\sigma'_1 \vee L(c^l)), \ldots, \mathbf{true} \Rightarrow \bigcirc(\neg D_n\sigma'_n \vee L(c^l)) \in \mathcal{N}'_i$. Hence, as $\mathbf{true} \Rightarrow \bigcirc(\neg D_i\sigma'_i \vee L(c^l)) \leq_s \mathbf{true} \Rightarrow \bigcirc(\neg C_i \vee L(c^l))$ holds for every i, $1 \leq i \leq n$, we obtain from Lemma 4.2.22 that there exists a terminating step clause $D \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$ with $D \leq_s C$. Again, we have to note that there cannot be a clause $C \in \mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathcal{U}' \cup \mathrm{LT}(\mathcal{S}'))$ that subsumes one of the clauses that participates in the derivation of the terminating step clause $D \Rightarrow \bigcirc\bot$ and leads to the derivation of a clause $\mathcal{D}'$ with $\mathcal{D}' <_s D \Rightarrow \bigcirc\bot$ as this would imply that $C \Rightarrow \bigcirc\bot \notin \mathcal{M}'_{i+1}$. □

The following three lemmata establish that the Subsumption-Restricted-FG-BFS algorithm only requires finitely many iterations when it is applied on any clausified monodic temporal problem.

Corollary 4.3.6. *Let $\mathsf{P}' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}\rangle$ be a clausified monodic temporal problem. Additionally, let $\mathcal{M}'_0, \mathcal{M}'_1, \ldots$ be the sets of terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem P' for an eventuality $\Diamond L(x) \in \mathcal{E}$ (without considering potential derivations of the empty clause). Finally, let for all i, $1 \leq i \leq n$, $\mathcal{M}''_i \subseteq \mathcal{T}(\mathsf{P}')$ be the representatives of the equivalence classes w.r.t. the equivalence relation $=_X$ that correspond to the terminating step clauses contained in the set $\mathcal{M}'_i$.*

Then it holds for all $i \geq 1$ that:

$$\mathcal{M}''_{i+1} \subseteq \mathcal{T}_{|\mathcal{M}''_i}(\mathsf{P}') \text{ and } \mathcal{T}_{|\mathcal{M}''_{i+1}}(\mathsf{P}') \subseteq \mathcal{T}_{|\mathcal{M}''_i}(\mathsf{P}')$$

Proof. Follows from Lemma 4.3.5. □

Lemma 4.3.7. *Let $\mathsf{P}' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}\rangle$ be a clausified monodic temporal problem and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality. Additionally, let $\mathcal{M}'_0, \mathcal{M}'_1, \ldots$ be the sets of terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem P' for the eventuality $\Diamond L(x) \in \mathcal{E}$.*

Then there exists an index $i \geq 1$ with either $\mathcal{M}'_i = \emptyset$ or $\emptyset \neq \mathcal{M}'_i =_X \mathcal{M}'_{i+1}$.

Proof. Follows from Corollary 4.3.6 and from the fact that the set $\mathcal{T}(\mathsf{P}')$ is finite. □

Corollary 4.3.8. *Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ be a monodic temporal problem and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality clause. Then the Subsumption-Restricted-FG-BFS algorithm applied on the problem P for the eventuality $\Diamond L(x)$ will require only finitely many iterations.*

The subsequent proposition still establishes a basic result about the terminating step clauses constructed in an iteration of the algorithm.

Lemma 4.3.9. *Let $\mathsf{P}' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}\rangle$ be a clausified monodic temporal problem and let $\Diamond L(x) \in \mathcal{E}$ be an eventuality. Additionally, let $\mathcal{M}'_0, \mathcal{M}'_1, \ldots$ be the sets of terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem P' for the eventuality $\Diamond L(x) \in \mathcal{E}$ (without considering potential derivations of the empty clause).*

Then it holds for every $i \in \mathbb{N}$ and for every step clause $D \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$ that there does not exist a step clause $C \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$ with $C <_s D$.

Proof. Follows immediately from the construction of the different clause sets $\mathcal{M}'_i$ as subsumed clauses are removed. □

The next four lemmata are crucial for proving Lemmata 4.3.15 and 4.3.18. They show that the loop condition, i.e. the logical equivalence of the formulae $R_i(x)$ and $R_{i+1}(x)$ computed in two subsequent iterations, is in fact equivalent to the equality of the sets containing the minimal terminating step clauses with respect to the subsumption relation $\leq_s$ constructed in the iterations i and $i+1$.

Lemma 4.3.10. *Let $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E} \rangle$ be a clausified monodic temporal problem. Additionally, let $C \Rightarrow \bigcirc\bot$ and $D \Rightarrow \bigcirc\bot$ be two terminating step clauses derived by subsumption-compatible ordered fine-grained resolution with selection from $P' \cup \mathrm{LT}(\mathcal{S}) \cup \mathrm{Cls}(\textbf{true} \Rightarrow \bigcirc\Phi(c^l))$ such that $C \leq_s D$.*

Then it holds that the formula $\forall x((\tilde{\exists} D)\{c^l \mapsto x\} \Rightarrow (\tilde{\exists} C)\{c^l \mapsto x\})$ is valid.

Proof. First of all, let $D \equiv \bigwedge_{y \in var(D)} \bigwedge_{j=1}^{m_y} P_j^y(y) \wedge \bigwedge_{j=1}^{n_1} Q_j(c_j) \wedge \bigwedge_{j=1}^{n_2} R_j(c^l) \wedge \bigwedge_{j=1}^{n_3} q_j$, where $c_1, \ldots, c_{n_1}$ are constants such that $\{c_1, \ldots, c_{n_1}\} \cap \{c^l\} = \emptyset$ and $q_1, \ldots, q_{n_3}$ are nullary predicate symbols. Additionally, let σ be the substitution with $C\sigma \subseteq D$. Hence, we have

$$C \equiv \bigwedge_{y \in var(C) \setminus dom(\sigma)} \bigwedge_{j \in \mathcal{I}_y} P_j^y(y) \wedge \bigwedge_{y \in var(C) \cap dom(\sigma)} \bigwedge_{j=1}^{m''_y} \tilde{P}_j^y(y) \wedge \bigwedge_{j \in \mathcal{I}_1} Q_j(c_j)$$
$$\wedge \bigwedge_{j \in \mathcal{I}_2} R_j(c^l) \wedge \bigwedge_{j \in \mathcal{I}_3} q_j$$

with $\mathcal{I}_y \subseteq \{1, \ldots, m_y\}$ for every variable $y \in var(C) \setminus dom(\sigma)$, $\mathcal{I}_1 \subseteq \{1, \ldots, n_1\}$, $\mathcal{I}_2 \subseteq \{1, \ldots, n_2\}$ and $\mathcal{I}_3 \subseteq \{1, \ldots, n_3\}$.

Moreover, let $\mathfrak{M} = (\mathcal{D}, \mathcal{I})$ be a first-order interpretation over the signature of the formulae C and D. Additionally, let $\mathfrak{a}\colon X \to \mathcal{D}$ be a variable assignment. We also assume that $\mathfrak{M} \models^{\mathfrak{a}} (\tilde{\exists} D)\{c^l \mapsto x\}$ holds.

We can thus infer that $\mathfrak{M} \models^{\mathfrak{a}} \exists y \bigwedge_{j \in \mathcal{I}_y} P_j^y(y)$ for every variable $y \in var(C) \setminus dom(\sigma)$, $\mathfrak{M} \models^{\mathfrak{a}} Q_j(c_j)$ for every $j \in \mathcal{I}_1$, $\mathfrak{M} \models^{\mathfrak{a}} R_j(x)$ for every $j \in \mathcal{I}_2$ and $\mathfrak{M} \models^{\mathfrak{a}} q_j$ for every $j \in \mathcal{I}_3$. Finally, for a variable $y \in var(C) \cap dom(\sigma)$ we distinguish between the following cases.

For $\sigma(y) = z \in X$, it holds that $\mathfrak{M} \models^{\mathfrak{a}} \exists z \bigwedge_{j=1}^{m_z} P_j^z(z)$, which implies that $\mathfrak{M} \models^{\mathfrak{a}} \exists y \bigwedge_{j=1}^{m''_y} \tilde{P}_j^y(y)$. If $\sigma(y) = c^l$, then we have $\mathfrak{M} \models^{\mathfrak{a}} \bigwedge_{j=1}^{m'_y} P_j^y(x)$, from which it follows that $\mathfrak{M} \models^{\mathfrak{a}} \exists y \bigwedge_{j=1}^{m''_y} \tilde{P}_j^y(y)$. For $\sigma(y) = c$ and $c \neq c^l$, we obtain $\mathfrak{M} \models^{\mathfrak{a}} \bigwedge_{j=1}^{m'_y} P_j^y(c)$, from which we can infer that $\mathfrak{M} \models^{\mathfrak{a}} \exists y \bigwedge_{j=1}^{m''_y} \tilde{P}_j^y(y)$.

We can hence conclude that:

$$\mathfrak{M} \models^a \bigwedge_{y \in var(C) \setminus dom(\sigma)} \left(\exists y \bigwedge_{j \in \mathcal{I}_y} P_j^y(y) \right) \wedge \bigwedge_{y \in var(C) \cap dom(\sigma)} \left(\exists y \bigwedge_{j=1}^{m''_y} \tilde{P}_j^y(y) \right) \\ \wedge \bigwedge_{j \in \mathcal{I}_1} Q_j(c_j) \wedge \bigwedge_{j \in \mathcal{I}_2} R_j(x) \wedge \bigwedge_{j \in \mathcal{I}_3} q_j$$

And finally, we can infer that $\mathfrak{M} \models^a (\tilde{\exists} C)\{c^l \mapsto x\}$ holds. □

Definition 4.3.11. *For two first-order clauses $\mathcal{C}$ and $\mathcal{D}$, we write $\mathcal{C} \vdash_f \mathcal{D}$ if and only the clause $\mathcal{D}$ results from the clause $\mathcal{C}$ through an arbitrary factoring inference on a pair of positive or negative literals. The reflexive and transitive closure of the relation $\vdash_f$ will be denoted by $\vdash_f^*$.*

Lemma 4.3.12. *Let $C \Rightarrow \bigcirc\bot$ and $D_1 \Rightarrow \bigcirc\bot, \ldots, D_n \Rightarrow \bigcirc\bot$ be terminating step clauses derived by subsumption-compatible ordered fine-grained resolution with selection from $P \cup \mathrm{LT}(\mathcal{S}) \cup \mathrm{Cls}(\boldsymbol{true} \Rightarrow \bigcirc\Phi_1(c^l)) \cup \mathrm{Cls}(\boldsymbol{true} \Rightarrow \bigcirc\Phi_2(c^l))$. We also assume that the formula*

$$\forall x((\tilde{\exists} C)\{c^l \mapsto x\} \Rightarrow \bigvee_{j=1}^{n} (\tilde{\exists} D_j)\{c^l \mapsto x\})$$

is valid. Then there exists an index j with $1 \leq j \leq n$ and a negative clause $\mathcal{D}$ such that $D_j \vdash_f^* \mathcal{D}$ and $\mathcal{D} \leq_s \neg C$.

Proof. Let $\mathcal{F} := \forall x((\tilde{\exists} C)\{c^l \mapsto x\} \Rightarrow \bigvee_{j=1}^{n} (\tilde{\exists} D_j)\{c^l \mapsto x\})$, and let

$$C = \bigwedge_{y \in var(C)} \bigwedge_{j=1}^{m_y} P_j^y(y) \wedge \bigwedge_{j=1}^{n_1} Q_j(c_j) \wedge \bigwedge_{j=1}^{n_2} R_j(c^l) \wedge \bigwedge_{j=1}^{n_3} q_j,$$

where $c_1, \ldots, c_{n_1}$ are constants such that $\{c_1, \ldots, c_{n_1}\} \cap \{c^l\} = \emptyset$ and $q_1, \ldots, q_{n_3}$ are nullary predicate symbols. The clausification of the formula $\neg\mathcal{F} \equiv \exists x((\tilde{\exists} C)\{c^l \mapsto x\} \wedge \bigwedge_{j=1}^{n} (\tilde{\forall} \neg D_j)\{c^l \mapsto x\})$ yields the following set of clauses $\mathcal{N}$:

$$\mathcal{N} = \bigcup_{y \in var(C)} \bigcup_{j=1}^{m_y} \{P_j^y(sk_y)\} \cup \bigcup_{j=1}^{n_1} \{Q_j(c_j)\} \cup \bigcup_{j=1}^{n_2} \{R_j(sk)\} \cup \bigcup_{j=1}^{n_3} \{q_j\} \\ \cup \bigcup_{j=1}^{n} \{\neg D_j\{c^l \mapsto sk\}\}$$

where $\{sk\} \cup \{\, sk_y \mid y \in var(C) \,\}$ are fresh Skolem constants. As the formula $\neg\mathcal{F}$ is unsatisfiable, it follows from refutational completeness of (regular) resolution that there exists a derivation $\mathcal{R}$ of the empty clause from the clause set $\mathcal{N}$. Additionally, as the clauses $\neg D_j\{c^l \mapsto sk\}$ for $1 \leq j \leq n$ only contain negative literals, we can observe that the derivation $\mathcal{R}$ essentially consists in variable-free unit clauses resulting from the clausification of the formula $(\tilde{\exists} C)\{c^l \mapsto sk\}$ being resolved together with a single clause $\neg D_j\{c^l \mapsto sk\}$ for a j, $1 \leq j \leq n$.

Let σ denote the accumulation of the different substitutions used in the derivation $\mathcal{R}$[2]. Then, the substitution σ maps every variable from its domain into a constant. We now exhaustively apply the (negative) factoring rule on every pair of literals $\neg P(x)$ and $\neg P(y)$ with $\sigma(x) = \sigma(y)$ or $\neg P(x)$ and $\neg P(c)$ with $\sigma(x) = c$ (for arbitrary predicate symbols P and constants c). After performing some additional elimination of duplicate literals we obtain a negative clause $\mathcal{D}$ such that $D_j \vdash_f^* \mathcal{D}$ holds. We can also infer that there exists a new refutation $\mathcal{R}'$ which involves the clause $\mathcal{D}$ and which results from the derivation $\mathcal{R}$. Additionally, as every literal in the clause $\mathcal{D}\sigma$ occurs at most once and as variables occurring in the clause $\mathcal{D}$ are consistently replaced by the same constants in the derivation $\mathcal{R}'$, which implies that every unit clause is used at most once, it is easy to see that there exists a substitution σ' with $\mathcal{D}\sigma' \subseteq \neg C$, i.e. $\mathcal{D} \leq_s \neg C$ holds. □

Lemma 4.3.13. *Let $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E} \rangle$ be a clausified monodic temporal problem. Furthermore, let $C_1 \Rightarrow \bigcirc\bot, \ldots, C_m \Rightarrow \bigcirc\bot$ and $D_1 \Rightarrow \bigcirc\bot, \ldots, D_n \Rightarrow \bigcirc\bot$ be terminating step clauses derived by subsumption-compatible ordered fine-grained resolution with selection from $P' \cup \mathrm{LT}(\mathcal{S}) \cup \mathrm{Cls}(\boldsymbol{true} \Rightarrow \bigcirc\Phi_1(c^l)) \cup \mathrm{Cls}(\boldsymbol{true} \Rightarrow \bigcirc\Phi_2(c^l))$. We also assume that the formula*

$$\forall x(\bigvee_{i=1}^{m}(\tilde{\exists}C_i)\{c^l \mapsto x\} \Rightarrow \bigvee_{j=1}^{n}(\tilde{\exists}D_j)\{c^l \mapsto x\})$$

is valid. Then it holds that for every i, $1 \leq i \leq m$ there exists an index j, $1 \leq j \leq n$ and a clause $\mathcal{D}$ with $D_j \vdash_f^* \mathcal{D}$ and $\mathcal{D} \leq_s \neg C_i$.

Proof. First of all, we can infer that the following formula is valid as well:

$$\forall x\Big(\bigwedge_{i=1}^{m}\Big[(\tilde{\exists}C_i)\{c^l \mapsto x\} \Rightarrow \bigvee_{j=1}^{n}(\tilde{\exists}D_j)\{c^l \mapsto x\}\Big]\Big)$$

Consequently, we obtain that the subsequent formula is also valid:

$$\bigwedge_{i=1}^{m}\forall x\Big((\tilde{\exists}C_i)\{c^l \mapsto x\} \Rightarrow \bigvee_{j=1}^{n}(\tilde{\exists}D_j)\{c^l \mapsto x\}\Big)$$

We can conclude that for every i, $1 \leq i \leq m$ the formula

$$\forall x((\tilde{\exists}C_i)\{c^l \mapsto x\} \Rightarrow \bigvee_{j=1}^{n}(\tilde{\exists}D_j)\{c^l \mapsto x\})$$

is valid and thus, by Lemma 4.3.12 for every i, $1 \leq i \leq m$, there exists an index j, $1 \leq j \leq n$, and a clause $\mathcal{D}$ with $D_j \vdash_f^* \mathcal{D}$ and $\mathcal{D} \leq_s \neg C_i$. □

Lemma 4.3.14. *Let $\mathcal{M}$ and $\mathcal{M}'$ be sets of terminating step clauses derived by subsumption-compatible ordered fine-grained resolution with selection from $P \cup \mathrm{LT}(\mathcal{S}) \cup \mathrm{Cls}(\boldsymbol{true} \Rightarrow \bigcirc\Phi_1(c^l)) \cup \mathrm{Cls}(\boldsymbol{true} \Rightarrow \bigcirc\Phi_2(c^l))$. We also assume that the sets $\mathcal{M}$ and $\mathcal{M}'$ are closed under*

[2] As the unit clauses do not contain free variables, there is no need to rename variables.

the application of the (unordered) factoring rule. Finally, let $\mathcal{N} = \{C_1 \Rightarrow \bigcirc\bot, \dots, C_m \Rightarrow \bigcirc\bot\} \subseteq M$ and $\mathcal{N}' = \{D_1 \Rightarrow \bigcirc\bot, \dots, D_n \Rightarrow \bigcirc\bot\} \subseteq M'$ be the sets of all the minimal step clauses with respect to the relation $\leq_s$ contained in the sets $\mathcal{M}$ and $\mathcal{M}'$, respectively.

Then the following statements are equivalent:

(i) $\mathcal{N} =_X \mathcal{N}'$

(ii) the formula $\forall x(\bigvee_{i=1}^{m} (\tilde{\exists} C_i)\{c^l \mapsto x\} \Leftrightarrow \bigvee_{j=1}^{n} (\tilde{\exists} D_j)\{c^l \mapsto x\})$ *is valid*

(iii) $\forall\, i, 1 \leq i \leq m\, \exists\, j, 1 \leq j \leq n\colon D_j \leq_s C_i$ *and* $\forall\, j, 1 \leq i \leq n\, \exists\, i, 1 \leq j \leq m\colon C_i \leq_s D_j$

Proof. The implication $(i) \Rightarrow (ii)$ is obvious. By Lemma 4.3.13 and by closedness under factoring inferences the implication $(ii) \Rightarrow (iii)$ holds. For the remaining implication $(iii) \Rightarrow (i)$ let $C \Rightarrow \bigcirc\bot \in \mathcal{N}$. It then follows from the assumptions that there exists a step clause $D \Rightarrow \bigcirc\bot \in \mathcal{N}'$ such that $D \leq_s C$. We obtain again from the assumptions that there exists a step clause $C' \Rightarrow \bigcirc \in \mathcal{N}$ with $C' \leq_s D$. Thus, we can conclude that there exists a variable renaming σ with $C\sigma = D$ and $C\sigma \Rightarrow \bigcirc\bot \in \mathcal{N}'$ as otherwise there would exist a step clause $C' \Rightarrow \bigcirc\bot \in \mathcal{N} \subseteq \mathcal{M}$ with $C' <_s C$, which would contradict the minimality of the step clause $C \Rightarrow \bigcirc\bot$.

The inclusion $\mathcal{N}' \subseteq_X \mathcal{N}$ can be shown analogously. □

We now show that the sets of minimal clauses with respect to the subsumption relation remain unchanged in subsequent iterations (up to variable names) once the loop search condition has been fulfilled and the algorithm has been kept iterating.

Lemma 4.3.15. *Let $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E} \rangle$ be a clausified monodic temporal problem. Additionally, let $\mathcal{M}'_0, \mathcal{M}'_1, \dots$ be the sets of terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem P' for an eventuality $\Diamond L(x) \in \mathcal{E}$ (without considering potential derivations of the empty clause). Finally, let $i \in \mathbb{N}$ be an iteration index such that the formula $\forall x(\bigvee_{C \Rightarrow \bigcirc\bot \in \mathcal{M}'_i} (\tilde{\exists} C)\{c^l \mapsto x\} \Leftrightarrow \bigvee_{D \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i+1}} (\tilde{\exists} D)\{c^l \mapsto x\})$ is valid.*

Then it holds for every $j \geq 0$ that $\mathcal{M}'_i =_X \mathcal{M}'_{i+j}$.

Proof. First of all, let $\mathcal{N}'_0, \mathcal{N}'_1, \dots$ be the initial sets of universal and step clauses in each iteration and let $\mathcal{T}'_0, \mathcal{T}'_1, \dots$ be the sets of all the terminating step clauses that can be derived from the sets $\mathcal{N}'_0, \mathcal{N}'_1, \dots$, respectively. Then, by Lemma 4.3.14 it follows from the validity of the formula $\forall x(\bigvee_{C \Rightarrow \bigcirc\bot \in \mathcal{M}'_i} (\tilde{\exists} C)\{c^l \mapsto x\} \Leftrightarrow \bigvee_{D \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i+1}} (\tilde{\exists} D)\{c^l \mapsto x\})$ that $\mathcal{M}'_i =_X \mathcal{M}'_{i+1}$.

We now show by induction on j that $\mathcal{M}'_{i+j} =_X \mathcal{M}'_{i+j+1}$ for every $j \geq 0$. For $j = 0$ there remains nothing to be shown. If $j > 0$, then it follows from the induction hypothesis that $\mathcal{M}'_{i+j-1} =_X \mathcal{M}'_{i+j}$.

Let $C \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i+j}$ and let $\mathbf{true} \Rightarrow \bigcirc(\neg D_1 \vee L(c^l)), \dots, \mathbf{true} \Rightarrow \bigcirc(\neg D_n \vee L(c^l))$ be all the newly added step clauses to the set $\mathcal{N}'_{i+j}$ which are involved in the derivation of the terminating step clause $C \Rightarrow \bigcirc\bot$. By Lemma 4.3.4, there exist variable renamings $\sigma_1, \dots, \sigma_n$

such that $D_k\sigma_k \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i+j-1}$ for every k, $1 \leq k \leq n$. Thus, it follows from the induction hypothesis that there exist variable renamings $\sigma'_1, \ldots, \sigma'_n$ with $D_k\sigma'_k \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i+j}$ for every k, $1 \leq k \leq n$. Consequently, by Lemma 4.3.4 there exist variables renamings $\sigma''_1, \ldots, \sigma''_n$ such that $\mathbf{true} \Rightarrow \bigcirc(\neg D_k\sigma''_k \vee L(c^l)) \in \mathcal{N}'_{i+j+1}$ for every k, $1 \leq k \leq n$. It is hence easy to see that there exists a variable renaming σ with $C\sigma \Rightarrow \bigcirc\bot \in \mathcal{T}'_{i+j+1}$. If we now assume that the step clause $C\sigma \Rightarrow \bigcirc\bot \in \mathcal{T}'_{i+j+1}$ is not minimal in the set $\mathcal{T}'_{i+j+1}$ with respect to the relation $\leq_s$, then let $C' \Rightarrow \bigcirc\bot \in \mathcal{T}'_{i+j+1}$ be a minimal terminating step clause with $C' <_s C\sigma$ (there cannot be a universal clause $\mathcal{D} \in \mathrm{Res}_{\mathrm{Sub}}(\mathcal{U})$ with $\mathcal{D} <_s C\sigma \Rightarrow \bigcirc\bot$ as this would imply that $C \Rightarrow \bigcirc\bot \notin \mathcal{M}'_{i+j}$). Let $\mathbf{true} \Rightarrow \bigcirc(\neg E_1 \vee L(c^l)), \ldots, \mathbf{true} \Rightarrow \bigcirc(\neg E_m \vee L(c^l))$ be all the newly added step clauses to the set $\mathcal{N}'_{i+j+1}$ which are involved in the derivation of the terminating step clause $C' \Rightarrow \bigcirc\bot$. Thus, it follows again from Lemma 4.3.4 that there are variable renamings $\theta_1, \ldots, \theta_m$ such that $E_1\theta_1 \Rightarrow \bigcirc\bot, \ldots, E_m\theta_m \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i+j} =_X \mathcal{M}'_{i+j-1}$. Hence, by Lemma 4.3.4 again there exist variable renamings $\theta'_1, \ldots, \theta'_m$ with $\mathbf{true} \Rightarrow \bigcirc(\neg E_1\theta'_1 \vee L(c^l)), \ldots, \mathbf{true} \Rightarrow \bigcirc(\neg E_m\theta'_m \vee L(c^l)) \in \mathcal{N}'_{i+j}$. We can infer that there is a variable renaming θ with $C'\theta \Rightarrow \bigcirc\bot \in \mathcal{T}'_{i+j}$ and $C'\theta <_s C$, which contradicts the minimality of the step clause $C \Rightarrow \bigcirc\bot$. We can conclude that $C\sigma \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i+j+1}$.

The inclusion $\mathcal{M}'_{i+j+1} \subseteq_X \mathcal{M}'_{i+j}$ can be shown analogously using $\mathcal{M}'_{i+j} \subseteq_X \mathcal{M}'_{i+j+1}$. □

We now prove a lemma that links together two runs of the Subsumption-Restricted-FG-BFS algorithm for the same eventuality $\Diamond L(x)$ on two sets of universal clauses $\mathcal{U}$ and $\tilde{\mathcal{U}}$ with $\mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathcal{U}) \leq_s \tilde{\mathcal{U}} \setminus \mathrm{taut}(\tilde{\mathcal{U}})$.

Lemma 4.3.16. *Let $\tilde{P} = \langle \tilde{\mathcal{U}}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ be a clausified monodic temporal problem. Additionally, let $\tilde{\mathcal{M}}'_0, \tilde{\mathcal{M}}'_1, \ldots$ be the sets of terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem $\tilde{P}$ for an eventuality $\Diamond L(x) \in \mathcal{E}$ (without considering potential derivations of the empty clause). Furthermore, let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ be a clausified monodic temporal problem with $\mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathcal{U}) \leq_s \tilde{\mathcal{U}} \setminus \mathrm{taut}(\tilde{\mathcal{U}})$ and let $\mathcal{M}'_0, \mathcal{M}'_1, \ldots$ be the sets of terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem P for the eventuality $\Diamond L(x) \in \mathcal{E}$ (without considering potential derivations of the empty clause).*

Then it holds for every i that $\mathcal{M}'_i \cup \mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathcal{U} \cup \mathcal{S}) \leq_s \tilde{\mathcal{M}}'_i$.

Proof. By induction on i using Lemmata 4.2.23 and 4.3.4. □

Before we can prove Lemma 4.3.18, we have to establish the following result. It states that a set of universal clauses $\mathcal{U}$ logically implies the formula $\forall x \neg(\tilde{\exists} D)\{c^l \mapsto x\}$, where D is the left-hand side of the terminating step clause $D \Rightarrow \bigcirc\bot$ constructed by the loop search algorithm, if a clause C with $C \leq_s \neg D$ can be derived by resolution inferences from clauses contained in $\mathcal{U}$.

Lemma 4.3.17. *Let $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}\rangle$ be a clausified monodic temporal problem. Additionally, let $C \in \mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathcal{U})$ be a universal clause and $D = \bigwedge_{j=1}^{n} L_j$, where for every j, $1 \leq j \leq n$, L_j is an at most monadic literal with $const(L_j) \subseteq const(P') \cup \{c^l\}$, and such that $C \leq_s \neg D$.*

Then it holds that $\mathcal{U} \models \forall x \neg(\tilde{\exists} D)\{c^l \mapsto x\}$.

Proof. First of all, it follows from the definition of the subsumption relation $\leq_s$ on temporal clauses that there exists a substitution σ with $C\sigma \subseteq \neg D$. As $c^l \notin \mathit{const}(\mathsf{P}')$, it is easy to see that there also exists a substitution σ' with $C\sigma' \subseteq \neg D\{c^l \mapsto x\}$ for a fresh variable x, which implies that the formula $\tilde{\forall} C \Rightarrow \tilde{\forall}(\neg D\{c^l \mapsto x\})$ is valid. We can thus conclude that the formula $\tilde{\forall} C \Rightarrow \forall x \neg(\tilde{\exists} D)\{c^l \mapsto x\})$ is also valid, and it follows from soundness of subsumption-compatible ordered fine-grained resolution with selection that $\mathcal{U} \models \forall x \neg(\tilde{\exists} D)\{c^l \mapsto x\}$. □

Finally, we show that for a loop formula $\bigvee_{j=1}^{m} \mathcal{A}_j(x)$ w.r.t. a satisfiable set of universal clauses $\mathcal{U}$ and an eventuality $\Diamond L(x)$, where $\mathit{Prenex}(\mathcal{A}_j(x)) = (\tilde{\exists} D_j)\{c^l \mapsto x\}$ for $1 \leq j \leq m$, the Subsumption-Restricted-FG-BFS algorithm applied on the clausified monodic temporal problem (saturated under the fine-grained step resolution inference rules of subsumption-compatible ordered fine-grained resolution with selection) involving the set $\mathcal{U}$ for the eventuality $\Diamond L(x)$ computes a formula $R(x) = \bigvee_{i=1}^{n} (\tilde{\exists} C_i)\{c^l \mapsto x\}$, $n \geq 0$, such that for every $j \in \{1, \ldots, m\}$ there either exists a universal clause $C \in \mathcal{U}$ with $C \leq_s \neg D_j$ or there exists $i \in \{1, \ldots, n\}$ such that $C_i \leq_s D_j$.

Lemma 4.3.18. *Let* $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ *be a clausified monodic temporal problem saturated under the fine-grained step resolution inference rules of* $\mathfrak{I}^{S,\succ}_{FG,Sub}$ *such that the set* $\mathcal{U}$ *is satisfiable, let* $\forall x(\mathcal{A}_j(x) \Rightarrow \bigcirc \mathcal{B}_j(x))$ *for* $j \in \{1, \ldots, m\}$ *and* $m \geq 1$ *be full merged step clauses with* $\mathit{Prenex}(\mathcal{A}_j(x)) = (\tilde{\exists} D_j)\{c^l \mapsto x\}$ *for every* $j \in \{1, \ldots, m\}$ *and let* $\Diamond L(x) \in \mathcal{E}$ *be an eventuality clause such that the loop side conditions* $\forall x(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow \neg L(x))$ *and* $\forall x(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow \bigvee_{j=1}^{m} \mathcal{A}_j(x))$ *are valid for all* j, $1 \leq j \leq m$.

Then the Subsumption-Restricted-BFS algorithm applied on the temporal problem P *for the eventuality* $\Diamond L(x)$ *returns a formula* $R(x) = \bigvee_{i=1}^{n} (\tilde{\exists} C_i)\{c^l \mapsto x\}$, $n \geq 0$, *such that for every* $j \in \{1, \ldots, m\}$ *there either exists a universal clause* $C \in \mathcal{U}$ *with* $C \leq_s \neg D_j$ *or there exists* $i \in \{1, \ldots, n\}$ *such that* $C_i \leq_s D_j$.

Proof. First of all, for every full merged step clause $\forall x(\mathcal{A}_j(x) \Rightarrow \bigcirc \mathcal{B}_j(x))$, $j \in \{1, \ldots, m\}$, such that $\mathit{Prenex}(\mathcal{A}_j(x)) = (\tilde{\exists} D_j)\{c^l \mapsto x\}$ and such that there exists a universal clause $C \in \mathcal{U}$ with $C \leq_s \neg D_j$, it follows from Lemma 4.3.17 that $\mathcal{U} \models \forall x \neg \mathcal{A}_j(x)$. We can thus infer that for every $j' \in \{1, \ldots, m\}$ with $j' \neq j$, the formula $\forall x(\mathcal{U} \wedge \mathcal{B}_{j'}(x) \Rightarrow \bigvee_{j'' \in \{1,\ldots,m\} \setminus \{j\}} \mathcal{A}_{j''}(x))$ is valid. Consequently, we may assume without loss of generality that for every full merged step clause $\forall x(\mathcal{A}_j(x) \Rightarrow \bigcirc \mathcal{B}_j(x))$, $j \in \{1, \ldots, m\}$, with $\mathit{Prenex}(\mathcal{A}_j(x)) = (\tilde{\exists} D_j)\{c^l \mapsto x\}$ there does not exist a universal clause $C \in \mathcal{U}$ with $C \leq_s \neg D_j$.

Now, let $\mathcal{M}'_0, \mathcal{M}'_1, \ldots, \mathcal{M}'_l$ be the sequence of sets and $R_0(x), R_1(x), \ldots, R_l(x)$ be the sequence of formulae constructed by a run of the Subsumption-Restricted-BFS algorithm applied on the temporal problem P for the eventuality $\Diamond L(x)$. We show by induction on i with $i \geq 1$ that for all $\forall x(\mathcal{A}_j(x) \Rightarrow \bigcirc \mathcal{B}_j(x))$, $j \in \{1, \ldots, m\}$, with $\mathit{Prenex}(\mathcal{A}_j(x)) = (\tilde{\exists} D_j)\{c^l \mapsto x\}$ there exists a terminating step clause $C^i_j \Rightarrow \bigcirc \bot \in \mathcal{M}'_i$ such that $C^i_j \leq_s D_j$.

For $i = 1$, we first of all observe for all j, $1 \leq j \leq m$, that the formula $\forall x(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow \neg L(x))$ is valid. By Lemma 3.4.40 it follows for every j, $1 \leq j \leq m$, that there exists a derivation of a terminating step clause $E \Rightarrow \bigcirc\bot$ from $\mathsf{P} \cup \mathrm{LT}(\mathcal{S}) \cup \mathrm{Cls}(\mathbf{true} \Rightarrow \bigcirc L(c^l))$ by ordered fine-grained resolution with selection without the clause conversion rule such that $E \leq_s D_j$. Then, by Lemma 4.2.23 there exists a clause $\tilde{C} \in \mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathcal{U} \cup \mathcal{S} \cup \mathrm{LT}(\mathcal{S}) \cup \mathrm{Cls}(\mathbf{true} \Rightarrow \bigcirc L(c^l))$ with $\tilde{C} \leq_s E \Rightarrow \bigcirc\bot$. If we assume that there exists a clause $\tilde{C}$ with $\tilde{C} \leq_s \bot$, i.e. $\tilde{C} = \bot$, then it would follow that $\tilde{C} = \bot \in \mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathcal{U}) = \mathcal{U}$ as in saturations computed by the Subsumption-Restricted FG-BFS algorithm universal clauses can only be derived through universal clauses. Consequently, it would hold that the set $\mathcal{U}$ is unsatisfiable, which contradicts our assumptions. We can thus infer from our assumptions that there exists a terminating step clause $C^1_j \Rightarrow \bigcirc\bot \in \mathcal{M}'_1$ with $C^1_j \leq_s E \leq_s D_j$.

If $i > 1$, then it follows from the induction hypothesis for all $\forall x(\mathcal{A}_j(x) \Rightarrow \bigcirc\mathcal{B}_j(x))$, $j \in \{1, \ldots, m\}$, with $\mathit{Prenex}(\mathcal{A}(x)) = (\tilde{\exists} D_j)\{c^l \mapsto x\}$ that there exists a terminating step clause $C^{i-1}_j \Rightarrow \bigcirc\bot \in \mathcal{M}'_{i-1}$ such that $C^{i-1}_j \leq_s D_j$. By Lemma 4.3.10 we can then infer that the formulae $\forall x(\mathcal{A}_j(x) \Rightarrow (\tilde{\exists} C^{i-1}_j)\{c^l \mapsto x\}))$ and $\forall x(\mathcal{A}_j(x) \Rightarrow R_{i-1}(x))$ are valid for every j, $1 \leq j \leq m$. We obtain for all j, $1 \leq j \leq m$, from the validity of the formula $\forall x(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow \bigvee^n_{j=1} \mathcal{A}_j(x))$ that the formula $\forall x(\mathcal{U} \wedge \mathcal{B}_j(x) \Rightarrow (R_{i-1}(x) \wedge \neg L(x)))$ is also valid. By Lemma 3.4.40 it follows again for every j, $1 \leq j \leq m$, that there exists a derivation of a terminating step clause $E \Rightarrow \bigcirc\bot$ from $\mathsf{P} \cup \mathrm{LT}(\mathcal{S}) \cup \mathrm{Cls}(\mathbf{true} \Rightarrow \bigcirc(\neg R_i(c^l) \vee L(c^l)))$ by ordered fine-grained resolution with selection without the clause conversion rule such that $E \leq_s D_j$. Similarly, to the previous case one can infer that there exists a terminating step clause $C^i_j \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$ with $C^i_j \leq_s E \leq_s D_j$. □

4.3.3 Refutational Completeness

In this section we prove the refutational completeness of subsumption-compatible ordered fine-grained resolution with selection where applications of the eventuality resolution rules are restricted to loops found by the Subsumption-Restricted-FG-BFS algorithm. The proof of refutational completeness is based on simulating a $\mathfrak{I}^{S,\succ}_{FG}$-derivation in $\mathfrak{I}^{S,\succ}_{FG,Sub}$.

First, we analyse the formulae computed by the Restricted-FG-BFS and Subsumption-Restricted-FG-BFS algorithms applied for the same eventuality $\Diamond L(x)$ on two clausified temporal problems $\tilde{\mathsf{P}} = \langle\tilde{\mathcal{U}}, \tilde{\mathcal{I}}, \tilde{\mathcal{S}}, \mathcal{E}\rangle$ and $\mathsf{P} = \langle\mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$, respectively, where $\mathcal{I} \leq_s \tilde{\mathcal{I}} \setminus \mathrm{taut}(\tilde{\mathcal{I}})$, $\mathcal{U} \leq_s \tilde{\mathcal{U}} \setminus \mathrm{taut}(\tilde{\mathcal{U}})$, and $\mathcal{S} \leq_s \tilde{\mathcal{S}} \setminus \mathrm{taut}(\tilde{\mathcal{S}})$.

Lemma 4.3.19. *Let $\tilde{\mathsf{P}} = \langle\tilde{\mathcal{U}}, \tilde{\mathcal{I}}, \tilde{\mathcal{S}}, \mathcal{E}\rangle$ be a clausified monodic temporal problem. Additionally, let $\tilde{H}(x) \neq \boldsymbol{false}$ be the formula computed by the Restricted-FG-BFS algorithm applied on the temporal problem $\tilde{\mathsf{P}}$ for an eventuality $\Diamond L(x) \in \mathcal{E}$. Furthermore, let $\mathsf{P} = \langle\mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ be a clausified monodic temporal problem saturated under the fine-grained step resolution inference rules of subsumption-compatible ordered fine-grained resolution with selection and such that $\mathcal{I} \leq_s \tilde{\mathcal{I}} \setminus \mathrm{taut}(\tilde{\mathcal{I}})$, $\mathcal{U} \leq_s \tilde{\mathcal{U}} \setminus \mathrm{taut}(\tilde{\mathcal{U}})$, $\mathcal{S} \leq_s \tilde{\mathcal{S}} \setminus \mathrm{taut}(\tilde{\mathcal{S}})$. Finally, let $H(x)$ be the loop formula computed constructed by the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem P' for the eventuality $\Diamond L(x) \in \mathcal{E}$.*

Then it holds that $\mathrm{Cls}(\forall x \neg H(x)) \cup \mathcal{U} \leq_s \mathrm{Cls}(\forall x \neg \tilde{H}(x))$.

Proof. First of all, if $\mathcal{U} \models \forall x \neg L(x)$, we can infer that $H(x) = \mathbf{true}$. Furthermore, we obtain $\mathrm{Cls}(\forall x \neg H(x)) = \{\bot\} \leq_s \mathrm{Cls}(\forall x \neg \tilde{H}(x))$.

Otherwise, $\mathcal{U} \not\models \forall x \neg L(x)$ holds. We can assume that the set $\mathcal{U}$ is satisfiable and that $H(x) \neq \mathbf{true}$. If $\tilde{H}(x) = \mathbf{true}$, we would have $\mathcal{U} \models \tilde{\mathcal{U}} \models \forall x \neg L(x)$ by Lemma 4.2.7. Thus, let $\tilde{H}(x) = \bigvee_{i=1}^{m} (\tilde{\exists} D_j)\{c^l \mapsto x\}$ and $H(x) = \bigvee_{i=1}^{n} (\tilde{\exists} C_i)\{c^l \mapsto x\}$. Then it follows from Lemma 4.3.18 for all $j \in \{1, \ldots, m\}$ that there either exists a universal clause $\mathcal{C} \in \mathcal{U}$ with $\mathcal{C} \leq_s \neg D_j$ or there exists $i \in \{1, \ldots, n\}$ such that $C_i \leq_s D_j$. As $c^l \notin \mathit{const}(\mathsf{P}) \cup \mathit{const}(\tilde{\mathsf{P}})$, $\mathrm{Cls}(\forall x \neg \tilde{H}(x)) = \{\neg D_1\{c^l \mapsto z\}, \ldots, \neg D_m\{c^l \mapsto z\}\}$ and $\mathrm{Cls}(\forall x \neg H(x)) = \{\neg C_1\{c^l \mapsto y\}, \ldots, \neg C_n\{c^l \mapsto y\}\}$, where y, z are fresh variables, it is easy to see that for all $j \in \{1, \ldots, m\}$ there either exists a universal clause $\mathcal{C} \in \mathcal{U}$ with $\mathcal{C} \leq_s \neg D_j\{c^l \mapsto z\}$ or there exists $i \in \{1, \ldots, n\}$ such that $\neg C_i\{c^l \mapsto y\} \leq_s \neg D_j\{c^l \mapsto z\}$. □

We can now prove the refutational completeness of subsumption-compatible ordered fine-grained resolution with selection where applications of the eventuality resolution rules are restricted to loop formulae found by the Subsumption-Restricted-FG-BFS algorithm.

Theorem 4.3.20. *Let P be a monodic temporal problem and let P^c be its constant-flooded form. Let $\succ$ be an admissible atom ordering and S be a subsumption compatible selection function. Then P is unsatisfiable if and only if there exists a $\mathfrak{I}^{S,\succ}_{FG,Sub}$-refutation of* $\mathrm{Cls}(P^c)$ *with applications of the eventuality resolution rule restricted to loop formulae found by the Subsumption-Restricted-FG-BFS algorithm. Moreover,, P is unsatisfiable if and only if any fair $\mathfrak{I}^{S,\succ}_{FG,Sub}$-derivation with applications of the eventuality resolution rules restricted to loop formulae found by the Subsumption-Restricted-FG-BFS algorithm is a refutation of* $\mathrm{Cls}(P^c)$.

Proof. Soundness of subsumption-compatible ordered fine-grained resolution with selection follows from soundness of ordered fine-grained resolution with selection and soundness of the inference rules for arbitrary factoring in left-hand sides of terminating step clauses and for arbitrary factoring in (at most) monadic negative universal clauses.

We may now assume that the temporal problem P is unsatisfiable, which implies that the constant-flooded and clausified temporal problem $\mathrm{Cls}(\mathsf{P}^c)$ is also unsatisfiable. It then follows from Theorem 3.4.46 that there exists a refutation $\tilde{\Delta} = \tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_m = \bot$ by $\mathfrak{I}^{S,\succ}_{FG}$ from $\mathrm{Cls}(\mathsf{P}^c)$.

In the following we show by induction on the length of the derivation $\tilde{\Delta}$ that for every clause $\tilde{C}_i$ $(1 \leq i \leq m)$ which is not a tautology that there exists a clause $C_i \in \Delta$ which can be derived by subsumption-compatible ordered fine-grained resolution with selection from the temporal problem $\mathrm{Cls}(\mathsf{P}^c)$ such that $C_i \leq_s \tilde{C}_i$, which implies that $C_m = \bot$.

If a clause $\tilde{C}_i$ $(1 \leq i \leq m)$ is contained in the temporal problem $\mathrm{Cls}(\mathsf{P}^c)$ but not a tautology, then we simply define $C_i = \tilde{C}_i$.

In the case where a non-tautological clause $\tilde{C}_i$ $(1 \leq i \leq m)$ is not contained in the temporal problem $\mathrm{Cls}(\mathsf{P}^c)$ and has not been derived by the eventuality resolution rules, we consider the following cases. First of all, we can infer that not both parents $\tilde{C}_{1,i}$ and $\tilde{C}_{2,i}$ of the clause $\tilde{C}_i$ in the derivation $\tilde{\Delta}$ are tautologies as this would imply that $\tilde{C}_i$ is a tautology by Lemmata 4.2.9 and 4.2.10. If one of them is a tautology, i.e. without loss of generality the clause $\tilde{C}_{2,i}$ is not a tautology, we obtain that $\tilde{C}_{2,i} \leq_s \tilde{C}_i$ holds by Lemma 4.2.9. Thus, by using the induction hypothesis we obtain a clause $C_{2,i}$ derived by subsumption-compatible ordered fine-grained resolution with selection from the temporal problem $\mathsf{Cls}(\mathsf{P}^c)$ such that $C_{2,i} \leq_s \tilde{C}_{2,i} \leq_s \tilde{C}_i$ holds. Otherwise, neither the clause $\tilde{C}_{1,i}$ nor the clause $\tilde{C}_{2,i}$ are tautologies and we can derive a clause C_i with $C_i \leq_s \tilde{C}_i$ by subsumption-compatible ordered fine-grained resolution with selection from the temporal problem $\mathsf{Cls}(\mathsf{P}^c)$ by using the induction hypothesis and Lemma 4.2.23.

Finally, if a clause $\tilde{C}_i$ $(1 \leq i \leq m)$ is not contained in the temporal problem $\mathsf{Cls}(\mathsf{P}^c)$ and has been derived by one of the eventuality resolution rules applied on a loop formula $\tilde{H}(x)$, we first of all compute the saturation under the fine-grained step resolution rules of subsumption-compatible ordered fine-grained resolution with selection. If we obtain a contradiction, then we can extend the derivation Δ accordingly and define $C_i = \bot$. Otherwise, we can first of all infer that $\tilde{H}(x) \neq \mathbf{false}$ (as this would imply that $\tilde{C}_i$ is a tautology), and we add all the clauses computed during the saturation to the derivation Δ. We finally obtain the required clause C_i with $C_i \leq_s \tilde{C}_i$ by applying Lemma 4.3.19. □

4.4 Summary

The aim of this chapter was to provide a formal analysis of combining redundancy elimination with ordered fine-grained resolution with selection.

First, we focused on redundancy elimination in combination with the resolution-based inference rules of ordered fine-grained resolution with selection. We presented syntactic criteria for identifying tautologies among temporal clauses and we defined a subsumption relation on temporal clauses. We then described how the calculus had to be extended in order to remain compatible with the removal of subsumed clauses, resulting in the subsumption-compatible ordered fine-grained resolution with selection calculus. We also proved the subsumption lemmata for the subsumption-compatible calculus.

In the second part of the chapter we analysed the problem of combining redundancy elimination with the loop search process. We introduced a resolution-based loop search algorithm called Subsumption-Restricted-FG-BFS which eliminates subsumed clauses and tautologies during loop search computations. After having proved some of its properties, we showed the refutational completeness of subsumption-compatible ordered fine-grained resolution with selection where applications of the eventuality resolution rules are restricted to loops found by the Subsumption-Restricted-FG-BFS algorithm.

Chapter 5

Fair Derivations in Monodic Temporal Reasoning

5.1 Introduction

Having focused on theoretical aspects of the $\mathfrak{I}^{S,\succ}_{FG}$- and $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculi until now, this chapter is more geared towards analysing the behaviour of subsumption-compatible ordered fine-grained resolution with selection in practice. It has turned out that some obstacles can be encountered when it comes to constructing fair derivations for the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus in practice, i.e. in an automated theorem prover, for example.

Broadly speaking, the inference rules of (subsumption-compatible) ordered fine-grained resolution with selection can be classified into two different categories. The majority of the rules are based on standard first-order resolution between different types of temporal clauses. The remaining inference rules reflect the induction principle that holds for monodic temporal logic over a flow of time isomorphic to the natural numbers. The applicability of the rules in this second category is only semi-decidable. Consequently, fair derivations, i.e. derivations in which every non-redundant clause that is derivable from a given clause set is eventually derived, cannot be guaranteed in practice as the applicability check for an inference rule of the second category might not terminate. But as the ability to construct fair derivations is an essential requirement for maintaining the refutational completeness of an automated theorem prover, we have to look for ways to overcome the fairness problems of the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus if we want to develop a fair prover for monodic first-order temporal logic which is based on $\mathfrak{I}^{S,\succ}_{FG,Sub}$.

In this chapter we therefore present an inference procedure that can construct fair derivations for reasoning in monodic first-order temporal logic, and we prove its refutational completeness. The new inference mechanism is based on subsumption-compatible ordered fine-grained resolution with selection. We also show that the new inference mechanism can also be used as a decision procedure for some specific classes of temporal problems.

We proceed as follows: after having illustrated the fairness problems related to the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus in the beginning of Section 5.2, we introduce the new inference procedure which can guarantee fair derivations in practice. We conclude the chapter by proving the refutational completeness of the fair inference mechanism in Section 5.3 and we show that it can be used as a decision procedure for some specific monodic temporal problems in Section 5.4.

5.2 Constructing Fair Derivations

We start by examining the fairness problems of the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus.

5.2.1 Fairness Problems

As stated in Theorem 4.3.20, any fair derivation by subsumption-compatible ordered fine-grained resolution with selection from an unsatisfiable clausified monodic temporal problem will eventually include a monodic temporal problem containing the empty clause. However, due to the presence of the ground and non-ground eventuality resolution rules in our calculus, constructing a fair derivation is a non-trivial problem. The validity of the side conditions of loop formulae, i.e. $\forall x(\mathcal{U} \wedge \mathcal{B}_i(x) \Rightarrow \neg L(x))$ and $\forall x(\mathcal{U} \wedge \mathcal{B}_i(x) \Rightarrow \bigvee_{j=1}^{n} \mathcal{A}_j(x))$ in the non-ground case, is only semi-decidable. Thus, the construction of a derivation could potentially 'get stuck' while checking these side conditions. For example, if we clausify the unsatisfiable problem

$$\begin{aligned}\mathsf{P} = \langle\{&P(c,c),\\ &\forall x\exists y\forall v\forall w(Q(y,x) \wedge (P(x,v) \Rightarrow P(y,w)))\},\\ &\{R(c)\},\\ &\{R(x) \Rightarrow \bigcirc R(x), R(x) \Rightarrow \bigcirc\neg L(x)\},\\ &\{\Diamond L(x)\}\rangle\end{aligned}$$

in DSNF, then we obtain the following clausified temporal problem:

$$\begin{aligned}\mathsf{Cls}(\mathsf{P}) = \langle\{&P(c,c), Q(f(x),x),\\ &\neg P(x,v) \vee P(f(x),w)\},\\ &\{R(c)\},\\ &\{R(x) \Rightarrow \bigcirc R(x), R(x) \Rightarrow \bigcirc\neg L(x),\\ &\ R(c) \Rightarrow \bigcirc R(c), R(c) \Rightarrow \bigcirc\neg L(c)\},\\ &\{\Diamond L(x)\}\rangle,\end{aligned}$$

where f is a unary function symbol introduced during Skolemization. But the temporal problem $\mathsf{Cls}(\mathsf{P})$ possesses an infinite saturation on universal clauses as universal unit clauses of the form $P(f^k(x),v)$ can be derived for any $k \in \mathbb{N}$. Thus, an attempt at determining whether the formula $\forall x(\mathcal{U} \wedge R(x) \Rightarrow \neg L(x))$ is valid, where $\mathcal{U}$ is the set of universal clauses and $R(x)$ the right-hand side of the step clause $R(x) \Rightarrow \bigcirc R(x)$, might not terminate.

```
Function Subsumption-Restricted-FG-BFS
Input:   A set of universal clauses $\mathcal{U}$ and a set of step clauses $\mathcal{S}$, saturated under
         the fine-grained step resolution inference rules of subsumption-compatible
         ordered fine-grained resolution with selection, and an eventuality clause
         $\Diamond L(x) \in \mathcal{E}$.
Output:  A formula $R(x)$ with at most one free variable.
Method:  (1) Let $R_0(x) = \textbf{true}$; $M_0 = \emptyset$; $i = 0$
         (2) Let $\mathcal{N}'_{i+1} = \mathcal{U} \cup \mathrm{LT}(\mathcal{S}) \cup \{\textbf{true} \Rightarrow \bigcirc(\neg R_i(c^l) \vee L(c^l))\}$. Apply the
             fine-grained step resolution rules of subsumption-compatible ordered
             fine-grained resolution with selection except the clause conversion
             rule to $\mathcal{N}'_{i+1}$, together with the removal of tautological and subsumed
             clauses. If we obtain a contradiction, then return the loop true (in
             this case $\forall x \neg L(x)$ is implied by the universal part).
             Otherwise let $\mathcal{M}'_{i+1} = \{D_j \Rightarrow \bigcirc\bot\}_{j=1}^{n}$ be the set of all new termi-
             nating step clauses in the saturation of $\mathcal{N}'_{i+1}$.
         (3) If $\mathcal{M}'_{i+1} = \emptyset$, return false; else let $R_{i+1}(x) = \bigvee_{j=1}^{n}(\tilde{\exists} D_j)\{c^l \rightarrow x\}$
         (4) If $\forall x(R_i(x) \Rightarrow R_{i+1}(x))$, return $R_{i+1}(x)$.
         (5) i = i + 1; goto 2.
```

Figure 5.1: Breadth-First Search using Subsumption-Compatible Ordered Fine-Grained Step Resolution with Selection together with Redundancy Elimination

One might try to overcome this problem based on the following observation. While additional step clauses can be inferred in a derivation, these need not be used as premises for the ground and the non-ground eventuality resolution rule. Instead it is sufficient to only use step clauses in the original monodic temporal problem for the construction of merged derived step clauses and full-merged step clauses. Thus, there is actually a finite and static set of potential premises for any application of the ground and the non-ground eventuality resolution rule during a derivation. Hypothetically, the validity of the side conditions of each of these applications could be checked in parallel, allowing us to proceed with those applications where the side conditions hold. In the following we refer to this approach as *parallel exhaustive loop test*. What makes this approach impractical is the fact that while the set of premises is static and finite, the set of universal clauses $\mathcal{U}$ involved in the side conditions of the rules in questions is not. In particular, applications of the clause conversion, ground and non-ground eventuality rules may extend $\mathcal{U}$ in such a way that a previously invalid side condition becomes valid. Thus, each time the set $\mathcal{U}$ is extended we have to re-check the applicability of the ground and non-ground eventuality resolution rules to each potential set of premises.

The use of the Subsumption-Restricted-FG-BFS algorithm to systematically search for (ground) loop formulae does not solve the problem related to the semi-decidability of the side conditions used in the eventuality resolution rules. In step (2) of the algorithm we need to saturate the set of universal and step clauses $\mathcal{N}'_{i+1}$ using the fine-grained step resolution rules except the clause conversion rule. This saturation process may not terminate even if no loop formula exists for a given eventuality clause $\Diamond L(x) \in \mathcal{E}$ or $\Diamond l \in \mathcal{E}$ and the set of current universal and step clauses. Thus, the Subsumption-Restricted-FG-BFS algorithm also cannot guarantee fairness.

If one tries to solve the fairness problem by delaying the application of the eventuality resolution rules as long as possible, then one faces the problem that the saturation process under the rules of fine-grained step resolution may not terminate even if the original monodic temporal problem is unsatisfiable. Consequently, the strategy of executing the Subsumption-Restricted-FG-BFS algorithm only after the original temporal problem has been saturated under fine-grained resolution may still lead to unfairness.

5.2.2 The Fair Inference Procedure $\mathbb{F}$

We can thus see that achieving fairness in derivations is not a trivial task and that it can only be accomplished if the two potentially non-terminating types of saturations, which are the regular saturation under subsumption-compatible ordered fine-grained step resolution with selection on one hand, and the saturations required for loop search on the other hand, are not executed sequentially. We hence propose a way of combining these two types of saturations into one 'global' saturation process.

The first step towards a procedure which guarantees the construction of a fair derivation is based on an idea introduced in [41] for improving the efficiency of loop search in propositional linear-time temporal logic. It suggests a minor modification of the Subsumption-Restricted-FG-BFS algorithm. In step (2) of the modified algorithm we now add the clauses which result from clausification of the formula $C^L_{i+1} = s^L_{i+1} \Rightarrow \bigcirc(\neg H_i(c^l) \vee L(c^l))$ to $\mathcal{N}'_{i+1}$, where s^L_{i+1} is a proposition uniquely associated with index $i+1$ and the eventuality clause $\Diamond L(x)$ for which we search for a loop. The proposition s^L_{i+1} acts as a marker for these clauses which are generated purely as a means to conduct the search. As there are only occurrences of s^L_{i+1} in left-hand sides of step clauses, the application of inference rules to these clauses will 'propagate' the literal s^L_{i+1} to all clauses we derive from C^L_{i+1}. This also means that $\mathcal{M}'_{i+1}$ can now be defined as the set of all clauses of the form $s^L_{i+1} \wedge C_j \Rightarrow \bigcirc\bot$. While this makes the construction of $\mathcal{M}'_{i+1}$ operationally easier compared to the original version of Subsumption-Restricted-FG-BFS, it does not fundamentally change the algorithm. However, this small change allows us to take advantage of the following observations. Within iterations of the steps (2)–(5) of the modified algorithm, the clauses in the various sets $\mathcal{N}'_{i+1}$ are now separated by the 'marker' s^L_{i+1}. Thus, instead of using different sets $\mathcal{N}'_{i+1}$ we can use a single set $\mathcal{T}$ which is simply extended in each iteration of the steps (2)–(5). Furthermore, we can keep the set $\mathcal{T}$ between separate calls of the modified Subsumption-Restricted-FG-BFS procedure for different eventualities but also between repeated calls of the modified Subsumption-Restricted-FG-BFS algorithm for the same eventuality clause $\Diamond L(x) \in \mathcal{E}$. Finally, if we restrict the clause conversion rule so that it cannot be applied to any clause containing a 'marker' s^L_i, then there is no reason to separate the clauses in $\mathcal{T}$ from those in the current monodic temporal problem in clausified form stored by the prover. Figure 5.2

Initialization

$\langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle \Longrightarrow \mathcal{N} \,|\, \emptyset \,|\, \emptyset$ where

$$\mathcal{N} = \mathcal{U} \cup \mathcal{I} \cup \mathcal{S} \cup \{ P(c^l) \Rightarrow \bigcirc M(c^l) \mid P(x) \Rightarrow \bigcirc M(x) \in \mathcal{S} \}$$
$$\cup \{ s_0^L \Rightarrow \bigcirc L(c^l) \mid \Diamond L(x) \in \mathcal{E} \}$$

Tautology Deletion

$\mathcal{N} \cup \{\mathcal{C}\} \,|\, \mathcal{P} \,|\, \mathcal{O} \Longrightarrow \mathcal{N} \,|\, \mathcal{P} \,|\, \mathcal{O}$ if $\mathcal{C}$ is a tautology

Forward Subsumption

$\mathcal{N} \cup \{\mathcal{C}\} \,|\, \mathcal{P} \,|\, \mathcal{O} \Longrightarrow \mathcal{N} \,|\, \mathcal{P} \,|\, \mathcal{O}$ if some clause in $\mathcal{P} \cup \mathcal{O}$ subsumes $\mathcal{C}$

Backward Subsumption

$\mathcal{N} \,|\, \mathcal{P} \cup \{\mathcal{C}\} \,|\, \mathcal{O} \Longrightarrow \mathcal{N} \,|\, \mathcal{P} \,|\, \mathcal{O}$
$\mathcal{N} \,|\, \mathcal{P} \,|\, \mathcal{O} \cup \{\mathcal{C}\} \Longrightarrow \mathcal{N} \,|\, \mathcal{P} \,|\, \mathcal{O}$ if some clause in $\mathcal{N}$ properly subsumes $\mathcal{C}$

Clause Processing

$\mathcal{N} \cup \{\mathcal{C}\} \,|\, \mathcal{P} \,|\, \mathcal{O} \Longrightarrow \mathcal{N} \,|\, \mathcal{P} \cup \{\mathcal{C}\} \,|\, \mathcal{O}$ if none of the previous rules applies

Loop Search Contradiction

$\emptyset \,|\, \mathcal{P} \cup \{ s_i^L \Rightarrow \bigcirc\bot \} \,|\, \mathcal{O} \Longrightarrow \{\bot\} \,|\, \mathcal{P} \,|\, \mathcal{O} \cup \{ s_i^L \Rightarrow \bigcirc\bot \}$ for some i, L

Next Loop Search Iteration

$\emptyset \,|\, \mathcal{P} \cup \{ s_i^L \wedge C \Rightarrow \bigcirc\bot \} \,|\, \mathcal{O} \Longrightarrow$
$\{ s_{i+1}^L \Rightarrow \bigcirc\neg C \vee L(c^l) \} \,|\, \mathcal{P} \,|\, \mathcal{O} \cup \{ s_i^L \wedge C \Rightarrow \bigcirc\bot \}$ for some i, L and $C \neq \emptyset$

Clause Conversion

$\emptyset \,|\, \mathcal{P} \cup \{ C \Rightarrow \bigcirc\bot \} \,|\, \mathcal{O} \Longrightarrow \{ \Box\neg C \} \,|\, \mathcal{P} \,|\, \mathcal{O} \cup \{ C \Rightarrow \bigcirc\bot \}$ where no $s_i^L \in C$

Regular Inference Computation

$\emptyset \,|\, \mathcal{P} \cup \{\mathcal{C}\} \,|\, \mathcal{O} \Longrightarrow \mathcal{N} \,|\, \mathcal{P} \,|\, \mathcal{O} \cup \{\mathcal{C}\}$ if none of the previous rule applies and where $\mathcal{N} = \mathrm{Res}_{\mathrm{Sub}}(\mathcal{C}, \mathcal{O})$

Loop Testing

$\emptyset \,|\, \mathcal{P} \,|\, \mathcal{O} \Longrightarrow \mathcal{N} \,|\, \mathcal{P} \,|\, \mathcal{O}$ where

$$\mathcal{N} = \{ \Box\forall x \neg H_{i+1}^L(x) \mid \text{for all } i,\ L \text{ with } \models \forall x (H_i^L(x) \Leftrightarrow H_{i+1}^L(x)) \}$$

and $H_i^L(x) := \bigvee \{ (\tilde{\exists} C_j)\{c^l \to x\} \mid s_i^L \wedge C_j \Rightarrow \bigcirc\bot \in \mathcal{P} \cup \mathcal{O} \}$ for all $i,\ L$

Figure 5.2: Fair Inference Procedure **F**

depicts the inference procedure **F** based on these considerations in the presentation style of [10]. The inference procedure operates on states $(\mathcal{N} \,|\, \mathcal{P} \,|\, \mathcal{O})$ that are constructed from an initial temporal problem $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$.

A state $(\mathcal{N} \,|\, \mathcal{P} \,|\, \mathcal{O})$ consists of three sets of clauses $\mathcal{N}$ (the set of *new* clauses), $\mathcal{P}$ (the set of clauses that still have to be *processed*) and $\mathcal{O}$ (the set of *old* clauses). The set $\mathcal{N}$ collects the newly-derived clauses and the set $\mathcal{O}$ contains all the clauses that have already been used as premises in inference steps (or can never be used as premises). Finally, the set $\mathcal{P}$ contains all the clauses that still need to be considered as premises. In the initial state

$(\mathcal{N}_0 \mid \emptyset \mid \emptyset)$ constructed by the 'initialization' rule, the sets $\mathcal{P}$, $\mathcal{O}$ are empty and the set $\mathcal{N}_0$ contains all the clauses contained in a temporal problem $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$. Additionally, as motivated above, all the clauses required for loop search are added to $\mathcal{N}_0$. Subsequent states are obtained by applying one of the other inference rules depicted in Figure 5.2 on a given state.

The rules 'tautology deletion', 'forward subsumption' and 'backward subsumption' perform reductions on the clause sets. 'Tautology deletion' removes tautological clauses from set of newly-derived clauses $\mathcal{N}$. 'Forward subsumption' and 'backward subsumption' eliminate clauses that have been subsumed by other clauses. Finally, the 'clause processing' rule is responsible for moving a clause that has survived the previous reduction steps to the set $\mathcal{P}$. Once no further reductions are possible, additional clauses can be derived by the following inference rules.

For the 'loop search contradiction' rule note that the presence of $s_i^L \Rightarrow \bigcirc\bot$ in $\mathcal{P}$ indicates that we can apply the non-ground eventuality resolution rule, resulting in the conclusion $\Box\forall x \neg\top$, which is contradictory. The empty clause is then added as set $\mathcal{N}$ and the clause $s_i^L \Rightarrow \bigcirc\bot$ is moved to the set of old clauses $\mathcal{O}$. If the set $\mathcal{P}$ contains a clause $s_i^L \wedge C \Rightarrow \bigcirc\bot$ for some i, L and $C \neq \emptyset$, then such a clause would be part of the set $\mathcal{N}_i$ in the Subsumption-Restricted-FG-BFS procedure, which is used to define the formula $H_{i+1}(x)$, which in turn is used to define the clauses in $\mathcal{M}'_{i+1}$. Here, we directly define the one clause from $\mathcal{M}'_{i+1}$ which derives from $s_i^L \wedge C \Rightarrow \bigcirc\bot$ and add it as newly-derived clause set. Finally, if a clause $C \Rightarrow \bigcirc\bot$ (without a marker s_i^L) is contained in the set $\mathcal{P}$, then such a clause is a suitable premise for the application of the clause conversion rule and we add the universal clause $\Box\neg C$ as newly-derived clause set. The clause $C \Rightarrow \bigcirc\bot$ is moved to the set $\mathcal{O}$.

In the case where the set $\mathcal{P}$ contains a clause C that is not handled by one of the previous rules, we compute the set $\mathrm{Res}_{\mathrm{Sub}}(C, \mathcal{O})$, which consists of all the conclusions derivable from the clause C with itself and with the clauses in $\mathcal{O}$ by the step resolution inference rules of $\mathfrak{I}^{S,\succ}_{FG,Sub}$ except the clause conversion rule. The computed clauses are then added to the next state as set $\mathcal{N}$ and the clause C is moved to the set of old clauses $\mathcal{O}$. The remaining rule 'loop testing' is responsible for checking the loop side condition. First, the formulae H_i^L are computed for all eventuality clauses $\Diamond L(x) \in \mathcal{E}$ and all indices i used to create some marker s_i^L in the set $\mathcal{P} \cup \mathcal{O}$. We then check whether the loop condition $\forall x(H_i^L(x) \Leftrightarrow H_{i+1}^L(x))$ holds for every i and every L. If so, an application of the non-ground eventuality resolution rule is possible. We compute the conclusion of the application and add it to the set $\mathcal{N}$. This concludes the description of the fair inference procedure.

There are three important observations to be made about the 'loop testing' rule. First, we can observe that $H_i^L(x)$ and $H_{i+1}^L(x)$ are monadic first-order formulae. Thus, the validity of the loop condition is a decidable problem. Second, in Subsumption-Restricted-FG-BFS, in order to establish whether $\forall x(H_i^L(x) \Leftrightarrow H_{i+1}^L(x))$ is valid we only need to test whether $\forall x(H_i^L(x) \Rightarrow H_{i+1}^L(x))$ holds as the implication $\forall x(H_{i+1}^L(x) \Rightarrow H_i^L(x))$ is always valid by the construction of $H_i^L(x)$ and $H_{i+1}^L(x)$ in these procedures. However, in the context of the inference procedure $\mathbf{F}$ this is no longer the case and we need to test both implications.

Finally, whenever the loop condition holds, we have indeed found a loop formula, although it may not be equivalent to a formula returned by Subsumption-Restricted-FG-BFS. We will see that eventually an analogous formula (w.r.t. to the negation of the loop formula) will be computed by the procedure $\mathbb{F}$.

5.3 Refutational Completeness

We now prove the refutational completeness of the fair inference procedure. Before we can state the completeness theorem, we have to prove several auxiliary lemmata.

First of all, we define the notion of a *fair* derivation produced by the inference procedure $\mathbb{F}$. For the purpose of the completeness proof we assume that the inference procedure does not necessarily terminate whenever the empty clause has been derived but continues to derive clauses instead.

Definition 5.3.1 (Derivation). *A derivation Δ produced by the inference procedure $\mathbb{F}$ shown in Figure 5.2 from a temporal problem $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ is a sequence of states $\langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle \Longrightarrow \mathcal{N}_0 \,|\, \mathcal{P}_0 \,|\, \mathcal{O}_0 \Longrightarrow \mathcal{N}_1 \,|\, \mathcal{P}_1 \,|\, \mathcal{O}_1 \Longrightarrow \mathcal{N}_2 \,|\, \mathcal{P}_2 \,|\, \mathcal{O}_2 \Longrightarrow \ldots$ where each state $(\mathcal{N}_i \,|\, \mathcal{P}_i \,|\, \mathcal{O}_i)$, $i \geq 0$, results from an application of an inference rule shown in Figure 5.2 on the state $(\mathcal{N}_{i-1} \,|\, \mathcal{P}_{i-1} \,|\, \mathcal{O}_{i-1})$ if and only if $i > 0$ or on $\langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ if $i = 0$.*

If $\mathcal{N}_i = \emptyset$ and $\mathcal{P}_i = \emptyset$ for an index $i \in \mathbb{N}$, we define $(\mathcal{N}_i \,|\, \mathcal{P}_i \,|\, \mathcal{O}_i) = (\mathcal{N}_j \,|\, \mathcal{P}_j \,|\, \mathcal{O}_j)$ for every $j \geq i$.

A derivation Δ is said to be fair[1] *if and only if $\bigcup_{i=0}^{\infty} \bigcap_{j \geq i}^{\infty} \mathcal{P}_j = \emptyset$ and, whenever possible, every application of the 'regular inference computation' rule is eventually followed by an application of the 'loop testing' rule.*

We now prove for every clause $\mathcal{D}$ contained in a set $\mathcal{P}_i \cup \mathcal{O}_i$ for $i \geq 0$ that there exists an index $k \in \mathbb{N}$ and a clause $\mathcal{C}$ such that $\mathcal{C} \leq_s \mathcal{D}$ and $\mathcal{C} \in \mathcal{O}_j$ for every $j \geq k$.

Lemma 5.3.2. *Let $\Delta = (\mathcal{N}_i \,|\, \mathcal{P}_i \,|\, \mathcal{O}_i)_{i \in \mathbb{N}}$ be a fair derivation produced by the inference procedure $\mathbb{F}$ shown in Figure 5.2. Furthermore, let $\mathcal{D}$ be a initial, universal or step clause such that $\mathcal{D} \in \mathcal{P}_i \cup \mathcal{O}_i$ for $i \geq 0$. Then there exists a clause $\mathcal{C}$ and an index $j \in \mathbb{N}$ such that $j \geq i$, $\mathcal{C} \in \bigcap_{k \geq j} \mathcal{O}_k$ and $\mathcal{C} \leq_s \mathcal{D}$.*

Proof. If $\mathcal{D} \in \mathcal{P}_i$, then as the derivation Δ is fair, it follows that the clause $\mathcal{D}$ will eventually be extracted from a set $\mathcal{P}_l$ for $l \geq i$. Thus, there either exists a clause $\tilde{\mathcal{D}} \in \mathcal{N}_l$ with $\tilde{\mathcal{D}} <_s \mathcal{D}$ or the clause $\mathcal{D}$ is moved to the set $\mathcal{O}_l$. As there are only finitely many clauses $\tilde{\mathcal{D}}'$ with $\tilde{\mathcal{D}}' <_s \tilde{\mathcal{D}}$, there exists a clause $\tilde{\mathcal{D}}' \in \mathcal{P}_{l'}$ with $\tilde{\mathcal{D}}' <_s \mathcal{D}$ for an index $l' \in \mathbb{N}$ that is not removed by backward subsumption in the derivation Δ. As the derivation Δ is fair, there hence exists a clause $\tilde{\mathcal{D}}'' \in \mathcal{O}_{l''}$ with $\tilde{\mathcal{D}}'' <_s \mathcal{D}$ for an index $l'' \in \mathbb{N}$. We may therefore assume without loss of generality that $\mathcal{D} \in \mathcal{O}_i$ (otherwise we consider the clause $\mathcal{D} \in \mathcal{O}_l$ or $\tilde{\mathcal{D}}'' \in \mathcal{O}_{l''}$).

[1] Despite its more complex appearance, the definition of fairness for the inference procedure $\mathbb{F}$ follows the same lines as the Definitions 3.2.7 and 3.3.3, i.e. every inference that is possible at a given index will be performed eventually.

Let $\mathcal{N} = \{\mathcal{C} \in \bigcup_{k=i}^{\infty} \mathcal{O}_i \mid \mathcal{C} \leq_s \mathcal{D}\}$. By Lemma 4.2.6, the relation $<_s$ is well-founded. Thus, as $\mathcal{N} \neq \emptyset$, there exists a minimal clause $\mathcal{C} \in \mathcal{N}$ with respect to the relation $<_s$. Hence, let $\mathcal{C} \in \mathcal{O}_j$ for an index $j \geq i$. We now prove by induction that $\mathcal{C} \in \bigcap_{k \geq j} \mathcal{O}_k$. For $k = j$ there remains nothing to be shown. If $k > j$, then it follows from the induction hypothesis that $\mathcal{C} \in \mathcal{O}_{k-1}$. If we now assume that $\mathcal{C} \notin \mathcal{O}_k$, then it would follow that the clause $\mathcal{C}$ has been deleted by backward subsumption, i.e. it can be shown that there exists a clause $\mathcal{C}' \in \mathcal{O}_{k'}$ with $\mathcal{C}' <_s \mathcal{C}$ for a $k' \geq k$. We can conclude that $\mathcal{C}' <_s \mathcal{D}$ by transitivity of the relation $<_s$. Thus, $\mathcal{C}' \in \mathcal{N}$ and $\mathcal{C}' <_s \mathcal{C}$, which contradicts the minimality of the clause $\mathcal{C}$. □

The proof of refutational completeness for the fair inference architecture **F** is based on showing that for every non-tautological clause $\mathcal{D}$ that is contained in a refutation Δ by $\Im^{S,\succ}_{FG,Sub}$ there exists a clause $\mathcal{C}$ derived by the fair inference procedure such that $\mathcal{C} \leq_s \mathcal{D}$ holds. Unsurprisingly, it will be most difficult to show that clauses resulting from applications of the Subsumption-Restricted-FG-BFS algorithm are subsumed by the fair inference procedure.

The next lemma shows that non-tautological clauses which do not originate from loop search are subsumed by clauses derived by the fair inference procedure.

Lemma 5.3.3. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a clausified monodic temporal problem and let $\Delta = (\mathcal{N}_i \mid \mathcal{P}_i \mid \mathcal{O}_i)_{i \in \mathbb{N}}$ be a fair derivation produced by the inference procedure* **F** *shown in Figure 5.2 from the temporal problem P. Let $\Delta' = \mathcal{D}_1, \ldots, \mathcal{D}_n$ be a derivation produced by subsumption-compatible ordered fine-grained resolution with selection from a temporal problem $P' = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E} \rangle$ without applying the eventuality resolution rules and such that for every clause $\mathcal{C}_2 \in \mathcal{U}' \cup \mathcal{I}' \cup \mathcal{S}'$ with $\mathcal{C}_2 \notin \mathrm{taut}(P')$ there exists a clause $\mathcal{C}_1 \in \bigcup_{i=0}^{\infty} \mathcal{O}_i$ with $\mathcal{C}_1 \leq_s \mathcal{C}_2$.*

Then it holds for every i with $1 \leq i \leq n$ that

$$\mathcal{D}_i \notin \mathrm{taut}(P') \Rightarrow \exists j \in \mathbb{N}\, \exists \mathcal{C}_i \in \mathcal{O}_j : \mathcal{C}_i \leq_s \mathcal{D}_i$$

Proof. By induction on i using the fairness of the derivation Δ and Lemma 4.2.23. □

Essentially, the remaining lemmata are concerned with proving that clauses derived by the eventuality resolution rules will eventuality be subsumed by clauses derived by the fair inference procedure. The main difficulty lies in the fact that the order in which the terminating step clauses for the loop search process are derived is no longer guaranteed to be the same as for the Subsumption-Restricted-FG-BFS algorithm.

We now prove that for any derivation of the fair inference procedure there exists an iteration index which contains all the universal clauses that can ever be derived by either loop search or the clause conversion rule. The underlying reason behind this observation is again the finite number of step clauses that are contained in the original temporal problem. First, we define some auxiliary notation.

Definition 5.3.4. *Let $\mathcal{N}$ be a set of initial, universal or step clauses. Then we denote by* $\mathrm{Univ}(\mathcal{N})$ *the set of all the* universal *clauses contained in the set $\mathcal{N}$, and by* $\mathrm{Step}(\mathcal{N})$ *the set of all the* step *clauses from the set $\mathcal{N}$ which do not contain a loop search marker.*

Lemma 5.3.5. *Let $P = \langle \mathcal{U}'_0, \mathcal{I}', \mathcal{S}', \mathcal{E} \rangle$ be a clausified monodic temporal problem. Additionally, let $\Delta = (\mathcal{N}_i \mid \mathcal{P}_i \mid \mathcal{O}_i)_{i \in \mathbb{N}}$ be a derivation produced by the inference procedure* **F** *shown in Figure 5.2 from the temporal problem P. Then there exists an index $I \in \mathbb{N}$ such that:*

$$\forall C \in \mathrm{Univ}(\bigcup_{j=I}^{\infty} (\mathcal{P}_j \cup \mathcal{O}_j)) \colon C \in \mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathcal{P}_I \cup \mathcal{O}_I)$$

Proof. The only universal clauses (different from the empty clause) which are added to a set $\mathcal{P}_k \cup \mathcal{O}_k$ ($k \in \mathbb{N} \setminus \{0\}$) and which are possibly not contained in the set $\mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathcal{P}_{k-1} \cup \mathcal{O}_{k-1})$ result from loop formulae or from applications of the clause conversion rule. Each of these formulae stems from combining the left-hand sides of some step clauses in $\mathcal{S}'$. As there are only finitely many non-equivalent such combinations w.r.t. the relation $=_X$ that are free of duplicate atoms and which are such that each subset of the non-ground atoms occurring in the left-hand sides of the step clauses contained in the set $\mathcal{S}'$ occurs at most once in them (see also Definition 4.3.1) and as subsumed clauses are removed, it is easy to see that there exists an index $I \in \mathbb{N}$ such that every universal clause $C \in \mathcal{P}_j \cup \mathcal{O}_j$ with $j \geq i$ has been derived by the resolution or factoring-based rules from clauses contained in $\mathcal{P}_I \cup \mathcal{O}_I$. □

Remark 5.3.6. *The index $I \in \mathbb{N}$ obtained through Lemma 5.3.5 will be called the* universal clause termination index.

The next two lemmata link terminating step clauses together that have been derived by the fair inference procedure and the Subsumption-Restricted-FG-BFS algorithm.

Lemma 5.3.7. *Let $P' = \langle \mathcal{U}'_0, \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$ be a clausified temporal problem. Additionally, let $\Delta = (\mathcal{N}_i \mid \mathcal{P}_i \mid \mathcal{O}_i)_{i \in \mathbb{N}}$ be a fair derivation produced by the inference procedure* **F** *shown in Figure 5.2 from the temporal problem P'. Furthermore, let $\Diamond L(x) \in \mathcal{E}$ be an eventuality and let $\mathcal{M}'_0, \mathcal{M}'_1, \ldots$ be the sets of (all the) terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied for the eventuality $\Diamond L(x) \in \mathcal{E}$ on the temporal problem $\langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$, where $\mathcal{U}'$ is a finite set of universal clauses with $\mathcal{U}' \leq_s \mathcal{U}'_0 \setminus \mathrm{taut}(\mathcal{U}'_0)$ and $\bigcup_{j=0}^{\infty} \mathcal{O}_j \leq_s \mathcal{U}' \setminus \mathrm{taut}(\mathcal{U}')$.*

Then it holds for every iteration index $i \in \mathbb{N}$ that:

$$\forall C \Rightarrow \bigcirc\bot \in \mathcal{M}'_i \; \exists k \in \mathbb{N} \; \exists D \in \mathcal{O}_k \colon D \leq_s s_i^L \wedge C \Rightarrow \bigcirc\bot$$

Proof. Similar to the proof of Lemma 5.3.3 and by using the properties of the loop search literals s_i^L (for $i \in \mathbb{N}$ and $\Diamond L(x) \in \mathcal{E}$). □

Lemma 5.3.8. *Let $P' = \langle \mathcal{U}'_0, \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$ be a clausified monodic temporal problem. Additionally, let $\Delta = (\mathcal{N}_i \mid \mathcal{P}_i \mid \mathcal{O}_i)_{i \in \mathbb{N}}$ be a fair derivation produced by the inference procedure* **F** *shown in Figure 5.2 from the temporal problem P' and let I be its universal clause termination index. Furthermore, let $\Diamond L(x) \in \mathcal{E}$ be an eventuality and let $\mathcal{M}'_0, \mathcal{M}'_1, \ldots, \mathcal{M}'_i, \mathcal{M}'_{i+1} \cdots$ be the sets of (all the) terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied for the eventuality $\Diamond L(x) \in \mathcal{E}$ on the temporal problem $\langle \mathrm{Univ}(\mathcal{P}_I \cup \mathcal{O}_I), \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$.*

Then it holds for every $i \in \mathbb{N}$ and for every derivation index $k \in \mathbb{N}$ that:

$$\forall\, s_i^L \wedge C \Rightarrow \bigcirc\bot \in \mathcal{N}_k \cup \mathcal{P}_k \cup \mathcal{O}_k \;\exists D \in \mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathrm{Univ}(\mathcal{P}_I \cup \mathcal{O}_I) \cup \mathcal{S}') \cup \mathcal{M}'_i : \qquad D \leq_s C \Rightarrow \bigcirc\bot$$

Proof. By induction on i using Lemmata 4.2.23, 5.3.2 and by using the fact that every universal clause that is contained in $\mathrm{Univ}(\mathcal{P}_k \cup \mathcal{O}_k)$ with $k \geq I$ can be derived from universal clauses in $\mathrm{Univ}(\mathcal{P}_I \cup \mathcal{O}_I)$ without using the clause conversion rule. □

The next lemma establishes that there exists an index J in any derivation of the fair inference procedure after which no new terminating loop search clauses are derived for a given eventuality and a given loop search iteration. More specifically, every terminating step loop search clause for a given eventuality and a given loop search iteration which appears at some index $k \geq J$ (with $\mathcal{N}_k = \emptyset$) in a derivation of the fair inference procedure **F** has already been derived at the index J. Additionally, each considered terminating loop search clause that appears at the index J is not removed at a later index $k' \geq J$ (with $\mathcal{N}_{k'} = \emptyset$). The underlying reason for the existence of these indices lies again in the finite number of step clauses that are contained in a temporal problem.

Lemma 5.3.9. *Let $P' = \langle \mathcal{U}'_0, \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$ be a clausified temporal problem. Additionally, let $\Delta = (\mathcal{N}_i \,|\, \mathcal{P}_i \,|\, \mathcal{O}_i)_{i \in \mathbb{N}}$ be a fair derivation produced by the inference procedure **F** shown in Figure 5.2 from the temporal problem P', let $\Diamond L(x) \in \mathcal{E}$ be an eventuality and $i \in \mathbb{N}$ be a loop search iteration index.*

Then there exists an index $J \in \mathbb{N}$ such that $\mathcal{N}_J = \emptyset$ and

(i) *for all $k \in \mathbb{N}$ with $k \geq J$ and $\mathcal{N}_k = \emptyset$, and for every terminating step clause $s_i^L \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_k \cup \mathcal{O}_k$ it holds that $s_i^L \wedge C \Rightarrow \bigcirc\bot \in_X \mathcal{P}_J \cup \mathcal{O}_J$; and*

(ii) *for every terminating step clause $s_i^L \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_J \cup \mathcal{O}_J$ it holds that $s_i^L \wedge C \Rightarrow \bigcirc\bot \in_X \mathcal{P}_k \cup \mathcal{O}_k$ for every $k \geq J$ with $\mathcal{N}_k = \emptyset$.*

Proof. First of all, we note that for a terminating step clause of the form

$$s_i^L \wedge \bigwedge_{i=1}^{m_1} P_i(x) \wedge \bigwedge_{j=1}^{m_2} R_j(y) \wedge E \Rightarrow \bigcirc\bot$$

where $P_i(x) \Rightarrow \bigcirc Q_i(x) \in_X \mathcal{S}'$ for every i, $1 \leq i \leq m_1$, and $R_j(y) \Rightarrow \bigcirc S_j(y) \in_X \mathcal{S}'$ for every j, $1 \leq j \leq m_2$, such that $var(E) \cap \{x, y\} = \emptyset$ and $\{P_1(x), \ldots, P_{m_1}(x)\} \subseteq \{R_1(x), \ldots, R_{m_2}(x)\}$, an arbitrary factoring inference on two atoms $P_i(x)$ $(1 \leq i \leq m_1)$ and $R_j(x)$ $(1 \leq j \leq m_2)$ from its left-hand side with a substitution $\sigma = [x \mapsto y]$ yields the terminating step clause (after eliminating duplicate atoms)

$$s_i^L \wedge \bigwedge_{j=1}^{n} R_j(y) \wedge E \Rightarrow \bigcirc\bot,$$

which subsumes the former terminating step clause.

Then, as the number of (ground and non-ground) step clauses contained in the set S' is finite, there only exist finitely many (maximal) subformulae of the form $\bigwedge_{i=1}^{l} P_i(x)$ that do not contain duplicate atoms in the left-hand sides of step clauses. Additionally, as subsumed clauses are removed in the derivation Δ and as the removal of duplicate literals derives clauses that subsume their parent clauses, we can infer that there only exist finitely many different terminating step clauses of the form $s_i^L \wedge C \Rightarrow \bigcirc\bot$ in the set $\bigcup_{i=1}^{\infty} (\mathcal{P}_i \cup \mathcal{O}_i)$. Let now $I \in \mathbb{N}$ be the index of the derivation Δ in which all such terminating step clauses have been derived and which is such that $\mathcal{N}_I = \emptyset$. It is now easy to see that property (i) holds already for the index I due to the following observation. Let $s_i^L \wedge C \Rightarrow \bigcirc\bot \in \mathcal{N}_k \cup \mathcal{P}_k \cup \mathcal{O}_k$ $(i \in \mathbb{N})$ with $k < I$ such that $s_i^L \wedge C \Rightarrow \bigcirc\bot \notin_X \mathcal{P}_I \cup \mathcal{O}_I$. Hence, it must hold that there exists a clause $\mathcal{D} \in \mathcal{N}_I \cup \mathcal{P}_I \cup \mathcal{O}_I = \mathcal{P}_I \cup \mathcal{O}_I$ with $\mathcal{D} <_s s_i^L \wedge C \Rightarrow \bigcirc\bot$ as the step clause $s_i^L \wedge C \Rightarrow \bigcirc\bot$ is not a tautology. Thus, we obtain $s_i^L \wedge C \Rightarrow \bigcirc\bot \notin_X \mathcal{P}_{I'} \cup \mathcal{O}_{I'}$ for every $I' \geq I$ with $\mathcal{N}_{I'} = \emptyset$.

Then, let $\{\, s_i^L \wedge C_j \Rightarrow \bigcirc\bot \mid 1 \leq j \leq n \,\}$ be all the terminating step clauses for the eventuality $\Diamond L(x)$ and the iteration index i that occur in the set $\bigcup_{i=0}^{\infty} (\mathcal{P}_i \cup \mathcal{O}_i)$. For every terminating step clause $s_i^L \wedge C_j \Rightarrow \bigcirc\bot$, $1 \leq j \leq n$, such that $s_i^L \wedge C_j \Rightarrow \bigcirc\bot \notin_X \bigcap_{i=I,\mathcal{N}_i=\emptyset}^{\infty} (\mathcal{P}_i \cup \mathcal{O}_i)$, let $k_j \geq I$ be the minimal index such that $s_i^L \wedge C_j \Rightarrow \bigcirc\bot \notin_X \mathcal{P}_{k_j} \cup \mathcal{O}_{k_j}$ and $\mathcal{N}_{k_j} = \emptyset$. If $s_i^L \wedge C_j \Rightarrow \bigcirc\bot \in_X \bigcap_{i=I,\mathcal{N}_i=\emptyset}^{\infty} (\mathcal{P}_i \cup \mathcal{O}_i)$ for $1 \leq j \leq n$, we define $k_j = I$. Finally, we set $J = \max(\{\, k_j \mid 1 \leq j \leq n \,\})$.

Now, let $k > J$ such that $\mathcal{N}_k = \emptyset$ and let $s_i^L \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_k \cup \mathcal{O}_k$. As $k > I$, it holds that $s_i^L \wedge C \Rightarrow \bigcirc\bot \in_X \mathcal{P}_I \cup \mathcal{O}_I$. If we assume that $s_i^L \wedge C \Rightarrow \bigcirc\bot \notin_X \mathcal{P}_J \cup \mathcal{O}_J$, then it would follow that there exists a clause $\mathcal{D}' \in \mathcal{N}_J \cup \mathcal{P}_J \cup \mathcal{O}_J = \mathcal{P}_J \cup \mathcal{O}_J$ with $\mathcal{D}' <_s s_i^L \wedge C \Rightarrow \bigcirc\bot$ as the step clause $s_i^L \wedge C \Rightarrow \bigcirc\bot$ is not a tautology and $s_i^L \wedge C \Rightarrow \bigcirc\bot \in_X \mathcal{P}_I \cup \mathcal{O}_I$. Consequently, it would hold that $s_i^L \wedge C \Rightarrow \bigcirc\bot \notin \mathcal{P}_k \cup \mathcal{O}_k$, which is a contradiction. We can infer that property (i) holds for the index J.

Finally, let $s_i^L \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_J \cup \mathcal{O}_J$ and $k \in \mathbb{N}$ be an index with $k > J$ and $\mathcal{N}_k = \emptyset$. Note that then $s_i^L \wedge C_j \Rightarrow \bigcirc\bot \in_X \bigcap_{i=I,\mathcal{N}_i=\emptyset}^{J} (\mathcal{P}_i \cup \mathcal{O}_i)$. If we assume that $s_i^L \wedge C \Rightarrow \bigcirc\bot \notin_X \mathcal{P}_k \cup \mathcal{O}_k$ holds, then let $k' \in \mathbb{N}$ be the minimal such index among the indices with $k' > J$ and $\mathcal{N}_{k'} = \emptyset$. Hence, $J \geq k' > J$ must hold, which is a contradiction. We can therefore conclude that $s_i^L \wedge C \Rightarrow \bigcirc\bot \in_X \mathcal{P}_k \cup \mathcal{O}_k$ and that property (ii) holds for the index J. □

Remark 5.3.10. *The index $J \in \mathbb{N}$ obtained through Lemma 5.3.9 will be called the* i-th iteration termination index *for the eventuality $\Diamond L(x)$.*

Now, we establish a closer link between the terminating step clauses derived by the Subsumption-Restricted-FG-BFS algorithm and the loop search process performed by the fair inference procedure. For any execution of the loop search algorithm (for an arbitrary eventuality) and for any iteration it can be shown that there exists an iteration index in a run of the inference procedure **F** after which the same terminating step clauses (up to variable names and an extension with loop search markers) have been derived and remain present in any later derivation index k with $\mathcal{N}_k = \emptyset$.

Lemma 5.3.11. *Let $P' = \langle \mathcal{U}'_0, \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$ be a clausified monodic temporal problem. Additionally, let $\Delta = (\mathcal{N}_i \mid \mathcal{P}_i \mid \mathcal{O}_i)_{i \in \mathbb{N}}$ be a fair derivation produced by the inference procedure* **F** *shown in Figure 5.2 from the temporal problem P' and let I be its universal clause termination index. Furthermore, let $\Diamond L(x) \in \mathcal{E}$ be an eventuality and let $\mathcal{M}'_0, \mathcal{M}'_1, \ldots, \mathcal{M}'_i, \mathcal{M}'_{i+1} \ldots$ be the sets of (all the) terminating step clauses constructed as in a run of the Subsumption-Restricted-FG-BFS algorithm applied for the eventuality $\Diamond L(x) \in \mathcal{E}$ on the temporal problem $\langle \mathrm{Univ}(\mathcal{P}_I \cup \mathcal{O}_I), \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$. Finally, let $i \in \mathbb{N}$ and let L_i be the i-th iteration termination index of the derivation Δ for the eventuality $\Diamond L(x)$. Then it holds for every $k \in \mathbb{N}$ with $k \geq \max(I, L_i)$ and $\mathcal{N}_k = \emptyset$ that:*

$$\mathcal{M}'_i =_X \{ C \Rightarrow \bigcirc\bot \mid s_i^L \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_k \cup \mathcal{O}_k \}$$

Proof. First of all, let $k \in \mathbb{N}$ such that $k \geq \max(I, L_i)$ and $\mathcal{N}_k = \emptyset$.

Then, let $C \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$. It follows from Lemmata 5.3.2 and 5.3.7 that there exists an index $k' \geq k$ such that $\mathcal{N}_{k'} = \emptyset$ and a clause $C' \in \mathcal{P}_{k'} \cup \mathcal{O}_{k'}$ with $C' \leq_s s_i^L \wedge C \Rightarrow \bigcirc\bot$. If we assume that $C' <_s s_i^L \wedge C \Rightarrow \bigcirc\bot$ holds, then we have to distinguish between the following two cases. If C' is a universal clause or a step clause without the loop search marker s_i^L, it follows that $C' \in \mathrm{Res}_{\mathrm{sub}}(\mathrm{Univ}(\mathcal{P}_I \cup \mathcal{O}_I) \cup \mathcal{S}')$, which would imply that $C \Rightarrow \bigcirc\bot \notin \mathcal{M}'_i$, contradicting the assumptions. Otherwise, we have that C' is a step clause which contains the loop search marker s_i^L, for which it follows from Lemma 5.3.8 that $C \Rightarrow \bigcirc\bot \notin \mathcal{M}'_i$ holds again. Thus, we have $s_i^L \wedge C \Rightarrow \bigcirc\bot \in_X \mathcal{P}_{k'} \cup \mathcal{O}_{k'}$. By definition of the i-th iteration termination index, we can conclude that $s_i^L \wedge C \Rightarrow \bigcirc\bot \in_X \mathcal{P}_k \cup \mathcal{O}_k$.

Now, let $s_i^L \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_k \cup \mathcal{O}_k$. Then, we have by Lemma 5.3.8 that there exists a clause $\mathcal{D} \in \mathrm{Res}^{\infty}_{\mathrm{sub}}(\mathrm{Univ}(\mathcal{P}_I \cup \mathcal{O}_I) \cup \mathcal{S}') \cup \mathcal{M}'_i$ such that $\mathcal{D} \leq_s C \Rightarrow \bigcirc\bot$, which implies that $\mathcal{D} \notin \mathrm{taut}(\mathsf{P})$. If we assume that $\mathcal{D} \in \mathrm{Res}^{\infty}_{\mathrm{sub}}(\mathrm{Univ}(\mathcal{P}_I \cup \mathcal{O}_I) \cup \mathcal{S}')$, we can infer by Lemmata 5.3.2 and 5.3.3 that there exists an index $k' \geq k$ such that $\mathcal{N}_{k'} = \emptyset$ and also a clause $\mathcal{D}' \in \mathcal{P}_{k'} \cup \mathcal{O}_{k'}$ with $\mathcal{D}' \leq_s \mathcal{D} \leq_s s_i^L \wedge C \Rightarrow \bigcirc\bot$. As $\mathcal{D}'$ is either a universal clause or a step clause without the loop search marker s_i^L, we obtain $\mathcal{D}' <_s s_i^L \wedge C \Rightarrow \bigcirc\bot$. Thus, by definition of the i-th iteration termination index, we can conclude that $s_i^L \wedge C \Rightarrow \bigcirc\bot \notin_X \mathcal{P}_k \cup \mathcal{O}_k$, which contradicts our assumptions. Therefore, $\mathcal{D} = D \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$ such that $D \leq_s C$ holds. If we now assume that $D <_s C$ holds, then it would first of all follow from Lemmata 5.3.2 and 5.3.7 that there exists an index $k'' \geq k$ with $\mathcal{N}_{k''} = \emptyset$ and a clause $\mathcal{D}'' \in \mathcal{P}_{k''} \cup \mathcal{O}_{k''}$ such that $\mathcal{D}'' \leq_s s_i^L \wedge D \Rightarrow \bigcirc\bot <_s s_i^L \wedge C \Rightarrow \bigcirc\bot$. Again, by definition of the i-th iteration termination index, we can conclude that $s_i^L \wedge C \Rightarrow \bigcirc\bot \notin_X \mathcal{P}_k \cup \mathcal{O}_k$, contradicting the assumptions. We can conclude that $C =_X D$ and $C \Rightarrow \bigcirc\bot \in \mathcal{M}'_i$. □

We can now prove the analogous proposition to Lemma 5.3.3 for clauses derived by the Subsumption-Restricted-FG-BFS algorithm, i.e. we show that any non-trivial loop formula computed by the Subsumption-Restricted-FG-BFS algorithm there exist subsuming clauses for the different negated subformulae of the loop formula that are derived by the fair inference procedure.

Lemma 5.3.12. *Let $P' = \langle \mathcal{U}'_0, \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$ be a clausified monodic temporal problem. Additionally, let $\Delta = (\mathcal{N}_i \,|\, \mathcal{P}_i \,|\, \mathcal{O}_i)_{i \in \mathbb{N}}$ be a fair derivation produced by the inference procedure* **F** *shown in Figure 5.2 from the temporal problem P'. Let $\Diamond L(x) \in \mathcal{E}$ be an eventuality and* ***true*** *$\neq H(x) = \bigvee_{j=1}^{n} (\tilde{\exists} C_j)\{c^l \to x\}$ with $n \geq 1$ be a loop formula computed by an application of the Subsumption-Restricted-FG-BFS algorithm on the clausified temporal problem $\langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$, where $\mathcal{U}'$ is a finite set of universal formulae with $\mathcal{U}' \leq_s \mathcal{U}'_0 \setminus \mathrm{taut}(\mathcal{U}'_0)$ and $\bigcup_{j=1}^{\infty} \mathcal{O}_j \leq_s \mathcal{U}' \setminus \mathrm{taut}(\mathcal{U}')$. Finally, let $\neg C_1\{c^l \mapsto y\}, \ldots, \neg C_n\{c^l \mapsto y\}$ be the clausification of the formula $\forall y \neg H(y)$, where y is a fresh variable.*

Then there exists a index $N \in \mathbb{N}$ such that for every j with $1 \leq j \leq n$ there exists a universal clause $\mathcal{D}_j \in \mathcal{P}_N \cup \mathcal{O}_N$ with $\mathcal{D}_j \leq_s \neg C_j\{c^l \mapsto y\}$.

Proof. Let I be the universal clause termination index for the derivation Δ (see Lemma 5.3.5). Additionally, let $\tilde{\mathcal{M}}'_0, \tilde{\mathcal{M}}'_1, \ldots$ be the sets of terminating step clauses computed by the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem $\langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$.

Let $j_1 \in \mathbb{N}$ be the minimal index such that $\tilde{\mathcal{M}}'_{j_1+1} = \{\, C_i \Rightarrow \bigcirc\bot \mid 1 \leq i \leq n \,\}$ and the formula $\forall x (\bigvee_{D \Rightarrow \bigcirc\bot \in M_{j_1}} (\tilde{\exists} D)\{c^l \mapsto x\} \Leftrightarrow \bigvee_{i=1}^{n} (\tilde{\exists} C_i)\{c^l \mapsto x\})$ is valid. Then it follows from Lemma 4.3.15 that $\tilde{\mathcal{M}}'_{j_1} =_X \tilde{\mathcal{M}}'_k$ for every $k \geq j_1$, and in particular it holds that $\tilde{\mathcal{M}}'_k \neq \emptyset$ for every $k \geq j_1$.

Furthermore, let $\mathcal{M}'_0, \mathcal{M}'_1, \ldots$ be the sets of terminating step clauses computed by the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem $\langle \mathrm{Univ}(\mathcal{US}_I \cup \mathcal{O}_I), \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$. By Lemma 4.3.7 there exists an index $j_2 \in \mathbb{N} \setminus \{0\}$ such that either $\mathcal{M}'_{j_2} = \emptyset$ or $\emptyset \neq \mathcal{M}'_{j_2} =_X \mathcal{M}'_{j_2+1}$.

If we now assume that $\mathcal{M}'_{j_2} \neq \emptyset$ and $\mathcal{M}'_{j_2} =_X \mathcal{M}'_{j_2+1}$, then let L_{j_2} and L_{j_2+1} be the j_2-th and j_2+1-th iteration termination indexes of the derivation Δ. By Lemma 5.3.11 it would hold for every index $K \in \mathbb{N}$ with $K \geq \max(I, L_{j_2}, L_{j_2+1})$ and $\mathcal{N}_K = \emptyset$ that

$$\mathcal{M}'_{j_2} =_X \{\, C \Rightarrow \bigcirc\bot \mid s^L_{j_2} \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_K \cup \mathcal{O}_K \,\} \neq \emptyset$$

and

$$\mathcal{M}'_{j_2+1} =_X \{\, C \Rightarrow \bigcirc\bot \mid s^L_{j_2+1} \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_K \cup \mathcal{O}_K \,\} \neq \emptyset.$$

As $\mathcal{M}'_{j_2} =_X \mathcal{M}'_{j_2+1}$, we could infer the following set equality for every index $K \in \mathbb{N}$ with $K \geq \max(I, L_{j_2}, L_{j_2+1})$ and $\mathcal{N}_K = \emptyset$:

$$\begin{aligned} &\{\, C \Rightarrow \bigcirc\bot \mid s^L_{j_2} \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_K \cup \mathcal{O}_K \,\} \\ &\qquad\qquad =_X \{\, C \Rightarrow \bigcirc\bot \mid s^L_{j_2+1} \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_K \cup \mathcal{O}_K \,\} \end{aligned}$$

By Lemma 5.3.2 and through an application of the ‘Loop Testing’ rule using the fairness of the derivation Δ, we can conclude that there would exist an index $K' \geq \max(I, L_{j_2}, L_{j_2+1})$ with $\mathcal{N}_{K'} = \emptyset$ such that for every $s^L_{j_2} \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_l \cup \mathcal{O}_l$, where l is the minimal index with $l \geq \max(I, L_{j_2}, L_{j_2+1})$ and $\mathcal{N}_l = \emptyset$, there would exist a universal clause $\mathcal{D} \in \mathcal{O}_{K'}$ with $\mathcal{D} \leq_s \neg C\{c^l \mapsto y\} <_s s^L_{j_2} \wedge C \Rightarrow \bigcirc\bot$ as y is a fresh variable, i.e.

$$\{\, C \Rightarrow \bigcirc\bot \mid s^L_{j_2} \wedge C \Rightarrow \bigcirc\bot \in \mathcal{P}_{K'} \cup \mathcal{O}_{K'} \,\} = \emptyset.$$

We have obtained a contradiction.

Consequently, we have $\mathcal{M}'_{j_2} = \emptyset$, and it follows that $\mathcal{M}'_k = \emptyset$ for every $k \geq j_2$. Hence, there exists an index $J \in \mathbb{N}$ with $\tilde{\mathcal{M}}'_{j_1+1} = \tilde{\mathcal{M}}'_J$ and $\mathcal{M}'_J = \emptyset$. Thus, as $\mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathrm{Univ}(\mathcal{US}_I \cup \mathcal{O}_I)) \leq_s \mathcal{U}' \setminus \mathrm{taut}(\mathcal{U}')$ it follows from Lemma 4.3.16 that $\mathrm{Res}^{\infty}_{\mathrm{Sub}}(\mathrm{Univ}(\mathcal{US}_I \cup \mathcal{O}_I) \cup \mathcal{S}') \leq_s \tilde{\mathcal{M}}'_{j_1+1}$. By Lemma 5.3.3 we can infer for every i, $1 \leq i \leq n$, that there exists an index $j_i \in \mathbb{N}$ and a universal clause $\mathcal{D}_i \in \mathcal{O}_{j_i}$ with $\mathcal{D}_i \leq_s \neg C_i$. It is easy to see for every i, $1 \leq i \leq n$, that $\mathcal{D}_i \leq_s \neg C_i\{c^l \mapsto y\}$ as $c^l \notin \mathit{const}(\mathcal{D}_i)$. Finally, the required index $N \in \mathbb{N}$ is obtained through Lemma 5.3.2. □

We now have all the prerequisites in place to prove the refutational completeness of the inference procedure **F**.

Theorem 5.3.13. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}^c \rangle$ be a clausified and constant-flooded monodic temporal problem. Let $\succ$ be an atom ordering and S an instance compatible selection function. Additionally, let $\Delta = (\mathcal{N}_i \mid \mathcal{P}_i \mid \mathcal{O}_i)_{i \in \mathbb{N}}$ be a fair derivation produced by the inference procedure* **F** *shown in Figure 5.2 from the temporal problem P using the atom ordering $\succ$ and selection function S. Then it holds that:*

$$P \textit{ is unsatisfiable if and only if } \bot \in \bigcup_{j=0}^{\infty} \mathcal{O}_j$$

Proof. The implication "$\Leftarrow$" follows from the soundness of subsumption-compatible ordered fine-grained resolution with selection. For the remaining implication "$\Rightarrow$" we first of all obtain from Theorem 4.3.20 that there exists a derivation $\tilde{\Delta} = \mathcal{D}_1, \ldots, \mathcal{D}_n$ where $\mathcal{D}_n = \bot$ such that applications of the eventuality resolution rules are restricted to loops found by the Subsumption-Restricted-FG-BFS algorithm. We now inductively show for every i, $1 \leq i \leq n$, for which the clause $\mathcal{D}_i$ is not a tautology that there exists a clause $\mathcal{C}_i$ and an index $j \in \mathbb{N}$ with $\mathcal{C}_i \leq_s \mathcal{D}_i$ and $\mathcal{C}_i \in \mathcal{P}_j \cup \mathcal{O}_j$.

Now, let $i \in \mathbb{N}$ with $1 \leq i \leq n$ such that the clause $\mathcal{D}_i$ is not a tautology and has not been derived by a run of the Subsumption-Restricted-FG-BFS algorithm. It follows then from Lemma 5.3.3 that there exists a clause $\mathcal{C}_i$ and an index $j \in \mathbb{N}$ with $\mathcal{C}_i \leq_s \mathcal{D}_i$ and $\mathcal{C}_i \in \mathcal{P}_j \cup \mathcal{O}_j$.

Otherwise, if a clause $\mathcal{D}_i$ for $1 \leq i \leq n$ that is not a tautology has been obtained through the clausification of a formula $\forall x \neg H(x)$, where the formula $H(x)$ has been computed by the Subsumption-Restricted-FG-BFS algorithm for an eventuality $\Diamond L(x) \in \mathcal{E}^c$, we distinguish between the following cases.

For $H(x) = \mathbf{true}$ (and hence, $\mathcal{D}_i = \bot$), the run of the Subsumption-Restricted-FG-BFS algorithm just consisted of a single iteration in which either the empty clause has been derived from universal clauses $\mathrm{Univ}(\mathcal{D}_0, \ldots, \mathcal{D}_{i-1})$, or there exists a derivation of a step clause $\mathbf{true} \Rightarrow \bigcirc\bot$. If the empty clause was derived in the single iteration of the algorithm, then by Lemma 5.3.3 there exists an index $j \in \mathbb{N}$ such that $\mathcal{D}_i = \bot \in \mathcal{P}_j \cup \mathcal{O}_j$. In the situation where the Subsumption-Restricted-FG-BFS derived the step clause $\mathbf{true} \Rightarrow \bigcirc\bot$, we can apply Lemma 5.3.7 and obtain an index $j \in \mathbb{N}$ such that there exists a clause $\mathcal{D}_i \in \mathcal{P}_j \cup \mathcal{O}_j$ with $\mathcal{D}_i \leq_s s_1^L \Rightarrow \bigcirc\bot$, which implies that $\mathcal{D}_i = \bot \in \mathcal{O}_J$ for an index $J \geq j$.

Finally, in the case where $H(x) \neq$ **true**, we can apply Lemma 5.3.12 and obtain an index $J \in \mathbb{N}$ such that there exists a clause $\mathcal{C}_i \in \mathcal{P}_J \cup \mathcal{O}_J$ with $\mathcal{C}_i \leq_s \mathcal{D}_i$. □

5.4 $\mathbb{F}$ as a Decision Procedure

In this section we examine the possibility of the fair inference procedure to be used as a decision procedure.

Just as with subsumption-compatible ordered fine-grained resolution with selection, we will see that when the "first-order" reasoning that is necessary for constructing derivations with the fair inference algorithm becomes decidable, then derivations produced by the fair inference procedure are also guaranteed to terminate. Consequently, the inference algorithm **F** can act as a decision procedure for the unsatisfiability of suitably restricted monodic temporal problems.

We now state and prove the corresponding theorem.

Theorem 5.4.1. *Let $\mathcal{L}$ be a fragment of first-order logic without equality or (non-constant) function symbols for which the validity problem is decidable by ordered resolution with selection, and let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}^c \rangle$ be a constant-flooded monodic temporal problem such that*

(i) the sets $\mathcal{I}$ and $\mathcal{U}$ (without considering the preceding $\square$-operator) only contain formulae from the fragment $\mathcal{L}$, and

(ii) the sets $\mathcal{U} \cup \{\tilde{\forall}\mathcal{C}_1, \ldots, \tilde{\forall}\mathcal{C}_m\}$ are still contained in the fragment $\mathcal{L}$, where for every i, $1 \leq i \leq m$, $\mathcal{C}_i$ is a negative clause that contains at most monadic literals with predicate symbols stemming from the left-hand sides of step clauses contained in the set $\mathcal{S}$ and such that $const(\mathcal{C}_i) \subseteq const(P)$, and

(iii) for every eventuality $\Diamond L(x) \in \mathcal{E}$ and for full-merged step clauses $\forall x(\mathcal{A}_i(x) \Rightarrow \bigcirc\mathcal{B}_i(x))$, $1 \leq i \leq n$, built from the temporal problem P and a set of formulae $\mathcal{U} \cup \{\tilde{\forall}\mathcal{C}_1, \ldots, \tilde{\forall}\mathcal{C}_m\}$ defined as in the previous point, the formulae

$$\forall x(\mathcal{U} \cup \mathcal{B}_i(x) \cup \{\tilde{\forall}\mathcal{C}_1, \ldots, \tilde{\forall}\mathcal{C}_m\} \Rightarrow \bigvee_{j=1}^{n} \mathcal{A}_j(x))$$

and

$$\forall x(\mathcal{U} \cup \mathcal{B}_i(x) \cup \{\tilde{\forall}\mathcal{C}_1, \ldots, \tilde{\forall}\mathcal{C}_m\} \Rightarrow \neg L(x))$$

are contained in the fragment $\mathcal{L}$ for every i with $i \leq i \leq n$, and

(iv) the previous point also holds for merged derived step clauses and ground eventualities.

*Then the inference procedure **F** shown in Figure 5.2 is a decision procedure for the satisfiability of the temporal problem P.*

Proof. Let $\mathrm{Cls}(\mathsf{P}) = \langle \mathcal{U}', \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$, let $\Delta = (\mathcal{N}_i \mid \mathcal{P}_i \mid \mathcal{O}_i)_{i \in \mathbb{N}}$ be a fair derivation produced by the inference procedure $\mathbb{F}$ shown in Figure 5.2 from the temporal problem $\mathrm{Cls}(\mathsf{P})$ and let I be its universal clause termination index.

We can first of all observe that under the conditions *(i)* to *(iv)* described above the validity of the loop side conditions becomes decidable for any eventuality from the set $\mathcal{E}^c$ and the set of universal clauses $\mathcal{U}'$, potentially extended with new universal clauses obtained through applications of the clause conversion or eventuality resolution rules. It is also easy to see that the computation of saturations involving initial, universal or step clauses from the temporal problem $\mathrm{Cls}(\mathsf{P})$, potentially extended with new universal clauses obtained through applications of the clause conversion or eventuality resolution rules, always terminates.

Then, if the temporal problem P is unsatisfiable, it follows from Theorem 5.3.13 that there exists an index $J \in \mathbb{N}$ with $\bot \in \mathcal{P}_J \cup \mathcal{O}_J$. Otherwise, the temporal problem P is satisfiable, and we have to show that there exists an index J with $\mathcal{N}_J = \mathcal{P}_J = \emptyset$. An inspection of the inference rules depicted in Figure 5.2 reveals that the only problematic rule for ensuring termination is the 'Next Loop Search Iteration' rule because it might potentially generate an infinite number of loop search clauses $s^L_{i+1} \Rightarrow \bigcirc D$ for $i \in \mathbb{N}$ if there is an eventuality contained in the temporal problem $\mathrm{Cls}(\mathsf{P})$. We now distinguish between the following two cases.

If there does not exist a loop formula for the temporal problem $\mathsf{P}' = \langle \mathrm{Univ}(\mathcal{P}_I \cup \mathcal{O}_I), \mathcal{I}', \mathcal{S}', \mathcal{E}^c \rangle$ and eventuality $\Diamond L(x) \in \mathcal{E}^c$, then the Subsumption-Restricted-FG-BFS algorithm applied on the temporal problem P' terminates after k iterations because no new terminating step clauses can be derived and it returns **false**. It is thus easy to see that there exists a index $i \in \mathbb{N}$ such that the set $\mathcal{N}_i \cup \mathcal{P}_i$ does not contain a loop search step clause $s^L_i \Rightarrow \bigcirc D$. We can infer that there exists an index $J \in \mathbb{N}$ such that $\mathcal{N}_J = \mathcal{P}_J = \emptyset$.

Finally, if a loop formula $\mathbf{true} \neq H(x) = \bigvee_{j=1}^{n} (\tilde{\exists} C_j)\{c^l \rightarrow x\}$ computed by the Subsumption-Restricted-FG-BFS algorithm exists for the temporal problem P' and eventuality $\Diamond L(x) \in \mathcal{E}^c$ (the case $H(x) = \mathbf{true}$ cannot occur as the temporal problem P is assumed to be satisfiable), then it follows from Lemma 5.3.12 that there exists a index $N \in \mathbb{N}$ such that for every j with $1 \leq j \leq n$ there exists a universal clause $\mathcal{D}_j \in \mathcal{P}_N \cup \mathcal{O}_N$ with $\mathcal{D}_j \leq_s \neg C_j\{c^l \mapsto y\}$, where y is a fresh variable. Consequently, if a clause $\mathcal{D}_j$, $1 \leq j \leq n$, is obtained through the terminating step clause $s^L_i \wedge \mathcal{D}_j\{y \mapsto c^l\} \Rightarrow \bigcirc\bot$, we can see that $\mathcal{D}_j \leq_s s^L_i \wedge \mathcal{D}_j\{y \mapsto c^l\} \Rightarrow \bigcirc\bot$ holds, i.e. all such terminating loop search step clauses will be removed for the eventuality $\Diamond L(x)$. Thus, there exists an index M in the derivation Δ such that the 'Next Loop Search Iteration' rule cannot be applied any longer for the eventuality $\Diamond L(x)$ on $\mathcal{P}_k$ with $k \geq M$. Now, it is easy to see that there exists an index $J \in \mathbb{N}$ such that $\mathcal{N}_J = \mathcal{P}_J = \emptyset$. □

We can now state the following corollary.

Corollary 5.4.2. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a temporal problem in* PLTL. *Then the inference procedure* $\mathbb{F}$ *shown in Figure 5.2 is a decision procedure for the satisfiability of the temporal problem P.*

5.5 Summary

In this chapter we analysed some theoretical aspects of subsumption-compatible ordered fine-grained resolution with selection which can lead to problems when fair $\mathfrak{I}^{S,\succ}_{FG,Sub}$-derivations should be constructed in practice.

Due to the fact that the applicability of the eventuality resolution rules is only semi-decidable, it becomes impossible to guarantee the construction of fair derivations, i.e. derivations in which every non-redundant clause that is derivable from a given clause set is eventually derived, as the applicability check for those inference rules might not terminate.

As the ability to construct fair derivations is an essential requirement for maintaining the refutational completeness of an automated theorem, we presented an inference procedure that can construct fair derivations for reasoning in monodic first-order temporal logic based on the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus and we proved its refutational completeness. The design of the new inference mechanism is based on integrating the saturation steps related to loop search, which may not terminate in general, into the main saturation process. The proof of refutational completeness proceeded by showing that for every non-tautological clause contained in a $\mathfrak{I}^{S,\succ}_{FG,Sub}$-refutation for an unsatisfiable clausified monodic temporal problem there exists a subsuming clause computed in a fair derivation of the new inference procedure.

We also showed that the new inference mechanism can be used as a decision procedure for temporal problems in which the first-order formulae are restricted appropriately.

Chapter 6

TSPASS - a Fair Monodic Temporal Logic Prover

6.1 Introduction

As explained in Chapter 1, one advantage of using clausal resolution and the normal form DSNF is that the temporal clauses in clausified DSNF can be translated into first-order logic. For temporal reasoning it is then possible to use state-of-the-art theorem provers for first-order logic as most temporal inference rules of the subsumption-compatible ordered fine-grained resolution with selection calculus can be mapped onto first-order resolution and factoring rules. A special treatment of eventualities remains necessary, though.

Several automated theorem provers based on resolution have been developed for PLTL and monodic FOTL. TRP [52, 53] and TRP++ [49] are theorem provers for PLTL that implement the resolution-based calculus introduced in [37], which is similar to ordered fine-grained resolution with selection. TRP is written in Prolog, whereas a C++ implementation is available in TRP++. For monodic FOTL then, the theorem prover TeMP has been developed [50]. It is based on ordered fine-grained resolution with selection, and uses the first-order prover Vampire as inference kernel.

Now, an important part in the architecture of automated theorem provers consists in the selection of the clauses which are considered for computing inferences. The order in which clauses are selected can contribute significantly to the amount of time needed to solve a given problem. But most importantly, one has to ensure that clause selection is fair, i.e. every clause should eventually be selected for performing inferences in order to maintain the refutational completeness of the theorem prover.

As described in Chapter 5, the calculus of (subsumption-compatible) ordered fine-grained resolution with selection contains inference rules that only have semi-decidable applicability conditions. Consequently, fair derivations cannot be easily obtained in practice as the applicability check for these inference rules might not terminate. In this chapter we show that there indeed exist fairness problems with the architecture of TeMP which cannot be overcome by simply defining a more appropriate clause selection function. We also describe the implementation of the fair inference architecture described in Chapter 5 in the monodic first-order temporal logic prover TSPASS, which is based on the first-order theorem prover SPASS [92]. Additionally, we analyse the effectiveness of redundancy elimination and the proof search performance of TSPASS on several examples. Binaries for the TSPASS system and its source code are available at:

`http://www.csc.liv.ac.uk/~michel/software/tspass/`

The chapter is organised as follows. In Section 6.2 we describe the connection between ordered fine-grained step resolution and regular first-order resolution, i.e. we show how first-order resolution can be used to simulate fine grained-step resolution. Then, in Section 6.3 we first of all describe the architecture of TeMP, before we outline its fairness problems and discuss how the fair inference procedure **F** depicted in Figure 5.2 can be implemented in practice. Subsequently, the implementation of TSPASS is described in Section 6.4 in greater detail. We conclude this chapter in Section 6.5 with an analysis of some experimental results which demonstrate the effectiveness of redundancy elimination in TSPASS and which illustrate its proof search performance on PLTL and monodic FOTL problems.

6.2 Fine-Grained Step Resolution and First-Order Logic

The deduction rules of (subsumption-compatible) ordered fine-grained step resolution with selection are close enough to the rules of classical first-order resolution for allowing us to use state-of-the-art first-order resolution provers as a basis for the implementation of our calculus.

Let **S** be a temporal problem in clausal form. For every k-ary predicate, P, occurring in **S**, we introduce a new $(k+1)$-ary predicate $\widetilde{P}$. We will also use the constant 0 (representing the initial moment in time), and unary function symbols s (representing the successor function on time) and h, which we assume not to occur in **S**. Let φ be a first-order formula in the vocabulary of **S**. We denote by $[\varphi]^T$ the result of replacing all occurrences of predicates in φ by their "tilded" counterparts with T as the first argument (e.g. $P(x, y)$ is replaced with $\widetilde{P}(T, x, y)$). The term T will either be the constant 0 or the variable t (intuitively, t represents a moment in time). The variable t is assumed to be universally quantified.

Now, in order to realise fine-grained step resolution by means of classical first-order resolution, we define a set of first-order clauses **FO(S)** as follows.

- For every initial clause C from **S**, the clause $[C]^0$ is in **FO(S)**.
- For every universal clause D from **S**, the clause $[D]^t$ is in **FO(S)**.
- For every step clause $p \Rightarrow \bigcirc q$ from **S**, the clause $\neg\widetilde{p}(t) \vee \widetilde{q}(s(t))$ is in **FO(S)**, and for every step clause $P(x) \Rightarrow \bigcirc Q(x)$, the clause $\neg\widetilde{P}(t, x) \vee \widetilde{Q}(s(t), h(x))$ is in **FO(S)**.[1]

[1] The purpose of the function symbol h will be explained on page 126.

The key insight is that fine-grained step resolution on **S**, including (implicitly) the clause conversion rule, can be realised using classical ordered first-order resolution with selection (see, e.g. [10]) on **FO(S)**. For universal and initial resolution and factoring rules of $\mathfrak{I}^{S,\succ}_{FG}$, rules 1 to 3 and 7 (see page 36), this is obvious. For step resolution and (step) factoring, rules 4 and 5, we observe that if a clause contains a *next-state* literal, i.e. a literal whose first argument starts with the function symbol s, a factoring or resolution inference can only be performed on such a literal. This requirement can be enforced by an appropriate atom ordering.

The restrictions on the literal orderings required to realise resolution inferences with step clauses on the first-order level can be obtained for example with the Knuth-Bendix ordering (KBO) [6, 26, 56]. The definition of the KBO is given below for two terms or atoms s and t: $s \succ_{\text{KBO}} t$ if and only if

(KBO1) $\forall x$: $|s|_x \geq |t|_x$ and $w(s) > w(t)$

or

(KBO2) $\forall x$: $|s|_x \geq |t|_x$, $w(s) = w(t)$ and one of the following cases occurs:

(KBO2a) $s = f^n(x)$ and $t = x$ (for $n > 0$)

(KBO2b) $s = f(s_1, \dots, s_m)$, $t = g(t_1, \dots, t_n)$ with $f > g$

(KBO2c) $s = f(s_1, \dots, s_m)$, $t = f(t_1, \dots, t_m)$ with $s_1 = t_1, \dots, s_{i-1} = t_{i-1}$, and $s_i \succ_{\text{KBO}} t_i$ (for $m > 0$)

The function w computes the weights of variables, terms, and predicates after weights (i.e. natural numbers) have been assigned to specific constants, functional and predicate symbols. A global weight for variables is generally used. The computation of the weights for terms and atoms is based on summing up the weights of the individual signature symbols and variables occurring in the term or atom. The symbol $>$ denotes a strict partial ordering on signature symbols, and by $|t|_x$ we represent the number of occurrences of the variable x in the term or atom t.

It is easy to see that from $s \succ_{\text{KBO}} t$ it follows that $w(s) \geq w(t)$ holds for any two terms or atoms s and t. Thus, by choosing the weight of the temporal successor function such that it is greater than the weight of every literal occurring in the left-hand side of a step clause (increased by one), we can ensure that every literal occurring in the right-hand side of a translated step clause is not smaller w.r.t. the KBO than any literal occurring in the left-hand side.

Now, during resolution inferences the variables present in the right-hand sides of step clauses can change. For example, if the step clause $\neg\tilde{p}(t, x) \vee \tilde{q}(s(t), x)$ is resolved with the universal clause $\neg\tilde{q}(t', z) \vee \tilde{r}(t', y)$, the step clause $\neg\tilde{p}(t, x) \vee \tilde{r}(s(t), y)$ is obtained as a result. The variable x does not occur in the literal $r(s(t), y)$, whereas the variable y does not occur in the atom $p(t, x)$. Consequently, the condition on variables imposed by the KBO cannot be fulfilled; the literals $\neg\tilde{p}(t, x)$ and $\tilde{r}(s(t), y)$ become thus incomparable w.r.t. the KBO. As a consequence both literals are maximal in the step clause $\neg\tilde{p}(t, x) \vee \tilde{r}(s(t), y)$ and resolution inferences are possible on the literal $\neg\tilde{p}(t, x)$. Hence, special care needs to be taken when computing the maximal literals in a clause in order to prevent left-hand side literals from becoming maximal (see Section 6.4.9 for more details). Note that for propositional temporal problems the use of the KBO would be sufficient to restrict inferences to the right-hand sides of step clauses only as in PLTL clauses every predicate contains the same free variable, namely the temporal variable.

Additionally, note that all rules performing inferences on (non-terminating) step clauses impose the restriction on most general unifiers σ that σ does *not* map variables occurring in the left-hand side of a step clause into a constant or a functional term. On first-order clauses, this restriction is enforced by the function symbol h introduced by FO: Each temporal literal $\bigcirc Q(x)$ is mapped by FO to $\tilde{Q}(s(t), h(x))$, and the function symbol h "shields" the variable x from being instantiated by a constant or functional term.

No explicit clause conversion rule, rule 6, is required for the translated clauses.

Moreover, our translation ensures that the first-order clause $\neg\tilde{P}(t, x) \vee \tilde{Q}(s(t), h(x))$, stemming from $P(x) \Rightarrow \bigcirc Q(x)$, does not subsume the clause $\neg\tilde{P}(t, c) \vee \tilde{Q}(s(t), c)$, stemming from $P(c) \Rightarrow \bigcirc Q(c)$, which is important for the implementation of the calculus to be complete (see Section 4.2.2).

Finally, we still observe that the Skolemization process is not performed after the transformation to first-order logic but on the level of DSNF problems already. For example, for the monodic FOTL formula $\Box\forall x\exists y\, p(x, y)$ we obtain the translated clause $p(t, x, f(x))$ and not the clause $p(t, x, f(x, t))$ although the domain element that is assigned to the variable y can potentially vary in the different time points of a model. This requirement for Skolem constants and function symbols is already handled at the level of the $\mathfrak{I}^{S,\succ}_{FG}$-calculus as most general unifiers are only allowed to map variables which occur in the left-hand sides of step clauses into variables. In this way one can prevent that the interpretations of Skolem constants or function symbols are fixed to the same domain elements across different time points as the interpretations of constants (and implicitly function symbols) are assumed to be rigid.

6.3 Implementing a Fair Architecture for Monodic Temporal Reasoning

6.3.1 The Architecture of TeMP

In this section we describe the design principles behind the automated theorem prover TeMP, which is based on the $\mathfrak{I}^{S,\succ}_{FG}$-calculus. As we have seen in the previous section, the fine-grained step resolution inference rules of $\mathfrak{I}^{S,\succ}_{FG}$ can be easily implemented in an automated theorem prover that is based on ordered first-order resolution with selection. However, the ground and non-ground eventuality resolution rules 8 and 9 of $\mathfrak{I}^{S,\succ}_{FG}$ cannot simply be mapped onto the inference rules of first-order resolution as suitable premises have to be determined first before those rules can be applied.

Instead, in TeMP the (Subsumption-Restricted)-FG-BFS algorithm shown in the Figures 3.2 and 4.1 is implemented in order to find the full merged step clauses (or merged derived step clauses, respectively) required for the application of the eventuality resolution rules. The only difficulty related to an implementation of the (Subsumption-Restricted)-FG-BFS algorithm using first-order ordered resolution with selection is that in step (2) of the algorithm, the rules of fine-grained step resolution are applied with the exception of the clause conversion rule, rule 6. As no explicit clause conversion rule is required on **FO**$(\mathbf{S})$, this restriction cannot be enforced by disabling one of the deduction rules. Instead one can use a variant FO_{BFS} of FO which has the desired effect. Let $\mathbf{S}_{i+1}$ be a monodic temporal problem in clausified form as defined in step (2) of the (Subsumption-Restricted)-FG-BFS algorithm. Then $\mathrm{FO}_{BFS}(\mathbf{S}_{i+1})$ is defined as follows:

- For every universal clause D in $\mathbf{S}_{i+1}$, the clause $[D]^t$ is in $\mathrm{FO}_{BFS}(\mathbf{S}_{i+1})$.
- For every ground step clause $p \Rightarrow \bigcirc l$ in $\mathbf{S}_{i+1}$, the clause $\neg\widetilde{p}(0) \vee \widetilde{l}(s(t))$ is in $\mathrm{FO}_{BFS}(\mathbf{S}_{i+1})$, and for every non-ground step clause $P(x) \Rightarrow \bigcirc M(x)$ in $\mathbf{S}_{i+1}$, the clause $\neg\widetilde{P}(0,x) \vee \widetilde{M}(s(t),h(x))$ is in $\mathrm{FO}_{BFS}(\mathbf{S}_{i+1})$.

Recall that initial clauses do not contribute to loop search, so we do not include their translation into $\mathrm{FO}_{BFS}(\mathbf{S}_{i+1})$. Again, the motivation for FO_{BFS} is that saturation of $\mathbf{S}_{i+1}$ under the rules 1 to 5 and 7 corresponds to the saturation of $\mathrm{FO}_{BFS}(\mathbf{S}_{i+1})$ under ordered first-order resolution as described above. In particular, clauses consisting only of literals whose first argument is '0' in the saturation of $\mathrm{FO}_{BFS}(\mathbf{S}_{i+1})$ correspond to final clauses (up to negation). Using this criterion it is straightforward to extract those clauses from the saturation of $\mathrm{FO}_{BFS}(\mathbf{S}_{i+1})$ to form the set $\mathcal{M}_{i+1}$ which is the outcome of step (2) of the (Subsumption-Restricted)-FG-BFS algorithm and to proceed with step (3).

The logical consequence check in step (4) of the (Subsumption-Restricted)-FG-BFS algorithm is again delegated to a first-order prover: for every $C_i(x) \in H_i(x)$ we form a new clause set $C_i(x) \wedge \neg H_{i+1}(x)$; if all the resulting sets are unsatisfiable, $\forall x(H_i(x) \Rightarrow H_{i+1}(x))$ is valid.

Note that it is straightforward to see whether a clause in FO(**S**) is the result of translating an initial, a universal, or a (non-)ground step clause. This makes it possible to compute $\mathrm{FO}_{BFS}(\mathbf{S})$ from FO(**S**) instead of from **S**. Also, the conclusion of an application of one of the eventuality resolution rules can directly be computed as a set of first-order clauses of the appropriate form. Thus, there is no need to ever translate clauses in FO(**S**) back to DSNF clauses. Instead, after translating the input monodic temporal problem once using FO, we can continue to operate with first-order clauses.

The considerations presented above and in the previous section give rise to the main procedure for a monodic temporal logic theorem prover. The architecture depicted in Figure 6.1 has been implemented in the prover TeMP [50]. It consists of a loop where in each iteration (i) the set of temporal clauses is saturated under fine-grained step resolution, more

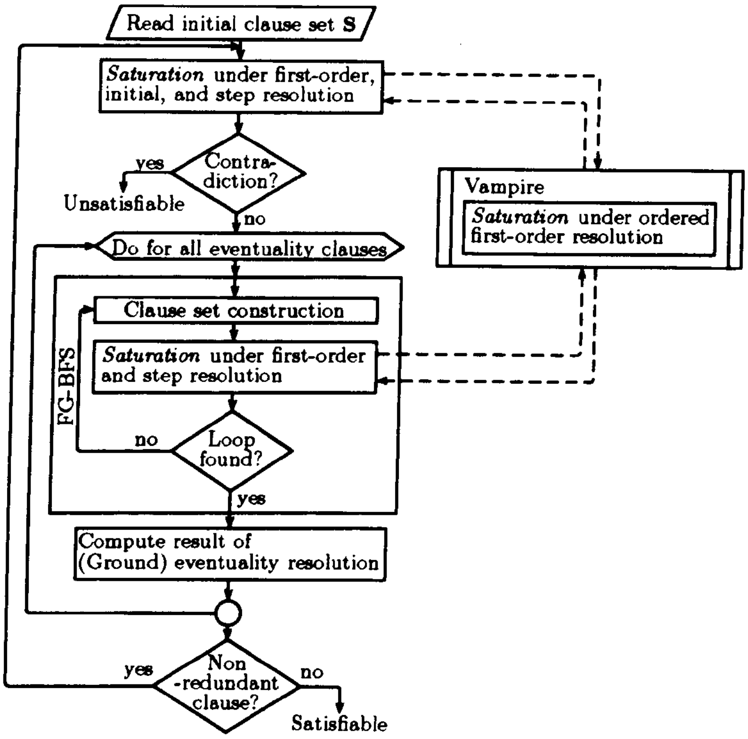

Read initial clause set S
Saturation under first-order, initial, and step resolution
Contra-diction?
yes
Unsatisfiable
no
Do for all eventuality clauses
Clause set construction
Saturation under first-order and step resolution
FG-BFS
Loop found?
no
yes
Compute result of (Ground) eventuality resolution
Non -redundant clause?
yes
no
Satisfiable
Vampire
Saturation under ordered first-order resolution

Figure 6.1: Main Procedure of TeMP (Using Breadth-First Search Strategy)

precisely, rules 1 to 7, and (ii) then for every eventuality clause in the clause set, an attempt is made to find a set of premises for an application of the (ground) eventuality resolution rule. If we find such a set, the set of clauses representing the conclusion of the application of the rule is added to the current set of clauses and the resulting set is saturated under application of the step resolution rules (this helps to identify whether the conclusion of the eventuality resolution rule is redundant or not). There are two control strategies concerning how to explore eventualities in loop search: either they are traversed one by one regardless of whether a loop is found and an application of the (ground) eventuality resolution rule derives new non-redundant clauses (a sort of breadth-first strategy) or the next iteration of the main loop is entered as soon as a loop is found for which an application of the (ground) eventuality resolution rule results in new non-redundant clauses (a sort of depth-first strategy). Figure 6.1 illustrates the breadth-first strategy. The main loop terminates if the empty clause is derived, indicating that the initial set of clauses is unsatisfiable, or if no new non-redundant clauses have been derived during the last iteration of the main loop, which in the absence of the empty clause indicates that the initial set of clauses is satisfiable.

In TeMP the task of saturating clause sets with classical resolution simulating fine-grained step resolution is delegated to the kernel of Vampire [78], which is linked to the whole system as a C++ library. TeMP communicates with the Vampire kernel in a direct way via the kernel API, thus avoiding expensive textual communication. Internally TeMP uses its own data structures, and there is a special module in TeMP which rewrites TeMP's data structures to, and from, Vampire's data structures. Note that minor adjustments have been made in the functionality of Vampire to accommodate step resolution: a special mode for literal eligibility has been introduced such that in a clause containing a next-state literal only next-state literals can become eligible.

6.3.2 Fairness Problems of TeMP

As we have seen in Chapter 5, the presence of the ground and non-ground eventuality resolution rules in the $\mathfrak{I}^{S,\succ}_{FG}$-calculus turns the construction of fair derivations into a non-trivial problem. Both these rules have side conditions which are only semi-decidable. Thus, the construction of a derivation could potentially 'get stuck' while checking these side conditions. Take, for example, an attempt to apply the ground eventuality resolution rule to premises $\mathcal{A}_1 \Rightarrow \bigcirc\mathcal{B}_1, \ldots, \mathcal{A}_n \Rightarrow \bigcirc\mathcal{B}_n$ such that for some i, $1 \leq i \leq n$, either the condition $\mathcal{U} \wedge \mathcal{B}_i \models \neg l$ or the condition $\mathcal{U} \wedge \mathcal{B}_i \models \bigvee_{j=1}^{n} \mathcal{A}_j$ does not hold. Assume $\mathcal{U} \wedge \mathcal{B}_i \models \neg l$ does not hold. Then, an attempt to establish whether in first-order logic $\neg l$ follows from $\mathcal{U} \wedge \mathcal{B}_i$ may not terminate. Thus, executions of the FG-BFS algorithm in TeMP are a first source of unfairness as those executions are not guaranteed to terminate in general, with the consequence that the construction of a derivation by the theorem prover might not proceed beyond this point

Another problem is that in the main procedure of TeMP, the FG-BFS algorithm will only be executed once the original monodic temporal problem has been saturated using the rules of fine-grained step resolution. Again, this saturation process may not terminate even if the original monodic temporal problem is unsatisfiable.

6.3.3 Implementation of the Fair Inference Procedure

The representation of the fair inference procedure **F** allowed us to prove its refutational completeness. However, Figure 5.2 does not provide a basis yet for implementing the fair architecture as, for example, it is not immediately obvious in which order that the inference rules should be applied in practice. Moreover, the presentation style of Figure 5.2 conceals the problem of selecting clauses for inference computation in such a way that fair derivations are obtained. Despite having solved the fairness problems related to the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus, the fairness of derivations can still be lost if the clause selection is performed in an inappropriate way. As we will see in Section 6.4.10, a clause selection function that is fair in the context of regular first-order resolution does not necessarily guarantee the fairness of derivations constructed by the fair architecture.

Figure 6.2, then, depicts the main procedure of a monodic temporal logic prover based on the fair inference procedure in a way that is more amenable for a practical implementation.

After reading the initial clause set **S** we add to **S** all clauses required for loop search right from the start. This is done in step (1). The remainder of the procedure is mostly identical to the main procedure of a resolution-based theorem prover, confer e.g., [78,81,91] and Section 6.4.1. This part of the main procedure operates on two sets of clauses, $\mathcal{US}$ (the

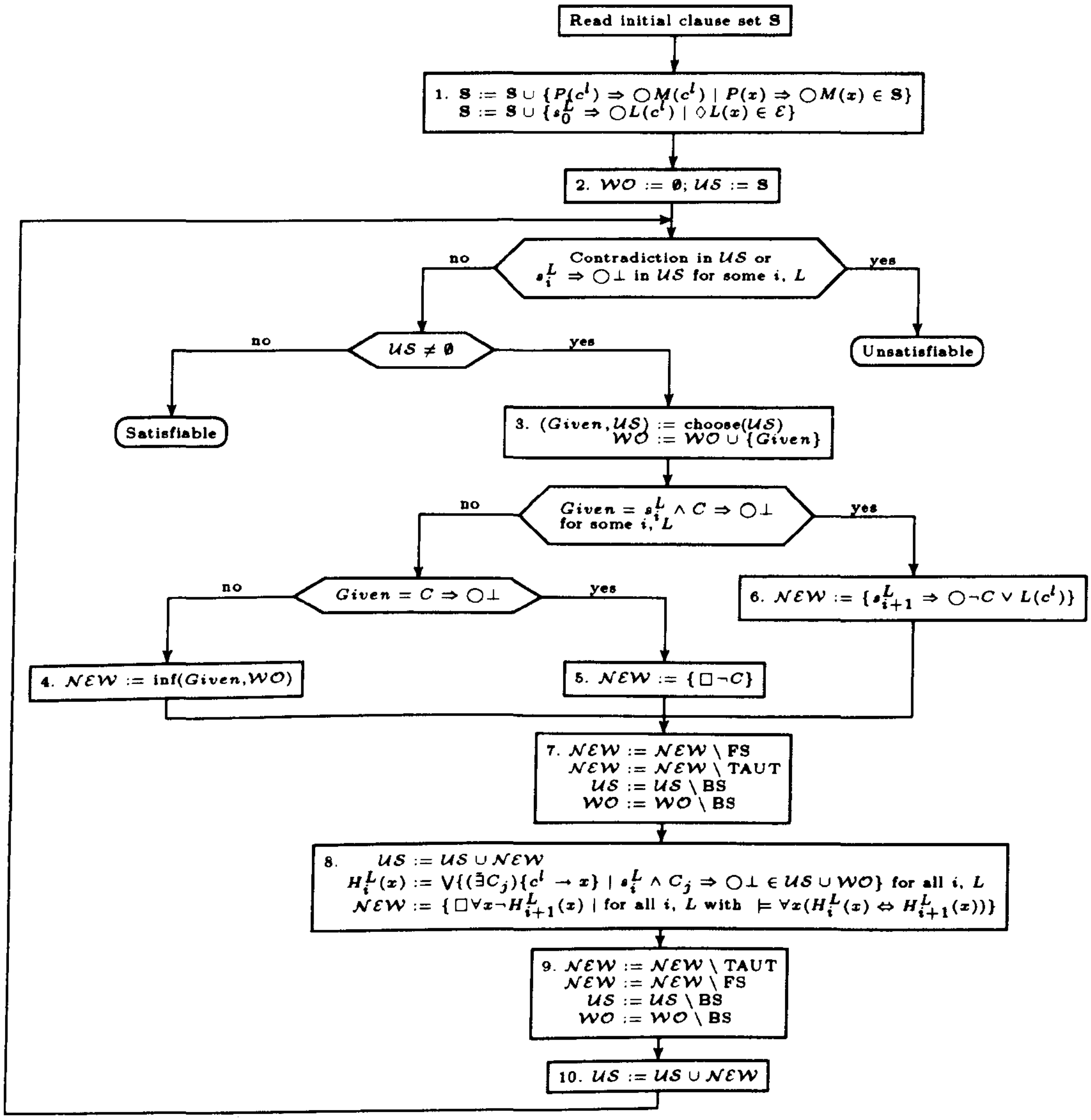

Read initial clause set S
1. $\mathbf{S} := \mathbf{S} \cup \{P(c^l) \Rightarrow \bigcirc M(c^l) \mid P(x) \Rightarrow \bigcirc M(x) \in \mathbf{S}\}$
$\mathbf{S} := \mathbf{S} \cup \{s_0^L \Rightarrow \bigcirc L(c^l) \mid \Diamond L(x) \in \mathcal{E}\}$
2. $\mathcal{WO} := \emptyset; \mathcal{US} := \mathbf{S}$
Contradiction in $\mathcal{US}$ or $s_i^L \Rightarrow \bigcirc\bot$ in $\mathcal{US}$ for some i, L
no
yes
Unsatisfiable
$\mathcal{US} \neq \emptyset$
no
yes
Satisfiable
3. $(Given, \mathcal{US}) :=$ choose$(\mathcal{US})$
$\mathcal{WO} := \mathcal{WO} \cup \{Given\}$
$Given = s_i^L \wedge C \Rightarrow \bigcirc\bot$ for some i, L
no
yes
$Given = C \Rightarrow \bigcirc\bot$
no
yes
6. $\mathcal{NEW} := \{s_{i+1}^L \Rightarrow \bigcirc\neg C \vee L(c^l)\}$
4. $\mathcal{NEW} :=$ inf$(Given, \mathcal{WO})$
5. $\mathcal{NEW} := \{\Box\neg C\}$
7. $\mathcal{NEW} := \mathcal{NEW} \setminus$ FS
$\mathcal{NEW} := \mathcal{NEW} \setminus$ TAUT
$\mathcal{US} := \mathcal{US} \setminus$ BS
$\mathcal{WO} := \mathcal{WO} \setminus$ BS
8. $\mathcal{US} := \mathcal{US} \cup \mathcal{NEW}$
$H_i^L(x) := \bigvee\{(\exists C_j)\{c^l \rightarrow x\} \mid s_i^L \wedge C_j \Rightarrow \bigcirc\bot \in \mathcal{US} \cup \mathcal{WO}\}$ for all i, L
$\mathcal{NEW} := \{\Box\forall x \neg H_{i+1}^L(x) \mid$ for all i, L with $\models \forall x(H_i^L(x) \Leftrightarrow H_{i+1}^L(x))\}$
9. $\mathcal{NEW} := \mathcal{NEW} \setminus$ TAUT
$\mathcal{NEW} := \mathcal{NEW} \setminus$ FS
$\mathcal{US} := \mathcal{US} \setminus$ BS
$\mathcal{WO} := \mathcal{WO} \setminus$ BS
10. $\mathcal{US} := \mathcal{US} \cup \mathcal{NEW}$

Figure 6.2: Saturation Architecture of TSPASS

set of *usable clauses*) and $\mathcal{WO}$ (the set of *worked-off clauses*). The set $\mathcal{WO}$ contains all the clauses that have already been used as premises in inference steps (or can never be used as premises) and the set $\mathcal{US}$ contains all the clauses that still need to be considered as premises. Initially, as set in step (2) of the procedure, the set $\mathcal{WO}$ is empty, while $\mathcal{US}$ contains all clauses of S. Next the procedure enters the main inference loop in which it remains as long as no contradiction has been derived and the set $\mathcal{US}$ is not empty. Note that the presence of $s_i^L \Rightarrow \bigcirc\bot$ in $\mathcal{US}$ indicates that we can apply the non-ground eventuality resolution rule resulting in the conclusion $\Box\forall x\neg\mathbf{true}$, which is contradictory. In the main loop, in step (3), we use the function *choose* to select and remove a clause, called the *given clause* from $\mathcal{US}$. The given clause is added to $\mathcal{WO}$. We then check whether the given clause is equal to $s_i^L \wedge C \Rightarrow \bigcirc\bot$ for some i and L. In the (Subsumption-Restricted)-FG-BFS algorithm, such a clause would be part of the set $\mathcal{M}_{i+1}$ which is used to define the formula $H_{i+1}(x)$, which in turn is used to define the clauses in $\mathcal{N}_{i+1}$. Here, in step (6), we directly define the one clause of $\mathcal{N}_{i+1}$ which derives from $s_i^L \wedge C \Rightarrow \bigcirc\bot$. If there is no such clause, but a clause of the form $C \Rightarrow \bigcirc\bot$ (without a 'marker' s_i^L), then such a clause is a suitable premise for the application of the clause conversion rule and we compute the conclusion of that application in step (5). Otherwise, the given clause is not subject to any special treatment. Instead, in step (4), we use the function *inf* to compute all conclusions derivable from the given clause and clauses in $\mathcal{WO}$ by the rules 1 to 5 and rule 7 of $\mathfrak{I}_{FG}^{S,\succ}$ and the arbitrary factoring in left-hand sides of terminating step clauses and in at most monadic negative universal clauses rules, i.e. the fine-grained step resolution rules of $\mathfrak{I}_{FG,Sub}^{S,\succ}$ except the clause conversion rule. In step (7) redundancy elimination, through backward (BS) and forward (FS) subsumption deletion, and the removal of tautologies (TAUT) is performed. The newly derived clauses that were not deleted in the previous step are added to $\mathcal{US}$ in step (8). We also prepare the loop condition test by computing the formulae H_i^L for all eventuality clauses $\Diamond L(x) \in \mathcal{E}$ and all indices i used to create some marker s_i^L in the clause set. We then check whether the loop condition $\forall x(H_i^L(x) \Leftrightarrow H_{i+1}^L(x))$ holds for every i and every L. If this is indeed the case, an application of the non-ground eventuality resolution rule is possible. We compute the conclusion of the application and add it to a set $\mathcal{NEW}$. In step (9) the set $\mathcal{NEW}$ is used in reductions again, and finally the remaining clauses are added to the set of usable clauses in step (10). This completes the main inference loop of the procedure.

Similarly to the inference procedure **F** depicted in Figure 5.2, there are several important observations to be made about step (8), which have already been partly described in Section 5.2.2.

First, we can observe that $H_i^L(x)$ and $H_{i+1}^L(x)$ are monadic first-order formulae. Thus, the validity of the loop condition is a decidable problem.

Second, in the (Subsumption-Restricted-)FG-BFS algorithm, in order to establish whether $\forall x(H_i^L(x) \Leftrightarrow H_{i+1}^L(x))$ is valid we only need to test whether $\forall x(H_i^L(x) \Rightarrow H_{i+1}^L(x))$ holds as the implication $\forall x(H_{i+1}^L(x) \Rightarrow H_i^L(x))$ is always valid by the construction of $H_i^L(x)$ and $H_{i+1}^L(x)$ in these procedures. However, in the context of the procedure in Figure 6.2 this is no longer the case and we need to test both implications.

Third, by Lemma 4.3.14 the validity of the loop search formula $\forall x(H_i^L(x) \Leftrightarrow H_{i+1}^L(x))$ is equivalent to the existence of mutual subsumptions for all the disjuncts that are found in the formulae

$$H_i^L(x) = \bigvee_{j=1}^{m} (\tilde{\exists} C_j)\{c^l \rightarrow x\} \text{ and } H_{i+1}^L(x) = \bigvee_{j=1}^{n} (\tilde{\exists} D_j)\{c^l \rightarrow x\},$$

respectively (if unordered factoring has been exhaustively applied on the sets $\{C_1, \ldots, C_m\}$ and $\{D_1, \ldots, D_n\}$). Additionally, by Lemma 4.3.10 it follows for two clauses $s_i^L \wedge C \Rightarrow \bigcirc\bot$, $s_i^L \wedge D \Rightarrow \bigcirc\bot$ with $C \leq_s D$ which have been derived during a run of the prover architecture shown in Figure 6.2 that the formula

$$\forall x((\tilde{\exists} D)\{c^l \rightarrow x\} \Rightarrow (\tilde{\exists} C)\{c^l \rightarrow x\})$$

is valid. We can thus infer that testing for subsumption on disjuncts of the formulae $H_i^L(x)$ and $H_{i+1}^L(x)$ is sufficient to find loop formulae in runs of the fair architecture. The subsumption test can lead to a significantly reduced number of derived clauses in contrast to using a logical validity test for testing loop search conditions.

Finally, whenever the loop condition holds, we have indeed found a loop formula, although it may not be equivalent to a formula returned by (Subsumption-Restricted)-FG-BFS as a loop formula. However, eventually, for each (negated) formula returned by the (Subsumption-Restricted)-FG-BFS algorithm as a loop formula the algorithm in Figure 6.2 will compute an analogous one.

In the context of Figure 6.2 a derivation is said to be *fair* if no clause remains indefinitely in the set of usable clauses $\mathcal{US}$. Consequently, provided that the function *choose* selects given clauses in a fair way using, for example, clause weight combined with clause age, the procedure in Figure 6.2 constructs a fair derivation. Thus, a derivation from an unsatisfiable monodic temporal problem in clausified form is guaranteed to eventually derive a contradiction. It is worthwhile to note that due to the introduction of new propositions during the derivation, some simpler clause selection functions which are fair for a fixed and finite signature may not be fair in the context of the algorithm in Figure 6.2 (see Section 6.4.10 for more details).

We conclude this section by stating the completeness of the fair architecture.

Theorem 6.3.1 (see Theorem 5.3.13). *Let P^c be a constant flooded and clausified monodic temporal problem. Let $\succ$ be an admissible atom ordering and S be an instance compatible selection function.*

Then P^c is unsatisfiable if and only if any fair run of the architecture depicted in Figure 6.2 using the atom ordering $\succ$ and selection function S returns "Unsatisfiable".

6.4 Implementation of TSPASS

The architecture depicted in Figure 6.2 has been implemented in the theorem prover TSPASS by modifying and extending the first-order prover SPASS 3.0 [92]. The transformation into

Algorithm 1 General Architecture of SPASS

```
 1: procedure SATURATION($N$)
 2:     $\mathcal{US} := N$
 3:     $\mathcal{WO} := \emptyset$
 4:     while $\mathcal{US} \neq \emptyset$ and $\bot \notin \mathcal{US}$ do
 5:         (Given, $\mathcal{US}$) := choose($\mathcal{US}$)
 6:         $\mathcal{WO} := \mathcal{WO} \cup \{\text{Given}\}$
 7:         $\mathcal{NEW} := \text{inf}(\text{Given}, \mathcal{WO})$
 8:         $\mathcal{NEW} := \text{red}(\mathcal{NEW}, \mathcal{US}, \mathcal{WO})$
 9:         $\mathcal{US} := \mathcal{US} \cup \mathcal{NEW}$
10:     end while
11:     if $\bot \in \mathcal{US}$ then
12:         print "Proof found."
13:     else
14:         print "Completion found."
15:     end if
16: end procedure
```

the normal form DSNF is performed by a separate program, whose output can be parsed by TSPASS.

We first recall the general architecture of SPASS (see [91]). Then, in Section 6.4.2 we present some basic considerations behind the implementation of TSPASS, before we describe TSPASS's general architecture in Section 6.4.3. In the subsequent section we briefly outline the implementation of the transformation into DSNF. The clausification and translation into first-order logic is described in Section 6.4.5, and efficient ways to access loop search clauses are presented in Section 6.4.6. The main inference method of TSPASS is explained in Section 6.4.7. Two ways to implement loop search tests are presented in the subsequent section. Finally, peculiarities related to the implementation of TSPASS are discussed in Section 6.4.9 and fairness problems regarding the clause selection function are analysed in Section 6.4.10.

6.4.1 Saturation Architecture of SPASS

The general architecture of the first-order prover SPASS is shown in Algorithm 1. Essentially, the prover operates on two sets of clauses $\mathcal{US}$ and $\mathcal{WO}$. The set $\mathcal{WO}$ of worked off clauses contains all the clauses on which all possible inferences (potentially with other clauses from the set $\mathcal{WO}$) have already been performed, whereas the set $\mathcal{US}$ of usable clauses contains clauses that still need to be considered for inferences with themselves or with clauses from the set $\mathcal{WO}$. We assume in the following that the sets of worked off and usable clauses are additionally represented by term index structures (see e.g. [77]).

In line 2 of Algorithm 1 the set of usable clauses $\mathcal{US}$ is initialised with the set of input clauses N. Lines 4 to 10 describe the saturation loop, which is executed as long as the set of usable clauses is not empty and the empty clause has not been derived. A clause is selected as "given clause" for inferences to be performed with through the "choose" function. The given clause is removed from the set of usable clauses and inserted into the set of worked-off clauses in lines 5 and 6. Then, inferences with the given clause are computed with respect to the set of worked-off clauses in line 7, and the resulting clauses from these inferences are stored in a set $\mathcal{NEW}$ of new clauses. In line 8 the new clauses are reduced with respect to the set of usable and worked-off clauses. Finally, the reduced set of new clauses is inserted into the set of usable clauses in line 9.

When the saturation loop terminates, the following two possibilities can therefore occur: either the set of usable clauses contains the empty clause, which implies that there exists a derivation of the empty clause from the initial set of clauses N. One can hence conclude that the set N is unsatisfiable. Alternatively, the set of usable clauses has become empty, from which we can infer that all possible non-redundant inferences have been computed. As the empty clause has not been derived, we can conclude that the initial set of clauses N is satisfiable.

6.4.2 Implementation Basics

We now start by describing some basic aspects regarding the implementation of TSPASS. Upon inspection of the temporal saturation architecture shown in Figure 6.2 one can see that different tasks are performed depending on the "type" of the given clause. In order to speed up the saturation process, a type field has been added to the clause data structure initially present in SPASS. The purpose of the type field is to store the temporal type of the clause. We distinguish between the following types:

- The initial clause type, representing clauses in which every literal contains the constant 0 as temporal argument, e.g. $\tilde{p}(0) \vee \neg\tilde{q}(0, x, y)$.

- The universal clause type, which characterizes clauses that only contain literals with one and the same variable as temporal argument, e.g. $\tilde{p}(t, x) \vee \tilde{q}(t) \vee \neg\tilde{r}(t, c)$.

- The loop search clause type identifies clauses that are used in the loop search process, i.e. they must contain exactly one loop search marker and at least one literal with a temporal successor term as temporal argument. Moreover, every literal with a variable as temporal argument must be negative and contain at most one term as argument (i.e. its arity must be less than or equal to 1). Finally, every literal must use the same temporal variable. An example of a loop search clause is the clause $\neg s_0^{L(x)} \vee \neg\tilde{p}(t, x) \vee \tilde{q}(s(t), h(x))$.

- The terminating loop search clause type is similar to the loop search clause type, except that literals with successor terms as temporal arguments must not occur, e.g. the clause $\neg s_0^{L(x)} \vee \neg\tilde{p}(t, x) \vee \tilde{q}(t)$ is a terminating loop search clause. One can observe that terminating loop search clauses are in fact universal clauses which contain a loop search marker.

- The step clause type represents clauses which contain at least one literal with a temporal successor term as temporal argument. Additionally, every literal with a variable as temporal argument must be negative and contain at most one argument. Finally, every literal must share the same temporal variable. An example for the step clause type is the clause $\neg\tilde{p}(t,x) \vee \tilde{q}(s(t),h(x))$.

The type of a clause is initialised whenever a clause is created, and updated whenever the clause is modified. Moreover, clauses are classified into the most restrictive type that applies.

The second important part of a problem in DSNF that needs to be implemented efficiently are eventualities. Firstly, in order to reduce the required implementation effort the different eventuality formulae $\forall x \Diamond L(x)$, $\Diamond L(c)$ or $\Diamond l$ are simply represented by unit clauses that contain the eventuality terms $L(x)$, $L(c)$ or l, respectively. The sometime operator $\Diamond$ does not occur in the eventuality clauses. Moreover, these eventuality clauses are kept separately from the other clauses in the implementation.

For the loop search process it is important that additional information is recorded for each of the eventuality clauses. In particular, if the validity of the loop search side conditions for a given eventuality $\Diamond L(x)$ is to be checked, the number of loop search markers that have been created so far has to be known. Additionally, the marker symbol for a given loop search iteration has to be easily retrievable so that the clauses of the form $s_i^{L(x)} \wedge C \Rightarrow \bigcirc\bot$ can be found for every $i \in \mathbb{N}$. Hence, in order to link eventualities with information about their loop search markers, we use the data structure depicted in Figure 6.3. The fields "map" and "mapSize" of the structure "Eventuality_Inf" describe an array of (loop search) marker symbols, which is required for quickly accessing a specific loop search marker symbol $s_i^{L(x)}$ given an eventuality predicate $L(x)$ and a loop search index i. The remaining field "markerCount" represents the number of markers that have been created so far for a given eventuality.

As every eventuality term is of the form $L(x)$, $L(c)$ or l, we use hash maps iteratively to provide fast access to the "Eventuality_Inf" objects for given eventuality clauses. On the first level two separate hash maps help distinguish between positive and negative eventualities. The hash map for negative eventualities and the hash table for positive eventualities map the top-level predicate symbol of unsigned eventuality terms to a second hash table. The two first level hash maps are accessed depending on the polarity of the given eventuality

```
typedef struct {
  SYMBOL*  map;
  NAT      mapSize;
  NAT      markerCount;
} EVENTUALITY_INF;
```

Figure 6.3: Data Structure Used to Link Eventualities with Loop Search Markers

term. Each of the entries in the two first level hash tables contains another hash table that maps the argument of the given unsigned eventuality term to an "Eventuality_Inf" object. In the case where the eventuality term does not contain an argument or the argument is a variable, we use the unique null symbol as entry key in the second hash table.

The initialisation of eventuality information objects for given eventualities is described in Algorithm 2 in greater detail. The input parameter of the procedure only consists of a set of eventuality clauses. An array of eventuality information objects is created in line 2 containing an entry for every considered eventuality, and the two top-level hashes are created in the lines 3 and 4. Then, a counter variable i is initialised (line 5) and an iteration is performed through all the eventualities in the lines 6 to 10. For every eventuality an eventuality information object is allocated, initialised and stored at position i in the array (line 7). The newly created eventuality information object is added to the two-level hash maps for the literal representing the considered eventuality by the procedure "AddEventualityInformation", which also handles the creation of second-level hash maps. Finally, the iteration counter is incremented in line 9.

The procedure for retrieving an eventuality information object given the literal representing an eventuality is shown in Algorithm 3. If the considered eventuality literal is negative, the variable "RootHash" is initialised with (a pointer to) the hash map representing negative eventualities (line 3), otherwise the hash map for positive eventualities is used (line 6). Additionally, the variable "EventualityLit" is made to point to the (positive) literal atom in the case where the eventuality literal is negative (line 4). The key for the hash map on the first-level is initialised in line 8 with the predicate symbol of the unsigned eventuality and the corresponding hash map is extracted in line 9. Then, in the case where the eventuality literal is propositional or the non-temporal argument of the literal predicate is a variable, the null symbol is used as key for the second-level hash map (line 11); otherwise, the constant (symbol) of the non-temporal literal argument is taken (line 13). Finally, the eventuality information object is extracted from the second-level hash map in line 15.

In the next section we present the general architecture of TSPASS.

Algorithm 2 Initialisation of Eventuality Information Objects

```
 1: procedure INITEVENTUALITYINFORMATION(EventualityClauses)
 2:     EventualityInfArray := new EventualityInf[Length(EventualityClauses)]
 3:     Create(PositiveEventualityHash)
 4:     Create(NegativeEventualityHash)
 5:     $i := 0$
 6:     for all Eventuality $\in$ EventualityClauses do
 7:         InitEventualityInformationStruct(EventualityInfArray[i], Eventuality)
 8:         AddEventualityInformation(GetEventualityLiteral(Clause),
                                       EventualityInfArray[$i$])
 9:         $i := i + 1$
10:     end for
11: end procedure
```

Algorithm 3 Retrieval of Eventuality Information Objects

```
procedure GetEventualityInformation(EventualityLit)
    if TopSymbol(EventualityLit) = Not then
        RootHash := NegativeEventualityHash
        EventualityLit := FirstArgument(EventualityLit)
    else
        RootHash := PositiveEventualityHash
    end if
    RootKey := TopSymbol(EventualityLit)
    SecondHash := GetValue(RootHash, RootKey)
    if NrOfArguments(EventualityLit) = 1 or IsVariable(SecondArg(EventualityLit)) then
        SecondKey := NullSymbol
    else
        SecondKey := TopSymbol(SecondArg(EventualityLit))
    end if
    return GetValue(SecondHash, SecondKey)
end procedure
```

6.4.3 General Architecture of TSPASS

The main method of TSPASS is described in Algorithm 4. The arguments of the main method are a set of formulae N in DSNF and a threshold parameter for loop search. In line 2 the set N is clausified and the resulting initial, universal or step clauses are translated into first-order logic by adding a temporal parameter to each literal. Additionally, four clause sets representing the different clause types in DSNF are returned, and in line 3 the set "InputClauses" is created, which will eventually contain the clauses on which the resolution and factoring inferences are performed. Lines 4 and 5 compute the set of constants that are initially contained in the problem; Skolem constants, which may be created during the clausification process, have to be removed from this set as step clauses and eventuality clauses do not need to be constant-flooded with Skolem constants. Then, the lines 6 to 9 perform the constant flooding using the set of clauses that was previously computed. The set of eventuality clauses is constant-flooded first and in a second step the loop search constant c^l is used in addition for performing the constant flooding on step clauses if there are eventualities contained in the problem. The newly obtained eventuality and step clauses are then added to the original sets of eventuality and step clauses, respectively (lines 10 to 12). The ordering on predicate and functional symbols that is necessary to maintain some restrictions imposed by the inference rules of the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus on the first-order level is computed in line 13 (a more detailed description is available in Section 6.4.5). In line 14 the eventuality information objects are initialised for the obtained eventuality clauses. The lines 15 to 19 then create a first loop search marker of iteration index 0 for each eventuality, which is then used in the construction of a first loop search clause for each eventuality. The second parameter of the "CreateLoopSearchStepClause" method specifies the iteration index for which the loop search clause will be built and the step literals that

Algorithm 4 Main Method

```
1: procedure MonodicProver($N$, LoopSearchThreshold)
2:     (InitialClauses, UniversalClauses, StepClauses, EventualityClauses)
                                                   = TemporalClausify($N$)
3:     InputClauses := InitialClauses ∪ UniversalClauses ∪ StepClauses
4:     Constants := ListOfConstants(InputClauses) ∪
                                   ListOfConstants(EventualityClauses)
5:     Constants := RemoveSkolemConstants(Constants)
6:     NewEventualityClauses := PerformConstantFlooding(EventualityClauses,
                                                               Constants)
7:     if length(EventualityClauses) $>$ 0 then
8:         Constants := Constants ∪ {LoopSearchConstant}
9:     end if
10:    NewStepClauses := PerformConstantFlooding(StepClauses, Constants)
11:    EventualityClauses := EventualityClauses ∪ NewEventualityClauses
12:    InputClauses := InputClauses ∪ NewStepClauses
13:    ComputeTemporalOrdering(StepClauses)
14:    InitEventualityInformation(EventualityClauses)
15:    for all Eventuality ∈ EventualityClauses do
16:        CreateLoopSearchMarker(Eventuality, 0)
17:        InitLoopSearchClause := CreateLoopSearchStepClause(Eventuality, 0, Nil)
18:        InputClauses := InputClauses ∪ {InitLoopSearchClause}
19:    end for
20:    TemporalSaturation(InputClauses, EventualityClauses, LoopSearchThreshold)
21: end procedure
```

should be contained in the loop search clause are given as third parameter. More precisely, "CreateLoopSearchStepClause(Eventuality, i, Literals)" creates and returns the loop search step clause $\neg s_i^L(t) \vee L(s(t), c^l) \vee \bigvee_{j=1}^{m} \neg A_j(s(t), t') \vee \bigvee_{j=1}^{n} \neg A'_j(s(t))$ if Eventuality $= \Diamond L(x)$ and Literals $= \{\neg A_1(t, t'), \ldots, \neg A_m(t, t'), \neg A'_1(t), \ldots, \neg A'_n(t)\}$. The loop search clauses are also added to the input clauses in line 18. Finally, in line 20 the temporal saturation method is invoked on the input and eventuality clauses (together with the loop search threshold), which then performs the main inference and reduction steps.

In the next section we describe the DSNF transformation.

6.4.4 Translation Into DSNF

For the implementation of the DSNF translation tool we have used some ideas originally introduced for the λ-calculus. Variables in formulae are represented by so-called de Bruijn indices (see, e.g., [54] for more details). A variable is represented by the number of quantifiers that need to be skipped in order to reach its binding quantifier in the formula's tree representation. For example, in de Bruijn notation the formula $\forall x \Box \exists y\, p(x, y)$ is represented by $\forall \Box \exists p(0, 1)$. The use of de Bruijn indices simplifies the comparison of formulae in the implementation.

An important part of the DSNF transformation consists in the verification of whether a

Algorithm 5 Verification of DSNF Format

```
procedure IsInDSNF(Formula)
    if $\neg$ContainsTemporalOperator(Formula) then
        return true
    end if
    if GetTopOperator(Formula) $\neq \Box$ then
        return false
    end if
    SubFormula := GetFirstSubformula(Formula)
    if $\neg$ContainsTemporalOperator(SubFormula) then
        return true
    end if
    if GetTopOperator(SubFormula) = $\forall$ then
        SubFormula := GetSecondSubformula(SubFormula)
        IsQuantified := true
    else
        IsQuantified := false
    end if
    if $\neg$IsQuantified and ContainsFreeVariables(SubFormula) then
        return false
    end if
    if GetTopOperator(SubFormula) = $\Rightarrow$ then
        LeftSubFormula := GetFirstSubformula(SubFormula)
        if GetTopOperator(GetSecondSubformula(SubFormula)) $\neq \bigcirc$ then
            return false
        end if
        RightSubFormula := GetFirstSubformula(GetSecondSubformula(SubFormula))
        if $\neg$IsPositiveLiteral(LeftSubFormula) or $\neg$IsLiteral(RightSubFormula) then
            return false
        end if
        if IsQuantified
            and ($\neg$ContainsAtMostTheFreeVariableZero(LeftSubFormula)
                or $\neg$ContainsAtMostTheFreeVariableZero(RightSubFormula) then
            return false
        end if
        return true
    else if GetTopOperator(SubFormula) = $\Diamond$ then
        SubFormula := GetFirstSubformula(SubFormula)
        if $\neg$IsLiteral(SubFormula) then
            return false
        end if
        if IsQuantified and $\neg$ContainsAtMostOneFreeVariable(SubFormula) then
            return false
        end if
        return true
    end if
    return false
end procedure
```

Algorithm 6 Clausification and Translation to First-Order Logic

```
1:  procedure TemporalClausify($N$)
2:      InitialClauses := $\emptyset$
3:      UniversalClauses := $\emptyset$
4:      StepClauses := $\emptyset$
5:      EventualityClauses := $\emptyset$
6:      for all Formula $\in N$ do
7:          if IsInitialFormula($N$) then
8:              NewClauses := FOClausify(Formula)
9:              InitialClauses := InitialClauses $\cup$
                                 {TranslateInitialClausesToFO(NewClauses)}
10:         else if IsUniversalFormula(Formula) then
11:             NewClauses := FOClausify(RemoveAlways(Formula))
12:             UniversalClauses := UniversalClauses $\cup$
                                 {TranslateUniversalClausesToFO(NewClauses)}
13:         else if IsStepFormula(Formula) then
14:             NewClause := TranslateStepFormulaToFO(Formula)
15:             StepClauses := StepClauses $\cup$ {NewClause}
16:         else if IsEventualityFormula(Formula) then
17:             NewClause := TranslateEventualityFormulaToFO(Formula)
18:             EventualityClauses := EventualityClauses $\cup$ {NewClause}
19:         end if
20:     end for
21:     return (InitialClauses, UniversalClauses, StepClauses, EventualityClauses)
22: end procedure
```

formula is already in DSNF format. The procedure described in Algorithm 5 is used for this purpose. The only input parameter is a formula and the value "true" is returned if and only if the input formula is in DSNF format.

If the input formula does not contain a temporal operator (line 2), then the given formula is an initial formula and "true" can be returned in line 3. Otherwise, the input must contain at least one temporal operator. If the formula does not start with an $\Box$ operator (line 5), it cannot be in DSNF format and the value "false" can safely be returned (line 6). In line 8 we can assume that the formula starts with an $\Box$ operator. The variable "SubFormula" is initialised with the subformula of the $\Box$ operator. Then, if the subformula does not contain a temporal operator (line 9), the input formula is of the type $\Box\varphi$ and we can return "true" (line 10). It is now also possible that "SubFormula" starts with a universal quantifier (line 12), in which case we assign the formula that is quantified over to "SubFormula" in line 13 as the remaining checks do not need to consider the universal quantifier. Additionally, the variable "IsQuantified" is set accordingly (lines 14 and 16). In the case where "SubFormula" does not start with a universal quantifier but contains free variables, we can return "false" (lines 18 to 20) as the input formula hence contains free variables.

If the top-most operator of "SubFormula" is an implication (line 21), the input formula could potentially be a step clause. The variable "LeftSubFormula" is initialised in line 22 with the left-hand side of the implication. Then, if the right-hand side of the implication does not start with a $\bigcirc$ operator (line 23), we can return "false" in line 24 as the input formula cannot be in DSNF format. Otherwise, the right-hand side of the implication is a formula starting with a $\bigcirc$ operator and the formula underneath the $\bigcirc$ operator is assigned to the variable 'RightSubFormula" (line 26). If the left-hand side of the considered implication is not an atom (i.e. a positive literal) or the immediate subformula of the $\bigcirc$ operator is not a literal (line 27), the input formula is not a step clause and the value "false" can be returned (line 28). The two remaining checks shown in the line 30 still verify that "LeftSubFormula" and "RightSubFormula" contain at most the free variable with index 0 if a preceding universal quantifier has been found in the input formula.

In the case where the immediate subformula of the $\Box$ operator (or universal quantifier) starts with an eventuality operator $\Diamond$ (line 34), the immediate subformula of the eventuality operator is assigned to the variable "SubFormula". If the immediate subformula is not a literal (line 36), we can return the value "false" (line 37). And similarly as in the case of step clauses, it is verified in the lines 39 to 41 that "SubFormula" only contains at most the variable with index 0 if a preceding quantifier has been detected in the input formula.

Finally, the value "false" is returned in line 44 if the immediate subformula of the $\Box$ operator (or universal quantifier) is not a formula that has an implication or an eventuality as top-most operator.

The implementation of the transformation into DSNF is based on the algorithm described in [58] and in Section 2.5. First of all, the input formula is brought into negation normal form. Then, innermost complex subformulae $\bigcirc\varphi(x)$, $\Diamond\varphi(x)$, $\Box\varphi(x)$, $\varphi(x)\,\mathsf{U}\,\psi(x)$, $\varphi(x)\,\mathsf{W}\,\psi(x)$ are successively renamed by new unary predicate symbols $P(x)$, introducing new defining formulae, called *pre-step* clauses,

$$\Box\forall x(P(x) \Rightarrow \Phi(x))$$

where $\Phi(x)$ is a formula starting with a temporal operator and every proper subformula of $\Phi(x)$ does not contain a temporal operator. In a final step the pre-step clauses that are not already in DSNF are transformed by using the fixed-point definitions of the temporal operators 'always' $\Box$, 'eventually' $\Diamond$, 'until' U and 'unless' W. In-between the different transformation steps the procedure depicted in Algorithm 5 is used to identify formulae that are already in DSNF, which are then excluded from future transformation steps.

In the next section we describe the procedures responsible for the clausification and translation into first-order logic of temporal formulae in DSNF.

6.4.5 Clausification and Translation to FOL

Algorithm 6 depicts the "TemporalClausify" method which is responsible for the clausification and the translation into first-order logic of a set of input formulae N in DSNF.

In the lines 2 to 5 the set of clauses that will contain the result of the clausification and temporal translation are initialised. Then, the temporal type of every formula contained

Algorithm 7 Computation of the Necessary Weight for the Temporal Successor Function

```
 1: procedure ComputeTemporalOrdering(Clauses)
 2:     LeftHandAtoms := Nil
 3:     MaxWeight := 0
 4:     for all Clause $\in$ Clauses do
 5:         if IsStepClause(Clause) then
 6:             LeftHandAtoms := LeftHandAtoms $\cup$ ExtractUniversalAtoms(Clause)
 7:         end if
 8:     end for
 9:     for all Atom $\in$ LeftHandAtoms do
10:         Weight := ComputeAtomWeight(Atom)
11:         MaxWeight := max(MaxWeight, Weight)
12:     end for
13:     SetAtomWeight(TemporalSuccessorFunction, MaxWeight + 1)
14: end procedure
```

in the set of input formulae N is determined in the lines 6 to 20 and the corresponding transformation method is invoked. In a last step the resulting clause is added to the appropriate output clause set in the lines 9, 12, 15 and 18.

The clausification process of an initial formula is straightforward and proceeds exactly as in the first-order case through the "FOClausify" method just as if no temporal operators are present. During the translation into first-order logic the arity of every predicate is increased by one argument and the temporal constant 0 is added as first argument to every literal contained in the initial clause. The clausification and translation into first-order logic of universal clauses is similar after the leading temporal operator $\Box$ has been removed, but instead of the temporal constant 0 a fresh variable (per clause) is added as temporal argument to the literals.

For a step clause $P(x) \Rightarrow \bigcirc Q(x)$ the clausification is trivial: it suffices to copy the literals and additionally negate the literal $P(x)$. Again, the arities of the predicates are increased by one unit and the literal $P(x)$ receives a variable fresh t as temporal argument, whereas the temporal argument of the literal $Q(x)$ will be the temporal successor term $s(t)$.

Finally, the clausification of eventualities is also straightforward. After having copied the literal to the clause, we still have to increase its arity by one and add a fresh variable as temporal argument.

The procedure for computing the symbol weight of the temporal successor function is shown in algorithm 7 (see Section 6.2 for more details). The algorithm only has one single input parameter, namely the set of (temporal) clauses that should be considered for the computation of the symbol weight. After the variables "LeftHandAtoms" and "MaxWeight" have been initialised in the lines 2 and 3, an iteration is then performed through all the input clauses (lines 4 to 8). If an input clause is identified to be a step clause, all its universal atoms, i.e. the atoms from its left-hand side, are added to the set "LeftHandAtoms" (lines 5 to 7). The weight of every left-hand atom is then computed and the maximum of these weights is calculated by consecutively adjusting the "MaxWeight" variable (lines 9 to 12).

Algorithm 8 Efficient Retrieval of Loop Search Clauses

```
procedure GetTerminatingLoopSearchStepClauses(Eventuality, $i$,
                                               $\mathcal{WO}$Index, $\mathcal{US}$Index)
    Clauses := GetTerminatingLoopSearchStepClausesFromIndex(Eventuality, i,
                                                            $\mathcal{US}$Index)
    return Clauses $\cup$ GetTerminatingLoopSearchStepClausesFromIndex(Eventuality, $i$,
                                                            $\mathcal{WO}$Index)
end procedure
```

Algorithm 9 Efficient Retrieval of Loop Search Clauses from an Index

```
procedure GetTerminatingLoopSearchStepClausesFromIndex(Eventuality, $i$,
                                                        Index)
    Clauses := Nil
    EventualityInf := GetEventualityInformation(Eventuality)
    LoopSearchMarker := GetLoopSearchMarker(EventualityInf, $i$)
    Terms := GetUnifiers(Index, LoopSearchMarker)
    for all Term $\in$ Terms do
        if IsVariable(Term) then
            continue
        end if
        for all Literal $\in$ GetAssociatedLiterals(Term) do
            Clause := LiteralOwningClause(Literal)
            if IsTerminatingStepClause(Clause) then
                Clauses := Clauses $\cup$ {Clause}
            end if
        end for
    end for
    return Clauses
end procedure
```

Finally, the weight of the temporal successor function is set to be the maximum of the left-hand side atom weights, increased by one unit (line 13).

6.4.6 Efficient Access to Loop Search Clauses

The temporal saturation procedure which is described in Section 6.4.7 requires fast access to the terminating loop search step clauses that are present in the sets of usable and worked-off clauses for a given eventuality and loop search iteration index. As the sets of usable and worked-off clauses can become quite large and as the terminating loop search step clauses need to be accessed frequently, it is, performance-wise, not feasible to iterate through the sets of usable and worked-off clauses every time the terminating loop search clauses need to be accessed.

In order to keep the retrieval performance at an acceptable level, it turns out that the same clause retrieval procedure that is responsible for finding unifiable literals during the computation of resolution inferences can be used for the retrieval of terminating loop search clauses. More concretely, given an eventuality $\Diamond L(x)$ and a loop search iteration index i the terminating loop search step clauses for the eventuality $\Diamond L(x)$ and the iteration index i can be found by searching for all the clauses which contain a literal that is unifiable with the loop search marker $s_i^{L(x)}$. As loop search markers do not possess any arguments and as they only occur in left-hand sides of loop search clauses, the aforementioned retrieval operation returns all the terminating and non-terminating loop search clauses related to the marker $s_i^{L(x)}$. An additional filtering step is required in order to only obtain the terminating loop search clauses.

The procedures depicted in Algorithms 8 and 9 show how the retrieval of terminating loop search clauses has been implemented. The method "GetTerminatingLoopSearchStepClauses" shown in Algorithm 8 uses the procedure described in Algorithm 9 to retrieve the terminating loop search clauses from the sets of usable and worked-off clauses given an eventuality and a loop search iteration index.

The main method for retrieving terminating loop search clauses is shown in Algorithm 9. Its input parameters are the eventuality and loop search iteration index in question, together with the indexing structure that should be used for the retrieval operations. The set "Clauses", which will contain the resulting clauses, is initialised in line 2 and the appropriate loop search marker is retrieved in line 4 through an eventuality information object, which is accessed in line 3 by the "GetEventualityInformation" procedure shown in Algorithm 3. The method "GetLoopSearchMarker" retrieves the loop search marker symbol for a given loop search iteration from a "Eventuality_Inf" object. Then, a set "Terms" is constructed in line 5, which contains all the atom (or variable) occurrences that can be unified with the considered loop search marker. In lines 6 to 16 an iteration is performed through the "Terms" set. Variables contained in the term set are discarded (lines 7 to 9) and for every non-variable the clauses associated to the atom in question are retrieved (lines 10 and 11). The procedure "GetAssociatedLiterals" first retrieves all the literal instances that contain a given atom, and then the method "LiteralOwningClause" fetches the clause that is associated with the literal instance in question. Finally, if a considered clause is a terminating step clause, it is added to the set of returned clauses (line 13).

In the subsequent section we describe the main inference and reduction procedure of TSPASS.

6.4.7 Temporal Saturation

The main inference and reduction method of TSPASS is depicted in Algorithm 10. Similarly to the saturation method of SPASS (see Algorithm 1), TSPASS also uses sets of worked off and usable clauses. The input parameters of TSPASS' saturation procedure are a set of first-order input clauses, a set of first-order eventuality clauses and a threshold parameter for performing loop search tests.

In line 2 the set of usable clauses is initialised with the set of input clauses, and the set of worked off clauses is set to be empty (line 3). The main saturation loop extends from line 5 to line 33; in line 4 a variable i is initialised which counts the iterations of the

Algorithm 10

```
procedure TemporalSaturation($N$, Eventualities, LoopSearchThreshold)
    $\mathcal{US} := N$
    $\mathcal{WO} := \emptyset$
    i := 0
    while $\mathcal{US} \neq \emptyset$ and $\bot \notin \mathcal{US}$ do
        (Given, $\mathcal{US}$) := choose($\mathcal{US}$)
        $\mathcal{WO} := \mathcal{WO} \cup \{\text{Given}\}$
        if IsTerminatingStepClause(Given)
          and ContainsExactlyOneLoopSearchMarker(Given) then
            if length(Given) = 1 then
                $\mathcal{NEW} := \{\bot\}$
            else
                $\mathcal{NEW}$ := {CreateNextLoopSearchStepClause(Given)}
            end if
        else
            $\mathcal{NEW}$ := inf(Given, $\mathcal{WO}$)
        end if
        if $\mathcal{NEW} \neq \emptyset$ then
            $\mathcal{NEW}$ := DeleteDifferentMarkerLoopSearchClauses($\mathcal{NEW}$)
            $\mathcal{US} := \mathcal{US} \cup \mathcal{NEW}$
            ($\mathcal{US}$, $\mathcal{WO}$) := red($\mathcal{US}$, $\mathcal{WO}$)
        end if
        if $\mathcal{US} = \emptyset$ or $i$ = LoopSearchThreshold then
            $\mathcal{NEW} := \emptyset$
            for all Eventuality $\in$ Eventualities do
                $\mathcal{NEW} := \mathcal{NEW} \cup$ PerformLoopSearchTest(Eventuality)
            end for
            if $\mathcal{NEW} \neq \emptyset$ then
                $\mathcal{US} := \mathcal{US} \cup \mathcal{NEW}$
                ($\mathcal{US}$, $\mathcal{WO}$) := red($\mathcal{US}$, $\mathcal{WO}$)
            end if
            $i := 0$
        end if
        $i := i + 1$
    end while
    if $\bot \in \mathcal{US}$ then
        return Unsatisfiable
    else
        return Satisfiable
    end if
end procedure
```

saturation loop. A clause is selected as "given clause" in line 6 and it is removed from the set of usable clauses, i.e. a new set $\mathcal{US}$ is returned from which the given clause has been removed. In line 7 the given clause is already added to the set of worked off clauses. Then, it is checked in line 8 whether the given clause is a terminating step clause and contains a loop search marker. Additionally, if the given clause only contains a single literal (line 9), we can conclude that the given clause is of the form $s_i^L \Rightarrow \bigcirc\bot$ (for an eventuality L and an iteration index i). We can thus infer that the initial temporal problem is unsatisfiable and we assign a singleton set which only contains the empty clause to the set of derivable clauses $\mathcal{NEW}$. This step replaces the test for the presence of the clause $s_i^L \Rightarrow \bigcirc\bot$ in the set of usable clauses $\mathcal{US}$ depicted in the case distinction following step 2 in Figure 6.2, which would lead to a poor performance of the prover architecture in practice. Then, if the given clause contains more than one literal, a new loop search clause based on the literals contained in the given clause is created in line 12 for the subsequent loop search iteration by the method "CreateNextLoopSearchStepClause". It is given a terminating loop search clause $\neg s_i^L(t) \vee \bigvee_{j=1}^m \neg A_i(t,t') \vee \bigvee_{j=1}^n \neg A_i'(t)$ as its only argument (for an eventuality $\Diamond L(x)$) and it returns the loop search step clause $\neg s_{i+1}^L(t) \vee L(s(t), c^l) \vee \bigvee_{j=1}^m \neg A_i(s(t),t') \vee \bigvee_{j=1}^n \neg A_i'(s(t))$ created through the "CreateLoopSearchStepClause" method. In line 12 the set $\mathcal{NEW}$ of derivable clauses then just consists of the new loop search clause. Otherwise, if the given clause is not a terminating step clause which contains a loop search marker, the given clause is used in line 15 to compute all the possible inferences with the set of worked off clauses. The newly obtained clauses are stored in the set of derivable clauses $\mathcal{NEW}$.

In the case where the set of derived clauses $\mathcal{NEW}$ is not empty, we have to perform reduction inferences in order to keep the sets of usable and worked off clauses fully reduced. An important reduction step consists in deleting clauses that contain at least two different loop search markers (see line 18) as they are not necessary for the overall saturation process. This deletion is performed by the method "DeleteDifferentMarkerLoopSearchClauses". Moreover, the removal of loop search clauses that contain at least two different loop search markers contributes to the possible termination of the saturation process for a satisfiable (first-order) temporal problem. After having added the set of newly derived clauses to the set of usable clauses (line 19), the sets of usable and worked off clauses are inter-reduced (line 20) and the reduced sets are returned by the reduction procedure.

Loop search tests are performed in the lines 22 to 31 whenever the set of usable clauses is empty or the iteration counter has reached the loop search threshold. Note that it is important to perform loop search tests when the set of usable clauses has become empty in order to avoid the situation where additional universal clauses could be derived through loop search but the loop search is not started due to the iteration counter not having reached the loop search threshold. Then, in the lines 23 to 26 loop search tests are performed for every eventuality and the possibly resulting universal clauses are collected in a set $\mathcal{NEW}$ of newly derived clauses. If new clauses have been obtained, they are added to the set of usable clauses in the lines 27 to 30, and the sets of usable and worked off clauses are inter-reduced again. Finally, the loop search iteration counter is reset in line 31.

Algorithm 11 Subsumption-based Loop Search Testing Procedure

```
 1: procedure PerformSubsumptionLoopSearchTest(Eventuality, $\mathcal{US}$, $\mathcal{WO}$)
 2:     LoopSearchClauses := $\emptyset$
 3:     LastIterationIndex := RetrieveLastIterationIndex(Eventuality)
 4:     if LastIterationIndex $\geq 1$ then
 5:         Clauses := GetTerminatingLoopSearchStepClauses(EventualityClause, 0, $\mathcal{US}$,
                                                                              $\mathcal{WO}$)
 6:     end if
 7:     for $i = 0, \ldots,$ LastIterationIndex $- 1$ do
 8:         NextClauses := GetTerminatingLoopSearchStepClauses(EventualityClause,
                                                            $i + 1$, $\mathcal{US}$, $\mathcal{WO}$)
 9:         if Clauses $\neq \emptyset$ and NextClauses $\neq \emptyset$ then
10:             if SetSubsumes(Clauses, NextClauses)
                   and SetSubsumes(NextClauses, Clauses) then
11:                 LoopSearchClauses := LoopSearchClauses $\cup$
                                          ComputeLoopSearchResult(NextClauses)
12:             end if
13:         end if
14:         Clauses := NextClauses
15:     end for
16:     return LoopSearchClauses
17: end procedure
```

Similarly to the saturation procedure of SPASS, if the empty clause is contained in the set of usable clauses, the set of input clauses N is unsatisfiable (lines 35 and 36). Otherwise, the set of usable clauses is empty, which implies that the set N is satisfiable (line 38).

In the next section we describe the implementation of the subsumption-based and logical loop search test that can be performed by TSPASS.

6.4.8 Loop Search Testing

The subsumption-based loop search test is depicted in Algorithm 11. The input parameters of the procedure are the eventuality for which the loop search tests have to be performed and (indexes for the) sets of usable and worked off clauses. In the lines 2 and 3 the set "LoopSearchClauses" of newly derived universal clauses (through an application of the eventuality resolution rule) is initialised and the last loop search iteration index for the given eventuality is retrieved. The remainder of the procedure works with two sets of clauses, "Clauses" and "NextClauses", which contain the terminating step clauses that have been derived for two consecutive loop search iterations i and $i + 1$, respectively. In line 5 the set "Clauses" is initialised with the terminating loop search clauses contained in the sets of usable and worked off clauses for the iteration index 0. Then, in the lines 7 to 15, the terminating loop search clauses are first of all retrieved for the subsequent loop search iteration index and are stored in the "NextClauses" set. If both the "Clauses" and "NextClauses" set are not empty (line 9), a loop search test can be performed for those two loop search iteration indexes. The procedure "SetSubsumes", which is invoked twice in line 10, checks whether

Algorithm 12 Loop Search Testing Procedure Based on Logical Equivalence

```
 1: procedure PerformLogicalLoopSearchTest(Eventuality, $\mathcal{US}$, $\mathcal{WO}$)
 2:     LoopSearchClauses := $\emptyset$
 3:     LastIterationIndex := RetrieveLastIterationIndex(Eventuality)
 4:     $H_i$ := Nil
 5:     $H_{i+1}$ := Nil
 6:     if LastIterationIndex >= 1 then
 7:         Clauses := GetTerminatingLoopSearchStepClauses(EventualityClause, 0, $\mathcal{US}$,
                                                                              $\mathcal{WO}$)
 8:         $H_i$ := BuildIntermediateLoopSearchResult(Clauses)
 9:     end if
10:     for $i = 0, \ldots,$ LastIterationIndex $- 1$ do
11:         NextClauses := GetTerminatingLoopSearchStepClauses(EventualityClause,
                                                          $i + 1, \mathcal{US}, \mathcal{WO}$)
12:         $H_{i+1}$ := BuildIntermediateLoopSearchResult(NextClauses)
13:         UnifyVariables($H_i$, $H_{i+1}$)
14:         if $H_i \neq$ Nil and $H_{i+1} \neq$ Nil then
15:             $x_t$ := TemporalVariable($H_i$)
16:             $x$ := FreeVariable($H_i$, $H_{i+1}$)
17:             $H := \forall x_t \forall x\colon H_i(x_t, x) \Leftrightarrow H_{i+1}(x_t, x)$
18:             if TemporalSaturation(Clausify($\neg H$), $\emptyset$, $-1$) = Unsatisfiable then
19:                 LoopSearchClauses := LoopSearchClauses $\cup$
                                    ComputeLoopSearchResult(NextClauses)
20:             end if
21:         end if
22:         Clauses := NextClauses
23:         $H_i := H_{i+i}$
24:     end for
25:     return LoopSearchClauses
26: end procedure
```

every clause contained in the set given as first argument subsumes at least one of the clauses contained in the set given as second argument, disregarding the loop search marker included in each clause. If every clause from the "Clauses" set subsumes at least one clause from the "NextClauses" set and vice versa, a loop has been found. The resulting universal clauses are computed in line 11 through the "ComputeLoopSearchResult" method. It is given a list of terminating loop search clauses as input and it computes the resulting universal clauses by copying the corresponding literals except the the loop search marker literal to a new clause and by replacing the loop search constant c^l by a fresh variable, i.e. for the terminating loop search clause $\neg s_i^L(t) \vee \bigvee_{j=1}^{m} \neg A_j(t, c^l) \vee \bigvee_{j=1}^{n} \neg A'_j(t, x_j) \vee \bigvee_{j=1}^{l} \neg A''_j(t)$ the method "ComputeLoopSearchResult" returns the universal clause $\bigvee_{j=1}^{m} \neg A_j(t, x) \vee \bigvee_{j=1}^{n} \neg A'_j(t, x_j) \vee \bigvee_{j=1}^{l} \neg A''_j(t)$, where x is a fresh variable. The resulting universal clauses are added to the set "LoopSearchClauses", which holds all the universal clauses that are obtained through successful loop search tests. Finally, in line 14 the set "NextClauses" is assigned to the set "Clauses" so that the set "NextClauses" can contain the terminating loop search clauses for the subsequent iteration index.

The procedure for logical loop search testing is shown in Algorithm 12. It is organised similarly to the subsumption-based loop search testing method. In addition to collecting clauses for consecutive loop search iterations in the two sets of clauses "Clauses" and "NextClauses", two formulae H_i and H_{i+1} are constructed that represent the disjunctions of the existentially closed (and freed of the loop search constant c^l) left-hand sides of the terminating step clauses obtained for two consecutive loop search iterations (see Figure 6.2 for details). The two formulae, or "terms" in the terminology of SPASS, are initialised in the lines 4 and 5. After having obtained the set of terminating step clauses for the loop search iteration that is currently being analysed (line 11) the corresponding formula H_{i+1} is constructed in line 12. The free variables, i.e. the temporal variable and the free variable resulting from replacing the loop search constant c^l, of the formulae H_i and H_{i+1} are unified in line 13, and if both formulae are defined, a loop search test can be performed (lines 14 to 21). In order to construct the equivalence formulae necessary for the loop search test, the temporal variable x_t and the free variable[2] x of the formulae H_i (and H_{i+1}) still have to be determined (lines 15 and 16). Then, in line 17 a formula H is defined which is valid if and only if the loop search test is successful for the terminating loop search clauses contained in the sets "Clauses" and "NextClauses" (after having removed the loop search markers). Note that the loop search equivalence formula shown in Figure 6.2 only quantifies universally over the free variable x, but the formula H also quantifies over the temporal variable x_t, which avoids the creation of new predicates only for the loop search test. It can be shown that the formula H is valid if and only if the formula shown in Figure 6.2, which uses the same predicates, but with reduced arities, is valid.

In order to test for validity the negation of the formula H is clausified in line 18 and the resulting set of clauses is saturated by the "TemporalSaturation" procedure (see Algorithm 10). As the value "-1" will be be passed to the"TemporalSaturation" procedure as loop search threshold (and the empty set for the parameter handling eventualities), the saturation which is performed is in fact equivalent to a regular first-order saturation. Then, if the "TemporalSaturation" procedure returns "Unsatisfiable", the formula H is valid and the loop search test has succeeded. Finally, the resulting universal clauses are computed in line 19 and added to the set "LoopSearchClauses", which collects the newly obtained universal clauses.

6.4.9 Implementation Peculiarities

In this section we describe some additional modifications that had to be carried out on the source code of SPASS in order to support ordered fine-grained resolution with selection.

First of all, the implementation of the (ordered) factoring inference rule had to be modified in order to accommodate the factoring in the (at most) monadic negative universal clauses rule. The modifications simply consist in switching to unordered factoring when

[2] H_i and H_{i+1} can have at most one free variable, but it is theoretically possible that at least one of the two formulae is closed.

factoring inferences on a (at most) monadic negative universal clause are to be performed.

Another peculiarity of implementing ordered fine-grained resolution with selection and the fair inference architecture is related to the missing distinction between terminating step clauses of the form $C \Rightarrow \bigcirc\bot$ and universal clauses on the first-order level. For example, if a terminating loop search clause $s_i^{L(x)} \wedge P(x) \Rightarrow \bigcirc\bot$ has been derived (for a unary predicate $P(x)$, a loop search iteration i and an eventuality $\Diamond L(x)$), the only possible inference on this clause in the original calculus would be an application of the clause conversion rule. However, such inferences are not allowed on loop search clauses in the fair architecture. On the other hand, on the first-order level the terminating step clause $s_i^{L(x)} \wedge P(x) \Rightarrow \bigcirc\bot$ is represented as the clause $\neg s_i^{L(x)} \vee \neg\tilde{P}(t,x)$, where t is a temporal variable. For instance, it would now be allowed to perform a resolution inference on the literal $\neg\tilde{P}(t,x)$ with the unit clause $\tilde{P}(t,x)$, which would derive the terminating loop search clause $s_i^{L(x)} \Rightarrow \bigcirc\bot$. The presence of the terminating step clause $s_i^{L(x)} \Rightarrow \bigcirc\bot$ would immediately result in the prover architecture shown in Figure 6.2 to qualify the currently considered problem as unsatisfiable, which could potentially lead to a loss of soundness. To avoid this, the implementation of the (ordered) resolution inference rule has been modified in such a way that inferences are not performed on terminating loop search clauses.

Finally, the problems mentioned in Section 6.2 related to the special ordering used for performing resolution with step clauses on the first-order level also had to be considered in the implementation. After a clause has been modified, a reinitialisation function is called in the original SPASS prover which ensures that the clause data structure remains in a well-defined state. This reinitialisation procedure has been extended in order to remove the maximal literal flag in left-hand sides of step clauses. The properties of the literal ordering used guarantee that at least one literal remains maximal in the right-hand side of a step clause even after the maximal literal flags have been cleared in the left-hand side.

6.4.10 Fairness Problems Related to the Regular Clause Selection Function

While conducting experiments with an earlier prototype of TSPASS it became apparent that the regular clause selection function of SPASS is not sufficient to guarantee the fairness of the architecture depicted in Figure 6.2.

In fact, TeMP's architecture (Figure 6.1) can be emulated by alternating between loop search and non-loop search clauses in the clause selection function whenever no clause of the currently considered type (i.e. either loop search or non-loop search) is contained in the set of usable clauses anymore.

Consequently, it must be ensured that non-loop search clauses are not continuously selected as "given" clauses as such a behaviour could lead to a loss of fairness.

On the other hand, special care needs to be taken when loop search clauses are to be chosen as "given" clauses. The main criterion that SPASS 3.0 considers when selecting a "given" clause is the size of clauses: smaller clauses are preferred over clauses with a larger number of literals. Now, in the situation when the selection strategy only consists in choosing a loop search clause as "given" clause after a certain amount of non-loop search clauses have been chosen, it could happen that the smallest clause contained in the set of usable clauses is a terminating loop search clause $C = s_i^l \wedge C \Rightarrow \bigcirc\bot$ (in high-level notation). If we additionally assume that a loop for the eventuality $\Diamond l$ exists and that the equally small terminating loop search clause $s_{i+1}^l \wedge C \Rightarrow \bigcirc\bot$ can easily be derived once the loop search clause C has been selected, the architecture shown in Figure 6.2 (page 130) can get "stuck" in the loop process. Due to the introduction of new clauses $s_{i+1}^l \Rightarrow \neg C \vee l$ once a terminating loop search clause $s_i^l \wedge C \Rightarrow \bigcirc\bot$ has been found, it might happen that the architecture continuously derives new terminating loop search clauses $s_{i+2}^l \wedge C \Rightarrow \bigcirc\bot$, $s_{i+3}^l \wedge C \Rightarrow \bigcirc\bot, \ldots$. Other loop search clauses, which would lead to a loop being found, will not be considered by the architecture.

As a remedy a special way of choosing loop search clauses as "given" clauses has been implemented in TSPASS. The implemented loop search selection strategy only tends to consider inferences with a clause $s_i^l \wedge C \Rightarrow \bigcirc D$ when the set of usable clauses does not contain a clause $s_j^l \wedge E \Rightarrow \bigcirc F$ with $j < i$. In order to guarantee fairness a free selection of loop search clauses is performed after a certain number of these "sequential" loop search clause selections.

Finally, in order to ensure that neither loop search clauses nor non-loop search clauses are continuously selected, and, for example, more and more marker symbols are introduced without considering non-loop search clauses, a mechanism which alternates between loop search and non-loop search clauses during the selection process has been implemented in TSPASS. The refined loop search clause selection strategy described above together with an alternation between loop search and non-loop search clause selections has proven to be quite effective in practice so far.

6.5 Experimental Results

In this section we describe the experiments that we have performed in order to evaluate the practical performance of TSPASS.

6.5.1 Considered Temporal Problems

For analysing the performance of TSPASS on PLTL problems, we have applied TSPASS on the benchmark classes for PLTL formulae that were introduced in [53]. There, the authors design two benchmark classes $\mathcal{C}_{ran}^1$ and $\mathcal{C}_{ran}^2$ which can be used to compare resolution and tableaux-based PLTL provers. Both benchmark classes consist of formulae which partly follow a pre-defined structure. The remaining part of those formulae is composed of randomly generated clauses. The classes are parametric with four parameters, l, n, k, and p. Regarding the randomly generated part of a formula, the parameter l determines the number of clauses, the parameter n specifies the number of propositions, the parameter k defines the number of literals per clause, and the parameter p determines the probability with which a literal is assigned negative polarity. The pre-defined part of a formula is only influenced by the parameter n.

Formulae of the class C^1_{ran} are of the form

$$\begin{aligned}
&\Box(\bigcirc L^1_1 \vee \ldots \vee \bigcirc L^1_k) \wedge \ldots \wedge \Box(\bigcirc L^l_1 \vee \ldots \vee \bigcirc L^l_k)\\
\wedge\ &\Box(\neg p_1 \vee \Diamond p_2)\\
\wedge\ &\Box(\neg p_2 \vee \Diamond p_3)\\
&\vdots\\
\wedge\ &\Box(\neg p_n \vee \Diamond p_1),
\end{aligned}$$

where for every i with $1 \leq i \leq l$ the literals $L^i_1, \ldots, L^i_k$ are determined by choosing k distinct propositional variables randomly from the set $\{p_1, \ldots, p_n\}$ of n propositional variables and by assigning negative polarity to each literal with probability p.

Formulae of the class C^2_{ran} are constructed as follows:

$$\begin{aligned}
&(r_1 \vee L^1_1 \vee \ldots \vee L^1_k) \wedge \ldots \wedge (r_1 \vee L^l_1 \vee \ldots \vee L^l_k)\\
\wedge\ &\Box(\neg r_n \vee \bigcirc r_1)\\
\wedge\ &\Box(\neg r_{n-1} \vee \bigcirc r_n)\\
&\vdots\\
\wedge\ &\Box(\neg r_1 \vee \bigcirc r_2)\\
\wedge\ &\Box(\neg r_n \vee \bigcirc \neg q_n) \wedge \ldots \wedge \Box(\neg r_1 \vee \bigcirc \neg q_n)\\
\wedge\ &(\neg r_1 \vee q_1) \wedge (\neg r_1 \vee \neg q_n)\\
\wedge\ &\Box(\neg q_1 \vee \Diamond s_2) \wedge \Box(\neg s_2 \vee q_2 \vee \bigcirc q_n \vee \ldots \vee \bigcirc q_3)\\
&\vdots\\
\wedge\ &\Box(\neg q_{n-1} \vee \Diamond s_n) \wedge \Box(\neg s_n \vee q_n)
\end{aligned}$$

where, again, for every i with $1 \leq i \leq l$ the literals $L^i_1, \ldots, L^i_k$ are determined by choosing k pairwise different propositional variables randomly from the set $\{p_1, \ldots, p_n\}$ of n propositional variables and by assigning negative polarity to each literal with probability p. In the experiments considered here, the values for k and p were set to 3 and 0.5, respectively, for both classes. For the parameter n we chose the two values 5 and 12.

The benchmark classes, C^1_{ran} $(n = 5)$ and C^1_{ran} $(n = 12)$ are designed in such a way that formulae contained in those classes can be theoretically solved more easily by resolution-based decision procedures, whereas the second set of benchmark classes, C^2_{ran} $(n = 5)$ and C^2_{ran} $(n = 12)$, are designed so that the formulae in them can be theoretically solved more easily by tableaux-based systems.

Then, in order to evaluate the performance of TSPASS on its main input language, the monodic fragment of FOTL, we consider several types of problems, some of which result from translating problem specifications expressed in the temporal logic of knowledge $KL_{(n)}$

into monodic FOTL. The logic $KL_{(n)}$ is the fusion of PLTL with the multi-modal S5 logic (see, e.g., [35] for more details); a description of the translation into monodic FOTL that we used can be found in [36].

In [27,28] a specification of the board game Cluedo [1] in $KL_{(n)}$ is given. The specification describes an example Cluedo game, for which one can formally prove that the different players can deduce additional knowledge over the course of the game. We have applied TeMP and TSPASS on six valid assertions that can be made in this example Cluedo game, after having transformed the $KL_{(n)}$ formulae into monodic FOTL.

Another application of the $KL_{(n)}$ logic for formal verification purposes can be found in [30,31]. There, a formal specification of the Needham-Schroeder protocol is given, which is designed to authenticate two communicating agents A and B against each other. In order to do so, several encrypted messages containing values private to A and B are exchanged between the two agents using public-key cryptography. We have used TeMP and TSPASS to prove several properties of the Needham-Schroeder algorithm using the translation into monodic FOTL of the $KL_{(n)}$ specification.

Additionally, we have considered other classes of problems that do not result from translations of $KL_{(n)}$ formulae. A formal model based on parameterised finite state machines which is suitable for specifying and verifying a range of communication protocols is given in [25]. Following the approach described in [39], the formal model can be used to express the behaviour of various cache coherence protocols in monodic FOTL. Such protocols are useful for ensuring cache data consistency in shared-memory multi-processor systems. We have applied TeMP and TSPASS on the specification of the MSI and Synapse N+1 cache coherence protocols (see, for example, [44]) and we have verified one correctness property.

Finally, in the last class of monodic FOTL problems that we consider temporal logic is primarily used as a formal language for specifying abstractions of foraging robot behaviour, which are then extended to the presence of multiple robots. The expressivity of first-order temporal logic (in particular the monodic fragment of FOTL) allows it to consider an arbitrary number of robots acting simultaneously in these problems, which then results in an analysis of swarm behaviour. TeMP and TSPASS have been employed to prove numerous properties of an example transition system that specifies foraging behaviour involving an arbitrary (potentially infinite) number of robots. More details about the specification can be found in [13].

All the experiments that we describe in the following were run on a PC with an Intel Core 2 Duo E6400 CPU and 2 GiB of main memory. We have conducted our experiments with version 0.92-0.16 of TSPASS.

6.5.2 Performance on Propositional Temporal Problems

In addition to the regular resolution and factoring inference rules, SPASS possesses additional inference and reduction rules that should increase the performance of the theorem prover. Before such an additional rule can be used in the context of temporal reasoning, one has to analyse its compatibility with the fair architecture in order to ensure that its refutational completeness is maintained. However, as the proof of refutational completeness is based on simulating $\mathfrak{I}^{S,\succ}_{FG,Sub}$-derivations, proving the refutational completeness of a such extended architecture is not a trivial task. It has turned out, though, that one can show fairly easily that one such rule, the *matching replacement resolution* rule, restricted to PLTL problems does not destroy the refutational completeness of the fair inference architecture.

The *matching replacement resolution* reduction rule implemented in SPASS is defined as follows:

$$\frac{C_1 \vee A \quad C_2 \vee \neg B}{C_1 \vee A, C_2}$$

if there exists a substitution σ such that

(i) $A\sigma = B$, and

(ii) $C_1\sigma \subseteq C_2$.

The effect of this rule is to remove the literal $\neg B$ from the clause $C_2 \vee \neg B$ if there exists a clause $C_1 \vee A$ and a substitution σ which fulfils the conditions (i) and (ii) described above. Regarding soundness of the inference rule, one can observe that a resolution inference on the literals A and $\neg B$ of the two premises using the substitution σ leads to the clause $C_1\sigma \vee C_2$. Thus, one obtains the clause C_2 after having eliminated duplicate literals, which obviously subsumes the clause $C_2 \vee \neg B$.

For propositional clauses $C_1 \vee A$ and $C_2 \vee \neg B$, the applicability conditions for the matching replacement rule reduce to the conditions $A = B$ and $C_1 \subseteq C_2$. Consequently, an application of matching replacement resolution in the propositional case corresponds to an unordered resolution inference, which is followed by an elimination of duplicate literals and the removal of the subsumed clause $C_2 \vee \neg B$. It is therefore easy to see that the refutational completeness of the fair inference procedure **F** is maintained if matching replacement resolution is added as an additional reduction rule. On the first-order level, then, one has to ensure that matching replacement resolution is not applied on literals that occur in terminating loop search clauses (as this would alter the results of loop search computations) and on literals contained in the left-hand sides of step clauses. In TSPASS the implementation of the matching replacement resolution rule has been modified accordingly. Moreover, conditions (i) and (ii) ensure that the clause $C_2 \vee \neg B$ cannot be a translated universal clause if the clause $C_1 \vee A$ is a translated initial or step clause. Thus, the refutational completeness of TSPASS is preserved if the matching replacement resolution rule is activated on propositional temporal problems.

We now analyse the performance of TSPASS on PLTL problems. For every value of l between 1 and $8n$, 100 random formulae from the classes C^1_{ran} and C^2_{ran} for $n = 5$ and $n = 12$ were generated and first tested for satisfiability (using TSPASS). The results obtained are depicted in the topmost graphs of Figures 6.4, 6.5, 6.6, and 6.7. We can observe that the probability of a formula from either the class C^1_{ran} or C^2_{ran} being unsatisfiable increases with the value of the quotient $\frac{l}{n}$, eventually culminating in all formulae being unsatisfiable for C^1_{ran} $(n = 5)$ or $(n = 12)$ and C^2_{ran} $(n = 12)$.

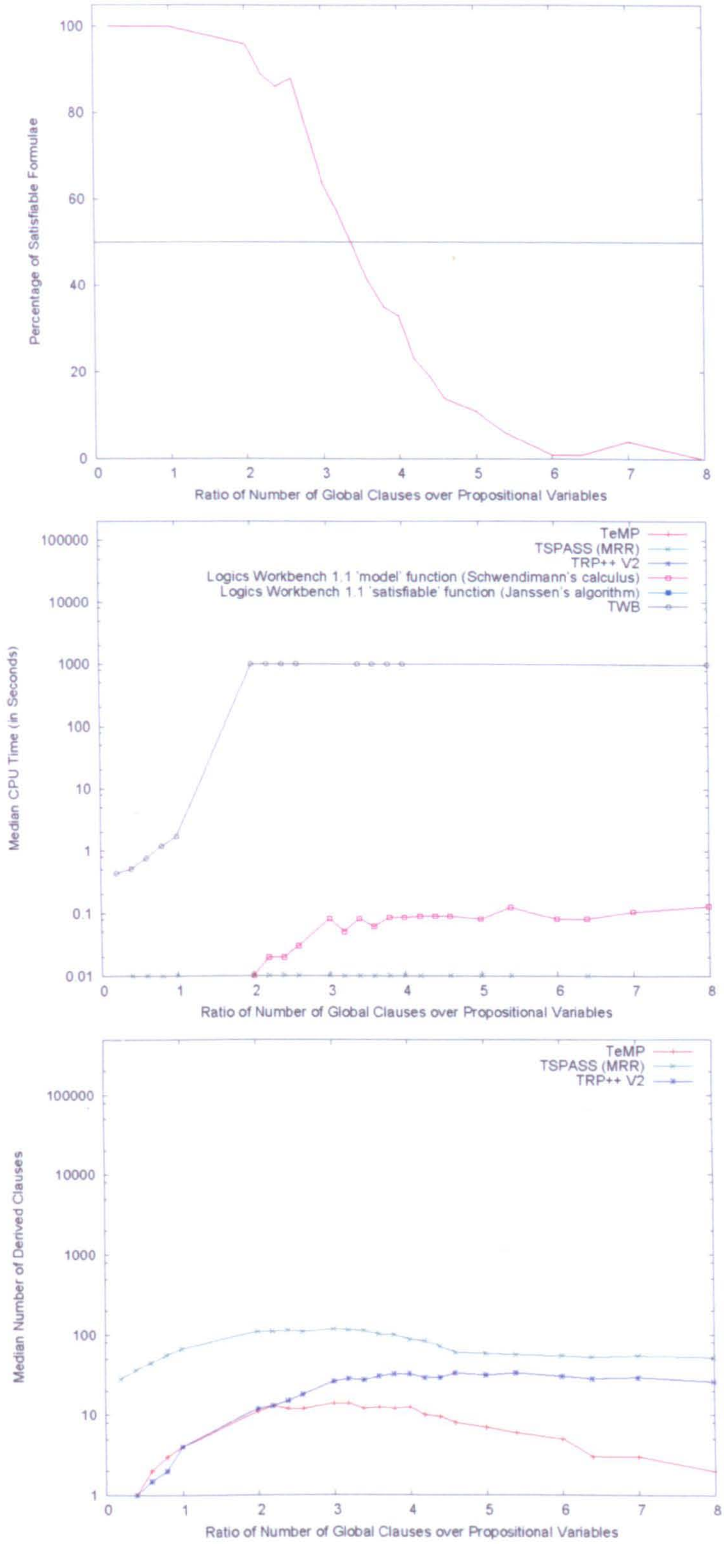


Figure 6.4: Experimental Results Obtained for Class $\mathcal{C}^1_{ran}$ $(n = 5)$

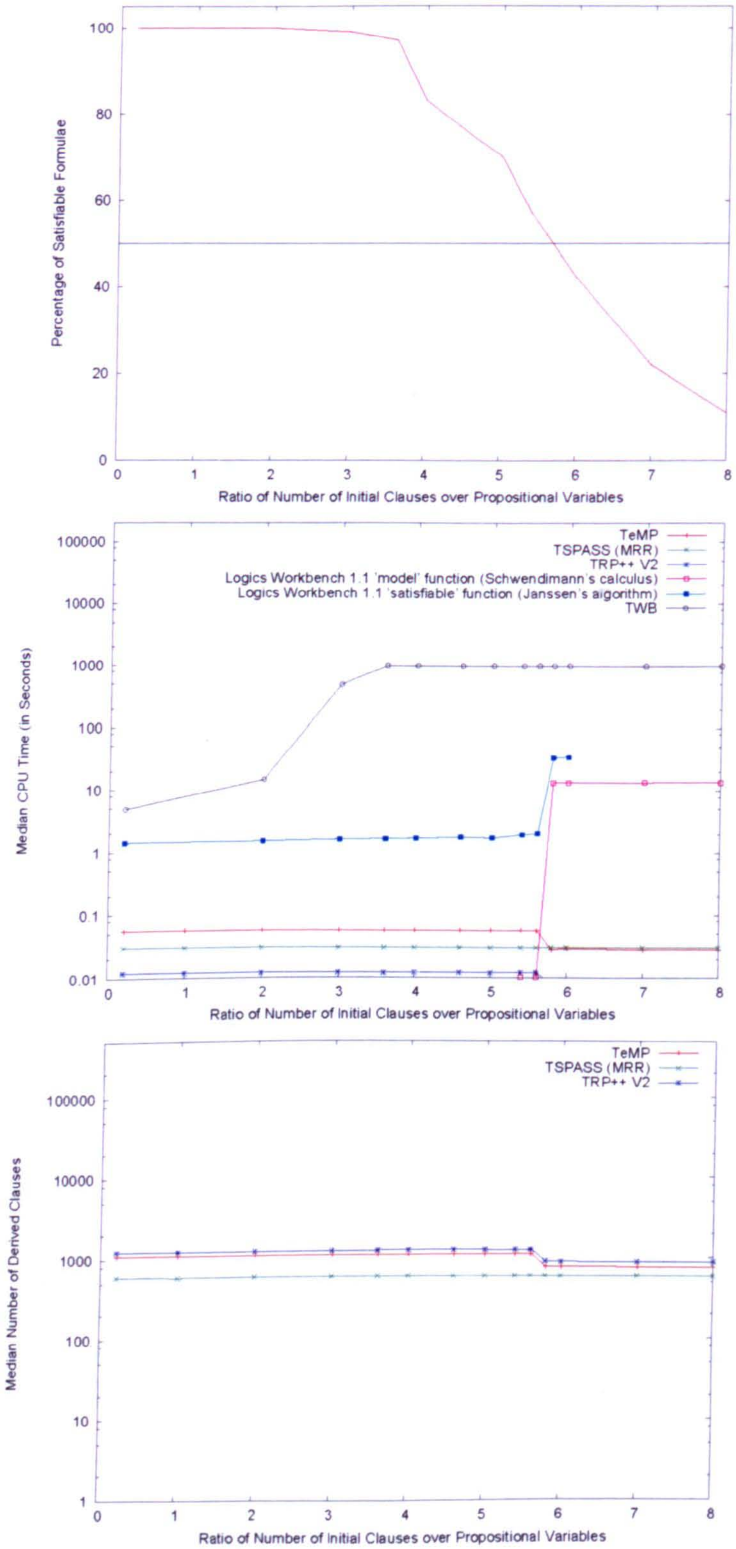


Figure 6.5: Experimental Results Obtained for Class $\mathcal{C}^2_{ran}$ ($n = 5$)

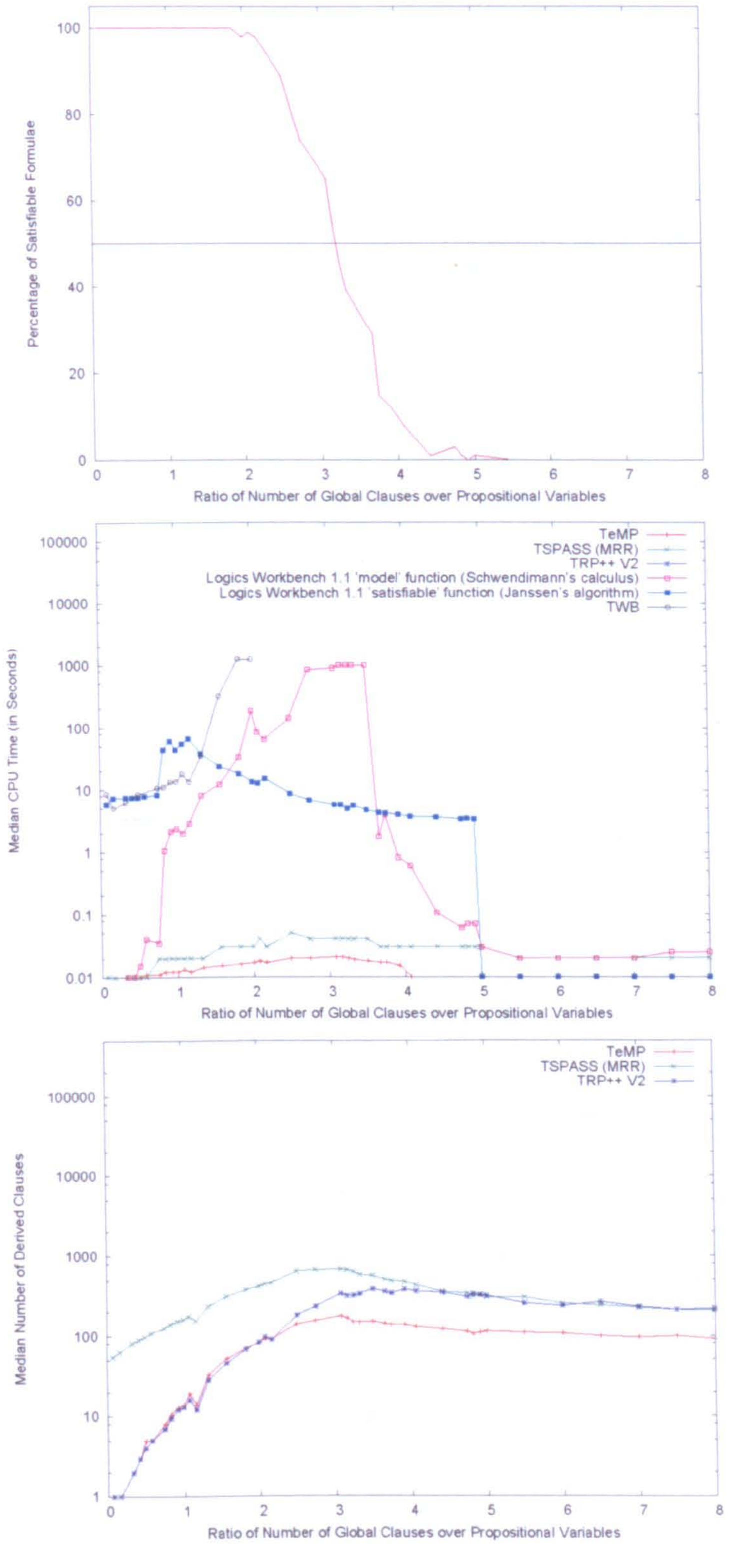


Figure 6.6: Experimental Results Obtained for Class $\mathcal{C}^1_{ran}$ $(n = 12)$

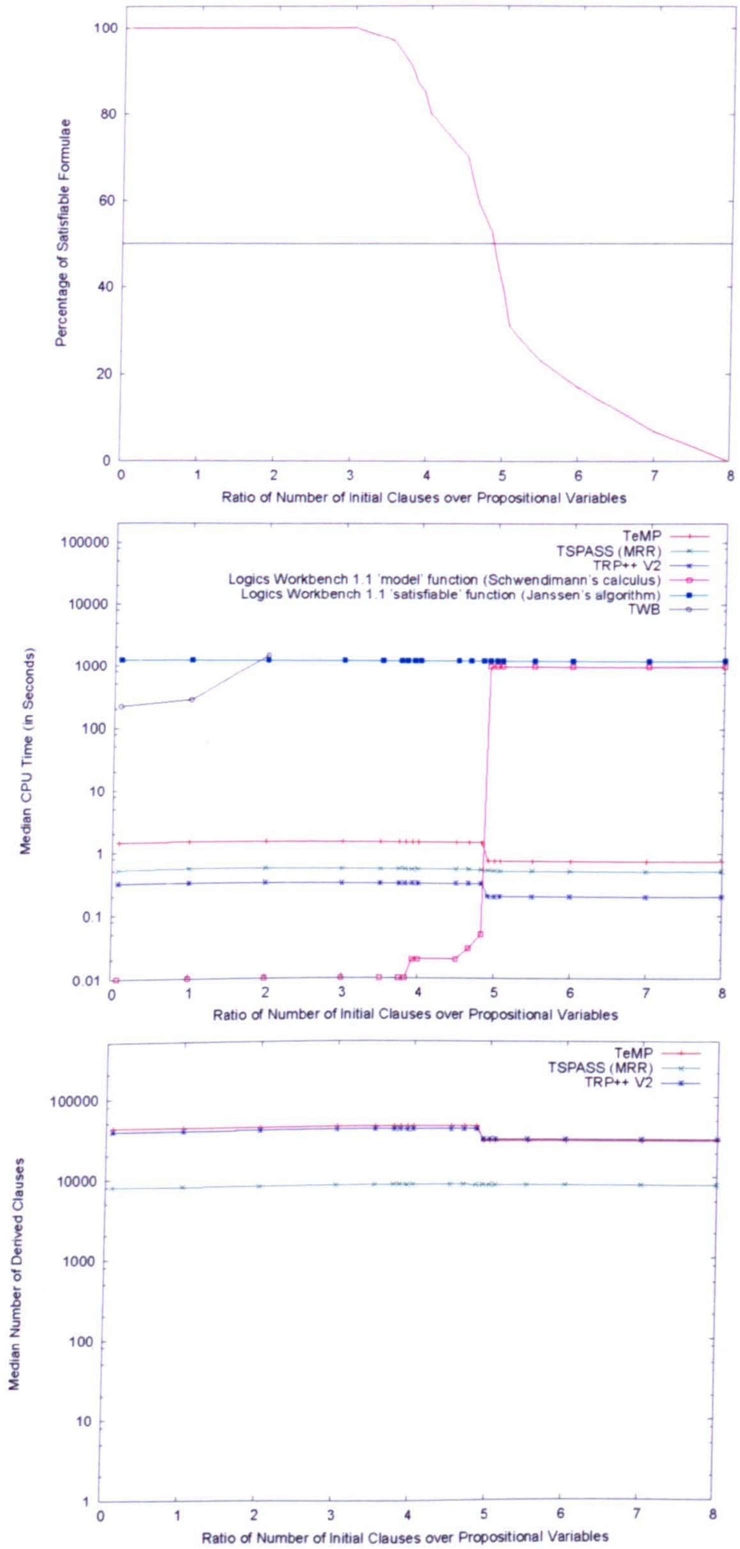
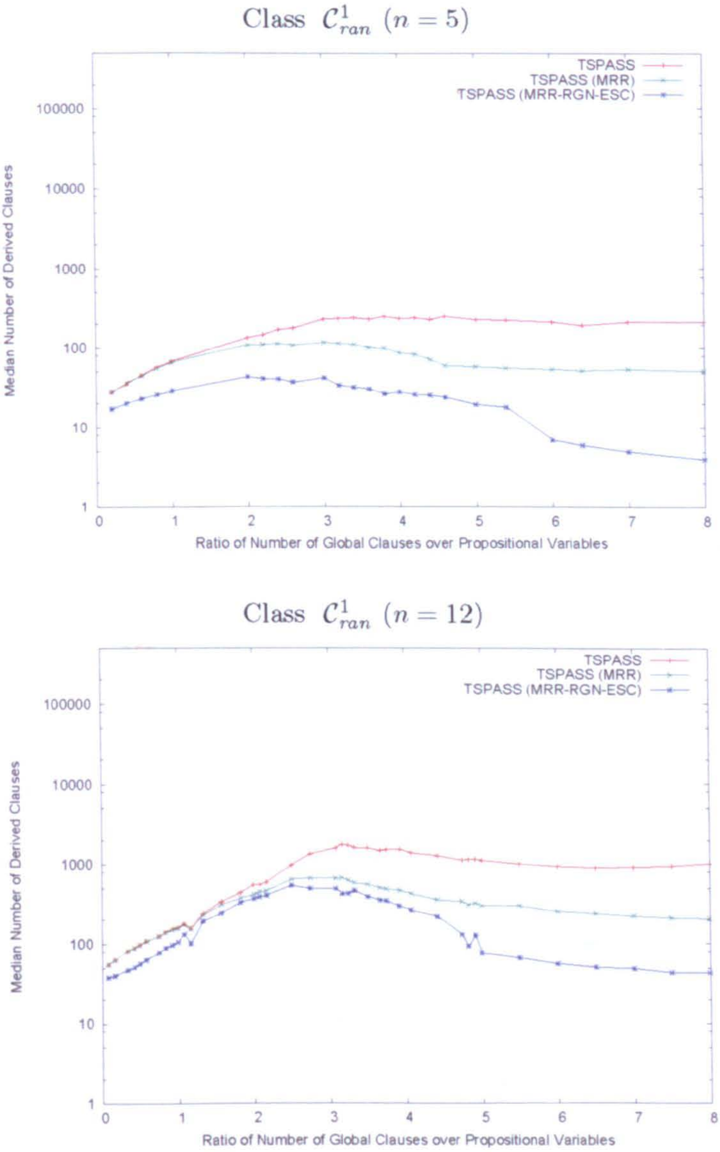
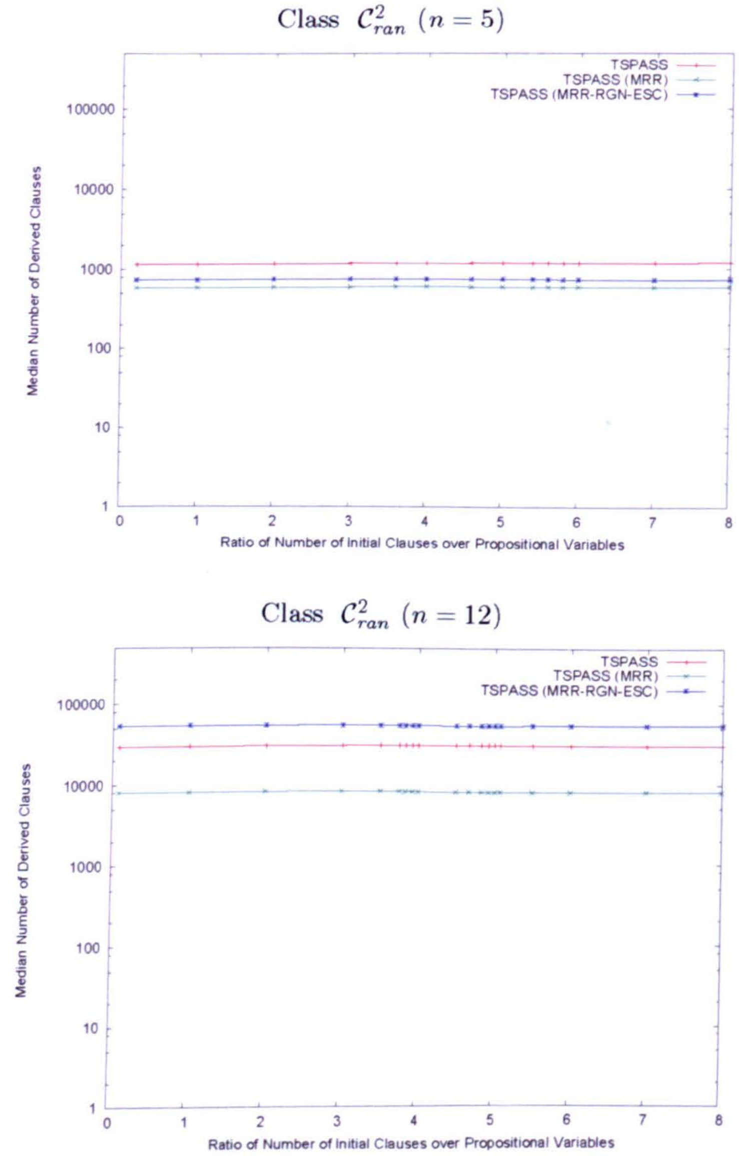


Figure 6.7: Experimental Results Obtained for Class $\mathcal{C}^2_{ran}$ $(n = 12)$


Class $\mathcal{C}^1_{ran}$ $(n = 5)$

Class $\mathcal{C}^1_{ran}$ $(n = 12)$

Figure 6.8: Experimental Results Obtained for Different Inference and DSNF Transformation Options of TSPASS on the Class $\mathcal{C}^1_{ran}$ with $n = 5$ and $n = 12$


Class $\mathcal{C}^2_{ran}$ $(n = 5)$

Class $\mathcal{C}^2_{ran}$ $(n = 12)$

Figure 6.9: Experimental Results Obtained for Different Inference Options of TSPASS on the Class $\mathcal{C}^2_{ran}$ with $n = 5$ and $n = 12$

For comparing the proof search performance, we have run TeMP, TSPASS with matching replacement resolution (MRR), TRP++, the Logics Workbench 1.1 using the 'model' and the 'satisfiable' function, and the Tableau Workbench on the generated formulae for the classes C^1_{ran} and C^2_{ran} with $n = 5$ and $n = 12$. TSPASS was instructed to perform subsumption-based loop search tests. The required median CPU execution time and the median number of derived clauses (for the resolution-based systems) on each problem set are shown in the middle and lowermost graphs, respectively, in the Figures 6.4, 6.5, 6.6, and 6.7. A time limit of 1000 CPU seconds was imposed on each problem.

For the class C^1_{ran} with $n = 5$, we can observe that the the Tableau Workbench shows the worst median CPU execution time. For the majority of the systems the execution time was negligible, and even below 0.01s. Regarding the number of derived clauses, one can see that TSPASS derives slightly more clauses than TeMP or TRP++, especially on the problems which are more likely to be satisfiable.

Then, for the class C^2_{ran} with $n = 5$, we first of all note that the 'satisfiable' function of the LWB cannot solve all the problems within the given time limit and that TRP++ requires the least execution time. Furthermore, TSPASS requires less execution time than TeMP on the problems that are more likely to be satisfiable. For the number of derived clauses, one can observe that TSPASS derives less clauses than TeMP or TRP++.

The results for the class C^1_{ran} with $n = 12$ show that the Tableau Workbench cannot solve all the problems within the given time limit. Furthermore, TSPASS requires more execution time than TeMP or TRP++, which needs less than 0.008s on all the problems, but the execution time of TSPASS is still negligible. Again, TSPASS derives slightly more clauses than TeMP or TRP++, in particular on the problems that are more likely to be satisfiable.

Finally, for the class C^2_{ran} with $n = 12$, the Tableau Workbench cannot solve all the problems within the given time limit. Moreover, the execution times of TSPASS are comparable with those of TRP++ on the problems that are more likely to be satisfiable. TRP++ requires slightly less execution time than TSPASS on the problems that are more likely to be unsatisfiable. Overall, we can observe that TeMP requires more time than TSPASS. Concerning the number of derived clauses, one can see that TSPASS derives less clauses than TeMP and TRP++.

In conclusion one can say that the performance of TSPASS is comparable with the performance of TeMP on PLTL problems. TRP++ is an optimised prover for PLTL, and consequently, it is natural that it performs slightly better than TeMP or TSPASS on PLTL problems. We still have to note that TSPASS does not exhibit the drop in execution time and in the number of derived clauses when the problems tend to be unsatisfiable. We attribute this observation to the design of the fair inference architecture which does not give any preference to either the loop search process or the main saturation at different instants of the proof search process. It might therefore take longer to derive the empty clause from the set of universal and initial clauses as the main saturation inferences are regularly "interrupted" by loop search computations. At the same time the fair architecture derives significantly less clauses on the problems that are harder to solve for resolution-based methods, i.e. on the class C^2_{ran}, which is due to the fact that the fair architecture does not have to compute different types of saturations in the proof search process.

In order to evaluate the effectiveness of the matching replacement resolution rule and also of different rewrite possibilities for the DSNF transformation, we have executed TSPASS with different inference and rewrite options on the classes C^1_{ran} and C^2_{ran} for $n = 5$ and $n = 12$. The median number of derived clauses for the class C^1_{ran} are depicted in Figure 6.8, whereas Figure 6.9 shows the median number of derived clauses for the class C^2_{ran}. The rewrite options for the DSNF transformation are

- a regrouping of formulae $(\bigcirc\varphi) \oplus (\bigcirc\psi)$ into $\bigcirc(\varphi \oplus \psi)$ for $\oplus \in \{\wedge, \vee\}$ called *regrouping of next-state operators* (RGN), and
- an extension of the definition of step clauses in DSNF format in such a way that *extended step clauses* (ESC) are allowed, i.e. step clauses which contain more than one literal in their left-hand or right-hand sides. This extension of the definition ideally leads to fewer renaming operations in the DSNF transformation process.

First of all, we have to note that the DSNF transformation options result in performance improvements in the proof search process for formulae of the class C^1_{ran}. On the class C^2_{ran}, however, these options cause a performance loss for $n = 12$.

We can finally observe that the use of matching replacement resolution reduces the number of derived clauses in both problem classes. An even greater reduction is obtained trough the use of the DSNF transformation options on the class C^1_{ran}.

6.5.3 Effectiveness of Redundancy Elimination

In order to show the effectiveness of tautology elimination, forward subsumption, and backward subsumption for the subsumption-compatible ordered fine-grained resolution with selection calculus, we have applied TSPASS on the Cluedo specification examples, which are composed of six valid assertions that can be made about an example Cluedo game. Assertion 4 is the only specification that contains eventuality formulae. A timeout (TO) of 1 CPU hour was imposed on each problem.

The results that we obtained are shown in Table 6.1. Here, '+B', '+F', and '+T' indicate that backward subsumption, forward subsumption, and tautology elimination, respectively, have been enabled while '-B', '-F', and '-T' indicate that they have been disabled. Given that all six assertions are valid, proofs can theoretically be found by a complete reasoner without the need for redundancy elimination. As the experiments indicate this is clearly not the case within a reasonable amount of time. On the other hand with all options for redundancy elimination enabled even the most difficult problem can be solved in little more than one second. As one might expect, forward subsumption is the most effective of the three options, followed by tautology elimination, while backward subsumption can on occasion slow down the process of finding a proof rather than speeding it up. Overall, the experiments confirm that redundancy elimination is crucial for effective resolution-based theorem proving in monodic first-order temporal logic.

6.5.4 Performance on Monodic Temporal Problems

For analysing the performance of TSPASS on monodic FOTL problems, we compared it against TeMP. Unless mentioned otherwise, a timeout of 12 minutes was imposed on each problem. For TeMP the input problems were first transformed into its clausal input form and then TeMP was started on this clausal input without any additional settings. TSPASS was instructed to perform subsumption-based loop search testing. Every prover run that terminated without triggering a timeout was repeated three times, and the median value of the process CPU time for the different runs is reproduced in the tables below. We also print the number of clauses that were derived by the provers for the different problems. An entry 'TO' for a time value or '-' as clause count indicates that the prover did not terminate within the imposed time limit.

The results for the Cluedo verification examples are shown in Table 6.2. Problem 4 is the only specification that contains eventuality formulae. We observe that TSPASS derives fewer clauses than TeMP on every problem. TSPASS especially seems to perform well on problem 4, which demonstrates the effectiveness of the fair architecture. The fact that TSPASS derives significantly less clauses than TeMP can be explained by the design of its architecture: the fair architecture does not require the computation of saturations under fine-grained step resolution in the loop search process and in the subsequent main saturation steps. Surprisingly, TeMP does not terminate within the time limit on problem 3 although no eventuality is contained in the problem specification.

The results obtained for the Needham-Schroeder protocol verification examples are

	-F/-B/-T		-F/+B/-T		-F/-B/+T		-F/+B/+T	
	Clauses	Time	Clauses	Time	Clauses	Time	Clauses	Time
1	—	TO	—	TO	239202	328.938s	17664	0.193s
2	—	TO	—	TO	—	TO	694574	10.959s
3	—	TO	—	TO	—	TO	—	TO
4	—	TO	—	TO	—	TO	529244	12.566s
5	—	TO	—	TO	—	TO	—	TO
6	—	TO	—	TO	—	TO	—	TO

	+F/-B/-T		+F/+B/-T		+F/-B/+T		+F/+B/+T	
	Clauses	Time	Clauses	Time	Clauses	Time	Clauses	Time
1	481	0.035s	480	0.039s	445	0.033s	444	0.038s
2	2354	0.130s	2262	0.129s	1926	0.115s	1892	0.124s
3	11065	1.375s	9912	1.310s	10102	1.350s	9170	1.278s
4	1460	0.087s	1559	0.097s	1125	0.074s	1343	0.093s
5	594	0.051s	594	0.052s	488	0.044s	488	0.049s
6	765	0.059s	765	0.055s	645	0.050s	645	0.054s

Table 6.1: Effectiveness of Redundancy Elimination Performed by TSPASS on the Cluedo Examples

Problem	Clauses Generated		Time		Result
	TeMP	TSPASS	TeMP	TSPASS	
1	1105	444	0.101s	0.056s	Unsatisfiable
2	4282	1892	0.423s	0.139s	Unsatisfiable
3	-	9170	TO	1.318s	Unsatisfiable
4	192640	1343	16.733s	0.116s	Unsatisfiable
5	3740	488	0.318s	0.069s	Unsatisfiable
6	4380	645	0.397s	0.079s	Unsatisfiable

Table 6.2: Results Obtained for the Cluedo Examples

Problem	Clauses Generated		Time		Result
	TeMP	TSPASS	TeMP	TSPASS	
0	571822	10744	48.655s	0.884s	Satisfiable
1	651599	4849	55.343s	0.520s	Unsatisfiable
2	891616	10665	77.646s	1.094s	Unsatisfiable
3	636257	2946	54.213s	0.324s	Unsatisfiable
4	616989	1274	52.625s	0.218s	Unsatisfiable

Table 6.3: Results Obtained for the Needham-Schroeder Protocol Verification Examples

shown in Table 6.3. Problem 0 represents the protocol specification itself and the remaining problems verify different properties of the protocol. All the problems contain eventuality formulae. On average TSPASS derives significantly less clauses and it requires less execution time on all the problems. For problem 0, TSPASS seems to be able to find loop formulae earlier, which then leads to a higher number of subsumed clauses. The performance improvements in TSPASS for the remaining problems again seem to be due to the fact that TSPASS does not have to compute saturations under fine-grained step resolution in the loop search process and in the main saturation steps for initial and universal clauses.

Then, the experimental results for the verification of the cache coherence protocols MSI and Synapse N+1 can be found in the Tables 6.4 and 6.5, respectively. Again, the problems 0 contain the protocol specifications, which do not contain eventualities. The

Problem	Clauses Generated		Time		Result
	TeMP	TSPASS	TeMP	TSPASS	
0	-	1340237	-	327.578s	Satisfiable
1	-	85651	-	4.384s	Unsatisfiable

Table 6.4: Results Obtained for the MSI Protocol Verification Examples

Problem	Clauses Generated		Time		Result
	TeMP	TSPASS	TeMP	TSPASS	
0	11159	13613	0.263s	0.35s	Satisfiable
1	22556	2101	0.438s	0.07s	Unsatisfiable

Table 6.5: Results Obtained for the Synapse N+1 Protocol Verification Examples

problems 1 each contain one single eventuality and represent the 'non co-occurrence of states' property. For the MSI protocol, TeMP does not terminate within 12 hours on both problems. On the other hand, TeMP solves both problems successfully in the case of the Synapse N+1 protocol. Also, TeMP derives fewer clauses than TSPASS and requires less execution time for establishing the satisfiability of the protocol specification. The performance improvements of TSPASS over TeMP again seem to be caused by the fact that TSPASS does not have to compute saturations of clause sets separately for loop search and the regular fine-grained resolution inferences. Note that an automated reduction from constant to expanding domains was performed for the verification examples that involve cache coherence protocols using the approach described in [58], which adds an exponential number of new clauses to the considered temporal problems.

Finally, Table 6.6 regroups the results obtained for the specification of some examples on simple foraging robots and associated properties. Each of these problems contains at least seven eventualities. TeMP and TSPASS both terminate on the satisfiable problems, but TeMP cannot solve 15 of the unsatisfiable problems within the given time limit. Additionally, on average TeMP derives more clauses than TSPASS. Again, we attribute this observation to the fact that inferences in TSPASS which have been computed once for a loop search instance do not have to be computed again for further loop search saturations. Additionally, the design principles of TeMP to compute saturations under fine-grained step resolution for the loop search and main saturation steps contribute to deriving more clauses than TSPASS.

Also, the problems on which TeMP does not terminate within the given time limit seem to exhibit its fairness problems as it remains 'stuck' whilst saturating a set of temporal clauses under ordered fine-grained resolution with selection.

6.6 Summary

The first goal of this chapter was to describe the implementation of the fair inference procedure **F**, which resulted in the automated theorem prover TSPASS. The second aim was to provide an experimental analysis of the effectiveness of redundancy elimination and of TSPASS's proof search performance on PLTL and monodic FOTL problems.

We first explained the connection between ordered fine-grained step resolution with selection and regular first-order resolution. Then, we continued by describing the architecture of TeMP, before we outlined its problems related to guaranteeing fair derivations. Moreover, we analysed how the fair inference procedure **F** can be implemented in practice. In the following, the implementation of TSPASS was described in greater detail.

We recalled the general architecture of SPASS before we explained some basic considerations behind the implementation of TSPASS. Next, we presented TSPASS's general architecture and we continued with a detailed description of its internals, like the clausification and translation into first-order logic, efficient techniques to access loop search clauses, the main inference procedure, and ways to implement loop search tests. We then discussed several peculiarities related to the implementation of TSPASS w.r.t. SPASS and we analysed some fairness problems regarding the regular clause selection function.

Finally, we examined the effectiveness of redundancy elimination in TSPASS and we evaluated its proof search performance on PLTL and monodic FOTL problems against TRP++, the Logics Workbench, the Tableau Workbench, and TeMP.

Problem	Clauses Generated		Time (s)		Result
	TeMP	TSPASS	TeMP	TSPASS	
0	19611	5707	0.498	0.407	Satisfiable
1	21812	833	0.538	0.094	Unsatisfiable
2	-	4834	TO	0.383	Unsatisfiable
3	-	15707	TO	1.223	Unsatisfiable
4	-	10722	TO	0.781	Unsatisfiable
5	-	10687	TO	0.807	Unsatisfiable
6	20019	6124	0.505	0.424	Satisfiable
7	23670	7230	0.585	0.508	Satisfiable
8	-	10083	TO	0.717	Unsatisfiable
9	-	9786	TO	0.694	Unsatisfiable
10	-	9064	TO	0.600	Unsatisfiable
11	3676	2075	0.111	0.184	Unsatisfiable
12	689	793	0.045	0.091	Unsatisfiable
13	4872	1072	0.138	0.116	Unsatisfiable
14	4799	2009	0.138	0.179	Unsatisfiable
15	4862	2062	0.142	0.190	Unsatisfiable
16	9030	6037	0.233	0.432	Unsatisfiable
17	746	549	0.044	0.076	Unsatisfiable
18	32395	5262	0.984	0.399	Unsatisfiable
19	590	839	0.041	0.096	Unsatisfiable
20	19716	6560	0.493	0.436	Satisfiable
21	22226	9522	0.554	0.590	Satisfiable
22	22796	8403	0.567	0.530	Satisfiable
23	23365	9363	0.579	0.558	Satisfiable
24	24043	10312	0.593	0.611	Satisfiable
25	23268	8791	0.577	0.533	Satisfiable
26	19716	7060	0.492	0.468	Satisfiable
27	24344	9467	0.604	0.619	Satisfiable
28	19716	7630	0.490	0.517	Satisfiable
29	-	8128	TO	0.608	Unsatisfiable
30	-	31484	TO	2.366	Unsatisfiable
31	-	24168	TO	1.752	Unsatisfiable
32	-	26708	TO	2.012	Unsatisfiable
33	21349	6188	0.531	0.440	Satisfiable
34	24603	7675	0.611	0.547	Satisfiable
35	-	27054	TO	1.935	Unsatisfiable
36	-	25437	TO	1.743	Unsatisfiable
37	-	23366	TO	1.702	Unsatisfiable
38	-	31313	TO	2.354	Unsatisfiable

Table 6.6: Results Obtained for the Robot Specification Examples

Chapter 7

Resolution-Based Model Construction for PLTL

7.1 Introduction

Besides clausal resolution-based methods like the $\mathfrak{I}^{S,\succ}_{FG}$-calculus, there are a variety of other proof methods for PLTL including, for instance, tableaux-based approaches [93]. An implementation of a one-pass tableau calculus [83] exists, for example, in the Logics Workbench [46]. In order to prove the validity of a formula φ both proof methods operate on the negated formula $\neg\varphi$. In the case of tableaux reasoning one essentially tries to construct a model for the formula $\neg\varphi$. If no model can be found, then one can conclude that the formula $\neg\varphi$ is unsatisfiable, which is equivalent to φ being valid. For resolution-based proof methods on the other hand the proof goal consists in deriving a contradiction from the formula $\neg\varphi$, from which one can conclude again that φ is valid.

It is therefore easy to see that formal verification by using tableaux-based systems bears the advantage that in case of a failure to prove the validity of a specific property a counter example demonstrating the erroneous behaviour has already been constructed. For clausal resolution-based reasoning a set of clauses on which every application of an inference rule will only derive redundant clauses, a so-called saturated set (up to redundancy), will have typically been constructed in that case. If the empty clause is not contained in this saturated set, one can conclude that the formula $\neg\varphi$ is satisfiable, which implies that φ is not valid. Thus, only the knowledge that the specification does not satisfy the required property is generally available for clausal resolution-based verification.

A way of constructing a model satisfying a saturated set (under ordered resolution) both for propositional and first-order logic has been devised in [10]. The model construction algorithm involves ordering the clauses by using an extension of the ordering on propositional symbols that has been used in the saturation of the clause set. One positive (maximal) literal is then satisfied per clause, whenever necessary, starting from the smallest clause w.r.t. the considered ordering. A term model, or so-called Herbrand model, representing the satisfied literals will be constructed in this way.

In this chapter we are trying to fill the gap in functionality that resolution-based techniques admit over tableaux-based reasoning for formal verification purposes based on PLTL. We present a method that allows us to construct a model for a satisfiable PLTL formula. Our approach is based on analysing the saturated clause set that has been computed under (subsumption-compatible) ordered fine-grained resolution with selection. A temporal model is then obtained by constructing models for sets of (non-temporal) propositional clauses at the different time points. The sets of clauses considered for the individual points in the time line will be constructed dynamically during the model construction process by taking those clauses into account that allow to express constraints among different time points. The whole model construction procedure is designed in such a way that it can be easily incorporated into existing resolution-based theorem provers for PLTL.

This chapter is organised as follows. We first recall the propositional model construction procedure in Section 7.2. Then, in Section 7.3 we introduce the resolution-based temporal model construction algorithm for PLTL and prove its correctness. We then consider practical aspects of the algorithm and its complexity in Section 7.4. We provide a brief overview of its implementation in the theorem prover TSPASS in Section 7.5 and we present some experimental results in Section 7.6. Subsequently, in Section 7.7 we describe alternative systems and approaches towards PLTL model construction, and we conclude with a short discussion of finding minimal models in Section 7.8.

7.2 Propositional Model Construction

In this section we briefly recall the model construction procedure for satisfiable sets of (non-temporal) propositional clauses as it was introduced in [10]. This model construction procedure uses an admissible ordering $\succ$ on propositional symbols (see Section 3.3.1), which is extended on literals by $\neg A \succ A$ and $(\neg)A \succ (\neg B)$ if and only if $A \succ B$. This ordering on literals is then extended on propositional clauses as its multiset extension $\succ_{\mathsf{mul}}$. For two propositional clauses $\mathcal{C}$ and $\mathcal{D}$, we define $\mathcal{C} \succ_{\mathsf{mul}} \mathcal{D}$ if and only if $\mathcal{C} \neq \mathcal{D}$ and for every literal L such that $\mathcal{C}(L) < \mathcal{D}(L)$, it holds that there exists a literal L' with $L' \succ L$ and $\mathcal{C}(L') > \mathcal{D}(L')$, where $\mathcal{C}(L)$, $\mathcal{D}(L)$, $\mathcal{C}(L')$, $\mathcal{D}(L')$ represent the number of occurrences of the literals L and L' in the clauses $\mathcal{C}$ and $\mathcal{D}$, respectively. The multiset extension $\succ_{\mathsf{mul}}$ of the ordering $\succ$ on literals is also simply denoted by $\succ$ when its intended meaning is clear from the context.

The propositional model is then constructed by considering which literals have to be satisfied in a given clause, starting from the smallest clause w.r.t. the clause ordering.

Definition 7.2.1 (Propositional Model Construction). *Let $\succ$ be an admissible ordering and S be a selection function. Additionally, let $\mathcal{N}$ be a set of propositional clauses.*

For a propositional clause $\mathcal{C} \in \mathcal{N}$ we inductively define a propositional model $I_{\succ,S}(\mathcal{C})$ and a set $\varepsilon_{\mathcal{C}}$ as follows.

Let $\mathcal{C} \in \mathcal{N}$ be a propositional clause. Then, we define $I_{\succ,S}(\mathcal{C}) = \bigcup_{\mathcal{C} \succ \mathcal{D}} \varepsilon_{\mathcal{D}}$, and if the clause $\mathcal{C}$

(i) is of the form $C' \vee A$, where A is the maximal literal in C,

(ii) is false in $I_{\succ,S}(C)$, and

(iii) if no negative literal is selected in C,

we define $\varepsilon_C = \{A\}$; otherwise we set $\varepsilon_C = \emptyset$. Finally, we define $I_{\succ,S}(\mathcal{N}) = \bigcup_{C \in \mathcal{N}} \varepsilon_C$.

In line with the definition of the semantics for PLTL given in Section 2.2 propositional symbols not contained in a propositional model $I_{\succ,S}(N)$ are false in the model[1].

It can be shown that for an arbitrary admissible ordering, an arbitrary selection function and for an arbitrary set of propositional clauses saturated under ordered propositional resolution with selection (w.r.t. to the given ordering) which does not contain the empty clause, the propositional model construction indeed constructs a model.

Theorem 7.2.2 (see [10], Theorem 3.16). *Let $\succ$ be an admissible ordering and S be a selection function. Moreover, let $\mathcal{N}$ be a set of propositional clauses that is saturated under inferences by the rules of ordered (propositional) resolution with selection and let $\mathcal{N}$ not contain the empty clause. Then it holds that $I_{\succ,S}(\mathcal{N}) \models \mathcal{N}$.*

7.3 Temporal Model Construction

First of all, we have to note that in this chapter we assume that propositional temporal problems in DSNF contain at most one single eventuality. This is not a limiting assumption as every propositional problem can be transformed in such a way that it contains at most one eventuality up to a linear increase in the size of the problem (see [22], Lemma 7).

Now, before we define the model construction formally, we present two examples that should illustrate the basic ideas behind the model construction procedure. Let us first consider the construction of a temporal model $\mathcal{M}_1$ for the following satisfiable temporal problem P_1:

$$\mathsf{P}_1 = \langle \{d \vee e\}, \{a\}, \{a \Rightarrow \bigcirc b, b \Rightarrow \bigcirc c, c \Rightarrow \bigcirc a\}, \emptyset \rangle.$$

We observe that P_1 does not contain an eventuality and that it is already saturated under (subsumption-compatible) ordered fine-grained resolution w.r.t. any admissible ordering (and selection function). Additionally, for $\mathcal{M}_1$ to be a model of P_1, $\mathcal{M}_1$ has to fulfil the initial (unit) clause a and the universal clause $d \vee e$ at the initial point in time. Thus, if we apply the standard propositional model construction on the propositional clause set $\{a, d \vee e\}$ with an ordering $\succ$ given by $a \succ b \succ c \succ d \succ e$, we obtain the propositional model $H_0 = \{a, d\}$. Then, for constructing the propositional model in the time point 1 we have to consider the universal clause $d \vee e$ again together with the right-hand sides of those (merged) step clauses whose left-hand sides were triggered at the initial time point. In this case only the step clause $a \Rightarrow \bigcirc b$ was triggered by the model H_0. Consequently, we construct a propositional

[1] More specifically, the terminology of "don't care" literals from SAT solvers does not apply here.

model for the clause set $\{d \vee e, b\}$ by using the ordering $\succ$ and obtain $H_1 = \{b, d\}$. Similarly, we can build the propositional model $H_2 = \{c, d\}$ for the time point 2. Now, we have to consider the clause set $\{d \vee e, a\}$ again for the time point 3, which results in the propositional model $H_3 = \{a, d\} = H_0$ through the standard propositional model construction with the ordering $\succ$. Hence, we can see that $\mathcal{M}_1 = (H_0, H_1, H_2, H_3, H_0, H_1, H_2, H_3, H_0, \dots)$ is a temporal model for P_1.

In the previous example one single ordering on propositional symbols was sufficient for constructing a temporal model. But as we will see in the following example, it can be necessary to change the ordering used for the propositional model construction. Let us consider the construction of a temporal model $\mathcal{M}_2$ for the following satisfiable temporal problem P_2:

$$\begin{aligned} \mathsf{P}_2 = \langle\{&\neg f \vee a, \\ &a \vee p, \\ &\neg f \vee b, \\ &\neg d \vee \neg l \vee e, \\ &f \vee g\}, \\ \{&a\}, \\ \{&a \Rightarrow \bigcirc\neg l, \\ &b \Rightarrow \bigcirc d, \\ &c \Rightarrow \bigcirc\neg e\}, \\ \{&\Diamond l\}\rangle. \end{aligned}$$

Here, the saturation of P_2 under (subsumption-compatible) ordered fine-grained resolution w.r.t. the ordering $\succ_0$ given by $a \succ_0 b \succ_0 c \succ_0 d \succ_0 e \succ_0 f \succ_0 g \succ_0 l \succ_0 p$ (and an empty selection function) will derive the merged step clauses $b \Rightarrow \bigcirc(\neg l \vee e)$ and $(b \wedge c) \Rightarrow \bigcirc\neg l$. There is no loop formula derivable from the problem P_2. We can see that the two (merged) step clauses $a \Rightarrow \bigcirc\neg l$ and $(b \wedge c) \Rightarrow \bigcirc\neg l$ imply the negation of the eventuality literal l at the next time point whenever their left-hand sides are fulfilled at the currently considered point of the time line. Now, if one wants to construct a model for a temporal problem that contains exactly one eventuality, then one has to ensure that the eventuality is satisfied infinitely often. The approach that we take here consists in fulfilling the eventuality at a given time point whenever the clauses that have to be considered for this point in the time line do *not* imply the negated eventuality. In this way we can add the eventuality unit clause l to the clause set and saturate the enlarged clause set under propositional ordered resolution with selection without deriving the empty clause.

Now, for the temporal problem P_2 we have to consider the clause set

$$\{f \vee g, \neg d \vee \neg l \vee e, \neg f \vee b, a, a \vee p, \neg f \vee a\}$$

for the initial time point. As this clause set does not imply the negated eventuality $\neg l$, we add the unit clause l and obtain the propositional model $H_0 = \{l, f, b, a\}$ by using the ordering $\succ_0$. Then, as the model H_0 triggers the merged step clauses $a \Rightarrow \bigcirc \neg l$, $b \Rightarrow \bigcirc d$ and $b \Rightarrow \bigcirc(e \vee \neg l)$, we have to additionally consider their right-hand sides for the propositional model construction in the time point 1, i.e. the clauses $\neg l$, d and $e \vee \neg l$. Consequently, as the clause set

$$\{\neg l, f \vee g, e \vee \neg l, d, \neg d \vee \neg l \vee e, \neg f \vee b, a \vee p, \neg f \vee a\}$$

implies the negated eventuality $\neg l$, we do not add the unit clause l to the clause set. The propositional model construction with the ordering $\succ_0$ yields the model $H_1 = \{f, d, b, a\}$.

We can see that the model H_1 again triggers the left-hand side of the step clause $a \Rightarrow \bigcirc \neg l$. Additionally, due to the universal clause $a \vee p$, the ordering $\succ_0$ will enforce that the symbol a is fulfilled (and thus l cannot be satisfied at the next time point) whenever the propositional model construction is performed with the ordering $\succ_0$ (the symbol a does not occur negatively in the temporal problem). Thus, if we want the temporal model construction to succeed we have to use a different ordering for constructing propositional models in some points of the time line. As the model H_1 also triggers the step clauses $b \Rightarrow \bigcirc d$ and $b \Rightarrow (e \vee \neg l)$, we have to consider the clause set

$$\{d, \neg f \vee a, \neg f \vee b, f \vee g, \neg l, e \vee \neg l, \neg d \vee \neg l \vee e, a \vee p\}$$

for the time point 2. If we now use the ordering $\succ_1$ given by $p \succ_1 l \succ_1 g \succ_1 f \succ_1 e \succ_1 d \succ_1 c \succ_1 b \succ_1 a$ for the propositional model construction, we first of all observe that the set is already saturated under ordered propositional resolution w.r.t. the ordering $\succ_1$. We hence obtain the model $H_2 = \{p, g, d\}$.

Finally, as H_2 does not trigger any of the step clauses, we only have to consider the clause set

$$\{l, f \vee g, \neg d \vee \neg l \vee e, \neg f \vee b, a \vee p, \neg f \vee a\},$$

which contains the eventuality unit clause l, for the propositional model construction. By using the ordering $\succ_0$ again we obtain the model $H_3 = \{l, f, b, a\} = H_0$. We can thus conclude that $\mathcal{M}_2 = (H_0, H_1, H_2, H_3, H_0, H_1, H_2, H_3, H_0, \ldots)$ is a temporal model for P_2.

As illustrated by these examples, the temporal model construction for a temporal problem $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ is based on using the regular propositional model construction for the different time points of a temporal model. For the initial time point 0 the regular propositional model construction will be performed over the set of universal clauses together with the set of initial clauses. For time points different from the initial point in time, the (merged) step clauses $C \Rightarrow \bigcirc D$ whose left-hand sides C were fulfilled at the previous moment in time have to be considered in addition to the set of universal clauses.

If the temporal problem P contains a single eventuality, i.e. $\mathcal{E} = \{\Diamond l\}$, special care has to be taken for allowing it to be satisfied infinitely often. We add the eventuality to the set of clauses used for the model construction in a specific time point if the newly-added eventuality unit clause does not lead to a contradiction. As a result, the constructed model will satisfy the eventuality in every time point in which the set of universal clauses and the right-hand sides of the step clauses whose left-hand sides were fulfilled at the previous time point do not imply the negated eventuality. The situations in which we might not be able to satisfy an eventuality in the next moment of time can be characterised by the presence of 'critical' merged step clauses $\mathcal{A} \Rightarrow \bigcirc\mathcal{B}$ such that $\mathcal{U} \cup \{\mathcal{B}\} \models \neg l$ and $\mathcal{U} \not\models \neg\mathcal{A}$. In particular one has to avoid that the left-hand side of one of these 'critical' merged step clauses is constantly fulfilled from any given time point onwards. One way of ensuring this requirement consists in varying the ordering on propositional symbols that is used to construct the models for the different time points, which is also the approach that we take here.

For example, if we were to construct a temporal model as described above for the temporal problem $\mathsf{P}_3 = \langle\{p \vee q\}, \emptyset, \{p \Rightarrow \bigcirc\neg l\}, \{\Diamond l\}\rangle$, we have to ensure that the propositional symbol p is not satisfied at every time point as otherwise we would obtain the sequence of propositional models $\{p, l\}, \{p\}, \{p\}, \ldots$ which obviously does not satisfy the formula $\Box\Diamond l$.

In the next subsection we describe the model construction procedure in a formal way and give an example for the construction of a model, while we prove the correctness of the procedure in the subsequent subsection.

7.3.1 Construction Principle

Before we can introduce the model construction procedure, we still need to give a couple of auxiliary definitions.

First of all, for a temporal problem P we associate with every set of merged step clauses $\mathcal{C}$ (and with the power set $\mathscr{P}(\mathcal{C})$) a set $\mathcal{O}_\mathcal{C}$ of strict total orderings on the set $\mathsf{Symbols}(\mathsf{P})$ of propositional symbols occurring in P.

Definition 7.3.1. *Let P be a propositional temporal problem in DSNF and let*

$$\mathcal{N} = \{\mathcal{A}_1 \Rightarrow \bigcirc\mathcal{B}_1, \ldots, \mathcal{A}_n \Rightarrow \bigcirc\mathcal{B}_n\}$$

be a set of merged step clauses built from the temporal problem P, where $\mathcal{A}_i = \bigwedge_{j=1}^{m_i} a_j^i$ for $1 \leq i \leq n$ and $a_1^i, \ldots, a_{m_i}^i$ are propositional symbols for $1 \leq i \leq n$.

We define $\mathcal{O}_\mathcal{N}$ to be the smallest set of admissible orderings on $\mathsf{Symbols}(P)$ *which contains for every tuple $(i_1, \ldots, i_n) \in \{1, \ldots, m_1\} \times \ldots \times \{1, \ldots, m_n\}$ exactly one ordering $\succ \in \mathcal{O}_\mathcal{N}$ with* $\mathsf{Symbols}(P) \setminus \{a_{i_1}^1, \ldots, a_{i_n}^n\} \succ a_{i_1}^1, \ldots, a_{i_n}^n$.

For the power set $\mathscr{P}(\mathcal{N})$ of $\mathcal{N}$ we define that $\mathcal{O}_{\mathscr{P}(\mathcal{N})} = \bigcup_{S \in \mathscr{P}(\mathcal{N})} \mathcal{O}_S$, where $\mathcal{O}_\emptyset = \emptyset$.

The next definition introduces the set $R^S(\mathfrak{M})$ which contains the right-hand sides of step clauses contained in a set $\mathcal{S}$ whose left-hand sides are *triggered* by a propositional model $\mathfrak{M}$.

Definition 7.3.2. *Let $P = \langle\mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$ be a propositional temporal problem such that $\mathcal{E} = \emptyset$ or $\mathcal{E} = \{\Diamond l\}$. Additionally, let $\mathcal{S}'$ be a set of step clauses derived by $\mathfrak{I}_{FG}^{S,\succ}$ (or $\mathfrak{I}_{FG,Sub}^{S,\succ}$) from P*

and let $\mathfrak{M}$ be a propositional model over Symbols(P). *Then we define:*

$$R^{S'}(\mathfrak{M}) = \{\, l_1 \vee \ldots \vee l_m \mid (p_1 \wedge \ldots \wedge p_m) \Rightarrow \bigcirc(l_1 \vee \ldots \vee l_m) \in S' \text{ and } \mathfrak{M} \models p_1 \wedge \ldots \wedge p_m \,\}$$

Next, we define the set $L^{\mathcal{E}}(\mathcal{N})$, which adds to a set $\mathcal{N}$ the unit clause l if $\mathcal{E} = \{\Diamond l\}$ and $\mathcal{N} \not\models \neg l$.

Definition 7.3.3. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a propositional temporal problem such that $\mathcal{E} = \emptyset$ or $\mathcal{E} = \{\Diamond l\}$. Furthermore, let $\mathcal{N}$ be a set of propositional clauses over* Symbols(P). *Then we define:*

$$L^{\mathcal{E}}(\mathcal{N}) = \begin{cases} \mathcal{N} \cup \{l\} & \text{if } \mathcal{E} = \{\Diamond l\} \text{ and } \mathcal{N} \not\models \neg l \\ \mathcal{N} & \text{otherwise} \end{cases}$$

Finally, for a set of propositional clauses $\mathcal{N}$ we denote by $\mathrm{Res}_{\succ,S}(\mathcal{N})$ the set of all the clauses obtained by an application of the ordered resolution with selection or the ordered factoring rule using the ordering $\succ$ and the selection function S to premises in $\mathcal{N}$. We also define that $\mathrm{Res}^0_{\succ,S}(\mathcal{N}) = \mathcal{N}$, $\mathrm{Res}^i_{\succ,S}(\mathcal{N}) = \mathrm{Res}_{\succ,S}(\mathrm{Res}^{i-1}_{\succ,S}(\mathcal{N}))$ for $i > 0$ and $\mathrm{Res}^\infty_{\succ,S}(\mathcal{N}) = \bigcup_{i \in \mathbb{N}} \mathrm{Res}^i_{\succ,S}(\mathcal{N})$.

We can now give the definition of the temporal model construction procedure.

Definition 7.3.4 (Temporal Model Construction). *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a propositional temporal problem in DSNF such that $\bot \notin \mathcal{U} \cup \mathcal{I}$, and $\mathcal{E} = \emptyset$ or $\mathcal{E} = \{\Diamond l\}$. Additionally, let S be a selection function, and if $\mathcal{E} = \{\Diamond l\}$, let $\mathcal{N} = \{\mathcal{A}_1 \Rightarrow \bigcirc\mathcal{B}_1, \ldots, \mathcal{A}_n \Rightarrow \bigcirc\mathcal{B}_n\}$ be the set of all the merged step clauses built from the temporal problem P and freed of duplicate propositional symbols such that for every i, $1 \leq i \leq n$:*

(i) $\mathcal{U} \cup \{\mathcal{B}_i\} \models \neg l$, and

(ii) $\mathcal{U} \not\models \neg\mathcal{A}_i$.

The merged step clauses from the set $\mathcal{N}$ will also be called critical *merged step clauses for the temporal problem P.*

We then define a sequence of propositional models $H_0, H_1, \ldots$ as follows:

$$H_0 = I_{\succ_0,S}(\mathrm{Res}^\infty_{\succ_0,S}(L^{\mathcal{E}}(\mathcal{U} \cup \mathcal{I})))$$

and for $i \geq 1$:

$$H_i = I_{\succ_i,S}(\mathrm{Res}^\infty_{\succ_i,S}(L^{\mathcal{E}}(\mathcal{U} \cup R^S(H_{i-1}))))$$

where $\succ_i$ ($i \in \mathbb{N}$) are admissible orderings on Symbols(P) *such that for every j, $j \geq 1$, with $H_j \models \bigvee_{k=1}^n \mathcal{A}_k$ and such that H_j occurs infinitely often,*

$$O_{\mathcal{P}(\mathcal{N})} \subseteq \{\, \succ_{t+1} \mid t \geq j \text{ and } H_t = H_j \,\}.$$

Additionally, for every H_j, $j \geq 1$ with $H_j \not\models \bigvee_{k=1}^n \mathcal{A}_k$ we have $\succ_{j+1} = \succ_0$.

Let $\mathcal{H} = (H_0, H_1, \ldots)$ denote the temporal model obtained in this way.

Remark 7.3.5. *For $i \geq 0$, the propositional model H_{i+1} is called a* successor model *for the propositional model H_i built with ordering $\succ_{i+1}$.*

One can observe that the process of ensuring that every ordering is used infinitely often is related to the notion of *fairness*, which is employed in the field of model checking [40].

As explained above, the sets of initial and universal clauses are considered for the model construction in the time point 0. Additionally, the eventuality is added to the clause set used for model construction if its presence does not lead to a contradiction. The propositional model construction is then performed through an initial ordering $\succ_0$ on Symbols(P) after the model construction clause set has been saturated under regular ordered resolution with selection using the ordering $\succ_0$. This saturation process is necessary in order to ensure the correctness of the propositional model construction.

For any time point other than the initial point of the time line, the universal clauses together with the right-hand side of any step clause whose left-hand was satisfied at the previous time point are used for the propositional model construction. Again, the eventuality is added to the considered set if it does not lead to a contradiction. As noted before, the ordering on propositional symbols under which the propositional resolution and model construction is performed has to be varied for the temporal model construction to succeed. The variation of the orderings on propositional symbols ensures that a propositional model is found eventually for a time point which does not trigger the left-hand side of any critical step clause.

For example, for the temporal problem $\mathsf{P}_3 = \langle\{p \vee q\}, \emptyset, \{p \Rightarrow \bigcirc\neg l\}, \{\Diamond l\}\rangle$ again, we cannot use the ordering $l \succ p \succ q$ at every time point as it would not lead to a correct temporal model. We have to use an ordering $\succ'$ with $q \succ' p$ at some time points instead.

We conclude this section by applying the temporal model construction procedure on a concrete example. We consider the temporal problem $\mathsf{P}_4 = \langle\{p \vee q\}, \{p\}, \{p \Rightarrow \bigcirc q, q \Rightarrow \bigcirc p\}, \{\Diamond\neg p\}\rangle$. Saturating the problem P_4 under (subsumption-compatible) ordered fine-grained resolution (with an empty selection function) using the ordering $p \succ q$ derives the universal clause $\neg p \vee \neg q$ (through loop search), the initial clause $\neg q$, and the step clause $q \Rightarrow \bigcirc\neg q$. The step clause $q \Rightarrow \bigcirc p$ is a critical step clause for the set of universal clauses as $\{p \vee q, \neg p \vee \neg q, p\} \models \neg\neg p$.

For the initial time point we hence consider the set of propositional clauses $\{\neg q, p, p \vee q, \neg p \vee \neg q\}$ for the propositional model construction procedure. With the symbol ordering $p \succ q$, we obtain the model $H_0 = \{p\}$.

Then, as the step clause $p \Rightarrow \bigcirc q$ has been triggered at the initial time point, we have to add the unit clause q to the considered clause set. As $\{q, p \vee q, \neg p \vee \neg q\} \not\models \neg\neg p$, we add the unit clause $\neg p$ and obtain the set $\{q, p \vee q, \neg p, \neg p \vee \neg q\}$, which is to be used for the propositional model construction. After saturation with the ordering $p \succ q$, the propositional model construction yields the propositional model $H_1 = \{q\}$ in the time point 1.

Finally, as the step clauses $q \Rightarrow \bigcirc p$, $q \Rightarrow \bigcirc\neg q$ have been triggered in time point 1, the unit clauses p and $\neg q$ have to be added to the clause set used for the propositional model construction. Additionally, as the set $\{\neg q, p, p \vee q, \neg p, \neg p \vee \neg q\}$ is unsatisfiable, the set $\{\neg q, p, p \vee q, \neg p \vee \neg q\}$ has be to be considered for the propositional model construction, which results in the model $H_2 = \{p\}$ with the ordering $p \succ q$.

As $H_0 = H_2$ the temporal model construction procedure will now construct models for the remaining time points analogously to ones shown above.

7.3.2 Proof of Correctness

In this section we prove the correctness of the construction procedure introduced in Definition 7.3.4, i.e. we show that the constructed sequence of propositional models is indeed a model for the considered temporal problem.

First of all, we introduce three lemmata that will be required for the subsequent correctness theorem.

Lemma 7.3.6. *Let $\mathcal{N}$ be a set of propositional clauses such that every clause contains at least one negative literal. Let $\succ$ be an arbitrary admissible ordering on* Symbols($\mathcal{N}$) *and let S be an arbitrary selection function.*

Then it holds that $I_{\succ,S}(\mathcal{N}) = \emptyset$.

Proof. We show by induction on $\mathcal{N}$ with respect to the well-founded (and total) multiset extension of the ordering $\succ$ on clauses that for every clause $C \in \mathcal{N}$ it holds that $\varepsilon_C = \emptyset$.

For the minimal clauses C we have $I_{\succ,S}(C) = \emptyset$. As C contains a negative literal, C is true in $I_{\succ,S}(C)$, and therefore, $\varepsilon_C = \emptyset$. The proof for remaining clauses proceeds along the same line. □

Lemma 7.3.7. *Let $\mathcal{N}$ be a satisfiable set of propositional clauses. Moreover, let $a_1, \ldots, a_n$ be propositional symbols and let $\succ$ be an admissible ordering on propositional symbols such that* Symbols($\mathcal{N}$) $\setminus \{a_1, \ldots, a_n\} \succ a_1, \ldots, a_n$. *Finally, let S be a selection function. Then it holds that:*

$$I_{\succ,S}(\mathrm{Res}^{\infty}_{\succ,S}(\mathcal{N})) \models a_1 \vee \ldots \vee a_n \quad \textit{iff} \quad \mathcal{N} \models a_1 \vee \ldots \vee a_n$$

Proof. The implication "$\Leftarrow$" follows from Theorem 7.2.2. For the remaining implication "$\Rightarrow$", we assume that $I_{\succ,S}(\mathrm{Res}^{\infty}_{\succ,S}(\mathcal{N})) \models a_1 \vee \ldots \vee a_n$. As clauses which contain a literal p or $\neg p$ with $p \succ a_1 \succ \ldots \succ a_n$ cannot produce an atom a_i $(1 \leq i \leq n)$ in the interpretation $I_{\succ,S}(\mathrm{Res}^{\infty}_{\succ,S}(\mathcal{N}))$, it follows that there exist clauses in the set $\mathrm{Res}^{\infty}_{\succ,S}(\mathcal{N})$ that only contain atom symbols from $\{a_1, \ldots, a_n\}$. Let $C_1, \ldots, C_m \in \mathrm{Res}^{\infty}_{\succ,S}(\mathcal{N})$ be all such clauses, i.e. for every i, $1 \leq i \leq m$, there exists an index j, $1 \leq j \leq n$, such that $C_i = C' \vee a_j$, Symbols(C') $\subseteq \{\, a_k \mid 1 \leq k \leq n \,\} \cup \{\, \neg a_k \mid 1 \leq k \leq n \,\}$ and $I_{\succ,S}(\mathrm{Res}^{\infty}_{\succ,S}(\mathcal{N})) \models a_j$. (Note that the index j could be the same for every i, $1 \leq i \leq m$.)

Then, if we assume that every clause C_i for $1 \leq i \leq m$ contains at least one negative literal, it would follow from Lemma 7.3.6 that $I_{\succ,S}(\{C_1, \ldots, C_m\}) = \emptyset$ and thus, $I_{\succ,S}(\mathrm{Res}^{\infty}_{\succ,S}(\mathcal{N})) \not\models a_1 \vee \ldots \vee a_n$, which contradicts with our assumptions. Thus, there exists a clause $C_i \in$

$\mathrm{Res}^{\infty}_{\succ,S}(\mathcal{N})$ $(1 \leq i \leq m)$ such that C_i is positive and $\mathrm{Symbols}(C_i) \subseteq \{a_1, \ldots, a_n\}$. We can infer that $\mathcal{N} \vDash a_1 \vee \cdots \vee a_n$. □

Lemma 7.3.8. *Let P be a propositional temporal problem and let $\mathcal{N}$ be a satisfiable set of propositional clauses which only uses propositional symbols from P. Additionally, let $\mathcal{N} = \{\mathcal{A}_1 \Rightarrow \bigcirc\mathcal{B}_1, \ldots, \mathcal{A}_n \Rightarrow \bigcirc\mathcal{B}_n\}$ be a set of merged step clause built from the temporal problem P, and let S be a selection function. Then it holds that:*

$$\mathcal{N} \vDash \bigvee_{i=1}^{n} \mathcal{A}_i \textit{ iff } \forall \succ \in \mathcal{O}_{\mathcal{N}}: I_{\succ,S}(\mathrm{Res}^{\infty}_{\succ,S}(\mathcal{N})) \models \bigvee_{i=1}^{n} \mathcal{A}_i$$

Proof. The implication "$\Rightarrow$" is obvious. For the implication "$\Leftarrow$", let $\mathcal{A}_i = \bigwedge_{j=1}^{m_i} a^i_j$ for $1 \leq i \leq n$ and propositional symbols $a^i_1, \ldots, a^i_{m_i}$ for $1 \leq i \leq n$. Then we have:

$$\bigvee_{i=1}^{n} \mathcal{A}_i \equiv \bigwedge_{(i_1,\ldots,i_n)\in\{1,\ldots,m_1\}\times\ldots\times\{1,\ldots,m_n\}} (a^1_{i_1} \vee \ldots \vee a^n_{i_n})$$

Furthermore, it follows from the assumptions that:

$$\forall \succ \in \mathcal{O}_{\mathcal{N}}\ \forall (i_1, \ldots, i_n) \in \{1, \ldots, m_1\} \times \ldots \times \{1, \ldots, m_n\}: I_{\succ,S}(\mathrm{Res}^{\infty}_{\succ,S}(\mathcal{N})) \models a^1_{i_1} \vee \ldots \vee a^n_{i_n}$$

Thus, as for every tuple $(i_1, \ldots, i_n) \in \{1, \ldots, m_1\} \times \ldots \times \{1, \ldots, m_n\}$ there exists an ordering $\succ \in \mathcal{O}_{\mathcal{N}}$ with $\mathrm{Symbols}(\mathcal{N}) \setminus \{a^1_{i_1}, \ldots, a^n_{i_n}\} \succ a^1_{i_1}, \ldots, a^n_{i_n}$, we obtain from Lemma 7.3.7:

$$\forall (i_1, \ldots, i_n) \in \{1, \ldots, m_1\} \times \ldots \times \{1, \ldots, m_n\}: \mathcal{N} \vDash a^1_{i_1} \vee \ldots \vee a^n_{i_n}$$

We can therefore conclude that $\mathcal{N} \vDash \bigvee_{i=1}^{n} \mathcal{A}_i$. □

We can now state and prove the correctness theorem for the model construction procedure.

Theorem 7.3.9. *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a propositional temporal problem with $\mathcal{E} = \emptyset$ or $\mathcal{E} = \{\Diamond l\}$ which is saturated under (subsumption-compatible) ordered fine-grained resolution with selection and does not contain the empty clause. Additionally, let $\mathcal{H} = (H_0, H_1, \ldots)$ be the corresponding sequence of propositional models obtained through temporal model construction. Then it holds that:*

$$\mathcal{H}_0 \models \mathcal{I} \wedge \Box\mathcal{U} \wedge \Box\mathcal{S} \wedge \Box\mathcal{E}$$

Proof. Let S be the selection function used in the saturation. Then, first of all, as the set $\mathcal{U} \cup \mathcal{I}$ does not contain the empty clause, it is easy to see that $\bot \notin \mathrm{Res}^{\infty}_{\succ_0,S}(L^{\mathcal{E}}(\mathcal{U} \cup \mathcal{I}))$. We can thus conclude that[2] $\mathcal{H}_0 \models \mathcal{I}$ and $\mathcal{H}_0 \models \mathcal{U}$.

We now show by induction on t that $\mathcal{H}_t \models \mathcal{S}$ and $\mathcal{H}_{t+1} \models \mathcal{U}$ for every $t \in \mathbb{N}$. For $t = 0$, we already have $\mathcal{H}_0 \models \mathcal{U}$, and if we assume that $\bot \in \mathrm{Res}^{\infty}_{\succ_1,S}(L^{\mathcal{E}}(\mathcal{U} \cup R^S(H_0)))$, then it would follow that $\bot \in \mathrm{Res}^{\infty}_{\succ_1,S}(\mathcal{U} \cup R^S(H_0))$. Thus, as $\bot \notin \mathrm{Res}^{\infty}_{\succ_1,S}(\mathcal{U})$ there would exist a derivation of a step clause $\mathcal{A} \Rightarrow \bigcirc\bot$ with $\mathcal{H}_0 \models \mathcal{A}$. Then, as the temporal problem P is

[2] The notation $\mathcal{H}_0$ was introduced in the definition of the truth-relation for PLTL given in Figure 2.1 on page 14

saturated, we would have $\mathcal{U} \models \neg\mathcal{A}$ and hence, $\mathcal{H}_0 \not\models \mathcal{A}$, which is a contradiction. We can infer that $\bot \notin \mathrm{Res}^{\infty}_{\succ_1, \mathcal{S}}(L^{\mathcal{E}}(\mathcal{U} \cup R^{\mathcal{S}}(H_0)))$, $\mathcal{H}_0 \models \mathcal{S}$ and $\mathcal{H}_1 \models \mathcal{U}$.

If $t > 0$, then it follows from the induction hypothesis that $\mathcal{H}_{t-1} \models \mathcal{S}$ and $\mathcal{H}_t \models \mathcal{U}$. Again, if we assume that $\bot \in \mathrm{Res}^{\infty}_{\succ_{t+1}, \mathcal{S}}(L^{\mathcal{E}}(\mathcal{U} \cup R^{\mathcal{S}}(H_t)))$, then there would exist a derivation of a clause $\mathcal{A} \Rightarrow \bigcirc\bot$ with $\mathcal{H}_t \models \mathcal{A}$. Additionally, as the temporal problem P is saturated, we would again have $\mathcal{U} \models \neg\mathcal{A}$ and $\mathcal{H}_t \not\models \mathcal{A}$, which is a contradiction. Thus, we obtain $\bot \notin \mathrm{Res}^{\infty}_{\succ_{t+1}, \mathcal{S}}(L^{\mathcal{E}}(\mathcal{U} \cup R^{\mathcal{S}}(H_t)))$, $\mathcal{H}_t \models \mathcal{S}$ and $\mathcal{H}_{t+1} \models \mathcal{U}$.

Finally, if $\mathcal{E} = \{\Diamond l\}$, let $t \in \mathbb{N}$. We still have to show that $\mathcal{H}_t \models \Diamond l$. If we assume for all $t' \in \mathbb{N}$ with $t' \geq t$ that $\mathcal{H}_{t'} \not\models l$, then for every $t' \geq t$ with $t' \geq 1$ it holds that $\mathcal{U} \cup R^{\mathcal{S}}(H_{t'-1}) \models \neg l$. It also holds that $\mathcal{U} \not\models \neg l$ as otherwise we could apply the eventuality resolution rule and have $\bot \in \mathcal{U}$. Additionally, for every $t' \geq t$ with $t' \geq 1$ there exists a merged clause $\mathcal{A}^{t'} \Rightarrow \bigcirc\mathcal{B}^{t'} \in \mathcal{N}$ with $\mathcal{U} \cup R^{\mathcal{S}}(H_{t'-1}) \equiv \mathcal{U} \cup \{\mathcal{B}^{t'}\}$, $\mathcal{H}_{t'-1} \models \mathcal{A}^{t'}$ and $\mathcal{U} \cup \{\mathcal{B}^{t'}\} \models \neg l$, where $\mathcal{N}$ is the set of merged step clauses from Definition 7.3.4. Then, as there are only finitely many valuations H_i ($i \in \mathbb{N}$), it follows that there exists an index $T \geq \max(t, 1)$ such that every valuation H_i with $i \geq T$ occurs infinitely often in the sequence $H_i, H_{i+1}, H_{i+2}, \ldots$. Furthermore, as there are only finitely many merged step clauses which have been freed of duplicate propositional symbols, there exist merged step clauses $\mathcal{A}'_1 \Rightarrow \bigcirc\mathcal{B}'_1, \ldots, \mathcal{A}'_m \Rightarrow \bigcirc\mathcal{B}'_m \in \mathcal{N}$ such that

$$\{\mathcal{A}'_1 \Rightarrow \bigcirc\mathcal{B}'_1, \ldots, \mathcal{A}'_m \Rightarrow \bigcirc\mathcal{B}'_m\} = \{\mathcal{A}^{t'} \Rightarrow \bigcirc\mathcal{B}^{t'} \mid t' \geq T\}.$$

By Lemma 7.3.8 it holds for every $t' \geq T$ that there exists a subset $\{i_1 \ldots, i_k\} \subseteq \{1, \ldots, m\}$ such that $\mathcal{U} \cup \{\mathcal{B}^{t'}\} \models \bigvee_{j=1}^{k} \mathcal{A}'_{i_j}$, from which we can infer that $\mathcal{U} \cup \{\mathcal{B}^{t'}\} \models \bigvee_{i=1}^{m} \mathcal{A}'_i$. Consequently, we obtain for every i with $1 \leq i \leq m$ that $\mathcal{U} \cup \{\mathcal{B}'_i\} \models \bigvee_{i=1}^{m} \mathcal{A}'_i$ and $\mathcal{U} \cup \{\mathcal{B}'_i\} \models \neg l$. We could hence apply the eventuality resolution rule and derive the set of universal clauses $\bigwedge_{j=1}^{n} \neg\mathcal{A}'_j$. Thus, as the temporal problem P is saturated under (subsumption-compatible) ordered fine-grained resolution with selection, we can infer that $\mathcal{H}_{T-1} \not\models \mathcal{A}^T$ holds, which is a contradiction. □

7.4 Practical Considerations and Complexity

The temporal model construction as described in the previous section constructs an infinite sequence of propositional models, as suggested by the definition of the semantics for PLTL given in Section 2.2. However, for practical applications, a finite representation of a temporal structure, as given by an ultimately periodic model is more useful.

Definition 7.4.1 (Ultimately Periodic Model). *Let $P = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a propositional temporal problem such that either $\mathcal{E} = \emptyset$ or $\mathcal{E} = \{\Diamond l\}$, and let $\mathcal{H} = (H_0, H_1, H_2, \ldots)$ be an infinite sequence of propositional models over* Symbols(P). *Furthermore, let $I, J, L \in \mathbb{N}$ be indices such that $I \leq L < J$, $H_I = H_J$ and $H_L \models l$ if $\mathcal{E} = \{\Diamond l\}$, $I = L$ otherwise.*

We then define a sequence of propositional models $\mathcal{H}' = (H'_0, H'_1, \ldots)$ as follows:

(i) $H'_i = H_i$ for every $0 \leq i \leq J$

(ii) $H'_i = H_{I+((i-I) \bmod (J-I))}$ *for every* $i \geq J+1$

It can be shown that if the sequence $\mathcal{H}$ is a model for P, then the sequence $\mathcal{H}'$ is also a model for P [84].

More concretely, in an implementation of the temporal model construction procedure one has to keep track of the ordering that has been used for the saturations used in the different time points. Whenever a previously considered set of clauses is encountered again, the symbol ordering used for the model construction in the considered time point has to be changed cyclically. Finally, the construction procedure can terminate whenever a previously encountered valuation has been computed again and the possibly present eventuality has been satisfied in between those two time points.

Moreover, it is easy to see that for a set $\mathcal{N} = \{\mathcal{A}_1 \Rightarrow \bigcirc\mathcal{B}_1, \ldots, \mathcal{A}_n \Rightarrow \bigcirc\mathcal{B}_n\}$ of critical merged step clauses for a temporal problem P the set $\mathcal{O}_{\mathcal{P}(\mathcal{N})}$ can be constructed from $\mathcal{P}(\bigcup_{i=1}^{n} \text{Symbols}(\mathcal{A}_i))$, the power set of all the propositional symbols occurring in left-hand sides of critical step clauses. Every ordering $\succ \in \mathcal{O}_{\mathcal{P}(\mathcal{N})}$ is characterised by the subset $\mathcal{P} \subseteq \bigcup_{i=1}^{n} \text{Symbols}(\mathcal{A}_i)$ such that $\text{Symbols}(\mathsf{P}) \setminus \mathcal{P} \succ p$ for every $p \in \mathcal{P}$. Thus, in an implementation it is sufficient to consider all the subsets of $\bigcup_{i=1}^{n} \text{Symbols}(\mathcal{A}_i)$ in order to construct the required orderings.

Furthermore, it is also possible to eliminate redundant cycles in constructed temporal models. For example, if one has built a model for a temporal problem P with a single eventuality $\Diamond l$ and the constructed model contains a sequence of valuations $H_i, H_{i+1}, \ldots, H_j$ such that $H_i = H_j$ and $H_k \not\models l$ for every $i \leq k \leq j$, then the sequence $H_i, \ldots, H_{j-1}$ can be removed from the final model as it does not contribute to satisfying the eventuality.

It is important to note that the model construction procedure is completely deterministic, that is, neither the basic building blocks given by $I_{\succ,S}$ and $\text{Res}^{\infty}_{\succ,S}$, nor the construction of the sequence of propositional models that form the ultimately periodic model involves any non-deterministic operation that in an implementation would force us to use a form of backtracking-search to find a model. On the other hand, just as for standard tableaux-based model generation procedures for PLTL, there is no guarantee that we will produce a minimal, that is, shortest possible, ultimately periodic model for a temporal problem or PLTL formula.

In particular, different choices of orderings can lead to different temporal models, which can greatly vary in size. However, it is not possible to construct every model of a temporal problem P though the model construction method introduced in this chapter. For example, the model $\{p\}, \{p\}, \ldots$ cannot be obtained for the temporal problem $\langle\{\neg p \vee \neg q\}, \emptyset, \emptyset, \emptyset\rangle$.

The computational complexity of the temporal model construction procedure is determined mainly by the time required to compute the saturation $\text{Res}^{\infty}_{\succ,S}(\mathcal{N})$ of a set $\mathcal{N}$ of clauses under ordered resolution with selection, which is exponential in the size of $\mathcal{N}$, the size of the $\text{Res}^{\infty}_{\succ,S}(\mathcal{N})$, which is also exponential in the size of $\mathcal{N}$, and the maximal length of the sequence of propositional models in an ultimately periodic model $\mathcal{H}'$ for a satisfiable temporal problem $\mathsf{P} = \langle\mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E}\rangle$, which is again exponential in the size of P. Overall, we obtain the following result.

	Time		
	LWB 'Model'	TSPASS	TSPASS Model Construction
C^1_{ran} (n=5)	0.06s	0.03s	0.05s (0.01s)
C^2_{ran} (n=5)	0.06s	0.06s	0.71s (0.03s)
C^1_{ran} (n=12)	1.5s	0.04s	0.89s (0.57s)
C^2_{ran} (n=12)	0.06s	1.13s	65.08s (0.60s)

	LWB 'Model' Length	TSPASS Model Properties		
		Model Length	Constructed Time Points	Minimal Critical Merged Step Clauses
C^1_{ran} (n=5)	7	2 (1)	2	0
C^2_{ran} (n=5)	2	2 (1)	2	0
C^1_{ran} (n=12)	39	17 (8)	17	17
C^2_{ran} (n=12)	2	2 (1)	2	0

Table 7.1: Median Results (per Class) for the TSPASS and LWB Model Construction Procedures Applied on the Benchmark Classes C^1_{ran} and C^2_{ran}

Theorem 7.4.2. *Let P = $\langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a satisfiable propositional temporal problem with $\mathcal{E} = \emptyset$ or $\mathcal{E} = \{\Diamond l\}$. Then an ultimately periodic model $\mathcal{H}$ for P can be constructed by the temporal model construction procedure in time exponential in the size of P.*

Since for a given PLTL formula φ an equi-satisfiable propositional problem P in DSNF can be computed in polynomial time and space, this result also implies the we can construct an ultimately periodic model for φ in time exponential in the size of φ.

It is important to remember that while the satisfiability problem of PLTL is PSPACE-complete, given that ultimately periodic models can be of exponential size in the worst case, we cannot hope for a model construction procedure of better complexity.

7.5 Implementation

The temporal model construction has been implemented as an extension of the theorem prover TSPASS. The implementation takes advantage of the fact that while the temporal problem is saturated by TSPASS, the minimal critical merged step clauses for the considered temporal problem are also computed as part of the overall loop search process (the terminating loop search clauses $s_0^L \wedge C \Rightarrow \bigcirc\bot$ for a given eventuality $\Diamond L(x)$). Consequently, no further computation is required to obtain these step clauses.

7.6 Experimental Results

We have compared the resolution-based model construction implemented in TSPASS 0.94-0.16 with the one-pass tableau calculus described in [83], which is implemented in the Logics

Workbench (LWB) version 1.1 [46]. We have applied both systems to numerous satisfiable PLTL formulae from the benchmark classes introduced in [53]. As described already in Section 6.5.1, two of the benchmark classes, C^1_{ran} ($n = 5$) and C^1_{ran} ($n = 12$), where n indicates the number of propositional symbols over which the formulae are constructed, are designed in such a way that they can be theoretically solved easily by resolution-based decision procedures, whereas the two other benchmark classes, C^2_{ran} ($n = 5$) and C^2_{ran} ($n = 12$), are designed so that the satisfiable formulae in them can be theoretically solved more easily by tableaux-based systems. In particular, in [53] the implementation of the one-pass tableau calculus in the LWB was indeed performing best on these formulae.

The experiments were run on a PC equipped with an Intel Core 2 Duo E6400 CPU and 3 GiB of main memory and an execution timeout of 5 minutes was imposed on each problem.

For TSPASS an empty selection function was used. The scheduling of the orderings on propositional symbols was done w.r.t. an increasing subset size and in such a way that a maximal number of different orderings were tried out: whenever a set of propositional clauses was encountered for the first time the next possible ordering was used for the propositional model construction instead of starting again from the initial ordering w.r.t. the critical symbols. Additionally, subsets that only involve propositional symbols occurring in left-hand sides of minimal critical merged step clauses were considered first for constructing the required orderings. Currently, TSPASS builds the remaining orderings which are necessary for the model construction to succeed with all the yet unused propositional symbols that occur in the left-hand sides of step clauses, i.e. no special processing is performed to exactly identify the symbols occurring in left-hand sides of critical merged step clauses that are not minimal. For the C^1_{ran} classes TSPASS was instructed to perform matching replacement resolution and formulae $\bigcirc(\phi) \vee \bigcirc(\psi)$ were rewritten to $\bigcirc(\phi \vee \psi)$ in order to reduce the number of required renamings; for the C^2_{ran} classes no special input rewriting was performed and no additional inference or reduction rules were activated.

The median results for all the satisfiable formulae of each class are shown in Table 7.1, with time values in the table being the average CPU time of three identical runs. We can observe that the number of generated clauses and the execution times increase for the model construction run of TSPASS, which is due to the transformation to single-eventuality problems and, as a result, an increased number of step clauses. Such a transformation is not performed if no model construction is required. Additionally, one can observe that the time spent on the transformation to DSNF is negligible. The numbers in brackets in the model construction time column indicate the amount of time actually spent on model construction w.r.t. the global execution time, and the numbers in brackets in the model length column represent the length of the periodic part. Finally, the median total number of constructed time points during the model construction in TSPASS is reproduced in the second last column, some of which are discarded during the elimination of redundant cycles.

The problem set for class C^1_{ran} (n=5) that we considered contains 2400 formulae in total of which 1217 formulae are satisfiable. On these satisfiable formulae, the model construction of TSPASS could solve all the problems, whereas the 'Model' function of the LWB did not finish on 26 problems within the given time limit. The set of problems for class C^2_{ran} (n=5) contains 1400 formulae in total of which 955 are satisfiable. All the models constructed by TSPASS and the LWB for this class were at most of length 2. No timeouts were incurred either in the TSPASS or the LWB runs. The collection of problems for class C^1_{ran} (n=12), then, contains 4000 formulae in total of which 2264 are satisfiable. The model construction of TSPASS did not finish on 30 satisfiable formulae in this class, whereas the 'Model' function of the LWB did not terminate within the given time limit on 284 of the satisfiable formulae. Finally, the problem set for class C^2_{ran} (n=12) contains 1900 formulae in total of which 1184 formulae are satisfiable. Again, no timeouts were incurred in either the TSPASS or LWB run, and all the models constructed by TSPASS and the LWB for this class were at most of length 2

As one might expect, the Logics Workbench can maintain its execution time advantage on C^2_{ran} ($n = 5$) and C^2_{ran} ($n = 12$). On the other hand, the model construction of TSPASS proves quite successful on C^1_{ran} ($n = 5$) and C^1_{ran} ($n = 12$), computing models of smaller median length than the LWB.

7.7 Alternative Model Construction Approaches

In this section we briefly discuss other relevant approaches and systems for the construction of models for satisfiable PLTL formulae.

First of all, we note that the Stanford Temporal Prover, STeP, [14,69] can also be used for model construction purposes. STeP is foremost a tool which supports the verification of PLTL properties for reactive systems. It combines model checking and deductive verification approaches. Temporal properties are verified through verification rules and verification diagrams. For this purpose, STeP provides features like automatic invariant generation and decision procedures for PLTL and for large classes of first-order formulae. A theorem prover based on non-clausal resolution and on paramodulation is also available in STeP. PLTL model construction is therefore possible in STeP either through model checking or the tableaux-based decision algorithm for PLTL given in [55], which is implemented in STeP. However, according to [53] the tableaux-based decision procedure of STeP for PLTL is outperformed on the benchmarking examples by the model 'function' of the LWB.

Additionally, one can easily see that a particularly close connection is present between the model construction mechanism described in this section and MetateM [11,12], which is a framework for using temporal logics, in particular PLTL, as an executable imperative language. Similarly to the model construction approach presented here, MetateM obtains a PLTL model for a clausified DSNF problem by sequentially executing all the temporal formulae present in the problem for the different time points. In this way the constraints that are expressed by the formulae contained in a temporal problem are satisfied in the different time points of a model. For instance, in MetateM terminology a step clause $p \Rightarrow \bigcirc q$ represents the "executable statement" that the atom q has to be executed at the next time point once the atom p is satisfied at the current time point. Different strategies are available in MetateM to handle the possible choices that can occur when disjunctions, i.e. initial or universal clauses, or multiple eventualities have to be fulfilled during the execution of a MetateM program. In particular, in order to recover from choices that do not lead to a correct model MetateM employs backtracking during the execution of temporal formulae. Note however that MetateM does not require temporal problems to be saturated under the inference rules of the $\mathfrak{I}^{S,\succ}_{FG}$-calculus before they can be executed. A Java-based interpreter for a first-order variant of MetateM is currently being developed [38].

7.8 Minimal Models

A natural question that arises when models for satisfiable formulae need to be constructed concerns the computation of "*minimal*" models. In contrast to the non-temporal case it turns out that it is no longer obvious how one can define a partial ordering on models that would lead to a formal notion which is intuitively perceived to represent minimal models for a PLTL problem. For instance, the characteristics of models that one can take into consideration for defining a partial ordering on PLTL models are the number of different propositional models in the infinite chain of the temporal model or the number of propositional variables assigned to "**true**" at a given time point. Both properties could be compared lexicographically, for example.

If we assume that the characterisation for a model to be considered as minimal refers to (at least) one of the two examples described above, then it is easy to see that the model construction presented here cannot guarantee the construction of minimal models. The number of time points that are present in a constructed model and the number of satisfied propositional variables heavily depend on the choice of the orderings that have been used for the construction, even for temporal problems without eventualities. For example, for the temporal problem $\langle \emptyset, \{p \vee q\}, \{p \Rightarrow \bigcirc r, r \Rightarrow \bigcirc p\}, \emptyset\rangle$ the model construction procedure presented here constructs the model $\{p\}, \{r\}, \{p\}, \{r\}, \ldots$ for an ordering $\succ_1$ with $p \succ_1 q$, whereas one obtains the model $\{q\}, \emptyset, \emptyset, \emptyset, \ldots$ for an ordering $\succ_2$ with $q \succ_2 p$.

Thus, in order to construct minimal models, one would have to explore the different successor models that one can obtain for a given time point by constructing the successor models with different orderings. As such, the search for minimal models would result in having to find a suitable path in a graph structure representing the different possible successors for every time point (see also, e.g., the notion of ***behaviour graph*** in [37]). One has to note, though, that not every propositional model can be obtained through an application of the propositional model construction on a clause set. For example, the model $\{p, q\}$ for the clause $p \vee q$ cannot be obtained through propositional model construction. Consequently, it remains unclear whether a suitable minimal model can indeed be found if the propositional successor models in the graph structure are constructed by propositional model construction.

7.9 Summary

The aim of this chapter was to present a fully-automatic procedure for constructing models for satisfiable PLTL formula, which contributes to closing the gap in functionality that separates resolution-based calculi from tableaux-based reasoning approaches for formal verification purposes. The procedure is based on computing saturations under ordered fine-grained resolution with selection while using the standard model construction for propositional clauses to construct models for the different time points. It is important to observe that the temporal model construction procedure is not based on performing a search with backtracking but the construction is guaranteed to succeed once the appropriate symbol orderings have been considered, and that it can always produce finite, ultimately periodic models. We also proved the correctness of the model construction algorithm and analysed some of its practical aspects. We then briefly introduced our implementation of the algorithm and provided an experimental analysis of the model construction procedure on the benchmarking classes for PLTL formulae. Subsequently, we described alternative approaches and systems for PLTL model construction, and we concluded with a short discussion of the problems related to finding minimal models.

Chapter 8

Conclusion

We conclude this thesis by summarising the results that we have described in the previous chapters and by giving an outlook on future research possibilities.

8.1 Summary of the Results

The general area in which the work presented in the previous chapters is situated is that of formal verification. Chapter 1 gave a brief overview of this field and summarised the main novel contributions of the thesis.

In Chapter 2 we formally introduced the syntax and semantics of the two formal languages we are considering in this thesis: propositional linear-time temporal logic and monodic first-order temporal logic. Both of these logics are interpreted over a model of time that is isomorphic to the natural numbers. We also defined some notions that were essential for the following parts of this thesis. Subsequently, we described the normal form for formulae of the two temporal logics we are considering. Finally, we demonstrated how the formulae that are in normal form can be clausified for the resolution-based calculus that was introduced in the subsequent chapter.

We then focused on the proof of refutational completeness for ordered fine-grained temporal resolution with selection in Chapter 3. First, we briefly recalled the inference rules of monodic temporal resolution and ordered fine-grained temporal resolution with selection. Subsequently, we defined a refined version of monodic temporal resolution, for which we also proved that it is refutationally complete. We then showed the Lifting Theorem for ordered fine-grained resolution with selection without the eventuality resolution rules and the arbitrary factoring in left-hand sides of terminating step clauses rule. Next, we proved that derivations of refined monodic temporal resolution can be simulated by ordered fine-grained resolution with selection. As refined monodic temporal resolution was shown to be refutationally complete for temporal problems that only contain step clauses with unique left-hand sides, we obtained a completeness result for ordered fine-grained resolution with selection restricted to those temporal problems. In a final step we then extended this completeness result to arbitrary temporal problems.

In Chapter 4 we provided a formal analysis of combining redundancy elimination with ordered fine-grained resolution with selection. We first focused on redundancy elimination in combination with the resolution-based inference rules of ordered fine-grained resolution with selection. We presented syntactic criteria for identifying tautologies among temporal clauses and we defined a subsumption relation on temporal clauses. We then described how the calculus had to be extended in order to remain compatible with the removal of subsumed clauses, resulting in the subsumption-compatible ordered fine-grained resolution with selection calculus ($\mathfrak{I}^{S,\succ}_{FG,Sub}$). We also proved subsumption lemmata for the subsumption-compatible calculus.

In the second part of Chapter 4 we analysed the problem of combining redundancy elimination with the loop search process. We introduced a resolution-based loop search algorithm called Subsumption-Restricted-FG-BFS which eliminates subsumed clauses and tautologies during loop search computations. After having proved some of its properties, we proved the refutational completeness of subsumption-compatible ordered fine-grained resolution with selection where applications of the eventuality resolution rules are restricted to loops found by the Subsumption-Restricted-FG-BFS algorithm.

In Chapter 5 we analysed some theoretical aspects of subsumption-compatible ordered fine-grained resolution with selection that can lead to problems when fair $\mathfrak{I}^{S,\succ}_{FG,Sub}$-derivations should be constructed in practice. Due to the fact that the applicability of the eventuality resolution rules is only semi-decidable, it becomes impossible to guarantee the construction of fair derivations, i.e. derivations in which every non-redundant clause that is derivable from a given clause set is eventually derived, as the applicability check for those inference rules might not terminate.

As the ability to construct fair derivations is an essential requirement for maintaining the refutational completeness of an automated theorem, we presented an inference procedure that can construct fair derivations for reasoning in monodic first-order temporal logic based on the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus and we proved its refutational completeness. The design of the new inference mechanism is based on integrating the saturation steps related to loop search, which may not terminate, into the main saturation process. The proof of refutational completeness proceeded by showing that for every non-tautological clause contained in a $\mathfrak{I}^{S,\succ}_{FG,Sub}$-refutation for an unsatisfiable clausified monodic temporal problem there exists a subsuming clause computed in a fair derivation of the new inference procedure.

We also showed that the new inference mechanism can also be used as a decision procedure for temporal problems in which the first-order formulae are restricted appropriately.

Then, in Chapter 6 we described the implementation of the fair inference procedure that had been introduced in Chapter 5, resulting in the automated theorem prover TSPASS. We also analysed the performance of TSPASS in practice. First, we discussed the connection between ordered fine-grained step resolution with selection and regular first-order resolution. We continued by describing the architecture of TeMP, before we outlined its problems related to guaranteeing fair derivations. We also analysed how the fair inference procedure described in Chapter 5 can be implemented in practice. In the following, the implementation of TSPASS was described in detail.

We recalled the general architecture of SPASS before we explained some basic considerations behind the implementation of TSPASS. Next, we presented the general architecture of TSPASS and we continued with a detailed description of its internals, like the the clausification and translation into first-order logic, efficient techniques to access loop search clauses, the main inference procedure, and ways to implement loop search tests. We then discussed several peculiarities related to the implementation of TSPASS w.r.t. SPASS and we analysed some fairness problems regarding the regular clause selection function.

Subsequently, we examined the effectiveness of redundancy elimination in TSPASS and we evaluated its proof search performance on PLTL and monodic FOTL problems against TRP++, the Logics Workbench, the Tableau Workbench, and TeMP.

Finally, in Chapter 7 we presented a fully-automatic procedure for constructing models for satisfiable PLTL formula. This automated construction procedure contributes to closing the gap in functionality that separates resolution-based calculi from tableaux-based reasoning approaches for formal verification purposes. The model construction procedure was based on computing saturations under ordered fine-grained resolution with selection while using the standard model construction for propositional clauses to construct models for the different time points. It is important to observe that the temporal model construction procedure is not based on performing a search with backtracking but the construction is guaranteed to succeed once the appropriate symbol orderings have been considered, and that it can always produce finite, ultimately periodic models. We proved the correctness of the model construction algorithm and analysed some of its practical aspects. We then briefly introduced our implementation of the algorithm and provided an experimental analysis of the model construction procedure on the benchmarking classes for PLTL formulae. We concluded the chapter with a description of alternative systems and approaches towards PLTL model construction, and we shortly discussed some of the problems related to finding minimal models.

8.2 Further Research Possibilities

We now briefly present some ideas for further research possibilities based on the results presented in this thesis. Regarding the fair inference procedure introduced in Chapter 5 and its implementation in the automated theorem prover TSPASS, it would be worthwhile to

- investigate whether several restrictions that had to be introduced for proving the refutational completeness of the different calculi in the previous chapters are necessary or only appear as a consequence of the chosen proof techniques. For example, the proof of refutational completeness for the $\mathfrak{I}^{S,\succ}_{FG}$-calculus given in Chapter 3 requires that selection function are instance compatible. However, such constraints on selection functions are not present in the setting of regular first-order resolution. Additionally, for proving the refutational completeness of the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus in Chapter 4 two additional inference rules, namely (arbitrary) factoring in left-hand sides of terminating step clauses and (arbitrary) factoring in at most monadic negative universal clauses, had to be introduced. Thus, one could try to find examples which demonstrate that these additional restrictions are in fact necessary for the completeness of the considered calculi. If such examples cannot be found, one could explore possibilities for alternative proofs that do not require additional restrictions. Furthermore, one could

- analyse the compatibility of the fair inference procedure with additional (first-order) inference and reduction rules, which are, for example, already available in the theorem prover SPASS. The reduction rule of matching replacement resolution in the general first-order case and the introduction of splitting rules are here of particular interest [91]. As the proof of refutational completeness for the fair inference procedure **F** is based on showing that for every clause derived by the $\mathfrak{I}^{S,\succ}_{FG,Sub}$-calculus there exists a subsuming clause computed by the procedure **F**, one can see that additional efforts would be required to show that an extended inference procedure remains refutationally complete if the newly added inference or reduction rule cannot be simulated by resolution inferences and subsequent subsumption steps (as it is indeed the case for propositional matching replacement resolution, for example). In particular, it would probably be most difficult to prove that new inference or reduction rules are compatible with the loop search algorithm, i.e. one has to guarantee that the conditions for finding loops can still be achieved. Also, one would have to to ensure that new inference or reduction rules do not cause undesired side effects such as interfering with the translation of temporal clauses into first-order logic and allowing inferences or reductions in this way which would not have been possible on the level of monodic FOTL. Along the same lines,

- one could analyse the possibilities of extending the fair architecture with inference rules capable of handling equality. A first attempt to include the handling of equality mainly into the monodic temporal resolution calculus was presented in [59]. For fine-grained temporal resolution a fine-grained superposition calculus is also briefly described in [59], which could form the starting point for an extension of the fair inference procedure with equality handling capabilities. Presumably, one could prove the refutational completeness of the fine-grained superposition calculus in analogy to proof of refutational completeness of the $\mathfrak{I}^{S,\succ}_{FG}$-calculus contained in Chapter 3, and consequently the subsumption lemmata could be proved for the fine-grained superposition calculus. In a final step the refutational completeness of the fair architecture extended with superposition rules could be established by linking the extended fair architecture with the fine-grained superposition calculus as it was described in Chapter 5. The implementation of TSPASS would also have to be modified in order to take the superposition inference rules of the fine-grained superposition calculus into account. Note, however, that even the monadic monodic two-variable fragment of FOTL with equality (but without function symbols) is not recursively enumerable [24], which brings us to the following point:

- another interesting application area of ordered fine-grained resolution with selection and the fair inference procedure consists in the full language of first-order temporal logic with or without equality. Despite not being refutationally complete, the extended inference rules for full FOTL would still be sound. One would have to extend the normal form DSNF in such a way that step clauses and eventuality clauses which involve predicates of arbitrary arity are allowed to occur. The implementation of the transformation to DSNF would also have to be modified. Additionally, one would have to change the implementation of the fair architecture in order to be able to handle eventuality clauses of arbitrary arity. However, TSPASS could hence be turned into a sound but not refutationally complete automated theorem prover for full FOTL.

Then, regarding the automated model construction procedure for satisfiable PLTL formulae described in Chapter 7, it would, for instance, be interesting to

- investigate the influence that the choice of orderings has on the constructed models. Hopefully, it would then be possible to guarantee the construction of small models, and ideally obtain minimal models (in specific cases). Also, one could try to
- reduce the number of renamings necessary for the transformation to single-eventuality problems. As a consequence, the number of propositional symbols that have to be considered during the construction of models will be reduced, which should result in shorter construction times required to build models for satisfiable PLTL formulae. Another interesting extension would be
- the answering of queries over models, i.e. given a satisfiable PLTL formula $\mathcal{F}$ and a PLTL query formula φ, one would like to know whether there exists a model $\mathcal{H}$ of the formula $\mathcal{F}$ such that $\mathcal{H} \models \varphi$ holds. Initially, one could limit the expressivity of the formula φ by not allowing it to contain eventualities, for example. Moreover, if one is not interested in the (full) model $\mathcal{H}$ itself, but only in the question whether such a model *can* exist, it is sufficient to construct time points for the model $\mathcal{H}$ until the formula φ is satisfied, which can potentially reduce the required construction efforts. Lastly, on a more technical side,
- one could also investigate in which cases an even stronger elimination of redundant cycles is possible. After the propositional symbols which have been introduced during the transformation to DSNF have been removed from a constructed model, experiments have shown that is possible to remove redundant cycles present in the reduced model in some cases while preserving correctness, i.e. the reduced model which has now been freed of redundant cycles remains a model for the original formula. A formal study of this problem could potentially identify criteria for when such an extended elimination of redundant cycles can be performed. Additionally,

- in the case where the validity of a formula $\varphi \Rightarrow \psi$ cannot be proved as the formula $\varphi \wedge \neg\psi$ is satisfiable, the constructed model $\mathcal{H}$ for the formula $\varphi \wedge \neg\psi$ shows why the formula $\neg\psi$ holds in the context of the formula φ. However, it would be worthwhile for the user to visualise how the model $\mathcal{H}$ satisfies the formula $\neg\psi$ alone by disregarding any propositional symbols that only occur in the formula φ, for instance. In this way one can avoid some potential clutter in the representation of the model $\mathcal{H}$ that can be caused by the formula φ. The user should thus be able to understand erroneous behaviour more easily.

Finally, we believe that it is feasible to extend the automated model construction method to CTL formulae, but an extension to first-order temporal logic will require greater efforts as the reduction to single-eventuality problems is not even generally possible for arbitrary monodic FOTL problems.

Bibliography

[1] Cluedo. http://www.hasbro.com.

[2] M. Abadi and Z. Manna. Nonclausal temporal deduction. In R. Parikh, editor, *Proceedings of the Conference on Logics of Programs, Brooklyn College, June 17-19, 1985*, volume 193 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1985.

[3] M. Abadi and Z. Manna. Nonclausal deduction in first-order temporal logic. *Journal of the ACM*, 37(2):279–317, 1990.

[4] P. Abate and R. Goré. The tableaux work bench. In M. C. Mayer and F. Pirri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2003, Rome, Italy, September 9-12, 2003. Proceedings*, volume 2796 of *Lecture Notes in Computer Science*, pages 230–236. Springer, 2003.

[5] A. Artale and E. Franconi. Introducing temporal description logics. In C. Dixon and M. Fisher, editors, *Proceedings of the Sixth International Workshop on Temporal Representation and Reasoning (TIME-99)*, pages 2–5. IEEE Computer Society Press, 1999.

[6] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, NY, USA, 1998.

[7] F. Baader and W. Snyder. Unification theory. In Robinson and Voronkov [80], chapter 8, pages 447–533.

[8] L. Bachmair, N. Derschowitz, and D. A. Plaisted. Completion without failure. In H. Ait-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures (Volume II): Rewriting Techniques*, pages 1–30. Academic Press, London, 1989.

[9] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[10] L. Bachmair and H. Ganzinger. Resolution theorem proving. In Robinson and Voronkov [80], chapter 2, pages 19–99.

[11] H. Barringer, M. Fisher, D. Gabbay, R. Owens, and M. Reynolds, editors. *The Imperative Future: Principles of Executable Temporal Logic.* Reasearch Studies Press, 1996.

[12] H. Barringer, M. Fisher, D. M. Gabbay, G. Gough, and R. Owens. MetateM: An introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.

[13] A. Behdenna, C. Dixon, and M. Fisher. Deductive verification of simple foraging robotic behaviours. *International Journal of Intelligent Computing and Cybernetics*, 2(4):604 – 643, 2009.

[14] N. Bjørner, Z. Manna, H. Sipma, and T. E. Uribe. Deductive verification of real-time systems using STeP. *Theoretical Computer Science*, 253(1):27–60, 2001.

[15] T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using Presburger arithmetic. In O. Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification, CAV '97, Haifa, Israel, June 22-25, 1997*, volume 1254 of *Lecture Notes in Computer Science*, pages 400–411. Springer, 1997.

[16] J. P. Burgess and Y. Gurevich. The decision problem for linear temporal logic. *Notre Dame Journal of Formal Logic*, 26(2):115–128, 1985.

[17] A. R. Cavalli and L. F. del Cerro. A decision method for linear temporal logic. In R. E. Shostak, editor, *7th International Conference on Automated Deduction, Napa, California, USA, May 14-16, 1984, Proceedings*, volume 170 of *Lecture Notes in Computer Science*, pages 113–127. Springer, 1984.

[18] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV version 2: An open source tool for symbolic model checking. In *Proceedings of the 14th International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364, Copenhagen, Denmark, July 2002. Springer.

[19] E. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, Cambridge, Massachusetts - London, England, 1999.

[20] E. M. Clarke and F. Lerda. Model checking: Software and beyond. *Journal of Universal Computer Science*, 13(5):639–649, 2007.

[21] E. M. Clarke, F. Lerda, and M. Talupur. An abstraction technique for real-time verification. In *Proceedings of the GM R&D Workshop on Next Generation Design and Verification Methodologies for Distributed Embedded Control System*, 2007.

[22] A. Degtyarev, M. Fisher, and B. Konev. A simplified clausal resolution procedure for propositional linear-time temporal logic. In U. Egly and C. G. Fermüller, editors, *Automated Reasoning with Analytic Tableaux and Related Methods, International*

Conference, TABLEAUX 2002, Copenhagen, Denmark, July 30 - August 1, 2002, Proceedings, volume 2381 of *Lecture Notes in Computer Science*, pages 85–99, London, UK, 2002. Springer-Verlag.

[23] A. Degtyarev, M. Fisher, and B. Konev. Monodic temporal resolution. *ACM Transactions On Computational Logic*, 7(1):108–150, 2006.

[24] A. Degtyarev, M. Fisher, and A. Lisitsa. Equality and monodic first-order temporal logic. *Studia Logica*, 72(2):147–156, 2002.

[25] G. Delzanno. Constraint-based verification of parameterized cache coherence protocols. *Formal Methods in System Design*, 23(3):257–301, 2003.

[26] J. Dick, J. Kalmus, and U. Martin. Automating the Knuth-Bendix ordering. *Acta Informatica*, 28(2):95–119, 1990.

[27] C. Dixon. Miss Scarlett in the ballroom with the lead piping. In R. L. de Mántaras and L. Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 995–996. IOS Press, 2004.

[28] C. Dixon. Specifying and verifying the game Cluedo using temporal logics of knowledge. Technical Report ULCS-04-003, University of Liverpool, 2004. Available at http://www.csc.liv.ac.uk/research/techreports/.

[29] C. Dixon, M. Fisher, B. Konev, and A. Lisitsa. Practical first-order temporal reasoning. *Proceedings of the 15th International Symposium on Temporal Representation and Reasoning, 2008. TIME '08.*, pages 156–163, June 2008.

[30] C. Dixon, M.-C. F. Gago, M. Fisher, and W. V. D. Hoek. Using temporal logics of knowledge in the formal verification of security protocols. Technical Report ULCS-03-022, University of Liverpool, 2003. Available at http://www.csc.liv.ac.uk/research/techreports/.

[31] C. Dixon, M.-C. F. Gago, M. Fisher, and W. van der Hoek. Using temporal logics of knowledge in the formal verification of security protocols. In *11th International Symposium on Temporal Representation and Reasoning (TIME 2004), 1-3 July 2004, Tatihou Island, Normandie, France*, pages 148–151. IEEE Computer Society, 2004.

[32] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 995–1072. 1990.

[33] E. A. Emerson and J. Srinivasan. Branching time temporal logic. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *REX Workshop*, volume 354 of *Lecture Notes in Computer Science*, pages 123–172. Springer, 1988.

[34] E. A. Euler, S. D. Jolly, and H. Curtis. The failures of the Mars Climate Orbiter and Mars Polar Lander: A perspective from the people involved. In *Proceedings of 24th Annual AAS Guidance and Control Conference*, number AAS 01-074. American Astronautical Society, 2001.

[35] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge.* MIT Press, 1995.

[36] M. C. Fernández-Gago, U. Hustadt, C. Dixon, M. Fisher, and B. Konev. First-order temporal verification in practice. *Journal of Automated Reasoning*, 34(3):295–321, 2005.

[37] M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, 2001.

[38] M. Fisher and A. Hepple. Executing logical agent specifications. In R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 1–27. Springer, June 2009.

[39] M. Fisher, B. Konev, and A. Lisitsa. Practical infinite-state verification with temporal reasoning. In E. M. Clarke, M. Minea, and F. L. Tiplea, editors, *Verification of Infinite-State Systems with Applications to Security, Proceedings of the NATO Advanced Research Workshop "Verification of Infinite State Systems with Applications to Security VISSAS 200", Timisoara, Romania, March 17-22, 2005*, volume 1 of *NATO Security through Science Series D: Information and Communication Security*, pages 91–100. IOS Press, 2005.

[40] N. Francez. *Fairness.* Springer-Verlag New York, Inc., New York, NY, USA, 1986.

[41] M.-C. F. Gago, M. Fisher, and C. Dixon. Algorithms for guiding clausal temporal resolution. In M. Jarke, J. Koehler, and G. Lakemeyer, editors, *KI 2002: Advances in Artificial Intelligence, 25th Annual German Conference on AI, Aachen, Germany, September 16-20, 2002, Proceedings*, volume 2479 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2002.

[42] O. Gasquet, A. Herzig, D. Longin, and M. Sahade. LoTREC: Logical tableaux research engineering companion. In B. Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings*, volume 3702 of *Lecture Notes in Computer Science*, pages 318–322. Springer, 2005.

[43] T. R. Halfhill. An error in a lookup table created the infamous bug in Intel's latest processor. In *BYTE*. March 1995.

[44] J. Handy. *The Cache Memory Book.* Academic Press, 1993.

[45] D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, volume F-13 of *NATO ASI Series*, pages 477–498. Springer, New York, NY, USA, 1985.

[46] A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. Propositional logics on the computer. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods, 4th International Workshop, TABLEAUX '95, Schloß Rheinfels, St. Goar, Germany, May 7-10, 1995, Proceedings*, volume 918 of *Lecture Notes in Computer Science*, pages 310–323. Springer, 1995.

[47] I. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable fragments of first-order temporal logics. *Annals of Pure and Applied Logic*, 106:85–134, 2000.

[48] G. J. Holzmann. *The SPIN Model Checker : Primer and Reference Manual.* Addison-Wesley, September 2003.

[49] U. Hustadt and B. Konev. TRP++ 2.0: A temporal resolution prover. In F. Baader, editor, *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2, 2003, Proceedings*, volume 2741 of *Lecture Notes in Computer Science*, pages 274–278. Springer, 2003.

[50] U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. TeMP: A temporal monodic prover. In D. A. Basin and M. Rusinowitch, editors, *Automated Reasoning - Second International Joint Conference, IJCAR 2004, Cork, Ireland, July 4-8, 2004, Proceedings*, volume 3097 of *Lecture Notes in Computer Science*, pages 326–330. Springer, 2004.

[51] U. Hustadt, B. Konev, and R. A. Schmidt. Deciding monodic fragments by temporal resolution. In R. Nieuwenhuis, editor, *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings*, volume 3632 of *Lecture Notes in Artificial Intelligence*, pages 204–218. Springer, 2005.

[52] U. Hustadt and R. A. Schmidt. Formulae which highlight differences between temporal logic and dynamic logic provers. In E. Giunchiglia and F. Massacci, editors, *Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*, Technical Report DII 14/01, pages 68–76. Dipartimento di Ingegneria dell'Informazione, Unversitá degli Studi di Siena, 2001.

[53] U. Hustadt and R. A. Schmidt. Scientific benchmarking with temporal logic decision procedures. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, editors, *Proceedings of the Eighth International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, pages 533–546. Morgan Kaufmann, 2002.

[54] F. Kamareddine. Reviewing the classical and the de Bruijn notation for λ-calculus and pure type systems. *Logic and Computation*, 11(3):363–394, 2001.

[55] Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A decision algorithm for full propositional temporal logic. In C. Courcoubetis, editor, *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 1993.

[56] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

[57] B. Konev, A. Degtyarev, C. Dixon, M. Fisher, and U. Hustadt. Towards the implementation of first-order temporal resolution: the expanding domain case. In *Proceedings of the 10th International Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic (TIME-ICTL 2003)*, pages 72–82. IEEE Computer Society, 2003.

[58] B. Konev, A. Degtyarev, C. Dixon, M. Fisher, and U. Hustadt. Mechanising first-order temporal resolution. *Information and Computation*, 199(1-2):55–86, 2005.

[59] B. Konev, A. Degtyarev, and M. Fisher. Handling equality in monodic temporal resolution. In M. Y. Vardi and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 10th International Conference, LPAR 2003, Almaty, Kazakhstan, September 22-26, 2003, Proceedings*, volume 2850 of *Lecture Notes in Computer Science*, pages 214–228. Springer, 2003.

[60] R. Kontchakov, C. Lutz, F. Wolter, and M. Zakharyaschev. Temporalising tableaux. *Studia Logica*, 76(1):91–134, 2004.

[61] G. L. Lann. An analysis of the Ariane 5 flight 501 failure - a system engineering perspective. In *1997 Workshop on Engineering of Computer-Based Systems (ECBS '97), March 24-28, 1997, Monterey, CA, USA*, pages 339–246. IEEE Computer Society, 1997.

[62] J.-L. Lassez, M. J. Maher, and K. Marriott. Unification revisited. In M. Boscarol, L. C. Aiello, and G. Levi, editors, *Foundations of Logic and Functional Programming*, volume 306 of *Lecture Notes in Computer Science*, pages 67–113. Springer, 1986.

[63] A. Leitsch. *The Resolution Calculus.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 1997.

[64] B. Löchner and T. Hillenbrand. A phytography of WALDMEISTER. *AI Communications*, 15(2-3):127–133, 2002.

[65] M. Ludwig and U. Hustadt. Fair derivations in monodic temporal reasoning. In R. A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 261–276. Springer, 2009.

[66] M. Ludwig and U. Hustadt. Redundancy elimination in monodic temporal reasoning. In N. Peltier and V. Sofronie-Stokkermans, editors, *FTP 2009 Workshop Proceedings*, pages 34 - 48. University of Oslo, Department of Informatics, 2009. Research Report 386.

[67] M. Ludwig and U. Hustadt. Resolution-based model construction for PLTL. In C. Lutz and J.-F. Raskin, editors, *TIME 2009, 16th International Symposium on Temporal Representation and Reasoning*, pages 73 – 80. IEEE Computer Society, 2009.

[68] M. Ludwig and U. Hustadt. Implementing a fair monodic temporal logic prover. *AI Communications*, 23(2-3):69–96, 2010.

[69] Z. Manna, N. Bjørner, A. Browne, E. Y. Chang, M. Colón, L. de Alfaro, H. Devarajan, A. Kapur, J. Lee, H. Sipma, and T. E. Uribe. STeP: the Stanford temporal prover. In P. D. Mosses, M. Nielsen, and M. I. Schwartzbach, editors, *TAPSOFT'95: Theory and Practice of Software Development, 6th International Joint Conference CAAP/FASE, Aarhus, Denmark, May 22-26, 1995, Proceedings*, volume 915 of *Lecture Notes in Computer Science*, pages 793–794. Springer, 1995.

[70] W. McCune and L. Wos. Otter - the CADE-13 competition incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997.

[71] B. C. Moszkowski and Z. Manna. Reasoning in interval temporal logic. In E. M. Clarke and D. Kozen, editors, *Logic of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 371–382. Springer, 1983.

[72] A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In Robinson and Voronkov [80], pages 335–367.

[73] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[74] A. Pnueli and Z. Manna. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer, 1991.

[75] A. Prasad Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(2-3):217–237, 1987.

[76] A. Prior. *Past, Present and Future.* Oxford University Press, 1967.

[77] I. V. Ramakrishnan, R. C. Sekar, and A. Voronkov. Term indexing. In Robinson and Voronkov [80], pages 1853–1964.

[78] A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2):91–110, 2002.

[79] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[80] J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[81] S. Schulz. E - a brainiac theorem prover. *AI Communications*, 15(2-3):111–126, 2002.

[82] R. L. Schwartz, P. M. Melliar-Smith, and F. H. Vogt. An interval logic for higher-level temporal reasoning. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, pages 173–186, 1983.

[83] S. Schwendimann. A new one-pass tableau calculus for PLTL. In H. C. M. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX '98, Oisterwijk, The Netherlands, May 5-8, 1998, Proceedings*, volume 1397 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 1998.

[84] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[85] M. Steedman. Temporality. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 895–935. Elsevier, 1997.

[86] C. Stirling. Modal and temporal logics. In S. Abramsky, D. M. Gabbay, and S. E. Maibaum, editors, *Handbook of Logic in Computer Science (vol. 2): Background: Computational Structures*, pages 478–551. Oxford University Press, New York, NY, USA, 1992.

[87] A. Szalas. Concerning the semantic consequence relation in first-order temporal logic. *Theoretical Computer Science*, 47(3):329–334, 1986.

[88] A. Szalas and L. Holenderski. Incompleteness of first-order temporal logic with until. *Theoretical Computer Science*, 57:317–325, 1988.

[89] A. U. Tansel, J. Clifford, S. K. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993.

[90] G. Venkatesh. A decision method for temporal logic based on resolution. In S. N. Maheshwari, editor, *Foundations of Software Technology and Theoretical Computer Science, Fifth Conference, New Delhi, India, December 16-18, 1985, Proceedings*, volume 206 of *Lecture Notes in Computer Science*, pages 272–289. Springer, 1985.

[91] C. Weidenbach. Combining superposition, sorts and splitting. In Robinson and Voronkov [80], pages 1965–2013.

[92] C. Weidenbach, R. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System description: SPASS version 3.0. In F. Pfenning, editor, *Automated Deduction CADE-21 : 21st*

International Conference on Automated Deduction, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 514–520, Bremen, Germany, 2007. Springer.

[93] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.

[94] F. Wolter and M. Zakharyaschev. Axiomatizing the monodic fragment of first-order temporal logic. *Annals of Pure and Applied Logic*, 118:133–145, 2002.

Index