CRutoN: Automatic Verification of a Robotic Assistant's Behaviours *

Paul Gainer[1], Clare Dixon[1], Kerstin Dautenhahn[2], Michael Fisher[1], Ullrich Hustadt[1], Joe Saunders[2], and Matt Webster[1]

[1] University of Liverpool, Liverpool, UK
{p.gainer, cldixon, mfisher, u.hustadt, matt}@liverpool.ac.uk
[2] University of Hertfordshire, Hatfield, UK
{k.dautenhahn, j.1.saunders}@herts.ac.uk

Abstract. The Care-O-bot is an autonomous robotic assistant that can support people in domestic and other environments. The behaviour of the robot can be defined by a set of high level control rules. The adoption and further development of such robotic assistants is inhibited by the absence of assurances about their safety. In previous work, formal models of the robot behaviour and its environment were constructed by hand and model checkers were then used to check whether desirable formal temporal properties were satisfied for all possible system behaviours. In this paper we describe the details of the software CRutoN, that provides an automatic translation from sets of robot control rules into input for the model checker NuSMV. We compare our work with previous attempts to formally verify the robot control rules, discuss the potential applications of the approach, and consider future directions of research.

1 Introduction

Robot assistants are autonomous robots that can help with home and work-oriented activities collaborating closely with humans. Personal care robots assist those who might be vulnerable due to illness, age or disability. In 2014 a new ISO safety standard for personal care robots was published, providing guidelines to manufacturers of personal care robots to ensure the safety of their design, construction and application [7]. However, the development and deployment of robotic assistants has been restricted by the lack of formal assurances of their safety.

Formal verification is the application of mathematical techniques to determine whether or not a system conforms exactly to its specification. These techniques are used in the development of software and hardware systems, notably in the development of critical systems where system failure can have drastic human or economic repercussions. Formal verification has already been applied

* This work was supported by both the Sir Joseph Rotblat Alumni Scholarship at Liverpool and the EPSRC Research Programme EP/K006193/1 *Trustworthy Robotic Assistants*.

to robotic systems, for instance, Cowley and Taylor [2] used linear logic and dependent-type theory to verify assembly robots. A different approach using hybrid automata and hybrid statecharts was employed by Mohammed et al. [10] to formally model and verify multirobot systems, and control algorithms for a surgical robot were verified by Kouskoulas et al. in [8]. Applying formal verification to the behaviours of robotic assistants can help to support their safety and trustworthiness by demonstrating that the robot always behaves in accordance with a set of formal requirements. We might, for instance, want to show that for all possible executions of robot behaviours the robot eventually performs some good action, or conversely that some bad action is never performed.

The Care-O-bot® is a robotic assistant that has been deployed in a domestic-style house at the University of Hertfordshire. The house is equipped with sensors which provide real-time information on the state of the house and its occupants. Studies have already been conducted to apply formal verification to the behaviours of the Care-O-bot in this environment. Model checking, an automated algorithmic verification technique, was applied in [3]. A model of the robot and its environment was manually constructed and the model checker NuSMV [1] was used to prove several properties relating to the priority and interruptibility of behaviours. Sensor data pertaining to the house was modelled by non-deterministically selecting one of several possible values for every sensor at any moment in time. In [17] algorithmic verification was again applied to the set of Care-O-bot behaviours. Here, models of the robot and its environment were manually constructed using the intelligent agent modelling and simulation language Brahms [15]. The *BrahmsToPromela* tool [16] was then used to automatically translate Brahms models into PROMELA, the input language for the model checker Spin [6]. This approach differed from the first in that the model of sensor data was more restrictive. The non-deterministic choice of sensor values was constrained using data taken from an activity log for a real 6 hour execution period of the robot.

These studies clearly demonstrated that the high level decision making of the Care-O-bot could indeed be verified using model checking, however both approaches had limiting factors. Firstly, effort was required to manually construct formal models for a fixed set of robot behaviours. Furthermore, a new set of control rules means the model needs to be constructed again. Secondly, in [3] the timing constraints in the control rules were dealt with in an ad-hoc manner. Additionally, the properties checked focused on the operation of the robot control rules rather than general requirements of the robot. In this paper we describe the software CRutoN[1], developed in [5], that proves to be an effective solution to the problem. CRutoN automates the generation of formal input models for the NuSMV model checker, and minimises the time needed to apply model checking to different sets of robot behaviours. Given a set of rules at the design stage, generated models could be used to find unexpected behaviour in the model. Less effort would then be required to refine the design since modification of the for-

[1] The software, sample output files, and input used in this paper, are available at `https://github.com/PaulGainer/CRutoN`.

mal model is automated. We show that the models generated using the software retain the same desirable properties as those constructed manually, and describe features of the software that facilitate modulation of the granularity of temporal aspects of the robot behaviours.

The Care-O-bot, the environment in which it operates, and the behaviours that determine the actions of the robot are introduced in Sec. 2. An overview of the CRutoN software, and the translation of control rules for the robot into an intermediate form representation are given in Sec. 3. In Sec. 4 we formalise the behaviour of the robot using linear time temporal logic, and we specify the expected behaviour of the robot in any generated formal model. Section 5 describes the translation from the intermediate form representation into input for the model checker NuSMV. In Sec. 6 the results of applying this translation to sets of control rules are presented, and we discuss the limitations of our approach. We give concluding remarks in Sec. 7 and outline some future work.

2 The Care-O-bot and the Robot House

The Robot House is a typical suburban house in Hertfordshire, UK. The house is appropriately furnished, and is equipped with a number of sensors. The sensors provide real time information about the state of the house and its occupants, for instance pressure sensors that detect when an occupant is seated, or electrical sensors indicating whether doors are open or closed [4,13].

The house provides a realistic setting in which experiments can be conducted using a number of robots, including the Care-O-bot. The Care-O-bot is a commercially available robot assistant developed at the Fraunhofer Institute for Manufacturing Engineering and Automation [12]. The robot has an articulated torso with a manipulator arm and tray, stereo optical sensors, LED lights, and appropriate sensors that provide information about its current state. The robot software is based on the Robot Operating System [11].

Control Rules and Behaviours. The high level decision making of the Care-O-bot is specified by a set of behaviours whose execution is controlled by a scheduler. Each *behaviour* consists of a sequence of atomic preconditions and a sequence of actions. Atomic preconditions and actions are also called *control rules*. *Actions* correspond to operations executed by the robot, including the setting of internal variables, implemented as ROS scripts. For instance, the robot may move to the living room and say "It's time for your medicine", or may turn its inbuilt lights to yellow. *Atomic preconditions* are propositional statements that check either the internal state of the robot or the state of the environment in which the robot operates. They may include an additional constraint requiring the condition to have remained *true* for some period of time, or requiring the condition to have been *true* at least once within some period of time.

The atomic preconditions of a behaviour are linked by Boolean operators into a propositional *precondition*. This precondition must evaluate to true for the behaviour to be scheduled for execution. If the sequence of atomic preconditions for a behaviour is empty then the associated propositional precondition,

(a) (b)

Fig. 1. The Care-O-bot (a) operating in the Robot House in Hertfordshire, and the floor plan of the house (b).

the empty conjunction, is always true. Behaviours without atomic preconditions can be declared to be *subroutines*. Subroutines cannot be picked by the scheduler for execution but can be executed directly by actions in other behaviours. Behaviours that are not subroutines are also called *schedulable*.

Behaviour Scheduling. The algorithm in Figure 3 describes how the Care-O-bot schedules its behaviours. When all preconditions of a behaviour are true, then the behaviour can be selected by the scheduler for execution. Only one behaviour can be executed at a time and when a behaviour is executed, the robot will sequentially perform its sequence of actions. Each behaviour has a *priority*, given by a natural number, which is used by the scheduler to decide which behaviour to execute if the preconditions of several behaviour are true at the same time. Behaviours with higher priorities will be scheduled before those with lower priorities, and if more than one behaviour shares the highest priority then one of these will be non-deterministically selected for scheduling. A behaviour can be declared to be *interruptible*. The execution of the sequence of actions of an interruptible behaviour can be interrupted by another schedulable behaviour having a priority greater than the priority of the executing behaviour, if, and only if, the preconditions of the interrupting behaviour hold. Any remaining actions in the interrupted behaviour are lost, and the behaviour must again wait to be scheduled as usual. All subroutines are uninterruptible.

Figure 2 shows the control rules for the S1-Med-5PM-Remind behaviour. Rules 1 and 2 are atomic preconditions and rules 3−9 are actions. For each control rule in the database there is a flag indicating if that rule is a precondition or an action. These are omitted here for simplicity. Rule 1 requires the time to be after 5pm and rule 2 requires the robot's internal flag `::502::5PM-MedicineDue`

1. `Time is on or after 17:00:00`
2. `::502::5PM-MedicineDue is true`
3. `Turn light on ::0::Care-O-Bot 3.2 to yellow`
4. `move ::0::Care-O-Bot 3.2 to ::14:: Living Room Sofa Area in the Living Room and wait for completion`
5. `Turn light on ::0::Care-O-Bot 3.2 to white and wait for completion`
6. `::0::Care-O-Bot 3.2 says 'Its time for your medicine' and wait for completion`
7. `::0::Care-O-Bot 3.2 GUI,S1-Set-GoToKitchen, S1-Set-ReturnHome, S1-Set-WaitHere`
8. `SET ::502::5PM-MedicineDue TO false`
9. `SET ::503::5PM-MedicineReminder TO true`

Fig. 2. The S1-Med-5PM-Remind Behaviour

to be *true*. When both preconditions hold, and this behaviour is scheduled for execution, the robot will sequentially execute actions 3−9. It first turns its lights to yellow, then moves to the living room, near the sofa and then turns its lights to white. The robot then tells the occupant "It's time to take your medicine", and displays some options on its GUI that allow the occupant to either send the robot to the kitchen, send the robot to its charging point, or instruct the robot to do nothing. The values of two internal variables are then set to *true* and *false* respectively.

3 Intermediate Form Translation

As a first step to the transformation of Care-O-bot control rules into input for NuSMV, we translate the extracted control rules into a succinct intermediate form representation that represents all of the information extracted from the control rules. This intermediate form facilitated the final translation into SMV. This was so that translations from the intermediate form into input for other model checkers could potentially be defined and implemented.

Precondition Classification. For the translation into intermediate form, we distinguish two categories of atomic preconditions: Value Check and Time Constraint. A Value Check checks the value of some variable corresponding to the internal state of the robot or the state of the environment. A Time Constraint requires the current time of day to be within some given time interval. Table 1 gives examples of preconditions for each of the two categories. The first Value Check checks the value of the Boolean variable `::502::5PM-MedicineDue`, which equates to an internal variable of the robot, the second Value Check checks that the location of the robot is in the living room, and the two Time Constraints require the current time to be at or after 5pm, or between midnight and 5pm.

Action Classification. We also distinguish four categories of actions: Value Assignment, Behaviour Execution, Behaviour Selection, and Delay. A Value Assign-

Table 1. Precondition Categories

Value Check	`::502::5PM-MedicineDue is true` `::0::Care-O-Bot 3.2 location is ::14::the Living Room`
Time Constraint	`Time is on or after 17:00:00`
	`Time is between 00:00:00 and 16:59:00`

ment assigns some value to a variable. This can represent a change in the internal state of the robot, or a change in the state of the environment. A Behaviour Execution transfers control from some scheduled behaviour to another behaviour. A Behaviour Selection again transfers control to another behaviour, however, here a choice of possible behaviours is presented to the occupant of the house via the GUI of the Care-O-bot. A Delay instructs the robot to do nothing for a given number of seconds. Table 2 gives examples of actions for each of the four categories. The first Value Assignment assigns the value *true* to the internal variable `::503::5PM-MedicineReminder`, while the second Value Assignment selects a phrase to be vocalised by the robot. The Behaviour Execution executes the behaviour S1-sleep, and the Behaviour Selection allows the occupant to choose to execute one of the `S1-Set-GoToKitchen`, `S1-Set-ReturnHome`, and `S1-Set-WaitHere` behaviours. Finally, the Delay instructs the robot to do nothing for 1 second.

Parsing and Translation. As can be observed from the tables, the control rules can have a wide variety of syntactic forms. To ensure that future control rules that may use additional categories of preconditions or actions can be translated correctly, the parsing and extraction of information from the rules is not hard coded into the software. The translator takes as input a set of Grammar Rules defining the syntax of the control rules, and a set of Data Extraction Rules that define how information should be extracted from the control rules. The Grammar Rule for a new control rule allows an automaton to be constructed at run time that can be used to parse this new syntactic rule form. This Grammar Rule has a corresponding Data Extraction Rule that describes how meaningful information can be extracted from the text parsed by the automaton. The software

Table 2. Action Categories

Value Assignment	`SET ::503::5PM-MedicineReminder TO true`
	`::0::Care-O-Bot 3.2 says Its time for your medicine`
Behaviour Execution	`Execute sequence S1-sleep on ::0::Care-O-Bot 3.2`
Behaviour Selection	`::0::Care-O-Bot 3.2 GUI,S1-Set-GoToKitchen, S1-Set-ReturnHome, S1-Set-WaitHere`
Delay	`Wait for 1 seconds on ::0::Care-O-Bot 3.2`

```
procedure BehaviourScheduling
  sched ← none
  while true do
    S ← all schedulable behaviours
          whose preconditions hold
    P ← b ∈ S with highest priority
    if sched = none and S≠∅ then
      sched ← P
    else if interruptible(sched) and
priority(P) > priority(sched) then
      sched ← P
    else if all actions executed then
      sched ← none
    else execute next action
    end if
  end while
end procedure
```

Fig. 3. Care-O-bot Behaviour Scheduling.

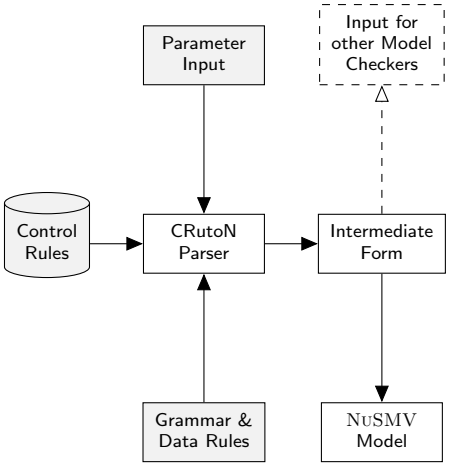

Fig. 4. A System Diagram of CRutoN.

has options that regulate the level of automation of the translation process and determine when a user should be prompted to disambiguate input if necessary.

Figure 4 shows a diagram of the system. Given a set of Care-O-bot behaviours $\mathcal{S}$, a set of Grammar Rules and Data Extraction Rules, and parameter input for the parser, we construct an intermediate form representation (IFR). The IFR of $\mathcal{S}$ is a tuple $\mathrm{I}(\mathcal{S}) = \langle \Gamma, \Omega, f_{sch}, f_{int} \rangle$, where Γ is a set of behaviours in intermediate form, f_{sch} and f_{int} are predicates over Γ that are *true* if, and only if, a behaviour is respectively schedulable or interruptible, and Ω is a set of variables that represent the internal state of the robot and the state of the environment. A behaviour $\mathcal{B} \in \Gamma$ is a tuple $\mathcal{B} = \langle P, \mathcal{A}, \rho \rangle$, where P is a formula formed by combining the atomic preconditions of the behaviour, $\rho \in \mathbb{N}$ is the priority of the behaviour, and $\mathcal{A}$ is a sequence of actions. Each individual action $\alpha \in \mathcal{A}$ is a tuple of values. A Value Assignment is a pair (ω, ν), where $\omega \in \Omega$ is a variable and ν is a value from the domain of ω, a Behaviour Execution is a subroutine $\mathcal{B} \in \Gamma$, a Behaviour Selection is a set of subroutines $\mathfrak{B} \subseteq \Gamma$, and a Delay is a natural number D.

4 Property Specification

There was no available formal semantics specifying the behaviours of the Care-O-bot. An analysis of the set of control rules modelled in [3,17] and input from the development team led to the formulation of desirable properties that would be expected to hold in any model resulting from the translation into model checker input. Linear-time temporal logic (LTL) was used to specify these properties.

Linear Temporal Logic. In LTL the model of time is isomorphic to the natural numbers, and a model for a formula is a sequence of states $\Sigma = \sigma_0, \sigma_1, \ldots$ such that each state σ_i is a valuation for the set of variables $\mathcal{V}$ at the i^{th} moment in time, and $\sigma_i(\omega)$ is finite for every $\omega \in \mathcal{V}$ and $i \geq 0$. The set of LTL formulae can be defined inductively as

$$\varphi ::= \top \mid \bot \mid (\omega{=}\nu) \mid \neg\varphi \mid (\varphi \vee \psi) \mid (\varphi \wedge \psi) \mid (\varphi \Longrightarrow \psi) \mid \bigcirc\varphi \mid \Box\varphi \mid \Diamond\varphi$$

where $\omega \in \mathcal{V}$ and φ and ψ are LTL forumlae. If ω is a Boolean variable we will often use ω for (ω=*true*), and $\neg\omega$ for (ω=*false*). We can define $(\Sigma, i) \models \Phi$, the truth of a formula Φ in Σ at time i, as follows:

$(\Sigma, i) \models (\omega{=}\nu)$	**iff** $\sigma_i(\omega){=}\nu$
$(\Sigma, i) \models \bigcirc\Phi$	**iff** $(\Sigma, i+1) \models \Phi$
$(\Sigma, i) \models \Diamond\Phi$	**iff** for some $k \in \mathbb{N}, (k \geq i)$ and $(\Sigma, k) \models \Phi$
$(\Sigma, i) \models \Box\Phi$	**iff** for all $k \in \mathbb{N}, (k \geq i)$ implies $(\Sigma, k) \models \Phi$.

The semantics of propositional operators is defined as usual.

Formal Model of the System State. Given a set of Care-O-bot behaviours $\mathcal{S}$, and its intermediate form representation $\mathrm{I}(\mathcal{S}) = \langle \Gamma, \Omega, f_{sch}, f_{int} \rangle$, we define variables SCHED and STEP that are used to model the behaviour scheduling procedure of the robot. The scheduling variable SCHED ranges over the values $\{none\} \cup \{sched^{\mathcal{B}} \mid \mathcal{B} \in \Gamma\}$ and its value indicates the behaviour that is currently scheduled by the robot with *none* indicating that no behaviour is scheduled. The step variable STEP ranges over the values $\{none\} \cup \{step^1, \ldots, step^k\}$ where $k = \mathbf{max}\{|\mathcal{A}| \mid \langle P, \mathcal{A}, \rho \rangle \in \Gamma\}$, and its value indicates the index of an action that is being executed in the sequence of actions for a behaviour that is currently scheduled by the robot, with *none* indicating that no action is being executed. A model of our system is a sequence of states $\Sigma = \sigma_0, \sigma_1, \ldots$, and each state is a valuation for $\Omega \cup \{\text{SCHED}, \text{STEP}\}$.

Specification. We now formally express properties that we would expect to hold in any generated NuSMV model. The following schematic formulae intend to capture the behavioural semantics of the robot with regards to behaviour scheduling and action execution. Let $\Gamma^{sch} = \{\mathcal{B} \in \Gamma \mid f_{sch}(\mathcal{B})\}$ be the set of all behaviours that are schedulable, and let $\Gamma^{int} = \{\mathcal{B} \in \Gamma \mid f_{int}(\mathcal{B})\}$ be the set of all behaviours that are interruptible. For every $\mathcal{B} = \langle P, \mathcal{A}, \rho \rangle \in \Gamma$ we define $\triangle^{\mathcal{B}} = \{\langle P', \mathcal{A}', \rho' \rangle \in \Gamma \mid \rho' > \rho\}$ to be the set of all behaviours in Γ that have a higher priority than $\mathcal{B}$.

If the k^{th} action of a scheduled behaviour $\mathcal{B}$ is the Value Assignment (ω, ν) and this action is executed, then the variable ω should have the value ν in the next moment in time.

$$\Box\Big[\big(\text{SCHED} = sched^{\mathcal{B}} \wedge \text{STEP} = step^k\big) \implies \bigcirc\big[(\omega = \nu)\big]\Big] \tag{1}$$

Scheduling. If no behaviours are scheduled and the preconditions to at least one behaviour hold, then in the next moment in time a behaviour will be scheduled.

$$\Box\left[\left(\begin{array}{c}\textsc{sched} = \mathit{none}\\ \wedge \bigvee_{\langle P,\mathcal{A},\rho\rangle\in\Gamma^{sch}} P\end{array}\right) \implies \bigcirc\left[\begin{array}{c}\textsc{step} = \mathit{step}^1\\ \wedge \bigvee_{\mathcal{B}\in\Gamma^{sch}} \textsc{sched} = \mathit{sched}^{\mathcal{B}}\end{array}\right]\right] \tag{2}$$

Termination. For any behaviour $\mathcal{B}$ that executes a Value Assignment or Delay as its k^{th} action, and this is the last action of the behaviour, if the preconditions to no other schedulable behaviour hold then in the next moment in time no behaviour should be executing.

$$\Box\left[\left(\begin{array}{c}\textsc{sched} = \mathit{sched}^{\mathcal{B}} \wedge \textsc{step} = \mathit{step}^k\\ \wedge \bigwedge_{\langle P,\mathcal{A},\rho\rangle\in\Gamma^{sch}} \neg P\end{array}\right) \implies \bigcirc\, \textsc{sched} = \mathit{none}\right] \tag{3}$$

Prioritisation. If no behaviour is scheduled, and the preconditions to one or more behaviours hold, then in the next moment in time the schedulable behaviour with the highest priority will be executing its first action.

$$\Box\left[\begin{array}{c}\left(\textsc{sched} = \mathit{none} \wedge \bigvee_{\langle P,\mathcal{A},\rho\rangle\in\Gamma^{sch}} P\right) \implies\\ \bigvee_{\mathcal{B}\in\Gamma^{sch}} \left(\begin{array}{c}\bigwedge_{\langle P',\mathcal{A}',\rho'\rangle\in\triangle^{\mathcal{B}}\cap\Gamma^{sch}} \neg P'\\ \wedge \bigcirc[\textsc{sched} = \mathit{sched}^{\mathcal{B}} \wedge \textsc{step} = \mathit{step}^1]\end{array}\right)\end{array}\right] \tag{4}$$

Persistence. For every uninterruptible behaviour $\mathcal{B}$, if it has been scheduled and has an action sequence of length k, then it should always eventually execute its last (k^{th}) action.

$$\Box\left[\textsc{sched} = \mathit{sched}^{\mathcal{B}} \implies \Diamond(\textsc{sched} = \mathit{sched}^{\mathcal{B}} \wedge \textsc{step} = \mathit{step}^k)\right] \tag{5}$$

Continuity. For any interruptible behaviour $\mathcal{B}$ that executes a Value Assignment or Delay as its k^{th} action, where that action is not the last action, in the next moment in time the behaviour should be executing its $(k+1)^{\text{th}}$ action if no other behaviour can interrupt $\mathcal{B}$.

$$\Box\left[\begin{array}{c}(\textsc{sched} = \mathit{sched}^{\mathcal{B}} \wedge \textsc{step} = \mathit{step}^k \wedge \bigwedge_{\langle P,\mathcal{A},\rho\rangle\in\triangle^{\mathcal{B}}\cap\Gamma^{sch}} \neg P)\\ \implies \bigcirc[\textsc{sched} = \mathit{sched}^{\mathcal{B}} \wedge \textsc{step} = \mathit{step}^{k+1}]\end{array}\right] \tag{6}$$

Note that without the 3^{rd} conjunct we can also show continuity for uninterruptible behaviours executing Value Assignment or Delay actions.

Discontinuity. For all interruptible behaviours it should be the case that if the behaviour can be interrupted in the next moment in time, the schedulable behaviour having the highest priority of all the behaviours that can interrupt should be executing its first action.

$$\bigwedge_{\mathcal{B}\in\Gamma^{int}} \Box\left[\bigvee_{\mathcal{B}'\in\triangle^{\mathcal{B}}\cap\Gamma^{sch}} \left(\begin{array}{c}\bigcirc[\textsc{sched} = \mathit{sched}^{\mathcal{B}'} \wedge \textsc{step} = \mathit{step}^1]\\ \wedge \bigwedge_{\langle P,\mathcal{A},\rho\rangle\in\triangle^{\mathcal{B}'}\cap\Gamma^{sch}} \neg P\end{array}\right)\right] \tag{7}$$

Delegation. For every behaviour $\mathcal{B}$ executing a Behaviour Execution, $\mathcal{B}_{ex}$, as its k^{th} action, if $\mathcal{B}$ is not interrupted by another behaviour then in the next moment in time $\mathcal{B}_{ex}$ should be scheduled and executing its first action.

$$\Box\left[\begin{pmatrix}\text{SCHED} = sched^{\mathcal{B}} \wedge \text{STEP} = step^{k}\\ \wedge \bigwedge_{\langle P,\mathcal{A},\rho\rangle\in\triangle^{\mathcal{B}}\cap\Gamma^{sch}} \neg P\end{pmatrix} \implies \bigcirc\left[\begin{matrix}\text{SCHED} = sched^{\mathcal{B}_{ex}}\\ \wedge \text{STEP} = step^{1}\end{matrix}\right]\right] \quad (8)$$

Resumption. For every behaviour $\mathcal{B}$ executing a Behaviour Execution $\mathcal{B}_{ex}$ as its k^{th} action, where that action is not the last action, if $\mathcal{B}$ is not interrupted by another behaviour then at some time after that $\mathcal{B}_{ex}$ should have finished executing its actions, and the original behaviour should be executing its $(k{+}1)^{\text{th}}$ action.

$$\Box\left[\begin{pmatrix}\text{SCHED} = sched^{\mathcal{B}} \wedge \text{STEP} = step^{k}\\ \wedge \bigwedge_{\langle P,\mathcal{A},\rho\rangle\in\triangle^{\mathcal{B}}\cap\Gamma^{sch}} \neg P\end{pmatrix} \implies \Diamond\left[\begin{matrix}\text{SCHED} = sched^{\mathcal{B}}\\ \wedge \text{STEP} = step^{k+1}\end{matrix}\right]\right] \quad (9)$$

Selection. For every behaviour $\mathcal{B}$ executing a Behaviour Selection $\mathfrak{B}$ as its k^{th} action, if $\mathcal{B}$ is not interrupted by another behaviour then in the next moment in time a behaviour in $\mathfrak{B}$ should be scheduled and executing its first action.

$$\Box\left[\begin{pmatrix}\text{SCHED} = sched^{\mathcal{B}}\\ \wedge \text{STEP} = step^{k}\\ \wedge \bigwedge_{\langle P,\mathcal{A},\rho\rangle\in\triangle^{\mathcal{B}}\cap\Gamma^{sch}} \neg P\end{pmatrix} \Longrightarrow \bigvee_{\mathcal{B}_{ex}\in\mathfrak{B}} \bigcirc\left[\begin{matrix}\text{SCHED} = sched^{\mathcal{B}_{ex}}\\ \wedge \text{STEP} = step^{1}\end{matrix}\right]\right] \quad (10)$$

Selection Resumption. For every behaviour $\mathcal{B}$ executing a Behaviour Selection $\mathfrak{B}$ as its k^{th} action, where that action is not the last action, if $\mathcal{B}$ is not interrupted by another behaviour then at some time after that any subroutine in $\mathfrak{B}$ should have finished executing its actions, and the original behaviour should be executing its $(k{+}1)^{\text{th}}$ action.

$$\Box\left[\begin{pmatrix}\text{SCHED} = sched^{\mathcal{B}}\\ \wedge \text{STEP} = step^{k}\\ \wedge \bigwedge_{\langle P,\mathcal{A},\rho\rangle\in\triangle^{\mathcal{B}}\cap\Gamma^{sch}} \neg P\end{pmatrix} \implies \bigvee_{\mathcal{B}_{ex}\in\mathfrak{B}} \Diamond\left[\begin{matrix}\text{SCHED} = sched^{\mathcal{B}}\\ \wedge \text{STEP} = step^{k+1}\end{matrix}\right]\right] \quad (11)$$

5 Translation into SMV

NuSMV [1] is a symbolic BDD-based model checker. The model checker accepts as input a finite state transition system defined using the modelling language SMV [9]. NuSMV input models can be decomposed into separate modules. Every model has at least one module, the main module, and some number of additional parameterisable modules. Each model consists of three sections: VAR, ASSIGN, and DEFINE. The VAR section defines variables and instances of modules; variables can be Booleans, symbolic enumerated types, or finitely bound integers. The global state of a NuSMV model is a valuation for all variables in the model. Initial states of the model and transitions between states are defined in the ASSIGN section. Finally, macro expressions can be defined in the DEFINE section.

Given a set of Care-O-bot behaviours $\mathcal{S}$, and its intermediate form representation $\text{I}(S)=\langle\Gamma,\Omega,f_{sch},f_{int}\rangle$, we construct an input model for NuSMV as follows.

State Variables. For every $\omega \in \Omega$ there is a variable declaration in the VAR section of the main module. Variables are either Booleans or enumerated types – variables over a set of symbolic constants. Variables that correspond to internal flags of the robot are initialised to *false*, and initial values for variables corresponding to the environment are non-deterministically chosen by NuSMV in the initial state. Initial values for both types of variable can also be explicitly specified in the form of additional input to the software. As in Sec. 4 we also define the enumerated variables scheduled and step. Initially, both variables have the value *none*. The additional variable scheduled_last ranges over the same values as scheduled, and is used to record a behaviour that executes a Behaviour Execution or Behaviour Selection so that it can be rescheduled once the subroutine has executed all of its actions. The initial value of scheduled_last is *none*. Since we only use one variable to record behaviours that should be rescheduled, we can only construct models where the level of nesting of behaviour executions is at most 1. Introducing additional variables would allow us to extend the nesting level, at the expense of model size.

The time variable records the time of day in the Robot House. Values for time are defined by partitioning a 24 hour period into intervals during which different subsets of the set of all Time Constraints in the model hold. The initial value of time is non-deterministically chosen in the initial state, and the value then remains constant throughout a run of the system. If desired, time can be set to an exact time of day using a parameter to the software, but still remains constant throughout. Note that this is a limitation relating to explicit time.

Behaviours. For every $\mathcal{B} = \langle P, \mathcal{A}, \rho \rangle \in \Gamma$ there is a corresponding macro expression for P in the main module; for non-schedulable behaviours this is simply *false*. In NuSMV a variable assignment occurs if some constraint holds. There is an ordering on variable assignments such that if the constraints hold for multiple assignments to a single variable then the assignment ordered first will be applied. We can exploit this to ensure that behaviours with higher priorities are scheduled before those with lower priorities i.e. scheduled is assigned a value corresponding to the behaviour having the highest priority of all those that can be scheduled. Assignments to scheduled are constrained by the expression corresponding to P, and by an interruptibility macro expression that evaluates to *true* if a behaviour with a lower priority is currently scheduled. For each Value Assignment in $\mathcal{A}$ there is a corresponding variable assignment, constrained by scheduled having a value corresponding to $\mathcal{B}$ and step having a value corresponding to the index of the action in $\mathcal{A}$. For each Behaviour Execution there is a corresponding assignment to scheduled_last of a value corresponding to $\mathcal{B}$, an assignment of 1 to step, and an assignment to scheduled of a value corresponding to that of the subroutine being executed. Additional variable assignments model the rescheduling of a behaviour that calls a subroutine, once the subroutine has executed all of its actions.

Temporal Constraints. Some Value Check rules have additional temporal constraints. There are two types of constraint, Been-In-State and Was-In-State, that require some variable $\omega \in \Omega$ to have respectively maintained some value ν dur-

Table 3. Priorities and Flags for Care-O-bot Behaviours

Name	Pri	Int	Sch
S1-Med-5PM-Reset	90	0	1
checkBell	80	0	1
unCheckBell	80	0	1
S1-remindFridgeDoor	80	0	1
answerDoorBell	70	0	1
S1-alertFridgeDoor	60	0	1
S1-Med5PM	50	1	1
S1-Med5PM-Remind	50	1	1
S1-gotoKitchen	40	1	1
S1-gotoSofa	40	1	1
S1-gotoTable	40	1	1
S1-kitchenAwaitCmd	40	1	1
Sw-sofaAwaitCmd	40	1	1
S1-tableAwaitCmd	40	1	1
S1-WaitHere	40	1	1
S1-ReturnHome	40	1	1
S1-continueWatchTV	35	1	1
S1-watchTV	30	1	1
S1-sleep	10	1	1

ing some previous period of time, or to have had the value ν at least once during some previous period of time. We model Been-In-State and Was-In-State constraints by introducing additional variables that record the number of transitions of the model since a variable last had a value other than ν or since a variable last had the value ν respectively. We associate with each state of the model some fixed length of time in seconds (duration), and this determines the number of values over which these additional variables range. The software has a parameter determining the granularity of these temporal aspects of the model.

6 Results and Discussion

We focus on the complete set of 31 behaviours developed as part of the EU Accompany project[2] and available from the project's Git repository[3]. The priority (Pri), interruptibility (Int), and schedulability (Sch) of the 19 schedulable behaviours is given in Table 3. There were also 12 unschedulable subroutines all with Pri, Int, and Sch set to 0. For every behaviour, and every action in those behaviours, we instantiated the corresponding properties specified in Sec. 4. The full specification for the system was the conjunction of all of these individually instantiated properties. All models generated using all behaviours with different sets of parameter input for CRutoN satisfied their corresponding specification.

Generated models can be used to check properties pertaining to specific behaviours and robot actions. We might, for instance, want to check whether a behaviour will eventually be scheduled, or if the robot will eventually perform some action, given the current state of the robot and environment. We validated our model by checking properties that were originally specified in [3]. For example, *"is it always the case that if the fridge door is open and the robot has not already alerted the user, then at some point in the future the robot will alert the user?"* We found that for all generated models the verification results matched those obtained by checking the properties in the manually constructed model.

[2] `http://accompanyproject.eu`

[3] `https://github.com/uh-adapsys/accompany`

Table 4. Model Size and Model Checking Times for Different Temporal Granularities.

Seconds per State	600	500	400	300	200	100	50
States	2^{49}	2^{49}	2^{49}	2^{50}	2^{51}	2^{53}	2^{56}
Reachable States	2^{22}	2^{22}	2^{23}	2^{23}	2^{24}	2^{26}	2^{28}
Model Build Time(s)	11.94	12.08	13.17	15.13	20.43	39.43	108.64
Model Checking Time(s)	0.73	0.82	0.99	1.16	1.75	3.77	8.95

Recall that CRutoN accepts parameters to allow modulation of the temporal granularity of the model, and associates a fixed length of time (duration) in seconds with every state in the formal model. Table 4 shows the effect of temporal granularity on the size of the models, and the time taken to perform model checking. The results correspond to the initial set of 31 behaviours. We generated models for 7 different durations for each state. The model checking times indicate the time taken to check the property $\Box((\mathsf{scheduled} = \mathsf{S1\text{-}alertFridgeDoor} \wedge \mathsf{step} = \mathsf{step}^1) \implies \Diamond(\mathsf{scheduled} = \mathsf{S1\text{-}alertFridgeDoor} \wedge \mathsf{step} = \mathsf{step}^9))$, which was *true* in each model as it is always the case that if the behaviour S1-alertFridgeDoor is executing its first action then is should eventually execute its last (9^{th}) action, since the behaviour is not interruptible. Note that this sample property was arbitrarily selected, and checking this property serves only to illustrate the effect of temporal granularity on model size, and hence model checking times. The results show that we can use a sensible duration of time for each state and still perform model checking within a reasonable amount of time. Using shorter durations per state would result in larger models, and hence longer model checking times. We can therefore extrapolate a trade-off between the time taken to perform model checking, and the time that would be required to manually extend existing formal models to include new behaviours.

For all investigated durations the corresponding model generated using CRutoN contains more reachable states than the manually constructed model described in [3]. One reason for this difference is that in the manual construction a distinction between value assignments to internal variables of the robot and all other actions was made. Only the latter result in a new state, the former do not. Whilst CRutoN currently does not make such a distinction and any action results in a new state, we could differentiate between value assignments and other actions (see Table 2) to group sequences of value assignments into one state.

It is clear that our modelling of temporal aspects is not ideal. Time Constraints are either always satisfied, or always not satisfied, along a run of the model, since the time of day is fixed for each path. Alternatively, we could allow the time of day to be non-deterministically chosen in each state, and then constrain that choice in the properties. We also note that it is sometimes difficult to determine a sensible value for the duration of time associated with a single state of the formal model, since setting this value too low can result in large models in which it is infeasible to check properties within a realistic amount of time, and setting this value to be too high results in unrealistic models where many of

the constraints imposed by Been-In-State and Was-In-State conditions are not included in the model as the durations of time to which they refer are too small.

We also applied our automatic transformations to an extended set of Care-O-bot behaviours provided by the development team working with the robot. The original set of 31 behaviours with 156 control rules was extended to 88 behaviours with 324 control rules. Some rules had new syntactic forms, however the expressiveness of the Grammar Rules and Data Extraction Rules allowed the software to parse all control rules, and automatically generate a formal NuSMV model that satisfied its specification.

7 Conclusion

We have described a translation from a set of control rules defining the behaviour of the Care-O-bot into both an intermediate form representation, and furthermore into input for the model checker NuSMV. We presented the software CRutoN that automates these translation processes. Formal models that satisfy their specifications are automatically generated for different sets of input parameters to the software, and the complexity of the generated models was evaluated with regards to the granularity of the temporal aspects. We aim to generalise our approach so that formal models could be automatically generated for other robot systems using similar rule constructs.

The generated intermediate form representation for a set of control rules could be used to develop further translations into input for other model checkers. We could, for instance, extend our models to incorporate uncertainty arising from faulty sensors or actuators, or the unpredictable behaviour of a human in the Robot House, and develop a translation into a probabilistic model checker. In Sec. 6 we discussed the limitations of our model with regards to the temporal aspects of robot behaviours. Translations into input for verification tools for real-time systems could be developed to refine our model of time.

Recent work in the robot house has allowed users to add their own behaviours, built upon existing primitives, via the 'TeachMe' system [14]. We have carried out a static analysis on the priorities and preconditions of newly added behaviours to advise users of potential problems. For example, the added behaviour will never be executed because an existing behaviour with a higher priority has a subset of the preconditions of the added behaviour. Evaluations have shown that users find this helpful when adding behaviours. The tool accomplishes this using an intermediate form representation of the behaviours generated by the CRutoN parser.

The analysis of scheduling issues arising from the prioritisation of behaviours could be complemented by formalising a set of properties relating to changes in the state of the robot and its environment resulting from robot actions, and automatically checking that these hold in generated models. Properties to be checked could include safety properties that would require the robot to never perform a specific action when in proximity to a human. A further issue that could be addressed is how counterexamples, generated when requisite properties fail to hold in the model, could be presented to a user in a comprehensible form.

References

1. Cimatti, A., Clarke, E.D., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002), doi:10.1007/3-540-45657-0_29
2. Cowley, A., Taylor, C.J.: Towards language-based verification of robot behaviors. In: Proceedings of IROS 2011. pp. 4776–4782. IEEE (2011)
3. Dixon, C., Webster, M., Saunders, J., Fisher, M., Dautenhahn, K.: "The fridge door is open"—temporal verification of a robotic assistant's behaviours. In: Mistry, M., Leonardis, A., Witkowski, M., Melhuish, C. (eds.) TAROS 2014. LNCS, vol. 9716, pp. 97–108. Springer, Cham (2014), doi:10.1007/978-3-319-10401-0_9
4. Duque, I., Dautenhahn, K., Koay, K.L., Willcock, L., Christianson, B.: Knowledge-driven user activity recognition for a smart house? Development and validation of a generic and low-cost, resource-efficient system. In: Proceedings of ACHI 2013. IARIA XPS Press (2013)
5. Gainer, P.: Verification for a Robotic Assistant. Tech. Rep. ULCS-17-003, Department of Computer Science, University of Liverpool, Liverpool, UK (2017)
6. Holzmann, G.J.: The Spin model checker: primer and reference manual. Addison-Wesley (2004)
7. ISO: Robots and robotic devices – safety requirements for personal care robots. ISO 13482:2014, Int. Organization for Standardization, Geneva, Switzerland (2014)
8. Kouskoulas, Y., Renshaw, D., Platzer, A., Kazanzides, P.: Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In: Proceedings of HSCC 2013. pp. 263–272. ACM (2013)
9. McMillan, K.L.: The SMV language. Tech. rep., Cadence Berkeley Labs (1999)
10. Mohammed, A., Stolzenburg, F., Furbach, U.: Multi-robot systems: Modeling, specification, and model checking. INTECH Open Access Publisher (2010)
11. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: Proceedings of the ICRA Workshop on Open Source Software in Robotics (2009)
12. Reiser, U., Connette, C., Fischer, J., Kubacki, J., Bubeck, A., Weisshardt, F., Jacobs, T., Parlitz, C., Hägele, M., Verl, A.: Care-o-bot® 3: Creating a product vision for service robot applications by integrating design and technology. In: Proceedings of IROS 2009. pp. 1992–1998. IEEE (2009)
13. Saunders, J., Burke, N., Koay, K.L., Dautenhahn, K.: A user friendly robot architecture for re-ablement and co-learning in a sensorised home. In: Proceedings of AAATE 2013. pp. 49–58. IOS Press (2013)
14. Saunders, J., Syrdal, D.S., Koay, K.L., Burke, N., Dautenhahn, K.: "Teach Me–Show Me"—end-user personalization of a smart home and companion robot. IEEE Trans. Human-Machine Systems 46(1), 27–40 (2016)
15. Sierhuis, M., Clancey, W.J.: Modeling and simulating work practice: A method for work systems design. IEEE Intelligent Systems 17(5), 32–41 (2002)
16. Stocker, R., Dennis, L., Dixon, C., Fisher, M.: Verification of Brahms human-robot teamwork. In: Fariñas del Cerro, L., Herzig, A., Mengin, J. (eds.) JELIA 2012. LNCS, vol. 7519, pp. 385–397. Springer, Berlin (2012), doi:10.1007/978-3-642-33353-8_30
17. Webster, M., Dixon, C., Fisher, M., Salem, M., Saunders, J., Koay, K.L., Dautenhahn, K., Saez-Pons, J.: Toward reliable autonomous robotic assistants through formal verification: A case study. IEEE Trans. Human-Machine System 46(2), 186–196 (2016)