Computer Science BSc (Hons) Dissertation

Verification for a Robotic Assistant

Author:
Paul Gainer
Student ID: 200867735

Supervisors:
Dr Clare Dixon
Dr Ullrich Hustadt

Abstract

The Care-O-Bot is an autonomous robotic assistant that is designed to provide companionship for a person living in a typical domestic environment. Currently the industrial development of the Care-O-Bot, and other robotic assistants, is inhibited by the absence of a coherent framework that can ensure their safety. The EPSRC funded project *Trustworthy Robotic Assistants* (TRA) aims to develop robots so that they can engage in advanced interactions with humans in a safe and trustworthy manner; this incorporates the development of new tools and techniques to verify and validate robotic assistants.

The functionality of the Care-O-Bot is determined by a set of control rules which are grouped together to form behaviours, high level rules which determine how the robot acts in its environment. This project is based on work undertaken as part of the TRA project in which model checking techniques were applied to verify a set of Care-O-Bot behaviours.

In previous work models of the robot behaviours, and its environment, were constructed by hand and used as input for the model checkers NuSMVand Spin, software tools that automate the model checking process. This report details the design, realisation and evaluation of an automatic translation from sets of Care-O-Bot control rules into input for the model checker NuSMV.

Initially an extensive analysis of a set of Care-O-Bot control rules is conducted. The results of this analysis are used to design an initial translation from control rules into a succinct intermediate form representation, then a second translation from this intermediate form into NuSMV input. The realisation of the design is discussed and the resulting software is tested and shown to be an effective solution to the problem. Finally, conclusions are drawn that determine a measure of success for the project.

Contents

1 Introduction

Robot assistants are intelligent, autonomous robots that can help with home and work-oriented activities. Of particular interest are personal care robots, whose purpose is to 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 [10].

The EPSRC funded project *Trustworthy Robotic Assistants*[1] (TRA) aims to enhance robot assistants so that they can safely interact with humans. As part of this project, work was undertaken at the University of Liverpool, in collaboration with the University of Hertfordshire, to apply formal verification techniques to the Care-O-Bot, a commercially available robot assistant developed at the Fraunhofer Institute for Manufacturing Engineering and Automation.

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.

1.1 Motivation

Model checking is a technique that given a model of a system and a property, exhaustively checks the property holds throughout every possible run of the system. In previous work models were constructed by hand to specify the behaviours – high level rules which determine how the robot acts in its environment – of the Care-O-Bot and the state of its environment. The Model checking software Spin [8] and NuSMV [2] were used to prove a number of properties in these models. The University of Hertfordshire provided a database populated with control rules for the Care-O-Bot; when combined these control rules form behaviours. The information stored in this database has changed over time, old behaviours have been removed and new behaviours have been added as different aspects of the robot's functionality have been studied. Further work is now needed to apply verification techniques to these new sets of behaviours.

1.2 Aims and Objective

This project aims to facilitate the verification of new sets of robot behaviours by developing an automated translation process that transforms control rules extracted from the database directly into input models for the model checking software NuSMV. The behaviours represented in the database will be first translated into a succinct intermediate form that adequately represents all of the information extracted from the database. This intermediate form will then be used for the final transformation into input for NuSMV, and could also prove useful for further work conducted as part of the TRA project, where different model checkers and verification techniques may be applied. Ideally full automation of the translation process will be realised, however in many instances it may be necessary to prompt the user to disambiguate any conflicting interpretations of the input.

1.3 Challenges

As there is no available formal specification defining the robot behaviours an in-depth analysis of their functionality and interactions is needed to determine if models produced as a result of the transformation process can be considered correct. Additional complications arise from the lack of a defined syntax for the rules that populate the database. Parsing these rules is non-trivial and

[1] www.robosafe.org

a system must be developed which can accommodate the introduction of new syntactic forms for rules.

A final consideration is the complexity of the NuSMV models produced by the transformation process. The state space of these models grows exponentially as new variables are introduced, therefore an appropriate level of abstraction is required to ensure that it is computationally feasible to check properties of the systems.

1.4 Solution

CRutoN, the software tool produced as a result of this work, proves to be an effective solution to the problem. The introduction of new syntactic forms of control rule is accommodated by allowing the user to define simple rules that describe both the syntax and semantics of each new form of rule. Given these additional rules the software can automatically translate sets of control rules into both the intermediate form and direct input for NuSMV, where input from the user is only required should ambiguity arise during the parsing of the control rules. Additional command line options facilitate the regulation of the complexity of the resultant models by allowing the user to select the level of abstraction used to represent the temporal aspects of the robot behaviours.

2 Background

2.1 Problem Background

Temporal logics are formalisms that allow propositions to be represented and reasoned about in terms of time. Given an underlying model describing the different states of a system in time, represented as a Kripke structure [11], temporal logic can be used to reason about whether certain properties hold in different states of the model. Model checking is the process by which this process of checking a property in a state is exhaustively applied to every state in a model.

Model checking has already been applied to the Care-O-Bot in [5] and [18] using a set of control rules provided by the University of Hertfordshire. These rules describe how the robot operates in a test environment called the Robot House, a typical suburban house equipped with furnishings and sensors to provide information on the state of the house and the activities of its occupant. This sensory data is used by the robot to respond accordingly to these external events, for instance if the doorbell rings the robot should notify the occupant of the house.

NuSMV is a popular model checker for temporal logic. A detailed description of NuSMV is provided in Section 9. In [5] a set of control rules defining the behaviours of the Care-O-Bot were translated into NuSMV input by hand and properties were proved using the model checker NuSMV[2]. Similarly in [18] a model was constructed using Brahms [16], an agent-based modelling language, and then translated into Promela, the input language for the model checker Spin [8]. The work in [18] included a more detailed model of the state of the robots environment, where the state of the environment changed according to the daily routine of the occupant of the house.

Since these studies were conducted the available set of control rules has been expanded to include new behaviours that have been developed for the robot. The software produced as a result of this project should allow models of these new sets of control rules to be automatically generated. These models can then be used to prove desirable properties of these new behaviours.

2.2 Research Conducted

The work presented in [5] and [18] was studied to fully understand the problem background and the techniques that had already been applied to formally verify the Care-O-Bot. Additional background research was conducted into temporal logics and model checking to develop a good

understanding of these principles in order to fully understand the problem, and to derive an appropriate solution.

2.2.1 Temporal Logic

Information presented in [3], [6], and [9] was studied extensively to develop an understanding of temporal logics, notably Linear Temporal Logic (LTL) which uses a discrete, linear model of time in which the states of the model correspond directly to the natural numbers. This knowledge of LTL was essential to understand the syntax and semantics of the properties expressed in [5] and [18], and allowed the author to formulate their own temporal properties that could be tested in the models produced as a result of the transformation process.

2.2.2 Model Checking

Model checking techniques detailed in [6] and [9] were studied extensively to develop a good understanding of model checking techniques and applications. As the project aimed to produce input for the model checker NuSMV, and if there was sufficient time for the model checker Spin, research was conducted to learn how to use these software systems and how to construct models in their respective input languages, SMV and Promela. In addition to reading the published literature for these systems [1, 2, 8] a number of other freely available educational resources were studied. Research papers describing translations of systems into NuSMV also helped to illustrate the process by which a system could be modelled using SMV, for instance [17] which describes the translation of Petri nets into NuSMV models. A significant proportion of the research into model checking was conducted to learn more about the model checker Spin, though there was insufficient time to realise an additional translation from Care-O-Bot control rules into Promela.

2.2.3 Semantics of Care-O-Bot Behaviours

As there were no available formal semantics specifying the behaviours of the Care-O-Bot a full analysis of both the set of control rules modelled in [5] and [18] was conducted in order to formulate temporal logic properties that would be expected to hold in any NuSMV model resulting from the translation process. Correspondence with the University of Hertfordshire helped to clarify aspects of the Care-O-Bot's functionality that were not fully understood. The functionality of the control rules as understood by this author is described in detail in Section 10.

2.2.4 Practical Model Checking

Small models were constructed by hand to gain familiarity with the model checking software. Initially, small state transition systems similar to that shown in Figure 15 were constructed by hand then modelled using NuSMV. The model described in [5] was rigorously studied then a small subset of the Care-O-Bot behaviours were selected and modelled by hand. A number of simple LTL specifications were then checked in the resulting models.

2.2.5 Additional Research

A string matching algorithm was devised using [4] to determine the similarity of two given strings, and was used to to automatically regulate inconsistency in the naming of variables and behaviours in the text representing the control rules in the database. Parsing techniques in [7] and [14] were studied, in particular left-recursive descent parsing. Due to an oversight made during the initial design phase many of the techniques described in these sources were not used in the final design.

2.3 Project Requirements

A software tool was expected that could translate sets of Care-O-Bot control rules into an intermediate form and into input for the model checker NuSMV. The intermediate form was required to be a succinct and adequate representation of the control rules. An intermediate form

representation of a set of control rules should then be translatable into input for the model checker NuSMV. Given input generated by the software, NuSMV was expected to generate a model in which temporal properties corresponding to the behaviours of the robot could be checked.

The produced system needed be flexible, allowing the introduction of new syntactic forms of rule. Flags to regulate the level of abstraction of temporal aspects of the Care-O-Bot behaviours and the level of automation of the translation process were also anticipated.

3 Design

3.1 Intermediate Form

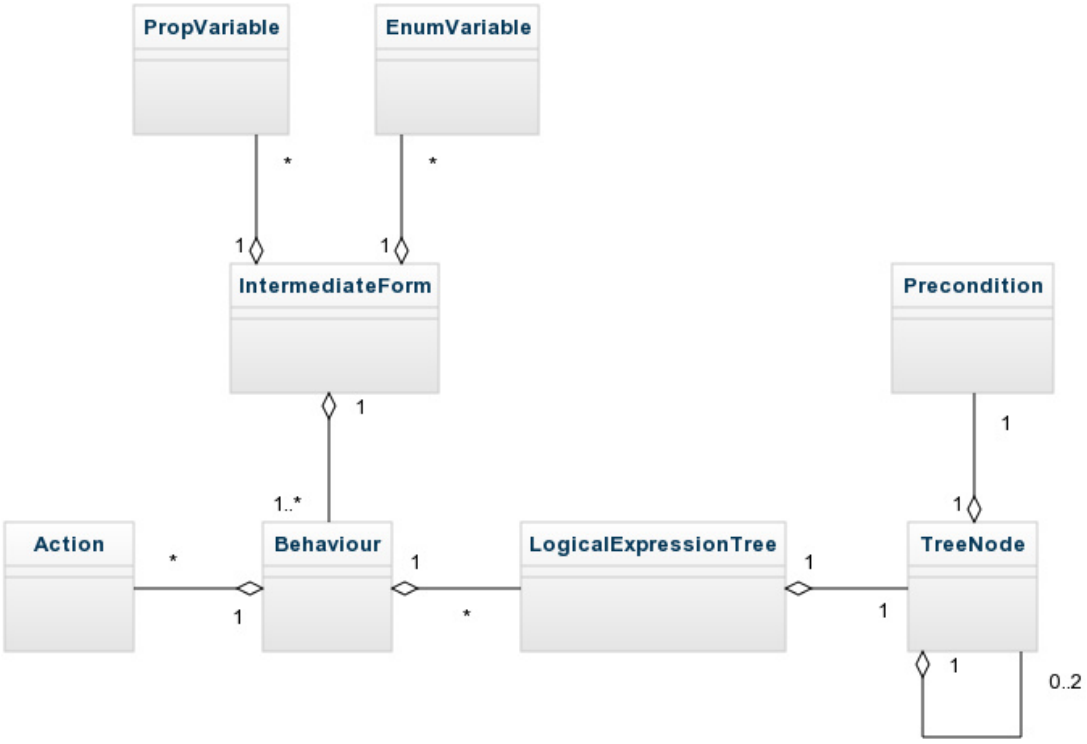
A complete list of Care-O-bot control rules was extracted from the provided database of Care-O-bot control rules. The individual precondition rules and action rules were analysed and distinct features of each type of rule were identified. Rules were then separated into groups where all rules in any group shared the same features. Appropriate data structures were then designed to represent each type of rule. The resulting class diagram is shown in Figure 1. The Intermediate Form consists of a set of propositional variables, a set of enumerated variables, and a set of behaviours. Each behaviour has a priority p with $p \in \mathbb{Z}$, a flag set to true iff the behaviour is interruptible, and a flag set to true iff the behaviour is schedulable. Furthermore, each behaviour consists of a (possibly empty) ordered list of actions and a (possibly empty) logical expression tree, where each internal tree node corresponds to one of the boolean connectives AND, OR, or NOT, each leaf node of the tree stores a reference to a precondition, and an expression evaluationg to true iff the preconditions of the behaviour hold can be constructed by performing an in-order traversal of the tree.

Preconditions Precondition rules are propositional statements that are either *true* or *false*. All precondition rules for a behaviour are linked by Boolean AND and OR operators [5] to form a logical expression that must evaluate to true for the behaviour to be scheduled for execution by the robot. The rules extracted from the control rules database have a variety of syntactic forms. An analysis was conducted to identify features of precondition rules that allowed the rules to be compartmentalized into groups. The identified features of precondition rules extracted from the database are detailed in Section 11.1. The precondition rules were then separated into disjoint groups, where all rules in a group shared the same features. A full listing of these groups is shown in Section 11.2.

Three categories of precondition rules were identified – propositional value checks, enumeration value checks, and time constraints. Propositional value checks and enumeration value checks should be constrainable by *been-in-state* and *was-in-state* conditions. Full details of these conditions, and the categorisation of the precondition rules are given in Section 11.3. A propositional value check evaluates to true iff for some Boolean variable b and some truth value $t \in \{true, false\}$ we have that $b = t$. An enumeration value check evaluates to true iff for some set of named values $\mathcal{V} = \{v_1, \ldots, v_n\}$, for some enumerated variable with a value $v \in \mathcal{V}$, and for some value $v' \in \mathcal{V}$, we have that $v = v'$. A Time constraint evaluates to true iff the current time of day is within a given time interval. These distinct forms of precondition rules will be represented as subclasses of a Precondition superclass.

Action Rules Each Care-O-bot control rule has an ordered list of action rules which are sequentially executed when the precondition rules hold. As stated in Section 3.1, the rules extracted from the database have many different syntactic forms. An analysis of the action rules identified distinct features of action rules which allowed the rules to be divided into groups. The identified

Figure 1: Structure of the Intermediate Form



features of action rules extracted from the database are detailed in Section 11.4. The precondition rules were then separated into disjoint groups, where all rules in a group shared the same features. A full listing of these groups is shown in Section 11.5.

Five categories of action rule were identified – propositional value assignments, enumeration value assignments, behaviour executions, non-deterministic behaviour executions, and delays. Full details of the categorisation of the precondition rules are given in Section 11.6. A propositional value assignment assigns some truth value $t \in \{true, false\}$ to some Boolean variable. For some set of named values $\mathcal{V} = \{v_1, \ldots, v_n\}$, an enumeration value assignment assigns some value $v \in \mathcal{V}$ to some enumerated variable over $\mathcal{V}$. A behaviour execution transfers control from some scheduled behaviour B to some other behaviour B'. If B still has actions to complete then control is returned to B once all actions in B' have been performed. For some set of behaviours $\mathcal{B} = \{B_1, \ldots, B_n\}$, a non-deterministic behaviour execution transfers control from some scheduled behaviour B to some behaviour $B' \in \mathcal{B}$, where B' is selected at random from $\mathcal{B}$. If B still has actions to complete then control is returned to B once all actions in B' have been performed. A delay forces the robot to do nothing for a given number of seconds. These distinct forms of action rules will be represented as subclasses of an Action superclass.

It is worth noting here that all actions represented in the intermediate form, and indeed in the final translation into model checker input, are considered to be atomic. They are either enacted in their entirety or not at all. The semantic implications of the suffix 'and wait for completion', which occurs multiples times in the set of control rules, will therefore be omitted entirely from the

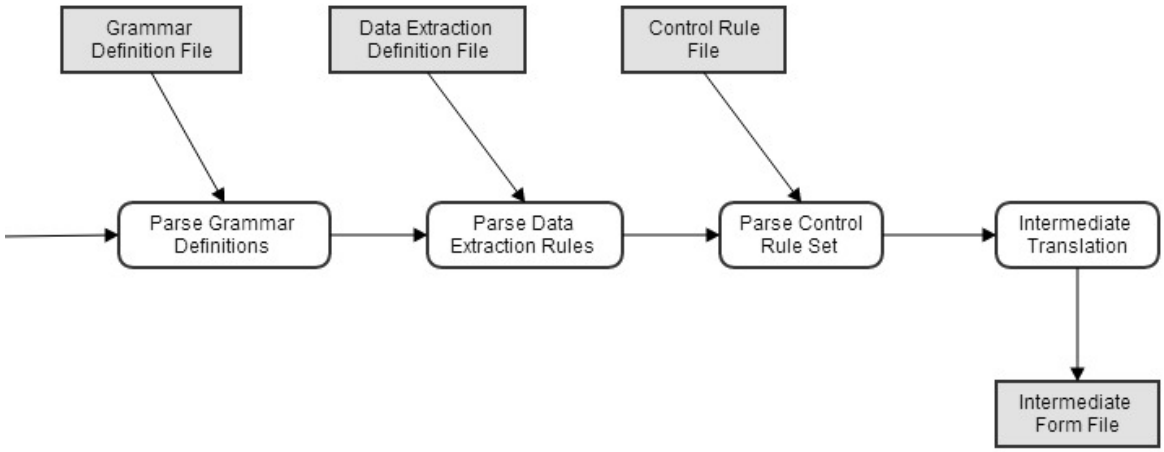
intermediate form.

3.2 Translation into Intermediate Form

As shown in Section 3.1, preconditions and actions have many distinct forms but can be separated into three distinct categories of precondition and five distinct categories of action. To ensure that additional sets of new control rules can be translated correctly the parsing and extraction of information from these rules is not hard coded into the resulting software. The translator takes as input not only a set of control rules but also a file containing a set of grammar rules (rules defining the syntax of control rules), and a file containing a set of data extraction rules (rules defining how information should be extracted from the control rules).

For a new form of control rule to be recognised and parsed correctly new definitions must be added to both the grammar rule file and the data extraction rule file. The grammar rule definition for a new control rule allows an automaton to be constructed at run time that can be used to parse this new form of rule. The data extraction rule definition for the new control rule describes how meaningful information can be extracted from the text parsed by constructed automaton. Figure 2 illustrates the process by which a set of control rules is translated into the intermediate form.

Figure 2: The Intermediate Form Translation Process



3.2.1 Grammar Rules

Grammar rules allow the user of the system to define the syntax of control rules. When a new syntactic form of control rule is encountered a new definition can be added which will allow the system to recognise parsed control rules of this new form.

Grammar Rule Syntax The syntax of grammar rules is a simple variation on the Backus-Naur Form notation *<symbol> ::= expression*, where *symbol* is the name of a non-terminal symbol, and *expression* is a non-empty sequence of lexical elements, namely terminal symbols and non-terminal symbols. Terminal symbols are of the form *'token'*, where *token* corresponds to an actual token of text expected in the control rule being parsed. For each symbol a deterministic finite automaton [7] is constructed that accepts as input tokens of text. The automaton is deemed to be *accepting* if after inputting some number of tokens the automaton is in an accepting state.

Grammar Rule Names For every distinct form of control rule there is an associated grammar rule. The name of each grammar rule must consist of a predefined prefix determining the category of precondition or action rule parsed by this grammar rule, suffixed with an integer. The predefined prefixes for preconditions are *pvc*, *evc* and *tc*, respectively corresponding to precondition value checks, enumeration value checks and time constraints. The predefined prefixes for actions are *pva*, *eva*, *exb*, *exbnd* and *del*, respectively corresponding to propositional value assignments, enumerated value assignments, behaviour executions, non-deterministic behaviour executions and delays. No two grammar rules corresponding to control rules in the same category may have the same identifying integer. The grammar rule definitions for the set of control rules used during the design stage can be found in Section 11.8.

Additional Symbols Additional non-terminal symbols can be defined to avoid multiple definitions in the grammar. For example, the non-terminal symbol *<ott> ::= 'one' 'two' 'three'* could be defined and then used in further definitions whenever the tokens *one*, *two* then *three* would be expected as sequential tokens in the input. Several built-in non-terminal symbols are included to increase the expressiveness of the grammar rules. Full details of these built-in symbols are given in Section 11.7.

Prefixes Prefixing any additional non-terminal symbol, built-in non-terminal symbol or terminal symbol with + indicates valid input for the constructed automaton may be repeated one or more times.Prefixing any additional non-terminal symbol, built_in non-terminal symbol or terminal symbol with a positive integer n indicates that valid input for the constructed automaton should be repeated exactly n times. For example, given the grammar definitions *<a> ::= 2'one'* and *<b> ::= +<a> 'two'* the symbol b would be in an accepting state if accepted as input any of the sequences *one one two*, *one one one one two* etc.

Limitations Symbols prefixed with + cannot be followed by any symbol that can accept more than one token as the parser is designed to have a look ahead of at most one token. In addition, any grammar rule defining a dynamically built automaton cannot be recursive, either directly or indirectly.

3.2.2 Data Extraction Rules

Figure 3 illustrates how several control rules, each having a distinct lexical composition, can all belong to a single category of control rule. As shown in Section 3.2.1, grammar rules can be defined for each distinct form of control rule in a category. Each grammar rule should have a corresponding data extraction rule. Data extraction rules describe how relevant information can be retrieved from each form of control rule after they have been parsed, for instance a data extraction rule for a propositional value check would include an indication of which of the (enumerated) symbols in the corresponding grammar rule definition is to be used to determine the name of the propositional variable.

Figure 3: Different Forms of Enumeration Value Assignment

1. `Turn light on ::0::Care-O-Bot 3.2 to white`
2. `::0::Care-O-Bot 3.2 says 'The fridge door is open!'`
3. `move ::0::Care-O-Bot 3.2 to ::31::  TV location in the Living Room`
4. `move tray on ::0::Care-O-Bot 3.2 to Lowered`
5. `move torso on ::0::Care-O-Bot 3.2 to the right`

There are eight distinct formats of extraction rule, one for each of the categories of precondition rules and action rules. Each format consists of a *rule_name*, the name of the corresponding non-terminal symbol in the set of grammar rules, then a number of variables to which different values are assigned. The types of value that can be assigned to variables are *boolean*, which can be true or false, *string*, which consists of any characters enclosed in quotation marks (*"text"*, for example), and finally *identifier*. Identifiers are either a *string* value, or a list of integers in the format $[1, 5, 2 \ldots]$ where each integer i in the list corresponds to any tokens accepted as input for the i^{th} symbol on the right hand side of the corresponding grammar rule definition. If multiple integers, say i, j and k are in the list then the parsed text for the i^{th}, j^{th} and k^{th} symbols would be concatenated with underscores to name a variable. Any variable type can be set to null as long as this is permitted in the definition of the rule. An example is shown below:

Given the grammar rule shown below, where each right hand symbol is enumerated:

`<tc2> ::= 'Time'`$_1$ `'is'`$_2$ `'between'`$_3$ `<time>`$_4$ `'and'`$_5$ `<time>`$_6$

We have the corresponding data extraction rule:

`tc2; start_time = [4]; end_time = [6];`

Which has the format:

rule_name; start_time = *identifier*; end_time = *identifier*.

The above data extraction rule tells the parser that when parsing the time constraint grammar rule *tc2*, the fourth symbol <*time*> should be used as the value for *start_time*, and the sixth symbol <*time*> should be used as the value for *end_time*.

A comprehensive list of the eight different formats of data extraction rule, and examples for each using rule definitions for the set of control rules used during the design stage, can be found in Section 11.9.

3.2.3 Parser Design

Input Files Input will consist of three separate files. The first file to be parsed will consist of the grammar rules for each of the categories of precondition and action. If a non terminal symbol appears in the right hand side of a grammar rule then it must have previously been defined in the file, or must correspond to a 'built in' symbol (see Section 11.7). The second input file should contain the set of data extraction rules, with one rule present for each defined grammar rule. The third input file will consist of tuples of data values extracted from the database of control rules,

one tuple per line. Each tuple has values corresponding to a singe control rule and consists of the following:

- the name of the behaviour to which the rule belongs
- an integer $p \in \mathbb{N}$ corresponding to the priority of the behaviour
- an integer $i \in \{0, 1\}$ where $i = 1$ if the behaviour can be interrupted
- an integer $s \in \{0, 1\}$ where $s = 1$ if the behaviour can be scheduled
- an integer determining the order in the which the rule appears in the behaviour
- a character $c \in \{R, A\}$, with $c = R$ if the control rule is a precondition rule or $c = A$ if the control rule is an action rule
- an integer $a \in \{0, 1, 2\}$, where if this rule is a precondition rule then if $a = 2$ a disjunction should be formed between this rule any any subsequent precondition rule in the behaviour, or if $a = 0$ or $a = 1$ a conjunction should be formed between this rule and any subsequent precondition rule in the behaviour
- an integer $n \in \{0, 1\}$ where if this rule is a precondition rule and $a = 1$ this rule should be negated
- some text defining the control rule itself

Data Types An abstract superclass Automaton is defined, having as fields a name, an integer giving the number of times to repeat (set to 0 for symbols prefixed with +, or set to the number of times to repeat otherwise), and procedures to check if the automaton is in an accepting state, to check if the automaton is in a dead state, to feed a token to the automaton, to get the input the automaton received to reach an accepting state, and finally to reset the automaton back to its initial state.

Two subclasses of Automaton are then defined, TerminalSymbol and NonTerminalSymbol. The name of a TerminalSymbol corresponds to the input needed to move the automaton to an accepting state. Each non-terminal symbol has a list of Automatons corresponding to the terminal and non-terminal symbols on the right-hand side of the rule.

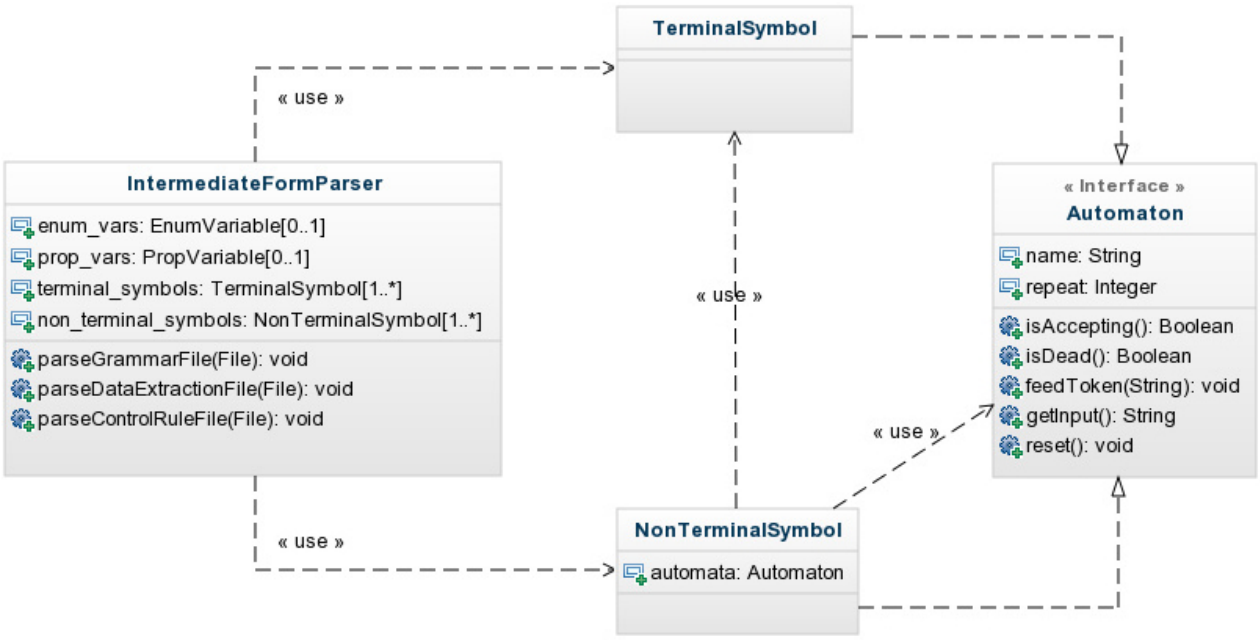
A class diagram illustrating the key fields and methods of these data structures and the parser itself is shown in Figure 4. Pseudocode for key parsing methods is included in Section 11.13.

In the original design an addition class AutomataChain was also defined. This class was included to allow the definition of grammar rules of the form *<symbol> ::= expression1 | expression2 | expression3*, where there is a choice of expressions for the symbol. It was decided to not allow multiple expressions to appear on the right hand side of a rule to simplify the parsing process, therefore the class AutomataChain is no longer factored into the design of the parser.

3.3 Translation into NuSMV Input

This section describes the process by which an intermediate form representation of a set of control rules (see Section 3.1) is translated directly into model checker input. The translation process produces as output a file defining a model that can be used as direct input for the model checker NuSMV. Each produced model consists of two modules, the required module `main` and an additional module `behaviour`.

Figure 4: Intermediate Form Parser Class Diagram



3.3.1 Main Module Variable Declarations

The VAR section of the main module contains a Boolean variable declaration for each propositional variable in the intermediate form, and an enumerated variable declaration for each enumerated variable in the intermediate form. Figure 5 shows the definitions that are added for a propositional variable *::501::TrayIsLowered* and an enumerated variable *light* with values *blue*, *yellow* and *red*. All characters not part of the input language for NuSMV are replaced by underscores, as shown on line 1.

Figure 5: Variable Declarations

```
1. __501__TrayIsLowered:  boolean;
2. light:  {blue, yellow, red};
```

Declarations for three other enumerated variables are included in every model. The first of these variables is *step*; this variable is used to record which action is being executed for any scheduled behaviour. The second variables is *schedule*; this variable records which behaviour is currently scheduled for execution. If a scheduled behaviour was executed by another behaviour then the *last_schedule* variable is used to 'remember' the executing behaviour.

An additional variable *time* is included in the model if there at least one behaviour in the given intermediate form that has a time constraint as a precondition. This variable is used to represent the current time of day. This variable was not discussed in the initial design as research was still being conducted as to how to represent the passing of time in the NuSMV models.

Given a set of intermediate form behaviours $\mathcal{B}$ the possible values for the variables are determined as follows:

- **step**

  The *step* variable has a value *step_none* in addition to another k distinct values *step_1*, ..., *step_k*, where $k = \max\{\text{numberOfActions}(b) \mid b \in \mathcal{B}\}$.

- **schedule**

  The *schedule* variable has a value *schedule_none* in addition to another $|\mathcal{B}|$ distinct values, where for each behaviour $B \in \mathcal{B}$ there is a corresponding value *schedule_* suffixed with the name of B.

- **last_schedule**

  The set of values for the *last_schedule* variable is equal to the set of values for the *schedule* variable.

- **time**

  Let $\mathcal{TC}$ be the set of all time constraint precondition rules in all behaviours in $\mathcal{B}$, and for each $tc \in \mathcal{TC}$ define a function f that maps each $tc \in \mathcal{TC}$ to a time interval $[tc_s, tc_e]$ where tc_s is the first moment in time in which tc holds, tc_e is the last moment in time in which tc holds, tc_s and tc_e correspond to some time of day in the 24 hour format, and tc_s corresponds to a time strictly earlier than tc_e. Given a time interval $[tc_s, tc_e]$ define $s([tc_s, tc_e])$ to be tc_s, $e([tc_s, tc_e])$ to be tc_e, and $\mathcal{S}([t_s, t_e])$ to be the set of time constraints such that $\mathcal{S}([t_s, t_e]) = \{tc \mid tc \in \mathcal{TC}$ **and** ($(s(f(tc)) \leq t_s$ **and** $e(f(tc)) \geq t_s)$ **or** $(s(f(tc)) \leq t_e$ **and** $e(f(tc)) \geq t_e)$)$\}$

  The values for *time* are constructed by first finding a set of time intervals $\mathcal{I} = \{[i_s^1, i_e^1], \ldots, [i_s^n, i_e^n]\}$, such that for every $[i_s^j, i_e^j] \in \mathcal{I}$ there is no $[i_s^k, i_e^k] \in \mathcal{I}$ where $1 \leq j \leq n$, $1 \leq k \leq n$ and $j \neq k$ such that $S([i_s^j, i_e^j]) = S([i_s^k, i_e^k]) \neq \emptyset$, and for every distinct time of day t' there is some $[t_s, t_e] \in \mathcal{I}$ with $t_s \leq t' \leq t_e$. Then for every interval $[i_s, i_e] \in \mathcal{I}$ where $S([i_s, i_e]) \neq \emptyset$ a value for *time* is added with the name *_i_s_to_i_e*. If there is some $i \in \mathcal{I}$ with $S(i) = \emptyset$ then the value *no_time_constraints_hold* is added.

3.3.2 Main Module Variable Assignments

The `ASSIGN` section of the main module has two statements for each propositional variable and enumerated variable in the intermediate form. The first statement assigns an initial value to the variable in the model, and the second statement is a sequence of expressions that determine the value that the variable will take in the next state.

The initial value for variables is set non-deterministically to one of the possible values for that variable unless an explicit initial value is given for the variable.

If a variable is flagged as non-deterministic (i.e. the variable represents some non-deterministic aspect of the Care-O-Bot's environment, for details of this see Section 11.9) then in every state the value of this variable in the next state is chosen non-deterministically.

The value of deterministic variables in the next state is determined by a sequence of expressions. Every expression in the sequence corresponds to some *propositional value assignment* action if the variable is a boolean, or some *enumeration value assignment* action if the variable is an enumeration, that assigns a value to the variable. Given a set of intermediate form behaviours $\mathcal{B}$ and some intermediate form variable *var*, for every assignment of some value to *var* in a behaviour $b \in \mathcal{B}$ an expression is added of the form:

`(schedule = schedule_N & step = step_n): v;`

where `N` is the name of b, n is the index of the assignment action in the sequence of actions for b, and v is the value to assign to *var*. Intuitively this expression means that if behaviour b is scheduled and is performing its n^{th} action, then in the next step the value of *var* will be equal to v. A final expression is added of the form:

`TRUE:` v.

As this expression is the last in the sequence it will only be evaluated should all other expressions evaluate to false. This expression ensures that if no new value has been assigned to *var* then it keeps its value in the next state.

Figure 6 shows the statements added to the model for two variables *v1* and *v2*. *v1* is a non-deterministic boolean variable, *v2* is a deterministic enumerated variable with possible values *value_1*, *value_2* and *value_3*, and *b1* is a behaviour which has as its 3^{rd} action an *enumerated value assignment* that assigns the value *value_1* to *v2*.

Figure 6: Variable Assignments

```
 1. init(v1):= {TRUE, FALSE};
 2. next(v1):= {TRUE, FALSE};
 3.
 4. init(v2):= {value_1, value_2, value_3};
 5. next(v2):=
 6.     case
 7.         (schedule = schedule_b1 & step = step_3):  value_1;
 8.
 9.         TRUE: v2;
10.     esac;
```

The following bullet points summarise how values are assigned to the variables *step*, *schedule*, *last_schedule* and *time*. A number of macro definitions are used that are described in detail in later sections although their intuitive meaning is given here. Detailed descriptions of how these variables are assigned values in the next state are shown in Section 11.10.

- **step**

  The initial value for this variable is *step_none* as no behaviour is performinag an action in the initial state. When a behaviour is scheduled *step* starts with the value *step_1*, and then increments to *step_2* and so forth as consecutive actions are performed.

- **schedule**

  The initial value for this variable is *schedule_none* as no behaviour is scheduled in the initial state. If a behaviour can be scheduled in the next moment in time as either no behaviour is scheduled or the current behaviour can be interrupted, then in the next moment in time *schedule* will have a value corresponding to this newly scheduled behaviour.

- **last_schedule**

  The initial value for this variable is *schedule_none* as no behaviour has been scheduled in the initial state. The value of this variable will always correspond to the value of the *schedule* variable in the previous state unless a behaviour has been executed by another behaviour, in which case the value keeps the value corresponding to the executing behaviour, allowing the executing behaviour to be 'remembered' so that control can be returned to this behaviour once the executed behaviour has finished performing its actions.

- **time**

  The time variable is assigned a random value initially and in the next moment in time.

3.3.3 Main Module Macro Definitions

The `DEFINE` section of the main module defines a number of macros that are used to succinctly express more complex logical expressions, increasing the readability of the code, decreasing the size of the code, and allowing properties checked in the model to be expressive yet easy to formulate. Brief descriptions of these macros are given here an full descriptions of how these macros are constructed are included in Section 11.11.

Preconditions For each behaviour in the intermediate form a macro is defined that evaluates to true iff the preconditions the behaviour hold.

Interrupts Each behaviour has a priority value. For every distinct priority value present in the intermediate form a macro is defined that evaluates to true iff a behaviour having that priority can interrupt a currently scheduled behaviour in the next moment in time.

Behaviour Scheduling Macros Six additional macros that relate to the scheduling of behaviours are defined here. The macros *an_executed_behaviour_is_ending_as_a_last_action*, *executed_behaviour_execute_next*, and *an_executed_behaviour_is_schedule* were not described in the original design.

- The *executed_behaviour_execute_next* macro evaluates to true iff in the next moment in time some behaviour will be executed by the currently scheduled behaviour.

- The *a_behaviour_can_be_scheduled* macro evaluates to true iff in the next moment in time some behaviour can be scheduled for execution.

- The *a_behaviour_is_ending* macro evaluates to true if some schedulable behaviour is performing its final action.

- The *an_executed_behaviour_is_ending_as_a_last_action* macro evaluates to true iff some some behaviour executed by another behaviour is performing its last action, and this executed behaviour was executed as the executing behaviours final action.

- The *an_executed_behaviour_is_ending* macro evaluates to true iff some non-schedulable behaviour is performing its final action.

- The *an_executed_behaviour_is_scheduled* macro evaluates to true iff some non-schedulable behaviour is currently being executed.

3.3.4 The Behaviour Module

A brief description of the behaviour module is given here, and a more detailed decription is given in Section 11.12. The parameterizable behaviour module, shown in Figure 7, instantiated once for each behaviour in the set of intermediate form behaviours, consists of a number of macro definitions. This purpose of this module is simply to provide macro definitions for individual behaviours. These macro definitions can then be used to easily construct otherwise complex expressions that are used both to define the state of variables in the `ASSIGN` section of the main module, and to formulate properties to test in the model.

Each behaviour accepts as parameters an expression that is true iff the preconditions of the behaviour hold, an expression that is true iff the behaviour can interrupt a currently scheduled behaviour, an expression that is true iff the behaviour can be interrupted in the next moment in time, references to the *schedule* and *step* variables, and the values for *schedule* and *step* corresponding to the behaviour and its final step respectively. Given these values the behaviour instance then provides macro definitions that be used to determine if the preconditions of the behaviour hold, the behaviour can be scheduled or interrupted, and if the behaviour is scheduled or executing its last step.

Figure 7: The Behaviour Module

```
1. MODULE behaviour(preconditions, can_interrupt, can_be_int, schedule,
   this_schedule, step, last_step)
2.
3.    DEFINE
4.       preconditions_hold:= preconditions;
5.       can_be_scheduled:= ((schedule = schedule_none | can_interrupt)
   & preconditions_hold);
6.       can_be_interrupted:= can_be_int;
7.       is_last_step:= (is_scheduled & step = last_step);
8.       is_scheduled:= (schedule = this_schedule);
```

3.3.5 Temporal Constraints

As described in Sections 10.3 and 3.1, *propositional value check* rules and *enumeration value check* rules can be additionally constrained by *been-in-state* and *was-in-state* conditions. These additional conditions respectively require that the the precondition has held throughout a previous period of time or has held at some point within a previous period of time.

Command line parameters are provided to allow the level of abstraction of temporal aspects of the control rules to be regulated to some extent. If the user has chosen a high level of abstraction then it is likely that many *been-in-state* and *was-in-state* conditions requiring a precondition to have held during or within a small period of time will not be represented in the produced code. For lower levels of abstraction additional lines of code are added to the model to correctly model the semantics of precondition rules constrained by *been-in-state* or *was-in-state* conditions.

3.4 Test Design

At the design stage it was stated that standard testing techniques would be applied to all classes and methods comprising unit testing, integration testing, component interface testing and

system testing. A set of grammar rules and data extraction rules corresponding to the set of control rules used in [5] and [18] would be constructed and the software would be used to produce an intermediate form representation of these rules. This intermediate form would then be checked by hand to ensure that it adequately represents the control rules. NuSMV input would then be generated using the intermediate form.

In [5] a sample of formal verification requirements were translated and their formalised properties were verified using NuSMV. These Linear Temporal Logic specifications were to be applied to models generated from model checker input produced by the automatic translation process and the results of the specifications were to be compared to those found in [5]. In addition to the aforementioned specifications additional desirable properties were to be tested. Many of these tests would correspond directly to the properties that are expected to hold in produced models, as discussed in Section 10.

3.5 Evaluation Design

The functionality of the produced software was to be compared to the expected functionality declared in the project requirements, and any extra features included in the software, and the reasons that they were implemented, were to be discussed.

In the original design it was stated that ideally full automation of the translation process would be realised however it may be necessary to prompt the user to disambiguate any conflicting interpretations of parsed rules, therefore an analysis of the level of automation provided by the software was to be conducted.

Model checker code produced by the translation from the intermediate form into model checker input would be compared to the hand-written code produced in [5] and comparisons were to be made to determine the efficiency of the automated translation process. This was to include a comparison between the size of the models produced using NuSMV input generated by the software and the size of the model produced by the hand written code produced in [5].

Finally, the translation of an additional set of control rules to those used in the design of the system was to be conducted to evaluate the flexibility of the software. If the new set of rules could not be correctly translated by the software then an analysis of the extent of modification required to correctly translate the additional rule set would be conducted.

4 Realisation

Standard testing techniques were applied to all classes and methods throughout the implementation of the software to identify and remove any errors in the code. This comprised unit testing, integration testing and component interface testing. Only the most important test results will be discussed later in this section.

4.1 Intermediate Form Implementation

The first step in the implementation of the design was to build the data structures constituting the intermediate form representation of a set of behaviours. Nearly all classes defined in the original intermediate form design were simple structures and their implementation was a straightforward process.

It was important that the produced software should be able to recognize discrepancies in the names of behaviours and variables in the input as there were some spelling errors present in the control rules extracted from the database. Command line switches were provided to guide the automatic identification of these spelling errors. Enabling automatic identifier matching tells the parser to try and automatically match parsed identifiers for behaviours or variables with those of behaviours and variables that have already been parsed, selecting the best match where multiple matches are found. An identifier is considered to match another existing identifier if the similarity of these two identifiers exceeds a given threshold. Enabling automatic case-insensitive parsing tells the parser to ignore letter case when matching the names of identifiers for behaviours or variables with those of identifiers that have already been parsed. These command line switches are shown in Figure 8, *-aim* toggles automatic identifier matching, *-smt* sets the string matching threshold (a required percentage of similarity), and *-cim* toggles automatic case-insensitive matching.

Section 13.1 lists the generic *getByName* procedure in the IntermediateForm class which, given the identifier of a behaviour or variable and a list of existing behaviour or variables, returns any corresponding behaviour or variable having that identifier or a null value if no match was found. If case-insensitive identifier matching is disabled then the user is prompted to confirm that two identifiers differing only by case are the same. If automatic identifier matching is disabled then a list of matches whose similarity exceeds a given threshold are displayed to the user who can then choose which match to consider and whether the existing identifier should be replaced, the existing identifier should be used instead of the parsed identifier, or to ignore the match and consider the parsed identifier as a new value.

4.2 Translation into Intermediate Form

As the translation process involved the parsing of three separate input files a collection of small helper functions was composed which facilitate the tokenizing and manipulation of the text to be extracted from these files.

4.2.1 Parsing Grammar Rules

As described in Section 3.2.1 the grammar rule file allows the user to define the syntactic forms of control rules using a simplfied and restricted Backus-Naur form notation. Parsing these grammar files was one of the most challenging aspects of the implementation. Despite its limitations the defined grammar was still very expressive and a lot of time was spent ensuring that the generated automata functioned correctly. The original design was reworked to simplify the construction of these automata, by disallowing syntax rules of the form $<symbol> ::= expression_1 \mid \ldots \mid expression_i$, where multiple expressions could appear on the right hand side of a rule. This change to the design did not affect the expressiveness of the grammar as separate rules could be defined

Figure 8: Command Line Parameters

```
CRuToN 1.0.0
usage: CRuToN -ic <file> -id <file> -ig <file> [options]
   required:
        -ic <file>       control rule file
        -id <file>       data extraction rule file
        -ig <file>       grammar rule file
   options:
        -ii <file>       initial values file
        -oi <file>       intermediate form output file
        -on <file>       NuSMV output file
        -tod <HH:MM:SS>  set time of day in the robot house
        -sms <1-oo>      set max seconds for was_in_state/been_in_state
                         default: 5000
        -smt <50-100>    set string matching threshhold
                         default: 90
        -sps <1-oo>      set seconds per state
                         default: 600
        -aim <0|1>       enable automatic identifier matching
                         default: enabled
        -cim <0|1>       enable automatic case-insensitive matching
                         default: enabled
        -dtv <0|1>       enable disallowing truth values as enum values
                         default: enabled
        -f <0|1>         enable behaviour flattening
                         default: enabled
        -mos <0|1>       enable min of 1 state for was_in_state/been_in_state
                         default: disabled
        -tnd <0|1>       enable true non-determinism in scheduling behaviours
                         default: enabled
```

for each separate expression, and allowed more time to be spent on other critical stages of the project such as the translation into NuSMV input.

Section 13.2 lists the *parseGrammarFile* procedure that parses a set of grammar rules defined in the file having the given name. For each rule in the file a new NonTerminalSymbol is constructed by constructing a list of automata for each of the symbols appearing on the right hand side of the rule. If an error occurs during the parsing of these rules, for instance if a rule is defined twice, or if the rule itself does not conform to the syntax, then an appropriate error message is constructed and then thrown to the calling procedure.

4.2.2 Parsing Data Extraction Rules

Each parsed data extraction rule corresponds directly to a grammar rule of one of the eight types described in Section 3.2.1 and is validated to ensure that all expected parameters are present and that the values given for these parameters are of the correct format. There should be one data extraction rule defined for every grammar rule defined in the grammar rule file.

During this stage of the implementation it was noticed that a number of additional expected parameters were needed in several data extraction rules to adequately represent the control rules. Fortunately these oversights in the original design were noticed at any early stage where necessary changes to the existing code base were relatively minor.

4.2.3 Parsing Control Rules

The control rule file consists of a set of tuples extracted from the database, where each tuple is a set of values for a single control rule. Each value is parsed in the order in which they are declared in Section 3.2.3. Adhering to the original design, the process of parsing precondition

rules was implemented using recursive descent parsing techniques, as it was initially believed that a precondition rule could consists of several preconditions linked by boolean AND and OR operators. Upon further examination of the database of rules it was noticed that two fields had been overlooked. With each precondition rule was associated two values, *andOrConnector* and *notConnector*. These fields respectively determined how each precondition rule was to be logically linked with the subsequent precondition rule, forming either a conjunction or disjunction, and whether a precondition should be negated.

This oversight in the original research resulted in a parsing solution that was both incorrect and much more complex than required. The parsing process was reimplemented resulting in a much simpler solution than before. In addition to the time spent reimplementing the parsing process, the process by which logical expression trees were constructed also needed to be reimplemented. Previously one logical expression tree was associated with each individual precondition rule, however in the amended design one logical expression tree was associated with each individual behaviour.

Each control rule is parsed by tokenizing the text describing the rule and using each token as input for each of the automata generated by either the precondition rule grammar definitions or the action rule grammar definitions, according to the type of control rule being parsed. Once all tokens in the text have been given as input to the corresponding precondition or action automata, those automata which are in an accepting state are considered to be candidates to match this parsed rule to some defined form of control rule. These candidates are then validated by checking that the parsed data corresponds to the input expected by the data extraction rule corresponding to the grammar rule used to generate the accepting automaton. If a single candidate remains after the rules have been validated then the parsed rule is automatically matched to the corresponding form of rule, otherwise the user is prompted to disambiguate the input.

4.2.4 Intermediate Form Translation Testing

A number of small handwritten control rules were used to test the process of translation into the intermediate form during the implementation. In Section 12.1 the translation of the set of control rules used in [5] and [18] is shown. First the three input files are listed followed by the string representation of the generated intermediate form. This intermediate form output was checked by hand to ensure that all information represented in the control rules was present.

4.3 Translation into NUSMV Input

4.3.1 Behaviour Lists

Given an intermediate form representation of a set of control rules the first step in the translation into NuSMV input is to build three lists of behaviours. The *buildBehaviourLists* procedure listed in Section 13.3 builds a list of schedulable behaviours, a list of behaviours that execute another behaviour, and a list of behaviours that are executed by another behaviour. These lists facilitate the construction of many of the macros described in Section 3.3.

4.3.2 Time Constraints

Once the lists of behaviours have been constructed a list of possible values for the *time* variable are built. The *time* variable is used to represent the current time of day. The *buildTimingConstraintMap* procedure listed in Section 13.4 builds the values for *time* and also constructs a map in which *time constraint* precondition rules are mapped to the periods of time (possible values for *time*) during which the precondition rule holds. Initially all *time constraint* rules are identified in the set of behaviours. If there is only a single rule then *time* simply has two values, a value corresponding to the period during which this rule holds, and another corresponding to all

other times when the rule does not hold. If there are multiple *time constraint* rules in the set of behaviours then the values for time are constructed by the process described in Section 3.3.1. The map constructed by this procedure is used later to construct the macro definitions for the preconditions of behaviours having a *time constraint* precondition rule.

The command line option *-tod* is provided (see Figure 8) to fix the time of day. If this option is used then the *time* variable is omitted entirely from the output and instead expressions corresponding to *time constraint* precondition rules are set directly to either `TRUE` or `FALSE` according to whether the precondition holds at the fixed time of day.

4.3.3 Main Module and Behaviour Module

The `VAR`, `ASSIGN` and `DEFINE` sections of the *main* module are sequentially constructed exactly as described in Section 3.3 where a definition and set of assignments are constructed for every variable in the intermediate form. Then definitions and assignments are constructed for the *step*, *schedule*, *last_schedule* and *time* variables. Finally the *behaviour* module is added and instances of this module are added to the *main* module, one for each of the behaviours in the intermediate form input.

4.3.4 Been-in-State and Was-In-State Conditions

The original design gave some initial ideas of how *been-in-state* and *was-in-state* conditions applied to precondition rules would be realised, and these ideas have now been expanded and fully implemented. Figure 9 demonstrates how code added to the NuSMV file for a *been-in-state* constrained precondition rule. The example first gives the precondition rules for the *S1-alertFridgeDoor* behaviour. Line 1 of the NuSMV code shows the introduction of a new variable which is added to the `VAR` section of the main module. The variable name is formed by prepending the name of the variable whose value is being checked (*Fridge_Freezer_In__ON_*) to the name of the value being checked (*TRUE*), then adding the suffix *BEEN_IN_STATE*.

Define S to be the number of seconds during which the precondition rule must have held. Depending on the level of abstraction used to represent timing constraints, each state in the model is considered to correspond to a fixed number of seconds s. The *been-in-state* variable therefore has values of the form $s_0, s_n, s_{2n}, \ldots, s_{S-n}, s_{final}$, where $n = \frac{S}{s}$. The initial value for the variable is always s_0. In any state if the precondition does not hold the variable resets to s_0, otherwise if the variable has the value s_i then in the next state the variable will have the value s_{i+n} until the variable has the value s_{final} .

The macro definition of the expression that is true iff the precondition rules of behaviour *S1-alertFridgeDoor* hold is shown on line 13 in Figure 9, where an additional expression checks that the *Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE* variable has the value s_{final} i.e. this precondition rule has remained true within the past 30 seconds.

Figure 9: *been-in-state* Example

S1-alertFridgeDoor precondition rules

1. Fridge Freezer In *ON* AND has been in this state for more than 30 seconds
2. ::514::GOAL-fridgeUserAlerted is false

NuSMV code

```
1. Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE: {s_0, s_final};
2.
3. init(Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE):= s_0;
4. next(Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE):=
5.     case
6.         Fridge_Freezer_In__ON_= FALSE: s_0;
7.         (Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE = s_0 &
   Fridge_Freezer_In__ON_= TRUE): s_final;
8.         Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE = s_final:  s_final;
9.
10.        TRUE: Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE;
11.    esac:
12.
13. pre_S1_alertFridgeDoor:= (!__514__GOAL_fridgeUserAlerted &
    Fridge_Freezer_In__ON_& Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE
    = s_final);
```

Figure 9 demonstrates how code added to the NuSMV file for a *was-in-state* constrained precondition rule. The example first gives the precondition rules for the *checkbell* behaviour. Line 1 of the NuSMV code shows the introduction of a new variable which is added to the VAR section of the main module. The variable name is formed by prepending the name of the variable whose value is being checked (*Fridge_Freezer_In__ON_*) to the name of the value being checked (*TRUE*), then adding the suffix *WAS_IN_STATE*.

Values for the *was-in-state* variable are constructed similarly to those for *been-in-state* variables and the initial value s_0 remains the same. The values for *was-in-state* variables in the next state differ from those of *been-in-state* variables as follows: In any state where the variable has the value s_i, if the precondition holds then in the next state the variable will have the value s_{i+n}. If the variable has the value s_{final} then in the next state the variable will reset to the value s_0.

The macro definition of the expression that is true iff the precondition rules of behaviour *S1-checkbell* hold is shown on line 13 in Figure 9, where an additional expression checks that the *Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE* variable does not have the value s_0 i.e. this precondition rule has been true at some point within the last 10 seconds.

Figure 10: *was-in-state* Example

checkbell precondition rules

1. Doorbell Last Wattage > 1 AND was in this state within the last 10 seconds

NuSMV code

```
1. Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE: {s_0, s_final};
2.
3. init(Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE):= s_0;
4. next(Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE):=
5.     case
6.         Doorbell_Last_Wattage___1 = TRUE: s_final;
7.         Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE = s_final:  s_0;
8.
9.         TRUE: Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE;
10.    esac:
11.
12. pre_checkBell:= Doorbell_Last_Wattage___1 &
    Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE != s_0;
```

In both previous examples the number of seconds during which the precondition must have either remained true, or have evaluated to true at least once, is quite small, resulting in a small number of possible values for the *been-in-state* and *was-in-state* variables. The fixed number of seconds corresponding to a single state in the model can be set using the *-sps* command line option (see Figure 8) and an upper bound on the number of seconds during which a precondition must have held or held at least once can be given using the *-sms* command line option. If the *-mos* command line switch is enabled then at least once previous state must be considered for *been-in-state* and *was-in-state* constraints, otherwise a constraint of s seconds, where s is less than half the number of seconds per state, will be ignored.

4.3.5 Non-Deterministic Behaviour Scheduling

In the set of control rules used in both [5] and [18], and during the development of this software, there were no two (or more) behaviours B_1 and B_2 such that the preconditions of B_1 held iff B_2 held, therefore true non-determinism was not needed in the NuSMV input as there would always be some possible state in which the preconditions of B_1 hold and the preconditions of B_2 did not hold, and vice versa. It may be the case that some other set of control rules may indeed have two (or more) behaviours B_1 and B_2 such that the preconditions of B_1 hold iff the preconditions of B_2 hold.

The command line switch *-tnd* allows true non-deterministic behaviour scheduling to be enabled. For a set of behaviours $\mathcal{B} = \{B_1, \ldots, B_n\}$ where for any $1 \leq i \leq n$ and $1 \leq j \leq n$, $\text{priority}(B_i) = \text{priority}(B_j)$ and for every distinct subset $\mathcal{B}' \subseteq \mathcal{B}$, an expression is added to determine the value of *schedule* in the next state such that if the preconditions to every behaviour in $\mathcal{B}'$ hold then in the next state *schedule* will have a value corresponding to some non-deterministically

chosen behaviour in $\mathcal{B}'$. These expressions are then added in order such that the subset $\mathcal{B}' = \mathcal{B}$ is considered first, and all singletons are considered last.

The generic procedures *vectorPowerSet* and *filterVectorByCharacteristicVector* listed in Section 13.5 are used to calculate the power set of all behaviours of equal priority in a set. The *vectorPowerSet* procedure generates all possible characteristic vectors for the given set of values. Each characteristic vector is used by the *filterVectorByCharacteristicVector* procedure to generate a subset of these values and then the list of all possible subsets of values, sorted into ascending or descending order of size, are returned by the *vectorPowerSet* procedure.

4.3.6 NuSMV Input Testing

Section 12.2.1 lists the NuSMV input generated using the intermediate form shown in Section 12.1.4. The command line switch *-mod* was enabled when constructing the NuSMV input. Additionally the command line option *-ii* specified a file containing a set of explicit values to assign to variables in the initial state of the model. The file consists of a number of lines each of the form *variable_name* = "*value*".

A number of LTL properties were formulated and tested in the resultant model. Details of all conducted tests are provided in Section 12.2.2. Many of these test specifications corresponded directly to the expected properties defined in Section 10.7.

4.4 Software Used

All code was implemented in C++ 11 using the Eclipse CDT 8.5.0 Integrated Development Environment for Eclipse Luna, and was compiled using the Cygwin v4.9.0 g++ compiler. NuSMV v2.5.4 was used to construct models using the generated NuSMV input files and to test properties in the models.

5 Evaluation

5.1 Software Functionality

The software produced as a result of this project was expected to translate a set of Care-O-Bot control rules into a succinct and adequate intermediate form representation. This intermediate form should then be translated into input for the model checker NuSMV in which it should be possible to check Linear Temporal Logic properties. Flags were also anticipated to regulate the abstraction of temporal aspects of the control rules.

5.1.1 Intermediate Form Translation

Section 12.1 lists the intermediate form and the control rule file, data extraction rule file, and grammar rule files used in its generation. This intermediate form was checked by hand and analysed to ensure that it correctly represented the set of control rules from which it was generated.

Variables Lines 4-22 of the intermediate form show the propositional variables identified in the set of control rules. The variable shown on line 9 was correctly identified as a non-deterministic variable by grammar rule `<pvc3>` and its corresponding data extraction rule. Those shown on lines 5 and 15 were correctly identified as non-deterministic variables by grammar rule `pvc2` and its corresponding data extraction rule. All other propositional variables correspond to internal flags of the robot and were correctly identified by grammar rules `pvc1` and `pva1` and their corresponding data extraction rules.

Lines 26-31 of the intermediate form show the enumerated variables identified in the set of control rules. Each enumerated variable was identified as expected and had all expected values, for instance the *says* variable shown on line 26 has all of the values identified for the variable on lines 3, 25, 71, 79 and 147, and has an addition *none* value as the variable as the data extraction rule `eva2` flags the variable as being resetting (see Section 11.9.5.

Behaviours All behaviours were present and each individual precondition rule and action rule was correctly identified. The rules defined on lines 133-135 of the set of control rules form the *S1-WaitHere* behaviour. Lines 269-278 of the intermediate form show the corresponding behaviour in the intermediate form. The behaviour is correctly identified as being both interruptible and schedulable, and has the expected priority of 40. The single precondition rule is correctly identified, checking that the variable *::512::Goal-waitHere* is true. The final action in this behaviour in the intermediate form corresponds directly to that in the control rule file, where the robot should non deterministically execute one of the *S1-Set-Waithere*, *S1-Set-ReturnHome* or *S1-Set-Continue* behaviours; this represents the user selecting one of the options shown on the robot's graphical user interface. Line 134 of the control rule file describes the second action of the *S1-WaitHere* behaviour with this behaviour executes another action, namely the *S1-sleep* behaviour. As both behaviours are interruptible the actions of the *S1-sleep* behavior are substituted into the *S1-WaitHere* behaviour, replacing the *execute behaviour* action, as described in Section 10.5.

Lines 44 and 74 of the intermediate form corresponding to lines 5 and 20 of the set of control rules show that *been-in-state* and *was-in-state* conditions are correctly identified. Lines 282-294 of the intermediate form show the logical expression tree constructed for the *S1-WatchTV* behaviour. Internal AND and OR nodes are represented by && and || respectively. Figure 11 shows the precondition rules with the *andOrConnector* field highlighted in red. A value of 2 signifies that a precondition should form a disjunction with the next and a value of 0 signifies that a conjunction should be formed. For simplicity we will refer to the seven preconditions as $p_1, \ldots, p_7$. We would then expect a logical expression tree to be constructed of the form

$((((((p_1 \vee p_2) \vee p_3) \vee p_4) \vee p_5) \wedge p_6) \wedge p_7)$. In the implementation the preconditions are in fact considered in reverse order giving the equivalent expression $(p_7 \wedge (p_6 \wedge (p_5 \vee (p_4 \vee (p_3 \vee (p_2 \vee p_1))))))$.

Figure 11: *S1-WatchTV* Precondition Rules

```
136. S1-watchTV 30 1 1 32 R 2 0 Living room sofa seat 1 is occupied
137. S1-watchTV 30 1 1 33 R 2 0 Living room sofa seat 2 is occupied
138. S1-watchTV 30 1 1 34 R 2 0 Living room sofa seat 3 is occupied
139. S1-watchTV 30 1 1 35 R 2 0 Living room sofa seat 4 is occupied
140. S1-watchTV 30 1 1 36 R 0 0 Living room sofa seat 5 is occupied
141. S1-watchTV 30 1 1 37 R 0 0 Television Wattage > 10
142. S1-watchTV 30 1 1 44 R 0 0 ::513::  GOAL-watchTV is false AND has
     beenin this state for more than 3600 seconds
```

The generated intermediate form fully represents all aspects of the behaviours defined in the set of control rules, albeit in a more succinct, legible form. The questionnaire listed in Section 14 was completed by a postdoctoral Research Associate who will be using the software for further work relating to the Trustworthy Robotic Assistants project. The response to question 5 states that the intermediate form will be very useful for translation from sets of control rules into languages for other model checkers.

5.1.2 NuSMVInput Translation

Section 12.2 shows the NuSMV file generated from the intermediate form discussed in the previous section, and demonstrates how properties that were expected to hold in a model produced as a result of the transformation do indeed hold in this generated model. The following properties are analogous to those checked in the hand-coded model produced in [5] and for each property tested the result is identical to that stated in [5].

Property 1 If the fridge door is open and *::514::GOAL_fridgeUserAlerted* is false then at some time in the future the Care-O-Bot will be in the living room and at some time after that it will say the fridge door is open. This is expected to be false as even though the preconditions for the behaviour *S1-alertFridgeDoor* hold the preconditions to a behaviour with a higher priority may hold and this behaviour will instead be scheduled.

Result

```
-- specification  G ((Fridge_Freezer_In__ON_ & !__514__GOAL_fridgeUserAlerted) -
>  F (location = __2___Living_Room &  F says = _The_fridge_door_is_open__))  is
false
```

Property 2 If the fridge door is open and *::514::GOAL_fridgeUserAlerted* is false and the *S1-alertFridgeDoor* behaviour has been scheduled then at some time in the future the Care-O-Bot will be in the living room and at some time after that it will say the fridge door is open. This is expected to be true as the S1-alertFridgeDoor behaviour is interruptible and should therefore execute all of its actions.

Result

```
-- specification  G ((((Fridge_Freezer_In__ON_ & !__514__GOAL_fridgeUserAlerted)
 & schedule = schedule_S1_alertFridgeDoor) & step = step_1) ->  F (location = __
2___Living_Room &  F says = _The_fridge_door_is_open__))  is true
```

Property 3 If the person selects the Care-O-Bot GUI choice 'Goto Kitchen', then at some time in the future it will be in the kitchen. This is expected to be false. Selecting this GUI options sets the internal flag *::505::GOAL-gotoKitchen* to be true which is the precondition for the *S1-gotoKitchen* behaviour, however the preconditions to a behaviour having a higher priority may hold or the *S1-gotoKitchen* behaviour may be interrupted.

Result

```
-- specification  G (schedule = schedule_S1_Set_GoToKitchen ->  F location = __7
___Kitchen_Entrance_in_the_Dining_Room)  is false
```

Property 4 If the person selects the Care-O-Bot GUI choice 'Goto Kitchen' and this behaviour is scheduled, then at some time in the future it will be in the kitchen. This is expected to be false. Selecting this GUI options sets the internal flag *::505::GOAL-gotoKitchen* to be true which is the precondition for the *S1-gotoKitchen* behaviour, however again the *S1-gotoKitchen* behaviour may be interrupted.

Result

```
-- specification  G ((schedule = schedule_S1_Set_GoToKitchen &  F schedule = sch
edule_S1_goToKitchen) ->  F location = __7___Kitchen_Entrance_in_the_Dining_Room
)  is false
```

Property 5 If one of the sofa seats is occupied, the television is on, and the goal to watch television has been false for at least an hour, then at some point in the future the Care-O-Bot will be at the sofa and at some point after that it will say 'Shall we watch Tv?'. This is expected to be false as although the preconditions to the *S1-watchTV* behaviour hold the preconditions to some other behaviour with a higher priority may hold, or if the *S1-watchTV* behaviour is scheduled it may be interrupted before completion.

Result

```
-- specification  G (((((((((seat_occupied = seat_1 | seat_occupied = seat_2) | s
eat_occupied = seat_3) | seat_occupied = seat_4) | seat_occupied = seat_5) & Tel
evision_Wattage___10) & !__513__GOAL_watchTV) & __513__GOAL_watchTV_FALSE_BEEN_I
N_STATE = s_final) ->  F (location = __14___Living_Room_Sofa_Area_in_the_Living_
Room & says = _Shall_we_watch_TV_together__))  is false
```

Property 6 If the raiseTray behaviour is executed then at some point in the future the physical tray is raised and at some pointer after that the robot's internal flag indiciating that the tray is raised should be true. This is expected to be true as the raiseTray behaviour is uninterruptible.

Result

```
-- specification  G (schedule = schedule_raiseTray ->  F (tray = Raised &  F __5
00__TrayIsRaised))  is true
```

Property 7 If the *S1-gotoKitchen* behaviour is scheduled and the preconditions to the *S1-alertFridgeDoor* behaviour hold then in the next moment in time the *S1-gotoKitchen* behaviour will not be scheduled. This is expected to be true as the *S1-gotoKitchen* behaviour is interruptible and the preconditions to at least one behaviour with a higher priority (namely *S1-alertFridgeDoor* hold.

Result

```
-- specification  G ((((schedule = schedule_S1_goToKitchen & Fridge_Freezer_In__
ON_) & !__514__GOAL_fridgeUserAlerted) & Fridge_Freezer_In__ON__TRUE_BEEN_IN_STA
TE = s_final) ->  X !(schedule = schedule_S1_goToKitchen))  is true
```

5.1.3 Anticipated Flags

Section 4.3.4 gives details of the *-sps* , *-sms* and *-mps* command line options. These options can be used to regulate the level of abstraction used to represent temporal aspects of the Care-O-Bot's behaviours, allowing larger models to be constructed from generated code when a higher granularity of abstraction is required.

There was sufficient time during the implementation stage of the project to include additional functionality, notable the inclusion of the *-tnd* and *-tod* command line options which respectively enable true non-determinism for behaviour scheduling (Section 4.3.5) and set a fixed time of day (Section 4.3.2).

5.2 Degree of Automation

The project requirements stated that ideally full automation of the translation process would be realised, however it may be necessary for the user to disambiguate input. Figure 12 shows a selection of the output generated by the software when translating the set of control rules into the intermediate form, then into NuSMV input.

During the entire transformation process user input was required only once, where a behaviour having no preconditions was found in the input. In all other cases small discrepancies in the naming of variables in the input were automatically regulated and corrected. The answer provided for question 1 in the completed questionnaire found in Section 14 rates the degree of automation as 9.5 out of 10 and states that only minimal user input is required during translation.

Figure 12: Automation of the Translation Process

```
------------------------------] Disambiguation [--------------------------

automatic similarity match
-------------------------------------------------------------------------------

matched propositional variable '::515:: GOAL-AnserDoorBell'
with existing propositional variable '::515::GOAL-AnserDoorBell'
having 96% similarity
------------------------------[ No Preconditions ]--------------------------

a schedulable behaviour has no preconditions
-------------------------------------------------------------------------------

schedulable behaviour 'S1-sleep' has no preconditions
        i: ignore and continue
        r: remove behaviour
        f: flag behaviour as non-schedulable
        enter selection: i
------------------------------] Disambiguation [--------------------------

automatic case-insensitive match
-------------------------------------------------------------------------------

matched behaviour 'S1-set-gotoSofa'
with existing behaviour 'S1-Set-GoToSofa'
------------------------------] Disambiguation [--------------------------

automatic case-insensitive match
-------------------------------------------------------------------------------

matched behaviour 'S1-set-gotoTable'
with existing behaviour 'S1-Set-GoToTable'
intermediate form written to file 'test_intermediate_form.txt' successfully
NuSMV code written to file 'nusmv_input.smv' successfully
```

5.3 Comparison to Hand-Written Model

In [5] as NuSMV input was constructed by hand and an NuSMV model was constructed using this input. Here a number of comparisons between this model and models generated from the automated translation process are made.

Firstly, a comparison between the sizes of the models is made. Figure 13 shows the size of the models respectively generated from the handwritten code produced [5], the code produced from translating the set of control rules using this software using the default settings, and the code produced by translating using the *-mos* command line switch.

Figure 13: Comparison of Generated NuSMV Models

Handwritten Code

```
################################################################################
system diameter: 105
reachable states: 130593 (2^16.9947) out of 7.79279e+015 (2^52.7911)
################################################################################
```

Generated Code (default settings)

```
################################################################################
system diameter: 164
reachable states: 1.8024e+006 (2^20.7815) out of 7.27327e+014 (2^49.3696)
################################################################################
```

Generated Code (*-mos* enabled)

```
################################################################################
system diameter: 171
reachable states: 7.91626e+006 (2^22.9164) out of 1.16372e+016 (2^53.3696)
################################################################################
```

The code generated using the default settings resulted in a model that was ∼8x smaller than the model produced using the handwritten code, and the code generated with *-mos* enabled resulted in a model that was ∼2x the size of the model produced using the handwritten code. States in an NuSMV model are defined as *n-tuple* where each element of the tuple corresponds to the value of some variable in the model. The code generated using the default settings had fewer variables than the handwritten model, resulting in a smaller model, while the code generated with *-mos* enabled had more variables than the handwritten model as additional variables were introduced to correctly model the *been-in-state* and *was-in-state* constraints that were ignored when generating code using the default abstraction settings. In both instances the handwritten code resulted in a model that had fewer reachable states than the model resulting from generated code. This is because the handwritten code uses additional abstraction to model certain aspects of the Care-O-Bot behaviours.

A number of behaviours of the form *S1-Set-...* are present in the set of control rules. These behaviours are scheduled for execution by the Care-O-Bot when the occupant of the house selects an option on its graphical user interface, and when executed set the value of a number of variables corresponding to the Care-O-Bot's internal flags. In the handwritten code a variable *gui_choice* was used to represent the choices made by the occupant of the house. When a choice is made in the next state all variable assignments shown in the corresponding *S1-Set...* behaviour in the control rules occur simultaneously In the generated models these behaviours are considered no differently than other executed behaviours, therefore when a GUI option is chosen the currently scheduled behaviour executes the corresponding *S1-Set-...* behaviour, which then assigns a value to each variable in turn before control is returned to the original behaviour.

5.4 Flexibility

It was essential that the produced software provided was capable of translating new sets of control rules, as its primary function is to be used as a tool for further formal verification of new sets of Care-O-Bot behaviour. A new set of control rules was made available by the University of Hertfordshire which were used to evaluate the performance of the software when translating new syntactic forms of control rule.

Examples of new syntactic forms of control rule present in the new set of rules are shown in Figure 14. For each new form of control rule the corresponding grammar rule and data extraction rule are also given. It was possible to construct grammar rules and data extraction rules for every new form of control rule. Although the construction of these rules takes time it takes considerable less time than would be required to produce NuSMV code corresponding to these sets of behaviours by hand. The feedback for question 3 in the questionnaire shown in Section 14 states that the software has been able to successfully translate all investigate control rules so far.

Figure 14: The Behaviour Module

Example 1:

```
Send the robot to the LivingRoom::2::, lowering tray if possible
(gotoLivingRoom::2::)

<exb4> ::= +<any_text> '(gotoLivingRoom::2::)'

exb4; behaviour_name = "gotoLivingRoom::2::";
```

Example 2:

```
Wait for 0.02 seconds on ::3::Care-O-Bot 3.6

<del1> ::= 'Wait' 'for' <float> 'seconds' 'on' <any_text> <any_text>

del1; seconds = [3];
```

Example 3:

```
ZUYD Cup is Empty

<pvc7> ::= +<any_text> 'is' <any_text>

pvc7; prop_name = [1]; truth_value = [3]; true = "full";
false = "empty"; non_deterministic = false;
```

Although the software can successfully translate all control rules tested so far it does have its limitations. It was assumed during the implementation that behaviours executed by *non-determinstic behaviour execution* action rules would have no preconditions, as this was always found to be the case when both sets of control rules were investigated. Any model constructed from NuSMV code generated for a set of control rules where this is the case would not accurately represent the Care-O-Bot behaviours.

5.5 Conclusion

This project has successfully met all of its aims and objectives. All essential features have been fully implemented and a number of additional features have also been included. A high level of automation has been achieved with the user only prompted for input when critical issues need to be addressed. The software produces models of a reasonable size in which it is feasible to check LTL properties. It has been shown that all properties that were expected to hold in models produced as a result of this translation process do indeed hold. Furthermore, the software proves to be flexible enough to correctly translate a different set of control rules to those used during the design and implementation.

The primary purpose of this project was to produce a tool that can be used in further work conducted as part of the Trustworthy Robotic Assistants project. The completed questionnaire shown in Section 14 gives details on the future applications of CRutoN. For this project to be truly deemed a success the software should prove to be an efficient and powerful tool when used for further formal verification of the Care-O-Bot.

Further work that could be conducted to extend the functionality of the software could include additional automatic translations into input for other model checkers.

6 Learning Points

My interest in the formalisms and applications of logic was first instilled by logic-based modules undertaken as part of my Computer Science degree. I chose this project as it would give me the opportunity to further my knowledge of logic and its applications, and to apply this knowledge to solve a challenging problem. As I intend to pursue a career in academic research this project also provided a unique opportunity to be involved in active research work whilst gaining experience of reading academic papers and developing my technical writing and analytical skills.

During the summer of 2014 I began reading about model checking and its applications. *"Logic in Computer Science: Modelling and Reasoning about Systems"* by Michael Huth and Mark Ryan [9] and *"Practical Formal Methods Using Temporal Logic"* by Michael Fisher [6] provided essential foundational knowledge of modal logics, linear temporal logics and model checking. Having only previously studied propositional and first-order logic this initial research was challenging but rewarding. This foundational knowledge allowed me to develop a good understanding of the problem background and to comprehend the research papers that related to this project.

I anticipated that the implementation stage would be challenging, and that a lot of code needed to be produced in order to meet the requirements stated in the specification document. I therefore decided to spend a significant amount of time designing both the translation into the intermediate form, and the translation into NuSMV input. The extra work conducted during the design stage of this project proved to be fruitful. The basic requirements were completed slightly ahead of schedule allowing more time to implement extra software functionality, reinforcing my belief that a good design can often lead to an easier implementation.

Having mainly used Java throughout my degree I decided to implement CRutoN using C++ to gain additional experience with this less familiar language. I believe that I have a much better understanding of this language than before and now feel as confident programming in C++ as I had in Java. I had never maintained a code base of the size before and as such I had to ensure that my code was coherent, structured and efficiently compartmentalised and documented. The insight I have gained into the management of large collections of code, and its corresponding version control, will prove to be invaluable in any future projects that I may undertake.

Planning the individual stages of this project was vital to its success. Several deadlines had to be met throughout the project's life cycle and it was essential that the project remained on or ahead of schedule so that there was time to deal with any unexpected delays and setbacks. I believe that I now have greatly improved my ability to manage my workload and to efficient allocate my time to different tasks.

Undertaking this project reinforced my desire to purse a career in academic research and led to me applying for postgraduate study relating to formal verification. Over the last year I have greatly developed my understanding of model checking and temporal logics and how these techniques and formalisms are used to address real problems, skills that are directly transferrable to my intended postgraduate research.

7 Professional Issues

The British Computer Society (BCS) Code of Conduct and Code of Practice respectively define expected professional standards and guidelines for the development of software systems.

Section 2a of the Code of Conduct states that work should only be undertaken if it is within your professional competence, while section 2b states that you should not claim any level of competence that you do not possess. I adhered to these principles as I believed that I had the necessary skills required to successfully complete this challenging project. Section 2c state that professionals should develop their professional knowledge, skills and competence on a continuing basic. This project has given me the unique opportunity to develop a deeper understanding of model checking and to develop the skill of applying these techniques to solve real problems. In accordance with section 2e of the Code of Conduct, which states that alternate viewpoints of work should be respected and honest criticisms of work should be sought out, I was in regular contact with the postdoctoral Research Associate who would be using my software. He provided me with invaluable criticisms and suggestions throughout the final stage of the implementation of my software.

The Code of Practice states that when research work is being conducted, work produced by other people and organisations should be investigated, analysed and acknowledged. All sources used when researching work relevant to this project have been fully cited and acknowledgement has been given where appropriate. Additionally, the Code of Practice states that all code produced should be well-structured and should be easy for other programmers to maintain by using meaningful naming conventions and avoiding unnecessarily complex programming techniques. All of the code produced as a result of this project is well-structured, documented and appropriate variable names have been used throughout to maintain clarity. Technial work should be documented such that others may take over the work if needed. As the produced software will be indeed be extended by others, as mentioned in the questionnaire, all code produced has been documented and this report gives detailed descriptions of how the software was designed and implemented.

Professionals are expected to manage their workload efficiently and should not commit themselves to work that they feel they cannot complete in time. Time management has been an integral part of this project and only through effective time management has this project been completed successfully.

While it was important to adhere to all standards given in Code of Conduct some sections of the Code of Practice were not applicable at all to this project, for instance sections 5.4 and 5.6 of the Code of Practice which respectively give guidelines for designing training courses and establishing support and maintenance services.

8 Bibliography

[1] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

[2] A. Cimatti, E. D. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer Berlin Heidelberg, 2002.

[3] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[5] C. Dixon, M. Webster, J. Saunders, M. Fisher, and K. Dautenhahn. The fridge door is open–temporal verification of a robotic assistant's behaviours. In *Advances in Autonomous Robotics Systems*, pages 97–108. Springer, 2014.

[6] M. Fisher. *An introduction to practical formal methods using temporal logic.* Wiley, 2011.

[7] D. Grune and C. H. Jacobs. *Parsing techniques : a practical guide.* Springer, 2008.

[8] G. J. Holzmann. *The* Spin *model checker: primer and reference manual.* Addison-Wesley, 2004.

[9] M. Huth and M. Ryan. *Logic in computer science: modelling and reasoning about systems.* Cambridge University Press, 2004.

[10] ISO. Robots and robotic devices – safety requirements for personal care robots. ISO 13482:2014, International Organization for Standardization, Geneva, Switzerland, 2014.

[11] S. A. Kripke. Semantical considerations on modal logic. In *Proceedings of a Colloquium on Modal and Many-Valued Logics*, 1963.

[12] K. L. McMillan. *Symbolic model checking*. Springer, 1993.

[13] K. L. McMillan. The smv language. Technical report, Cadence Berkeley Labs, 1999.

[14] R. C. Moore. Removing left recursion from context-free grammars. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 249–255. Association for Computational Linguistics, 2000.

[15] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[16] M. Sierhuis and W. J. Clancey. Modeling and simulating work practice: A method for work systems design. *IEEE Intelligent Systems*, 17(5):32 – 41, 2002.

[17] M. Szpyrka, A. Biernacka, and J. Biernacki. Methods of Translation of Petri Nets to NuSMV Language. In *Proceedings of the 23rd International Workshop on Concurrency Specification and Programming (CSP 2014)*, volume 1269 of *CEUR Workshop Proceedings*, pages 245–256, 2014.

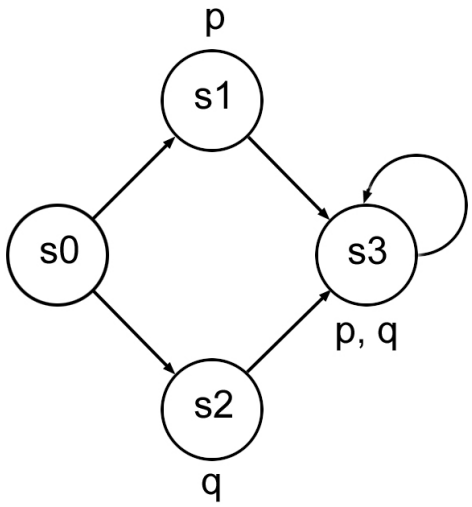
[18] M. Webster, C. Dixon, M. Fisher, M. Salem, J. Saunders, K. Koay, and K. Dautenhahn. Formal verification of an autonomous personal robotic assistant. In *Formal Verification and Modeling in Human-Machine Systems*, pages 74–79. AAAI, 2014.

9 Appendix A - NuSMV

NuSMV[2] is a popular model checker for temporal logic and is a reimplementation and extension of SMV. SMV is a binary decision diagram based model checker developed at Carnegie Mellon University [12]. Given a model of a system, and a property to test in the model, NuSMV exhaustively explores the possible runs of the system to check whether the property remains true. If it does then the checker returns *true* and we can say that the model *satisfies* the property, if not *false* will be returned and a counterexample can be generated showing a run of the system where the property does not remain true.

The model checker accepts as input a finite state transition system defined using the modelling language SMV [13]. A model can be represented as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$, where S is a set of states, $\mathcal{I} \subseteq \mathcal{S}$ is a set of initial states, $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a relation that specifies the transitions between states, and $\mathcal{L}$ is a labelling function that maps a state to a set of atomic propositions that are valid in that state [1]. States in an NuSMV model are defined as an *n-tuple* where each element of the tuple corresponds to the value of some variable in the model.

Figure 15: A State Transition System M



$\mathcal{M} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$

$\mathcal{S} = \{s_0, s_1, s_2, s_3\}$

$\mathcal{I} = \{s_0\}$

$\mathcal{R} = \{(s_0, s_1), (s_0, s_2), (s_1, s_3), (s_2, s_3), (s_3, s_3)\}$

$\mathcal{L} = \{s_1 \mapsto \{p\}, s_2 \mapsto \{q\}, s_3 \mapsto \{p, q\}\}$

Figure 16: An NuSMVModel of M

```
 1. MODULE main
 2.
 3. VAR
 4.   state:  {s0, s1, s2, s3};
 5.
 6. ASSIGN
 7.   init(state):= s0;
 8.
 9.   next(state):=
10.     case
11.       state = s0:  {s1, s2};
12.       state = s1:  s3;
13.       state = s2:  s3;
14.       state = s3:  s3;
15.     esac;
16.
17. DEFINE
18.   p:= (state = s1 | state = s3);
19.   q:= (state = s2 | state = s3);
20.
21. LTLSPEC
22.   F(G(p & q));
```

NuSMV allows the system to be decomposed into modules which can be instantiated multiple times. Every model has at least one module, the `main` module, and some number of additional parameterizable modules. Each model usually consists of two main sections, `VAR` and `ASSIGN`. The `VAR` section defines variables and instances of modules; variables can be Booleans, symbolic enumerated types or bounded integer ranges. The `ASSIGN` section defines the initial states (initial

values for the variables) and transitions between the states (assignments of values to variables). A third section `DEFINE` can also be included, where macros of the form *identifier*:= *expression* can be defined.

Figure 15 shows a simple state transition system M. M has four states, s_0, s_1, s_2 and s_3, an initial state s_0, and some transitions between states. States are labelled with the names of propositions that are true in that state. Beginning with the initial state s_0 a transition can be made to either s_1 or s_2, then from here to s_3, where self-transition occurs.

Figure 16 shows the NuSMVmodel describing this state transition system shown in Figure 15. This model consists of a single module, the required module `main`. On line 4 an enumerated variables `state` is declared, having a named value for each of the states in M. Line 7 assigns the initial value `s0` to state (s_0 is the initial state in $\mathcal{M}$). Lines 9-15 define the transitions from one state to another. The statements on lines 11-14 in the `case-esac` block are of the form *logical expression: value*, and are sequentially evaluated in the order that they appear in the code. If the logical expression evaluates to true then the variable has the given value in the next state, for instance line 12 says that if `state` is equal to `s1` then in the next state `state` will be equal to `s3`. If the logical expression evaluates to false then the next statement in the `case-esac` block is evaluated. Line 11 shows how non-determinism in a system can be modelled; here if `state` is equal to `s0` then in the next state `state` will be non-deterministically assigned one of the values `s1` or `s2`. In the `DEFINE` section two macros are defined. Line 18 declares a macro `p` which evaluates to true if `state` has the value `s1` or `s3` (as $p \in \mathcal{L}(s_1)$ and $p \in \mathcal{L}(s_3)$).

Propositional logic operators are represented in NuSMVby: `->` (implies), `<->` (equivalence), `&` (and), `|` (or) and `!` (not). Temporal operators are represented as `G` (always), `F` (sometimes), `X` (next) and `U` (until). On line 22 the Linear Temporal Logic property $\Diamond (\Box (p \wedge q))$ is given. The model satisfies the property as in every state there is always some future state (namely s_3), where in all future states both p and q hold.

10 Appendix B - Semantics of Care-O-Bot Control Rules

10.1 Introduction

The University of Hertfordshire provided a database populated with control rules for the Care-O-Bot. These rules are grouped together to form behaviours, high level rules which determine how the robot acts in its environment. There are two types of control rule: precondition rules and action rules. Precondition rules are propositional statements and action rules correspond to actual actions performed by the robot, including the setting of variables. Precondition rules are implemented as SQL queries and action rules are implemented as Robot Operating System [15] scripts.

A behaviour consists of a sequence of precondition rules linked by boolean operators, and a sequence of actions rules. A behaviour can be scheduled for execution by the robot if all of the behaviour's precondition rules hold. If a behaviour is scheduled for execution then the robot will execute the sequence of actions for that behaviour. Figure 17 shows the answerDoorBell behaviour. Rule 0 is a precondition rule and rules 1 and 2 are action rules. For each control rule in the database there is a field determining if the control rule is a precondition or action. These are omitted here for simplicity. Rule 0 is a precondition rule requiring the ::515::Goal-AnserDoorBell flag to be true. When this precondition rule holds, and this behaviour is scheduled for execution, the robot will sequentially execute action rules 1 and 2, first saying the word "Doorbell" then waiting for 5 seconds.

Figure 17: The answerDoorBell behaviour

```
0  ::515::GOAL-AnserDoorBell is true
1  ::0::Care-O-Bot 3.2 says 'Doorbell' and wait for completion
2  Wait for 5 seconds on ::0::Care-O-Bot 3.2
```

10.2 Behaviour Scheduling

Algorithm 1 describes the process by which behaviours are scheduled for execution. An integer value is associated with each behaviour, the behaviour's *priority*. Should the preconditions to two or more behaviours hold at any moment in time, the behaviour having the highest priority will be scheduled for execution next.

Each behaviour is flagged as being either interruptible or non-interruptible, and either schedulable or non-schedulable. The sequence of actions executed by an interruptible behaviour b_i can be interrupted by another behaviour b_j if the priority of b_j is greater than the priority of b_i and the preconditions of b_j hold. Behaviours flagged as non-schedulable cannot be scheduled for execution. Non-schedulable behaviours are instead executed directly by actions in other behaviours.

Algorithm 1 Care-O-bot behaviour scheduling.

```
1: procedure BehaviourScheduling
2:     let B be a set of behaviours
3:     scheduled = none
4:     while true do
5:         S ← {b ∈ B : preconditions_hold(b) and schedulable(b)}
6:         if (scheduled = none or scheduled has no other action to perform) and S ≠ ∅ then
7:             scheduled = b ∈ S s.t. priority(b) = max{priority(s) : s ∈ S}
8:         else if ∃b ∈ S.(priority(b) > priority(scheduled)) and interruptible(scheduled) then
9:             scheduled = b
10:        end if
11:        if scheduled has another action to perform then
12:            perform the next action in scheduled
13:        else
14:            scheduled = none
15:        end if
16:    end while
17: end procedure
```

10.3 Precondition Rules

Each robot behaviour has a possibly empty sequence of precondition rules (propositional statements) linked by boolean AND and OR operators; this sequence must evaluate to true for the behaviour to be scheduled. If the set of precondition rules for a behaviour is empty then the sequence of precondition rules is automatically satisfied.

Each propositional statement checks the state of the robot or the state of the environment in which the robot operates. Precondition rules may be suffixed with an additional constraint requiring the condition to have evaluated to true throughout a previous period of time, or requiring the condition to have evaluated to true at least once within a previous period of time.

Figure 18 shows some examples of precondition rules taken directly from The University of Hertfordshire database.

Figure 18: Examples of Precondition Rules

1. `Living room sofa seat 1 is occupied`

   This precondition rule is satisfied if a sensor in the Robot House is indicating that the occupier is sitting in one of the seats in the living room.

2. `::503::  5PM-MedicineReminder is true AND has been in this state for more than 60 seconds`

   This precondition rule is satisfied if the internal flag ::503::5PM-MedicineReminder is true, and has been true throughout the last 60 seconds. Intuitively when this rule is satisfied this indicates that the robot believes that it is time to remind the occupier to take their medicine.

10.4 Action Rules

Each robot behaviour has a non-empty sequence of action rules where each action rule corresponds to an actual action performed by the robot. When the precondition rules of a behaviour are satisfied, and that behaviour is scheduled for execution, the first action in the sequence of actions for that behaviour is performed. Upon completion of the action the subsequent action (if any) in the sequence is performed and so forth, until either there are no more actions to perform or the behaviour is interrupted by another behaviour.

Figure 19 shows some examples of action rules taken directly from The University of Hertfordshire database.

Figure 19: Examples of Action Rules

1. `Turn light on ::0::Care-O-Bot 3.2 to yellow`

   The Care-O-Bot is equipped with a light which can display a number of different colours. This action changes the colour of this light to yellow.

2. `Execute sequence 'lowerTray' on ::0::Care-O-Bot 3.2`

   This action directly executes the behaviour 'lowerTray'. If the preconditions to the lowerTray behaviour hold then control passes to this behaviour then returns when the executed behaviour terminates.

10.5 Nested Behaviour Executions

A behaviour B_1 may pass control to another behaviour B_2 as the result of performing a *behaviour execution* action. When B_2 has performed all of its actions control returns to B_1. In many instances this execution of another behaviour is in fact a macro expansion where the action rules of the executed behaviour are substituted for the *behaviour execution* rule in the original behaviour. As a behaviour may execute a second behaviour, which may in turn execute another behaviour and so forth, this process of macro expansion is applied iteratively to each behaviour until no more *behaviour execution* rules in a behaviour can be expanded.

If B_1 is flagged as being uninterruptible then the action rules of B_2 are substituted into B_1, replacing the *behaviour execution* action in B_1's sequence of action rules. If both B_1 and B_2 are flagged as being interruptible then again the action rules of B_2 are substituted into B_1. If B_1 is flagged as being interruptible, and B_2 is flagged as being non-interruptible then no substitutions are performed and control will instead pass to behaviour B_2.

It is worth nothing that as a result of this substitution process that for any remaining *behaviour execution* rules where a behaviour B_1 executes another behaviour B_2 it is always the case that B_2 is an uninterruptible behaviour.

10.6 Definitions

Let a Care-O-bot behaviour be a pair $B_i = \langle precon_i, \mathcal{A}_i \rangle$ where $precon_i$ is a proposition that is true iff the preconditions of B_i hold, and $\mathcal{A}_i = (A_i^1, \ldots, A_i^{n_i})$ is an ordered sequence of action rules where each action is executed sequentially should $precon_i$ hold and B_i be scheduled for execution. Each action rule A_i^j, $1 \leq j \leq n_i$ is either a *propositional value assignment*, *enumeration value assignment*, *behaviour execution*, *non-deterministic behaviour execution* or a *delay*. Let $\text{pre}(B_i) = precon_i$, $\text{action}_k(B_i) = A_i^k$, and $\text{num\_actions}(B_i) = |\mathcal{A}_i|$. Given a set of n behaviours $\mathcal{S} = \{B_1, \ldots, B_n\}$ define the following:

1. Let priority : $\mathcal{S} \rightarrow \mathbb{N}_0$ be a unary function that maps a behaviour $B_i \in \mathcal{S}$ to some $p \in \mathbb{N}_0$, where p is the priority of B_i.

2. Let interruptible(B_i) be a unary predicate which is true at all times iff B_i is flagged as being interruptible.

3. Let schedulable(B_i) be a unary predicate which is true at all times iff B_i is flagged as being schedulable.

4. Let isExecution(A) be a unary predicate which is true at all times iff the action A executes another behaviour, as described in Section 10.5.

5. For every $B_i \in \mathcal{S}$ define $scheduled_i$ to be a proposition that is true iff behaviour B_i has been scheduled for execution and is executing one of its actions.

6. Define $step_k$ to be a proposition that is true if there is some behaviour $B_i \in \mathcal{S}$ such that $scheduled_i$ holds and B_i is executing $\text{action}_k(B_i)$.

7. Define $scheduled\_none_S$ to be a proposition that is true iff $(\neg scheduled_1 \wedge \ldots \wedge \neg scheduled_n)$ holds.

8. Define $interrupted\_next_i$ to be a proposition that is true if

   $(scheduled_i \ \wedge \ \text{interruptible}(B_i) \ \wedge \ \exists B_j \in S.(\text{pre}(B_j) \ \wedge \ \text{schedulable}(B_j)$
   $\wedge \ (\text{priority}(B_j) > \text{priority}(B_i))))$

   holds.

9. Define $is\_final\_action_i$ to be a proposition that is true iff $scheduled_i$ holds and B_i is executing action $\text{action}_k(B_i)$ where $k = \text{num\_actions}(B_i)$.

10.7 Linear Temporal Logic Properties

Given a set of n behaviours $\mathcal{S} = \{B_1, \ldots, B_n\}$, and using the definitions from Section 10.6, the following LTL properties would be expected to hold in any NuSMV model produced for a set of behaviours.

1. Only one action can be executed by the robot at any point in time:

   $\Box(step_i \Rightarrow (\forall j, 1 \leq j \leq n.(j \neq i \Rightarrow \neg step_j))$

2. Only one behaviour can be scheduled by the robot at any point in time:

   $\Box(scheduled_i \Rightarrow (\forall j, 1 \leq j \leq n.(j \neq i \Rightarrow \neg scheduled_j))$

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

   $\Box((scheduled\_none_S \ \wedge \ \exists j.(\text{pre}(B_j) \ \wedge \ \text{schedulable}(B_j) \ \wedge \ \forall k \neq j.((\text{pre}(B_k) \ \wedge$
   $\text{schedulable}(B_k)) \Rightarrow (\text{priority}(B_k) \leq \text{priority}(B_j))))) \Rightarrow \bigcirc(scheduled)j \ \wedge \ step_1))$

4. If a behaviour B_i is interruptible, and the preconditions to one or more schedulable behaviours in S with a higher priority hold, then in the next moment in time the schedulable behaviour having the highest priority of all these behaviours will interrupt B_i and will be executing its first action:

   $\Box((\mathit{scheduled}_i \wedge \text{interruptible}(B_i) \wedge \exists j \neq i.(\text{pre}(B_j) \wedge \text{schedulable}(B_j) \wedge (\text{priority}(B_j) > \text{priority}(B_i)) \wedge \forall k \neq j.((\text{pre}(B_k) \wedge \text{schedulable}(B_k)) \Rightarrow (\text{priority}(B_k) \leq \text{priority}(B_j)))) \Rightarrow \bigcirc(\mathit{scheduled}_j \wedge \mathit{step}_1))$

5. If a behaviour B_i is scheduled and is executing its k^{th} action, this action does not execute another behaviour (see Section 10.5) and is not the final action for B_i, and in the next moment in time behaviour B_i will not be interrupted by another behaviour, then in the next moment in time behaviour B_i will be executing its $(k+1)^{th}$ action:

   $\Box((\mathit{scheduled}_i \wedge \mathit{step}_k \wedge \neg\text{isExecution}(\text{action}_k(B_i)) \wedge \neg \mathit{is\_final\_action}_i \wedge \neg \mathit{interrupted\_next}_i) \Rightarrow \bigcirc(\mathit{step}_{k+1}))$

6. If a behaviour B_i is scheduled and is executing its k^{th} action, this action executes another behaviour B_j whose preconditions hold, and in the next moment in time behaviour B_i will not be interrupted by another behaviour, then in the next moment in time behaviour B_j will be scheduled and will be executing its first action:

   $\Box((\mathit{scheduled}_i \wedge \mathit{step}_k \wedge \text{isExecution}(\text{action}_k(B_i)) \wedge \text{pre}(B_j) \wedge \neg \mathit{interrupted\_next}_i) \Rightarrow \bigcirc(\mathit{scheduled}_j \wedge \mathit{step}_1))$

7. If a behaviour B_i is scheduled and is executing its k^{th} action, this action executes another behaviour B_j whose preconditions hold, this action is not the final action for B_i, and in the next moment in time behaviour B_i will not be interrupted by another behaviour, then in the next moment in time behaviour B_j will be scheduled and will be executing its first action, and at some time after that behaviour B_i will again be scheduled and will be performing its $(k+1)^{th}$ action.

   $\Box((\mathit{scheduled}_i \wedge \mathit{step}_k \wedge \text{isExecution}(\text{action}_k(B_i)) \wedge \text{pre}(B_j) \wedge \mathit{is_final_action}_i \wedge \neg \mathit{interrupted\_next}_i) \Rightarrow \bigcirc(\mathit{scheduled}_j \wedge \mathit{step}_1 \wedge \Diamond(\mathit{scheduled}_i \wedge \mathit{step}_{k+1})))$

8. If a schedulable behaviour B_i is executing its final action, this action does not execute another behaviour and the preconditions to no other behaviour hold then in the next moment in time no behaviour will be executing.

   $\Box((\mathit{scheduled}_i \wedge \mathit{is\_final\_action}_i \wedge \neg\text{isExecution}(\text{action}_{\text{num\_actions}(B_i)}(B_i)) \wedge \text{schedulable}(B_i) \wedge \forall j \neq i.\neg\text{pre}(B_j)) \Rightarrow \bigcirc(\mathit{scheduled\_none}_S))$

9. When an uninterruptible behaviour B_i has been scheduled it is expected to execute all of its actions.

   $\Box((\mathit{scheduled}_i \wedge \neg\text{interruptible}(B_i) \wedge \mathit{step}_i) \Rightarrow \Diamond(\mathit{scheduled}_i \wedge \mathit{step}_{\text{num\_actions}(B_i)}))$

11 Appendix C - Additional Design Documentation

11.1 Identified Precondition Rule Features

The following features of precondition rules were identified:

- Propositional Value Check

  The value of a propositional variable is compared to a given truth value.

  Example: `::502::  5PM-MedicineDue is false`

- Been in State

  A suffix applied to preconditions stating that the precondition must also have held for more than n seconds.

  Example: `has been in this state for more than 60 seconds`

- Was in State

  A suffix applied to preconditions stating that the precondition must also have held at some point within the last n seconds.

  Example: `was in this state within the last 10 seconds`

- Relational Comparison

  Compares the value of an integer variable to an integer value.

  Example: `Television Wattage>10`

- Location Check

  Checks the location of the Care-O-bot .

  Example: `::0::Care-O-Bot 3.2 location is ::7::  Kitchen Entrance in the Dining Room`

- Time Between

  A time constraint stating that the current time must within a given range.

  Example: `Time is between 00:00:00 and 16:59:00`

- Time After

  A time constraint stating that the current time must be at or after a given time.

  Example: `Time is on or after 17:00:00`

- Seat Sensor Check

  Determines if a (numbered) sofa seat in the Robot House is occupied.

  Example: `Living room sofa seat 1 is occupied`

11.2 Precondition Rules Grouped by Features

Propositional Value Check

::512:: GOAL-waitHere is true
::502:: 5PM-MedicineDue is false
::500:: TrayIsRaised is true
::504:: TrayIsEmpty is true
::506:: GOAL-gotoCharger is true
::502:: 5PM-MedicineDue is true
::501:: TrayIsLowered is true
::509:: GOAL-waitAtKitchen is true
::508:: GOAL-gotoSofa is true
::511:: GOAL-waitAtTable is true
::510:: GOAL-waitAtSofa is true
::507:: GOAL-gotoTable is true
::513:: GOAL-watchTV is true
::515:: GOAL-AnswerDoorBell is true
::505:: GOAL-gotoKitchen is true
::514:: GOAL-fridgeUserAlerted is false

Propositional Value Check, Been in State

::503:: 5PM-MedicineReminder is true AND has been in this state for more than 60 seconds
::513:: GOAL-watchTV is false AND has been in this state for more than 3600 seconds
::515:: GOAL-AnswerDoorBell is true AND has been in this state for more than 10 seconds

Propositional Value Check [2], Been in State

Fridge Freezer Is *ON* AND has been in this state for more than 30 seconds

Relational Comparison, Was in State

Doorbell Last Wattage > 1 AND was in this state within the last 10 seconds

Relational Comparison

Television Wattage > 10

Location Check

::0::Care-O-Bot 3.2 location is ::7:: Kitchen Entrance in the Dining Room
::0::Care-O-Bot 3.2 location is ::14:: Living Room Sofa Area in the Living Room
::0::Care-O-Bot 3.2 location is ::23:: Living Room Table in the Living Room Sofa Area of the Living Room

Time Between

Time is between 00:00:00 and 16:59:00

[2]The second identified form of a propositional value check

Time After

```
Time is on or after 17:00:00
```

Seat Sensor Check

```
Living room sofa seat 1 is occupied
Living room sofa seat 2 is occupied
Living room sofa seat 3 is occupied
Living room sofa seat 4 is occupied
Living room sofa seat 5 is occupied
```

11.3 Categorization of Precondition Rules

Propositional Value Checks The following distinct forms of propositional value checks were identified:

1. `::502::  5PM-MedicineDue is false`
2. `Fridge Freezer Is *ON*`
3. `Television Wattage > 10`

Form 1 consists of a propositional variable name and a truth value, and form 2 consists of a proposition variable name and a value `*ON*` or `*OFF*`. Form 3, a relational comparison, has been identified as a propositional variable check. The comparison references some variable external to the Care-O-bot and as such could be toggled non-deterministically in any produced model.

Enumeration Value Checks The following distinct forms of enumeration value checks were identified:

1. `::0::Care-O-Bot 3.2 location is ::14::  Living Room Sofa Area in the Living Room`
2. `Living room sofa seat 1 is occupied`

In form 1 the location of the robot is checked and in form 2 the occupancy of a numbered seat is tested. As the robot can only be in one location at once, and as the robot's environment is designed to accommodate only a single person (who can sit on only one seat at a time), enumerations adequately represent the semantics of these statements.

Time Constraints The following distinct form of time constraint were identified:

1. `Time is on or after 17:00:00`
2. `Time is between 00:00:00 and 16:59:00`

Form 1 sets a lower bound on the current time, and form 2 sets both a lower and upper bound on the current time. As the behaviour database is periodically updated with new control rules it is assumed that a similar rule might eventually be defined which only imposes an upper bound on the current time.

11.4 Identified Action Rule Features

The following features of action rules were identified:

- Propositional Value Assignment

  A truth value is assigned to a given propositional variable.

  Example: `SET ::506::GOAL-gotoCharger TO false`

- Turn on Light

  The light on the Care-O-bot is set to the given colour.

  Example: `Turn light on ::0::Care-O-Bot 3.2 to yellow`

- Say Something

  The robot speaks a phrase to the occupier of the house.

  Example: `::0::Care-O-Bot 3.2 says 'Doorbell' and wait for completion`

- Wait

  Tells the robot to wait for n seconds.

  Example: `Wait for 1 seconds on ::0::Care-O-Bot 3.2`

- Move to Location

  Moves the robot from its current location to the given location.

  Example: `move ::0::Care-O-Bot 3.2 to ::999:: Current user Location`

- Change Tray Position

  Changes the position of the robot's tray.

  Example: `move tray on ::0::Care-O-Bot 3.2 to Raised`

- Set GUI Options

  Displays a given list of selectable options to be displayed on the robot's graphical user interface.

  Example: `::0::Care-O-Bot 3.2 GUI,S1-Set-ReturnHome,S1-Set-Continue`

- Execute Behaviour

  Executes the given behaviour on the robot.

  Example: `Execute sequence 'S1-sleep' on ::0::Care-O-Bot 3.2`

- Move Robot Torso

  Changes the position of the torso of the robot.

  Example: `move torso on ::0::Care-O-Bot 3.2 to the right`

- Wait for Completion

  A suffix applied to actions to indicate that the robot should wait for the action to complete before continuing.

  Example: `and wait for completion`

11.5 Action Rules Grouped by Features

Propositional Value Assignment

```
SET ::506::GOAL-gotoCharger TO false
SET ::506::GOAL-gotoCharger TO false
SET ::508::GOAL-gotoSofa TO true
SET ::507::GOAL-gotoTable TO false
SET ::505::GOAL-gotoKitchen TO true
SET ::512::GOAL-waitHere TO false
SET ::509::GOAL-waitAtKitchen TO true
SET ::511::GOAL-waitAtTable TO false
SET ::510::GOAL-waitAtSofa TO true
SET ::513::GOAL-watchTV TO true
SET ::500::TrayIsRaised TO false
SET ::501::TrayIsLowered TO true
SET ::504::TrayIsEmpty TO true
SET ::502::5PM-MedicineDue TO true
SET ::503::5PM-MedicineReminder TO false
SET ::514::GOAL-fridgeUserAlerted TO false
SET ::515::GOAL-AnserDoorBell TO false
```

Turn on Light

```
Turn light on ::0::Care-O-Bot 3.2 to white
Turn light on ::0::Care-O-Bot 3.2 to yellow
```

Turn on Light, Wait for Completion

```
Turn light on ::0::Care-O-Bot 3.2 to white and wait for completion
Turn light on ::0::Care-O-Bot 3.2 to yellow and wait for completion
```

Say Something, Wait for Completion

```
::0::Care-O-Bot 3.2 says 'Shall we watch TV together?'  and wait for
completion
::0::Care-O-Bot 3.2 says 'Have you taken your medicine' and wait for
completion
::0::Care-O-Bot 3.2 says 'Its time for your medicine' and wait for
completion
::0::Care-O-Bot 3.2 says 'Doorbell' and wait for completion
::0::Care-O-Bot 3.2 says 'The fridge door is open!'  and wait for
completion
```

Wait

```
Wait for 5 seconds on ::0::Care-O-Bot 3.2
Wait for 1 seconds on ::0::Care-O-Bot 3.2
```

Move to Location

```
move ::0::Care-O-Bot 3.2 to ::31::  TV location in the Living Room
move ::0::Care-O-Bot 3.2 to ::999::  Current user Location
```

Move to Location, Wait for Completion

```
move ::0::Care-O-Bot 3.2 to ::14::  Living Room Sofa Area in the Living
Room and wait for completion
move ::0::Care-O-Bot 3.2 to ::7::  Kitchen Entrance in the Dining Room
and wait for completion
move ::0::Care-O-Bot 3.2 to ::5::  ChargingStation Area in the Dining Room
and wait for completion
move ::0::Care-O-Bot 3.2 to ::2::  Living Room and wait for completion
move ::0::Care-O-Bot 3.2 to ::23::  Living Room Table in the Living Room
Sofa Area of the Living Room and wait for completion
```

Change Tray Position, Wait for Completion

```
move tray on ::0::Care-O-Bot 3.2 to Lowered and wait for completion
move tray on ::0::Care-O-Bot 3.2 to Raised and wait for completion
```

Set GUI Options

```
::0::Care-O-Bot 3.2 GUI,S1-Set-GoToKitchen,S1-Set-ReturnHome,
S1-Set-WaitHere
::0::Care-O-Bot 3.2 GUI,S1-Set-WaitHere,S1-Set-ReturnHome,S1-Set-Continue
::0::Care-O-Bot 3.2 GUI,S1-set-gotoSofa,S1-set-gotoTable,S1-Set-Continue,
S1-Set-WaitHere
::0::Care-O-Bot 3.2 GUI,S1-Set-ReturnHome,S1-Set-WaitHere,S1-Set-Continue
::0::Care-O-Bot 3.2 GUI,S1-Set-GoToKitchen,S1-Set-WaitHere
::0::Care-O-Bot 3.2 GUI,S1-Set-ReturnHome,S1-Set-Continue
::0::Care-O-Bot 3.2 GUI,S1-Set-Watch-TV,S1-Set-ReturnHome,S1-Set-Continue
```

Execute Behaviour

```
Execute sequence 'lowerTray' on ::0::Care-O-Bot 3.2
Execute sequence 'S1-sleep' on ::0::Care-O-Bot 3.2
Execute sequence 'raiseTray' on ::0::Care-O-Bot 3.2
Execute sequence 'T-moveTo-person' on ::0::Care-O-Bot 3.2
```

Move Robot Torso, Wait for Completion

```
move torso on ::0::Care-O-Bot 3.2 to the right and wait for completion
move torso on ::0::Care-O-Bot 3.2 to the back position and wait for
completion
```

11.6 Categorization of Precondition Rules

Propositional Value Assignment The following form of propositional value checks was identified:

1. `SET ::506::GOAL-gotoCharger TO false`

A truth value is assigned to a given propositional variable.

Enumeration Value Assignment The following distinct forms of enumeration value assignments were identified:

1. `Turn light on ::0::Care-O-Bot 3.2 to white`
2. `::0::Care-O-Bot 3.2 says 'The fridge door is open!'`
3. `move ::0::Care-O-Bot 3.2 to ::31::  TV location in the Living Room`
4. `move tray on ::0::Care-O-Bot 3.2 to Lowered`
5. `move torso on ::0::Care-O-Bot 3.2 to the right`

Form 1 sets the state of the light on the Care-O-bot , form 2 states that the robot should vocalize a statement, form 3 tells the robot to move to a given location, form 4 changes the position of the robot's tray and form 5 moves the torso of the robot to a given position. All of these forms can be adequately represented using an enumerated type for the following reasons: the robot has a single light which can be at most one colour at a time, only one statement can be vocalized at any time, the robot can only move to one location at any time, the tray can only be in one position at any time, and the robot's torso can only be moved to one new position at any time.

Non-Deterministic Behaviour Executions The following form of non-deterministic behaviour execution was identified:

1. `::0::Care-O-Bot 3.2 GUI,S1-Set-WaitHere,S1-Set-ReturnHome,`
   `S1-Set-Continue`

Here a list of behaviours represent the choices that are displayed on the robot's GUI. In any constructed model the behaviour to be executed will be assigned non-deterministically as the house occupant may chose any of the options.

Behaviour Executions The following form of behaviour executions was identified:

1. `Execute sequence 'S1-sleep' on ::0::Care-O-Bot 3.2`

Here control is transferred from one behaviour to another, and then returned upon completion of the executed behaviour.

Delays The following form of delays was identified:

1. `Wait for 5 seconds on ::0::Care-O-Bot 3.2`

The robot is told to wait for a given number of seconds.

11.7 Predefined Non-Terminal Symbols

These predefined symbols are included in the grammar decribed in Section 3.2.3:

- ⟨**integer**⟩

  Accepts any integer as input.

- ⟨**float**⟩

  Accepts any real floating-point number as input.

- ⟨**relational_operator**⟩

  Accepts any of >, <, =, ==, >= and <= as input.

- ⟨**time**⟩

  Accepts as input any valid time in one of the formats HH:MM:SS, HH:MM, or HH.

- ⟨**any_text**⟩

  This symbol accepts any token of text as valid input.

11.8 Grammar Rules

Figures 20 and 21 show the grammar rules used to respectively parse the precondition rule and action rules extracted from the University of Hertfordshire database.

Figure 20: Backus-Naur Form Definitions for Preconditions

⟨*pvc1*⟩ ::= +⟨*any_text*⟩ 'is' ⟨*any_text*⟩
⟨*pvc2*⟩ ::= +⟨*any_text*⟩ ⟨*relational_operator*⟩ ⟨*integer*⟩
⟨*pvc3*⟩ ::= +⟨*any_text*⟩ 'IN' ⟨*any_text*⟩
⟨*evc1*⟩ ::= '::0::Care-O-Bot' '3.2' 'location' 'is' +⟨*any_text*⟩
⟨*evc2*⟩ ::= 'Living' 'room' 'sofa' 'seat' ⟨*integer*⟩ 'is' 'occupied'
⟨*tc1*⟩ ::= 'Time' 'is' 'on' 'or' 'after' ⟨*time*⟩
⟨*tc2*⟩ ::= 'Time' 'is' 'between' ⟨*time*⟩ 'and' ⟨*time*⟩

Figure 21: Backus-Naur Form Definitions for Actions

⟨*pva1*⟩ ::= 'SET' +⟨*any_text*⟩ 'TO' ⟨*any_text*⟩
⟨*eva1*⟩ ::= 'Turn' 'light' 'on' '::0::Care-O-Bot' '3.2' 'to' ⟨*any_text*⟩
⟨*eva2*⟩ ::= '::0::Care-O-Bot' '3.2' 'says' +⟨*any_text*⟩
⟨*eva3*⟩ ::= 'move' '::0::Care-O-Bot' '3.2' 'to' +⟨*any_text*⟩
⟨*eva4*⟩ ::= 'move' 'tray' 'on' '::0::Care-O-Bot' '3.2' 'to' ⟨*any_text*⟩
⟨*eva5*⟩ ::= 'move' 'torso' 'on' '::0::Care-O-Bot' '3.2' 'to' +⟨*any_text*⟩
⟨*ebnd1*⟩ ::= '::0::Care-O-Bot' '3.2' 'GUI' +⟨*any_text*⟩
⟨*exb1*⟩ ::= 'Execute' 'sequence' ⟨*any_text*⟩ 'on' '::0::Care-O-Bot' '3.2'
⟨*d1*⟩ ::= 'Wait' 'for' ⟨*integer*⟩ 'seconds' 'on' '::0::Care-O-Bot' '3.2'

11.9 Data Extraction Rules

Eight different formats of data extraction rule are given here, one for each of the three types of precondition rule and one for each of the five types of action rule.

11.9.1 Propositional Value Check Data Extraction Rule

Format

rule_name; prop_name = *identifier*; truth_value = *identifier*; true = *string*; false = *string*; non_deterministic = *boolean*.

Variable Definitions

1. prop_name: a name for the propositional variable
2. truth_value: the symbol denoting the truth value
3. true: a string denoting the representation of *true* for the rule.
4. false: a string denoting the representation of *false* for the rule.
5. non_deterministic: this is set to true if this propositional variable represents the state of something external to the robot. Setting this to true implies that the truth value and definitions for *true* and *false* should be set to *null*.

Examples

1. **Grammar Rule**
   ⟨*pvc1*⟩ ::= +⟨*any_text*⟩ 'is' ⟨*any_text*⟩

   Data Extraction Rule
   pvc1; prop_name = [1]; truth_value = [3]; true = "true"; false = "false"; non_deterministic = false.

   For rule *pvc1* we have the propositional variable name as the first symbol in the rule definition, the truth value as the third symbol, true as "true", false as "false" and non-determinism set to false.

2. **Grammar Rule**
   ⟨*pvc2*⟩ ::= +⟨*any_text*⟩⟨*relational_operator*⟩⟨*integer*⟩

   Data Extraction Rule
   pvc2; prop_name = [1, 2, 3]; truth_value = null; true = null; false = null; non_deterministic = true;

   For rule *pvc2* we have the propositional variable name as the concatenation of symbols 1, 2 and 3, the truth value as null, true as null, false as null, and non-determinism set to true.

11.9.2 Enumeration Value Check Data Extraction Rule

Format

rule_name; enum_name = *identifier*; enum_value = *identifier*; has_none_value = *boolean*; non_deterministic = *boolean*;

Variable Definitions

1. enum_name: a name for the enumerated type
2. enum_value: a name for the enumeration value
3. has_none_value: set to true if an extra value *none* should be added to the list of possible values for the enumerated type.
4. non_deterministic: this is set to true if this enumerated variable represents the state of something external to the robot.

Examples

1. **Grammar Rule**
   ⟨*evc1*⟩ ::= '::0::Care-O-Bot' '3.2' 'location' 'is' +⟨*any_text*⟩

   Data Extraction Rule
   evc1; enum_name = [3]; enum_value = [5]; has_none_value = false; non_deterministic = false;

   For rule *evc1* we have the enumeration name as 'location' and the enumeration value as any number of ⟨*any_text*⟩ symbols. The enumerated variable should have no additional *value* and does not represents the state of something external to the robot.

2. **Grammar Rule**
   ⟨*evc2*⟩ ::= 'Living' 'room' 'sofa' 'seat' ⟨*integer*⟩ 'is' 'occupied'

   Data Extraction Rule
   evc2; enum_name = [4, 7]; enum_value = [5]; has_none_value = true; non_deterministic = true;

   For rule *evc2* we have the enumeration name as the concatenation of the 'seat' and 'occupied', and the enumeration value as the symbol ⟨*integer*⟩. This enumerated variable does represent the state of something external to the robot and additional possible value, *none*, as no living room sofa seats may be occupied.

11.9.3 Time Constraint Data Extraction Rule

Format

rule_name; start_time = *identifier*; end_time = *identifier*.

Variable Definitions

1. start_time: the start time.
2. end_time: the end time.

Examples

1. **Grammar Rule**
   ⟨*tc1*⟩ ::= 'Time' 'is' 'on' 'or' 'after' ⟨*time*⟩

   Data Extraction Rule
   tc1; start_time = [6]; end_time = null.

   For rule *tc1* we have the start time as the symbol ⟨*time*⟩, and a null value for end time. This would imply that we are checking to see if the time is *after* a certain point.

2. **Grammar Rule**
   ⟨*tc2*⟩ ::= 'Time' 'is' 'between' ⟨*time*⟩ 'and' ⟨*time*⟩.

   Data Extraction Rule
   tc2; start_time = [4]; end_time = [6].

   For rule *t2* we have the start time as the ⟨*time*⟩ symbol at position 4, and the end time as the ⟨*time*⟩ symbol at position 6. This would imply that we are checking to see if the time is *within* a given range.

If a rule is defined as having an end time but no start time this would imply that we are checking to see if the time is *before* a certain point.

11.9.4 Propositional Value Assignment Data Extraction Rule

Format

rule_name; prop_name = *identifier*; truth_value = *identifier*; true = *string*; false = *string*;

Variable Definitions

1. prop_name: the name of the propositional variable
2. truth_value: the symbol denoting the truth value
3. true: a string denoting the representation of *true* for the rule.
4. false: a string denoting the representation of *false* for the rule.

Examples

1. **Grammar Rule**
   ⟨*pva1*⟩ ::= 'SET' +⟨*any_text*⟩ 'TO' ⟨*any_text*⟩

   Data Extraction Rule
   pva1; prop_name = [2]; truth_value = [4]; true = "true"; false = "false".

   For rules *pva1* we have the propositional variable name as the second symbol in the rule definition (any number of ⟨*any_text*⟩ symbols which will be concatenated with underscores), the truth value as the fourth symbol, true as "true" and false as "false".

11.9.5 Enumeration Value Assignment Data Extraction Rule

Format

rule_name; enum_name = *identifier*; enum_value = *identifier*; resets = *boolean*.

Variable Definitions

1. enum_name: the name of the enumerated type
2. enum_value: the name of the enumeration value
3. resets: this is set to true if whenever the value of the enumeration is set to any value, it should then reset back to an additional *none* value in the next state. For instance if in the next state the robot says *"Hello!"*, then in the following state the robot should not be saying *"Hello!"*, unless of course this is intentional.

Examples

1. **Grammar Rule**
   ⟨*eva1*⟩ ::= 'Turn' 'light' 'on' '::0::Care-O-Bot' '3.2' 'to' ⟨*any_text*⟩

   Data Extraction Rule
   eva1; enum_name = [2]; enum_value = [7]; resets = false.

   For rule *eva1* we have the enumeration name as 'light' and the enumeration value as the seventh symbol, ⟨*any_text*⟩.

2. **Grammar Rule**
   ⟨*eva3*⟩ ::= 'move' '::0::Care-O-Bot' '3.2' 'to' +⟨*any_text*⟩

   Data Extraction Rule
   eva3; enum_name = "location"; enum_value = [5]; resets = false.

   For rule *eva3* we have the enumeration name as the given string "location", and the enumeration value as the fifth symbol (any number of ⟨*any_text*⟩ symbols).

11.9.6 Non-Deterministic Behaviour Execution Data Extraction Rule

Format

rule_name; behaviour_values = *identifier*; split_character = *string*

Variable Definitions

1. behaviour_values: the list of behaviours.
2. split_character: the string by used to split the list of behaviours into separate behaviours.

Examples

1. **Grammar Rule**
   ⟨*ebnd1*⟩ ::= '::0::Care-O-Bot' '3.2' 'GUI' +⟨*any_text*⟩

   Data Extraction Rule
   evand1; behaviour_values = [4]; split_character = ','.

   For rule *ebnd1* we have the list of behaviours as symbol 4 (any number of ⟨*any_text*⟩ symbols), and the separating character as ','.

11.9.7 Behaviour Execution Data Extraction Rule

Format

rule_name; behaviour_name = *identifier*;

Variable Definitions

1. behaviour_name: the name of the behaviour to be executed

Examples

1. **Grammar Rule**
   ⟨*exb1*⟩ ::= 'Execute' 'sequence' ⟨*any_text*⟩ 'on' '::0::Care-O-Bot' '3.2'

   Data Extraction Rule
   exb1; behaviour_name = [3].

   For rule *exb1* we have the behaviour name as the third symbol, ⟨*any_text*⟩.

11.9.8 Delay Data Extraction Rule

Format

rule_name; seconds = *identifier*;

Variable Definitions

1. seconds: the number of seconds to delay

Examples

1. **Grammar Rule**
   ⟨*d1*⟩ ::= 'Wait' 'for' ⟨*integer*⟩ 'seconds' 'on' '::0::Care-O-Bot' '3.2'

   Data Extraction Rule
   d1; seconds = [3].

   For rule *d1* we have the number of seconds as the third symbol, ⟨*integer*⟩.

11.10 NuSMV Variable Assignments

The following section gives full definitions of how values will be assigned to each of the *step*, *schedule* and *last_schedule* variables. Given a set of intermediate form beaviours $\mathcal{B}$:

step

The initial value for *step* is always set to *step_none* as no behaviour is scheduled in the initial state of the model.

The value for *step* in the next state is determined by the following ordered sequence of expressions. First the expression:

```
a_behaviour_can_be_scheduled:  step_1
```

states that if the macro *a_behaviour_can_be_scheduled* evaluates to true, then in the next state *step* has the value *step_1*. Intuitively, *a_behaviour_can_be_scheduled* evaluates to true if in the next state a behaviour can be scheduled for execution, either because no behaviour can be scheduled or because a behaviour can interrupt the currently scheduled behaviour.

For each behaviour $b \in \mathcal{B}$ that has a *behaviour execution* action or *non-deterministic behaviour execution* action as its n^{th} action an expression of the form:

```
(b_N.is_executing & step = step_n):  step_1
```

is added, where `N` is the name of b and `n` is the index of the executing action in the list of actions. The macro *b_N.is_executing* evaluates to true if b is currently scheduled for execution, therefore the expression intuitively means that if b is scheduled and is performing its n^{th} action, then in the next state *step* will have the value *step_1* as control will have been passed to some executed behaviour $b' \neq b \in \mathcal{B}$ which will then be performing its first action.

For each pair of behaviours $b, b' \in \mathcal{B}$ where b' is executed by b, an expression of the form:

```
(b_N'.is_last_step & last_schedule = schedule_N): step_n
```

is added, where `N` is the name of b, `N'` is the name of b' and n is the index of some action in b, where the $(n-1)^{th}$ action in b executed the behaviour b'. The macro *b_N'.is_last_step* evaluates to true if b' is performing its final action. This expression intuitively means that if the executed behaviour b' is performing its last action then in the next state control will have passed back to the executing behaviour b which will then be performing its next action.

The expressions:

```
a_behaviour_is_ending:  step_none
```

and

```
an_executed_behaviour_is_ending_as_a_last_action:  step_none
```

are then added. The macro *a_behaviour_is_ending* evaluates to true if the scheduled behaviour is performing its last action and in the next moment of time no other behaviour is scheduled. The macro *an_executed_behaviour_is_ending_as_a_last_action* evaluates to true if some behaviour $b' \in \mathcal{B}$ executed by some other behaviour $b \in \mathcal{B}$ is performing its last action, and b has no more actions to perform. In either case in the next state no behaviour is scheduled so the value of *step* is set to *step_none*.

As stated in Section 3.3.1 *step* has values *step_none* plus valus *step_1*,...,*step_k*, where $k = \max\{\text{numberOfActions}(b) \mid b \in \mathcal{B}\}$. For each $2 \leq i \leq k$ an expression is added of the form:

```
step = step_(i - 1):  step_i;
```

then finally an expression of the form:

```
step = step_i:  step_none;.
```

These expressions increment the current step until the last step is reached.

schedule

The initial value for *schedule* is always set to *schedule_none* as no behaviour is scheduled in the initial state of the model.

The value for *schedule* in the next state is determined by the following ordered sequence of expressions. For every schedulable behaviour $b \in \mathcal{B}$ an expression of the form:

```
b_N.can_be_executed:  schedule_N
```

is added, where `N` is the name of b. The macro *b_N.can_be_executed* evaluates to true if b can be scheduled in the next state as either no behaviour is currently scheduled or b can interrupt the currently scheduled behaviour. These expressions are ordered such that the first expression corresponds to the schedulable behaviour with the highest priority and the last expression corresponds to the schedulable behaviour with the lowest priority. This ensures that if two or more candidate behaviours can be scheduled for execution in the next moment in time then the behaviour having the highest priority of all candidate behaviours will be scheduled.

For every behaviour $b \in \mathcal{B}$ if the n^{th} action of b is a *non-deterministic behaviour execution* action then an expression of the form:

```
(b_N.is_executing & step = step_n):  {value_list}
```

is added, where `N` is the name of b and `value_list` is the comma separated list of all behaviours that can be non-deterministically selected for execution by the n^{th} action of b.

For every behaviour $b \in \mathcal{B}$ if the n^{th} action of b is a *non-deterministic behaviour execution* action then an expression of the form:

```
(b_N.is_executing & step = step_n):  {value_list}
```

is added, where `N` is the name of b and `value_list` is the comma separated list of all behaviours that can be non-deterministically selected for execution by the n^{th} action of b.

For every behaviour $b \in \mathcal{B}$ if the n^{th} action of b is a *behaviour execution* action then an expression of the form:

```
(b_N.is_executing & step = step_n
& b_N'.preconditions_hold):  schedule_N'
```

is added, where `N` is the name of b and `N'` is the name of b'. The macro *b_N'.preconditions_hold* evaluates to true if the precondition rules for b hold. Intuitively this statement means that when b executes b' in the next state b' is scheduled iff the precondition rules for b' hold.

Finally the expressions shown in Figure 22 are added. Lines 1, 2 and 4 reset *schedule* to *schedule_none* if no behaviour will be scheduled in the next moment in time. Line 3 sets the value of *schedule* to the value of *last_schedule* if some executed behaviour $b \in \mathcal{B}$ is ending, returning control to the behaviour that executed b. Line 6 ensures that if none of the previous expressions evaluate to true then *schedule* keeps its value in the next moment in time.

Figure 22: *schedule* Assignments

```
1. a_behaviour_is_ending:  schedule_none;
2. an_executed_behaviour_is_ending_as_a_last_action:  schedule_none;
3. an_executed_behaviour_is_ending & last_schedule != schedule):
   last_schedule;
4. an_executed_behaviour_is_ending:  schedule_none;
5.
6. TRUE: schedule;
```

last_schedule

The initial value for *last_schedule* is always set to *schedule_none* as no behaviour has been scheduled in the initial state of the model.

The expressions shown in Figure 23 determine the value of *last_schedule* in the next moment in time. The expression on line 5 assigns the value of *schedule* to *last_schedule* in the next moment in time, 'remembering' what behaviour was scheduled in the previous state. Lines 3 and 4 force *last_schedule* to retain its value in any state where some behaviour is being executed in this moment in time by the current behaviour and will be scheduled in the next moment in time, or if some executed behaviour is already scheduled. This ensures that when a behaviour is executed the executing behaviour is remembered.

Figure 23: *last_schedule* Assignments

```
1. next(last_schedule):=
2.     case
3.         executed_behaviour_execute_next:  last_schedule;
4.         an_executed_behaviour_is_executing:  last_schedule;
5.         TRUE: schedule;
6.     esac;
```

11.11 Main Module Macro Definitions

These macros are used to succinctly express more complex logical expressions, increasing the readability of the code, decreasing the size of the code, and allowing properties checked in the model to be expressive yet easy to formulate.

Preconditions Given a set of intermediate form behaviours $\mathcal{B}$, for each behaviour $b \in \mathcal{B}$ a macro is added of the form:

`pre_N:= expression`

where `N` is the name of b and `expression` is a logical expression that evaluates to true iff the precondition rules for b hold at some moment in time. If a behaviour b has no preconditions then a macro of the form:

`pre_N:= TRUE`

is added instead, as the preconditions of b will always hold. Figure 24 shows the logical expression tree for the *lowerTray* behaviour. The corresponding macro for this behaviour is:

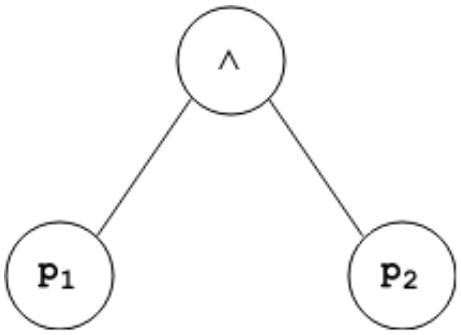
`pre_lowerTray:= (__504__TrayIsEmpty & __500__TrayIsRaised`

Explicit value checks of the form *variable* = *value* are omitted for propositional variable checks, instead using *variable* for *variable* = true and !*variable* for *variable* = false. Expressions corresponding to *time constraints* are of the form:

`time =` $\mathtt{t}_1$ `|` ... `|` $\mathtt{t}_n$

where each $\mathtt{t}_i$ corresponds to a period of time during which the precondition rule holds.

Figure 24: Logical expression tree for the *lowerTray* behaviour



p1: `::500::TrayIsRaised is true`
p2: `::504::TrayIsEmpty is true`

Interrupts Given a set of intermediate form behaviours $\mathcal{B}$, and a set of behaviour priorities $\mathcal{P} = \{\text{priority}(b) \mid b \in \mathcal{B} \text{ and schedulable}(b)\}$, for every $p \in \mathcal{P}$ a macro of the form:

`can_interrupt_p`

is defined; intuitively each macro evaluates to true if any behaviour $b \in \mathcal{B}$ with priority$(b) = p$ could interrupt any currently scheduled behaviour, as long as its preconditions hold. Initially a macro is defined for $p = \min(\mathcal{P})$ of the form:

`can_interrupt_p:= FALSE`

For any schedulable behaviour $b \in \mathcal{B}$ with priority$(b) = \min(\mathcal{P})$ it will never be the case that b can interrupt any other behaviour in $\mathcal{B}$. Then, given an ordered list of all priority values $p_1, \ldots, p_n \in \mathcal{P}$, where p_1 is the lowest priority, for every p_i with $2 \leq i \leq n$ a macro is defined of the form:

`can_interrupt_p`$_i$`:= (can_interrupt_p`$_{i-1}$ `| expression)`

where `expression` is a (possibly empty) disjunction of the form:

`(schedule = schedule_N`$_1$ `| ... | schedule = schedule_N`$_k$`)`

where each `N`$_j$ with $1 \leq j \leq k$, is the name of some interruptible behaviour $b_j \in \mathcal{B}$ with priority$(b_j) = p_{i-1}$.

Behaviour Scheduling Macros Six additional macros that relate to the scheduling of behaviours are defined here. Given a set of intermediate form behaviours $\mathcal{B}$:

- The *executed_behaviour_execute_next* macro evaluates to true iff in the next moment in time some behaviour in $\mathcal{B}$ will be executed by the currently scheduled behaviour in $\mathcal{B}$. The macro is of the form:

  `executed_behaviour_execute_next:= expression`

  where `expression` is a (possibly empty) disjunction of expressions of the form $E_1 \mid \ldots \mid E_n$ and each $E_{1 \leq i \leq n}$ is an expression of the form:

  `(!b_N.can_be_interrupted & b_N.is_executing & step = step_k)`

  where `N` is the name of some behaviour $b \in \mathcal{B}$ that has a *behaviour execution* action or *non-deterministic behaviour execution* action as its k^{th} action. An expression is included for every distinct *behaviour_execution* action in every behaviour in $\mathcal{B}$. The (negated) macro *!b_N.can_be_interrupted* evaluates to true if the behaviour having the name `N` cannot be interrupted by some other behaviour in the next moment in time.

- The *a_behaviour_can_be_scheduled* macro evaluates to true iff in the next moment in time some behaviour in $\mathcal{B}$ can be scheduled for execution. The macro is of the form:

  `a_behaviour_can_be_scheduled:= expression`

  where `expression` is a disjunction of expressions of the form $E_1 \mid \ldots \mid E_n$ and each $E_{1 \leq i \leq n}$ is an expression of the form:

  `b_N.can_be_scheduled`

  where `N` is the name of some schedulable behaviour $b \in \mathcal{B}$. An expression is included for every distinct schedulable behaviour in $\mathcal{B}$. The macro *b_N.can_be_scheduled* evaluates to true if the behaviour having the name `N` can be scheduled for execution in the next moment in time.

- The *a_behaviour_is_ending* macro evaluates to true iff some schedulable behaviour in $\mathcal{B}$ is performing its final action. The macro is of the form:

  `a_behaviour_is_ending:= expression`

  where `expression` is a disjunction of expressions of the form $E_1 \mid \ldots \mid E_n$ and each $E_{1 \leq i \leq n}$ is an expression of the form:

  `b_N.is_last_step`

where `N` is the name of some schedulable behaviour $b \in \mathcal{B}$. An expression is included for every distinct schedulable behaviour in $\mathcal{B}$.

- The *an_executed_behaviour_is_ending_as_a_last_action* macro evaluates to true iff some some behaviour $b \in \mathcal{B}$, executed by another behaviour $b' \in \mathcal{B}$, is performing its last action, and b' executed b as its final action. The macro is of the form:

  ```
  an_executed_behaviour_is_ending_as_a_last_action:=
  expression
  ```

  where `expression` is a disjunction of expressions of the form $E_1 \mid \ldots \mid E_n$ and each $E_{1 \leq i \leq n}$ is an expression of the form:

  `(b_N1.is_last_step & last_schedule = schedule_N2)`

  where $\mathtt{N_1}$ is the name of some executed behaviour $b_1 \in \mathcal{B}$ and $\mathtt{N_2}$ is the name of the behaviour $b_2 \in \mathcal{B}$ that executed b_1. An expression is included for every distinct pair of behaviours where one behaviour executes the other as a final action.

- The *an_executed_behaviour_is_ending* macro evaluates to true iff some non-schedulable behaviour in $\mathcal{B}$ is performing its final action. The macro is of the form:

  ```
  an_executed_behaviour_is_ending:= expression
  ```

  where `expression` is a disjunction of expressions of the form $E_1 \mid \ldots \mid E_n$ and each $E_{1 \leq i \leq n}$ is an expression of the form:

  ```
  b_N.is_last_step
  ```

  where `N` is the name of some non-schedulable behaviour $b \in \mathcal{B}$. An expression is included for every distinct non-schedulable behaviour in $\mathcal{B}$.

- The *an_executed_behaviour_is_scheduled* macro evaluates to true iff some non-schedulable behaviour in $\mathcal{B}$ is currently scheduled. The macro is of the form:

  ```
  an_executed_behaviour_is_scheduled:= expression
  ```

  where `expression` is a disjunction of expressions of the form $E_1 \mid \ldots \mid E_n$ and each $E_{1 \leq i \leq n}$ is an expression of the form:

  ```
  b_N.is_scheduled
  ```

  where `N` is the name of some non-schedulable behaviour $b \in \mathcal{B}$. The macro *b_N.is_scheduled* evaluates to true if the behaviour having the name `N` is currently scheduled. An expression is included for every distinct non-schedulable behaviour in $\mathcal{B}$.

11.12 The Behaviour Module

The parameterizable behaviour module, shown in Figure 25, is instantiated once for each behaviour in the set of intermediate form behaviours $\mathcal{B}$, and consists of a number of macro definitions. This purpose of this module is simply to provide macro definitions for individual behaviours. These macro definitions can then be used to easily construct otherwise complex expressions that are used both to define the state of variables in the `ASSIGN` section of the main module, and to formulate properties to test in the model.

For each behaviour $b \in \mathcal{B}$ a statement is added to the `VAR` section of the main module of the form:

```
b_N: behaviour(parameter_list)
```

where `N` is the name of b and `parameter_list` is a comma separated list of values passed as parameters to the module.

Figure 25: The Behaviour Module

```
1. MODULE behaviour(preconditions, can_interrupt, can_be_int, schedule,
   this_schedule, step, last_step)
2.
3.    DEFINE
4.       preconditions_hold:= preconditions;
5.       can_be_scheduled:= ((schedule = schedule_none | can_interrupt)
   & preconditions_hold);
6.       can_be_interrupted:= can_be_int;
7.       is_last_step:= (is_scheduled & step = last_step);
8.       is_scheduled:= (schedule = this_schedule);
```

Parameters For an instance of the behaviour module corresponding to a behaviour $b \in \mathcal{B}$ the following values are passed as parameters:

- **preconditions**

  The macro definition `pre_N`, corresponding to the expression that evaluates to true iff the preconditions of b hold, where `N` is the name of b.

- **can_interrupt**

  If b is schedulable:

  The macro definition `can_interrupt_n`, corresponding to the expression that evaluates to true iff b can interrupt a currently scheduled behaviour, where n is the priority of b.

  If b is not schedulable:

  `FALSE`

- **can_be_int**

  If b is interruptible:

  A disjunction $E_1 \mid \ldots \mid E_n$ where each $E_{1 \leq i \leq n}$ is an expression of the form:

`b_N.can_be_scheduled`

where `N` is the name of some $b' \in \mathcal{B}$ with priority(b') > priority(b). An expression is included for every distinct behaviour in $\mathcal{B}$ having a higher priority than b.

If b is not interruptible:

`FALSE`

- **schedule**

  A direct reference to the *schedule* variable defined in the main module.

- **this_schedule**

  A value for the enumerated variable *schedule*:

  `schedule_N`

  where `N` is the name of b.

- **step**

  A direct reference to the *step* variable defined in the main module.

- **last_step**

  A value for the enumerated variable *step*:

  `step_k`

  where `k` is the number of actions in b.

Macro Definitions For an instance of the behaviour module corresponding to a behaviour $b \in \mathcal{B}$ the following macros are defined:

- **preconditions_hold**

  This macro evaluates to true iff the value passed as the parameter for *precondition* holds.

- **can_be_executed**

  This macro evaluates to true iff the preconditions of b hold, and either no behaviour is currently scheduled or b can interrupt some scheduled behaviour.

- **can_be_interrupted**

  This macro evaluates to true iff the value passed as the parameter for *can_be_int* holds.

- **is_last_step**

  This macro evaluates to true iff b is scheduled and the value of *step* is equal to *step_k*.

- **is_scheduled**

  This macro evaluates to true iff the value of *schedule* is equal to *this_schedule*.

11.13 Pseudocode for Key Parsing Methods

Algorithm 2 shows the procedure for parsing the file containing the grammar rules for each rule. After opening the file a set of non-terminal symbols $\mathcal{N}$ is created and any 'built-in' non-terminal symbol definitions are added to $\mathcal{N}$. For each line of input in the file the name of the non-terminal symbol is read, the next token '::-' is then skipped, then a new NonTerminalSymbol is created. The next token on the line is then read, and is checked to see if it is prefixed with '+' or an integer. If it is then the number of repetitions of valid input required for the resultant automaton is set to either 0 for '+', or the number itself if the prefix is an integer. If the token is a terminal symbol (in quotation marks) then a new TerminalSymbol is created and added to the end of the list of automata for the new NonTerminalSymbol. If the token is a non-terminal symbol then we know that this has been previously defined; as previously stated in Section 3.2.3, if a non-terminal symbol appears in the right side of a rule then it must have been previously defined. The non-terminal symbol is then refactored if necessary and added to the new NonTerminalSymbol.

Algorithm 2 Parses a file containing the grammatical forms of rules.

```
1: procedure ParseGrammarRules(F)
2:     open(F)
3:     N ← predefined nonterminal symbols
4:     while hasInput(F) do
5:         L ← getNextLine(F)
6:         non_t ← getNextToken(L)
7:         getNextToken(L)
8:         new ← new NonTerminalSymbol
9:         while hasNextToken(L) do
10:            t ← getNextToken(L)
11:            p ← getSymbolPrefix(t)
12:            if p = " + " then
13:                repeat ← 0
14:            else if isInteger(p) then
15:                repeat ← parseInteger(p)
16:            else
17:                repeat ← 1
18:            end if
19:            if isTerminalSymbol(t) then
20:                terminal ← newTerminalSymbol(getText(t), repeat)
21:                addAutomaton(new, terminal)
22:            else
23:                non_terminal ← getNonTerminalByName(N, getText(t))
24:                addAutomaton(new, copyAutomaton(non_terminal, repeat))
25:            end if
26:        end while
27:        N ← N ∪ {new}
28:    end while
29:    close(F)
30:    returnN
31: end procedure
```

The copyAutomaton procedure takes as input a non-terminal symbol and a value, *repeat* and returns a copy of the given automaton with the new value for *repeat*

Once a line has been fully parsed the new NonTerminalSymbol is added to $\mathcal{N}$. Finally, the file is closed and the set of non-terminal symbols is returned.

Algorithm 3 Parses a file containing a set of control rules.

```
 1: procedure ParseControlRules(F)
 2:     open(F)
 3:     B ← ∅
 4:     while hasInput(F) do
 5:         L ← getNextLine(F)
 6:         behaviour_name ← getNextToken(L)
 7:         rule_order ← getNextToken(L)
 8:         rule_type ← getNextToken(L)
 9:         if rule_type = "R" then
10:             if existsBehaviourWithName(B, behaviour_name) then
11:                 b ← getBehaviourByName(B, behaviour_name)
12:                 addPrecondition(b, parsePrecondition(L))
13:             else
14:                 new ← newBehaviour(behaviour_name)
15:                 addPrecondition(new, parsePrecondition(L))
16:                 B ← B ∪ {new}
17:             end if
18:         else
19:             if existsBehaviourWithName(B, behaviour_name) then
20:                 b ← getBehaviourByName(B, behaviour_name)
21:                 addAction(b, parseAction(L))
22:             else
23:                 new ← newBehaviour(behaviour_name)
24:                 addAction(new, parseAction(L))
25:                 B ← B ∪ {new}
26:             end if
27:         end if
28:     end while
29:     close(F)
30: end procedure
```

Algorithm 3 describes the procedure for parsing a file containing a set of control rules. After opening the file an empty set of behaviours $\mathcal{B}$ is created. Then for each line of input the name of the behaviour, the rule order and the rule type is read. If the rule is a precondition then the rest of the line is parsed as a precondition, and then either added to an existing behaviour having the parsed name, or a new behaviour is created, the precondition is added, then the behaviour is added to $\mathcal{B}$. The process is similar for actions, where the rest of the line is instead parsed as an action. The addPrecondition and addAction methods will correctly insert each precondition and action into the corresponding lists in the behaviour according to the parsed rule order.

12 Appendix D - Test Results

12.1 Intermediate Form Translation Results

12.1.1 Control Rule File

```
1 name  priority  interruptable schedulable ruleOrder ruleType
      andOrConnector  notConnector  ruleActionText
2 answerDoorBell  70  0 1 0 R 0 0 ::515:: GOAL-AnserDoorBell is true
3 answerDoorBell  70  0 1 1 A 0 0 ::0::Care-O-Bot 3.2 says 'Doorbell' and
      wait for completion
4 answerDoorBell  70  0 1 2 A 0 0 Wait for 5 seconds on ::0::Care-O-Bot 3.2
5 checkBell 80  0 1 1 R 0 0 Doorbell Last Wattage > 1 AND was in this state
      within the last 10 seconds
6 checkBell 80  0 1 2 A 0 0 SET ::515::GOAL-AnserDoorBell TO  true
7 lowerTray 0 0 0 9 R 0 0 ::500:: TrayIsRaised is true
8 lowerTray 0 0 0 10  R 0 0 ::504:: TrayIsEmpty is true
9 lowerTray 0 0 0 11  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  yellow
10 lowerTray 0 0 0 12  A 0 0 move tray on ::0::Care-O-Bot 3.2 to Lowered and
      wait for completion
11 lowerTray 0 0 0 13  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  white and
      wait for completion
12 lowerTray 0 0 0 14  A 0 0 SET ::500::TrayIsRaised TO  false
13 lowerTray 0 0 0 15  A 0 0 SET ::501::TrayIsLowered TO  true
14 raiseTray 0 0 0 13  R 0 0 ::501:: TrayIsLowered is true
15 raiseTray 0 0 0 14  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  yellow
16 raiseTray 0 0 0 15  A 0 0 move tray on ::0::Care-O-Bot 3.2 to Raised and
      wait for completion
17 raiseTray 0 0 0 16  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  white and
      wait for completion
18 raiseTray 0 0 0 17  A 0 0 SET ::500::TrayIsRaised TO  true
19 raiseTray 0 0 0 18  A 0 0 SET ::501::TrayIsLowered TO  false
20 S1-alertFridgeDoor  60  0 1 27  R 0 0 Fridge Freezer In *ON* AND has been
      in this state for more than 30 seconds
21 S1-alertFridgeDoor  60  0 1 31  R 0 0 ::514:: GOAL-fridgeUserAlerted is
      false
22 S1-alertFridgeDoor  60  0 1 32  A 0 0 Turn light on ::0::Care-O-Bot 3.2
      to  yellow
23 S1-alertFridgeDoor  60  0 1 34  A 0 0 move ::0::Care-O-Bot 3.2 to ::2::
      Living Room and wait for completion
24 S1-alertFridgeDoor  60  0 1 35  A 0 0 Turn light on ::0::Care-O-Bot 3.2
      to  white and wait for completion
25 S1-alertFridgeDoor  60  0 1 36  A 0 0 ::0::Care-O-Bot 3.2 says 'The
      fridge door is open!' and wait for completion
26 S1-alertFridgeDoor  60  0 1 37  A 0 0 SET ::506::GOAL-gotoCharger TO
      false
27 S1-alertFridgeDoor  60  0 1 38  A 0 0 SET ::507::GOAL-gotoTable TO  false
28 S1-alertFridgeDoor  60  0 1 39  A 0 0 SET ::508::GOAL-gotoSofa TO  false
29 S1-alertFridgeDoor  60  0 1 40  A 0 0 ::0::Care-O-Bot 3.2
      GUI,S1-Set-GoToKitchen,S1-Set-WaitHere
30 S1-alertFridgeDoor  60  0 1 41  A 0 0 SET ::514::GOAL-fridgeUserAlerted
      TO  true
31 S1-continueWatchTV  35  1 1 11  R 0 0 ::513:: GOAL-watchTV is true
32 S1-continueWatchTV  35  1 1 12  R 0 0 Television Wattage > 10
33 S1-continueWatchTV  35  1 1 13  A 0 0 Turn light on ::0::Care-O-Bot 3.2
      to  yellow
```

```
34 S1-continueWatchTV  35  1 1 21  A 0 0 Execute sequence 'lowerTray' on
      ::0::Care-O-Bot 3.2
35 S1-continueWatchTV  35  1 1 22  A 0 0 move ::0::Care-O-Bot 3.2 to ::31::
      TV location in the Living Room
36 S1-continueWatchTV  35  1 1 23  A 0 0 Turn light on ::0::Care-O-Bot 3.2
      to  white
37 S1-continueWatchTV  35  1 1 24  A 0 0 move torso on ::0::Care-O-Bot 3.2
      to the right and wait for completion
38 S1-continueWatchTV  35  1 1 25  A 0 0 move torso on ::0::Care-O-Bot 3.2
      to the back position and wait for completion
39 S1-continueWatchTV  35  1 1 26  A 0 0 ::0::Care-O-Bot 3.2
      GUI,S1-Set-ReturnHome,S1-Set-Continue
40 S1-goToKitchen  40  1 1 31  R 0 0 ::505:: GOAL-gotoKitchen is true
41 S1-goToKitchen  40  1 1 32  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to
      yellow
42 S1-goToKitchen  40  1 1 43  A 0 0 Execute sequence 'lowerTray' on
      ::0::Care-O-Bot 3.2
43 S1-goToKitchen  40  1 1 44  A 0 0 move ::0::Care-O-Bot 3.2 to ::7::
      Kitchen Entrance in the Dining Room and wait for completion
44 S1-goToKitchen  40  1 1 45  A 0 0 Execute sequence 'raiseTray' on
      ::0::Care-O-Bot 3.2
45 S1-goToKitchen  40  1 1 46  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to
      white
46 S1-goToKitchen  40  1 1 47  A 0 0 SET ::505::GOAL-gotoKitchen TO  false
47 S1-goToKitchen  40  1 1 48  A 0 0 SET ::509::GOAL-waitAtKitchen TO  true
48 S1-gotoSofa 40  1 1 12  R 0 0 ::508:: GOAL-gotoSofa is true
49 S1-gotoSofa 40  1 1 13  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  yellow
50 S1-gotoSofa 40  1 1 14  A 0 0 Execute sequence 'lowerTray' on
      ::0::Care-O-Bot 3.2
51 S1-gotoSofa 40  1 1 15  A 0 0 move ::0::Care-O-Bot 3.2 to ::14:: Living
      Room Sofa Area in the Living Room and wait for completion
52 S1-gotoSofa 40  1 1 16  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  white
      and wait for completion
53 S1-gotoSofa 40  1 1 17  A 0 0 SET ::508::GOAL-gotoSofa TO  false
54 S1-gotoSofa 40  1 1 18  A 0 0 SET ::510::GOAL-waitAtSofa TO  true
55 S1-gotoTable  40  1 1 8 R 0 0 ::507:: GOAL-gotoTable is true
56 S1-gotoTable  40  1 1 9 A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  yellow
57 S1-gotoTable  40  1 1 10  A 0 0 Execute sequence 'lowerTray' on
      ::0::Care-O-Bot 3.2
58 S1-gotoTable  40  1 1 11  A 0 0 move ::0::Care-O-Bot 3.2 to ::23:: Living
      Room Table in the Living Room Sofa Area of the Living Room and wait
      for completion
59 S1-gotoTable  40  1 1 12  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to
      white and wait for completion
60 S1-gotoTable  40  1 1 13  A 0 0 SET ::507::GOAL-gotoTable TO  false
61 S1-gotoTable  40  1 1 14  A 0 0 SET ::511::GOAL-waitAtTable TO  true
62 S1-kitchenAwaitCmd  40  1 1 14  R 0 0 ::0::Care-O-Bot 3.2 location is
      ::7:: Kitchen Entrance in the Dining Room
63 S1-kitchenAwaitCmd  40  1 1 15  R 0 0 ::509:: GOAL-waitAtKitchen is true
64 S1-kitchenAwaitCmd  40  1 1 16  A 0 0 ::0::Care-O-Bot 3.2
      GUI,S1-set-gotoSofa,S1-set-gotoTable,S1-Set-Continue,S1-Set-WaitHere
65 S1-kitchenAwaitCmd  40  1 1 17  A 0 0 SET ::509::GOAL-waitAtKitchen TO
      false
66 S1-Med-5PM  50  1 0 42  R 0 0 Time is on or after 17:00:00
67 S1-Med-5PM  50  1 0 43  R 0 0 ::502:: 5PM-MedicineDue is true
68 S1-Med-5PM  50  1 0 44  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  yellow
```

```
S1-Med-5PM  50  1 0 45  A 0 0 move ::0::Care-O-Bot 3.2 to ::14:: Living Room Sofa Area in the Living Room and wait for completion
S1-Med-5PM  50  1 0 46  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  white and wait for completion
S1-Med-5PM  50  1 0 47  A 0 0 ::0::Care-O-Bot 3.2 says 'Its time for your medicine' and wait for completion
S1-Med-5PM  50  1 0 48  A 0 0 ::0::Care-O-Bot 3.2 GUI,S1-Set-GoToKitchen,S1-Set-ReturnHome,S1-Set-WaitHere
S1-Med-5PM  50  1 0 49  A 0 0 SET ::502::5PM-MedicineDue TO  false
S1-Med-5PM  50  1 0 50  A 0 0 SET ::503::5PM-MedicineReminder TO  true
S1-Med-5PM-Remind 50  1 1 11  R 0 0 ::503:: 5PM-MedicineReminder is true AND has been in this state for more than 60 seconds
S1-Med-5PM-Remind 50  1 1 12  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to yellow
S1-Med-5PM-Remind 50  1 1 13  A 0 0 move ::0::Care-O-Bot 3.2 to ::14:: Living Room Sofa Area in the Living Room and wait for completion
S1-Med-5PM-Remind 50  1 1 14  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to white and wait for completion
S1-Med-5PM-Remind 50  1 1 15  A 0 0 ::0::Care-O-Bot 3.2 says 'Have you taken your medicine' and wait for completion
S1-Med-5PM-Remind 50  1 1 16  A 0 0 SET ::503::5PM-MedicineReminder TO false
S1-Med-5PM-Reset  90  0 1 7 R 0 0 Time is between 00:00:00 and 16:59:00
S1-Med-5PM-Reset  90  0 1 10  R 0 0 ::502:: 5PM-MedicineDue is false
S1-Med-5PM-Reset  90  0 1 11  A 0 0 SET ::502::5PM-MedicineDue TO  true
S1-Med-5PM-Reset  90  0 1 12  A 0 0 SET ::503::5PM-MedicineReminder TO false
S1-remindFridgeDoor 80  0 1 0 R 0 0 ::514:: GOAL-fridgeUserAlerted is true AND has been in this state for more than 300 seconds
S1-remindFridgeDoor 80  0 1 1 A 0 0 SET ::514::GOAL-fridgeUserAlerted TO false
S1-ResetAllGoals  0 0 0 19  A 0 0 SET ::500::TrayIsRaised TO  false
S1-ResetAllGoals  0 0 0 20  A 0 0 SET ::501::TrayIsLowered TO  true
S1-ResetAllGoals  0 0 0 21  A 0 0 SET ::502::5PM-MedicineDue TO  true
S1-ResetAllGoals  0 0 0 22  A 0 0 SET ::503::5PM-MedicineReminder TO false
S1-ResetAllGoals  0 0 0 23  A 0 0 SET ::504::TrayIsEmpty TO  true
S1-ResetAllGoals  0 0 0 24  A 0 0 SET ::505::GOAL-gotoKitchen TO  false
S1-ResetAllGoals  0 0 0 25  A 0 0 SET ::506::GOAL-gotoCharger TO  false
S1-ResetAllGoals  0 0 0 26  A 0 0 SET ::507::GOAL-gotoTable TO  false
S1-ResetAllGoals  0 0 0 27  A 0 0 SET ::508::GOAL-gotoSofa TO  false
S1-ResetAllGoals  0 0 0 28  A 0 0 SET ::509::GOAL-waitAtKitchen TO  false
S1-ResetAllGoals  0 0 0 29  A 0 0 SET ::510::GOAL-waitAtSofa TO  false
S1-ResetAllGoals  0 0 0 30  A 0 0 SET ::511::GOAL-waitAtTable TO  false
S1-ResetAllGoals  0 0 0 31  A 0 0 SET ::512::GOAL-waitHere TO  false
S1-ResetAllGoals  0 0 0 32  A 0 0 SET ::513::GOAL-watchTV TO  false
S1-ResetAllGoals  0 0 0 33  A 0 0 SET ::514::GOAL-fridgeUserAlerted TO false
S1-ReturnHome 40  1 1 15  R 0 0 ::506:: GOAL-gotoCharger is true
S1-ReturnHome 40  1 1 16  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to yellow
S1-ReturnHome 40  1 1 17  A 0 0 Execute sequence 'lowerTray' on ::0::Care-O-Bot 3.2
S1-ReturnHome 40  1 1 18  A 0 0 move ::0::Care-O-Bot 3.2 to ::5:: ChargingStation Area in the Dining Room and wait for completion
S1-ReturnHome 40  1 1 19  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to
```

```
      white and wait for completion
107 S1-ReturnHome 40  1 1 20  A 0 0 SET ::506::GOAL-gotoCharger TO  false
108 S1-Set-Continue 0 0 0 1 A 0 0 SET ::512::GOAL-waitHere TO  false
109 S1-Set-Continue 0 0 0 2 A 0 0 SET ::513::GOAL-watchTV TO  false
110 S1-Set-GoToKitchen  0 0 0 3 A 0 0 SET ::505::GOAL-gotoKitchen TO  true
111 S1-Set-GoToKitchen  0 0 0 4 A 0 0 SET ::512::GOAL-waitHere TO  false
112 S1-Set-GoToSofa 0 0 0 2 A 0 0 SET ::508::GOAL-gotoSofa TO  true
113 S1-Set-GoToTable  0 0 0 2 A 0 0 SET ::507::GOAL-gotoTable TO  true
114 S1-Set-ReturnHome 0 0 0 6 A 0 0 SET ::506::GOAL-gotoCharger TO  true
115 S1-Set-ReturnHome 0 0 0 7 A 0 0 SET ::512::GOAL-waitHere TO  false
116 S1-Set-ReturnHome 0 0 0 8 A 0 0 SET ::513::GOAL-watchTV TO  false
117 S1-Set-WaitHere 0 0 0 1 A 0 0 SET ::512::GOAL-waitHere TO  true
118 S1-Set-Watch-TV 0 0 0 2 A 0 0 SET ::513::GOAL-watchTV TO  true
119 S1-Set-Watch-TV 0 0 0 3 A 0 0 SET ::512::GOAL-waitHere TO  false
120 S1-sleep  10  1 1 0 A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  white
121 S1-sleep  10  1 1 5 A 0 0 Wait for 1 seconds on ::0::Care-O-Bot 3.2
122 S1-sleep  10  1 1 6 A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  yellow
123 S1-sleep  10  1 1 7 A 0 0 Wait for 1 seconds on ::0::Care-O-Bot 3.2
124 S1-sleep  10  1 1 8 A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  white
125 S1-sofaAwaitCmd 40  1 1 15  R 0 0 ::0::Care-O-Bot 3.2 location is ::14::
      Living Room Sofa Area in the Living Room
126 S1-sofaAwaitCmd 40  1 1 16  R 0 0 ::510:: GOAL-waitAtSofa is true
127 S1-sofaAwaitCmd 40  1 1 17  A 0 0 ::0::Care-O-Bot 3.2
      GUI,S1-Set-ReturnHome,S1-Set-WaitHere,S1-Set-Continue
128 S1-sofaAwaitCmd 40  1 1 18  A 0 0 SET ::510::GOAL-waitAtSofa TO  false
129 S1-tableAwaitCmd  40  1 1 8 R 0 0 ::0::Care-O-Bot 3.2 location is ::23::
      Living Room Table in the Living Room Sofa Area of the Living Room
130 S1-tableAwaitCmd  40  1 1 9 R 0 0 ::511:: GOAL-waitAtTable is true
131 S1-tableAwaitCmd  40  1 1 10  A 0 0 ::0::Care-O-Bot 3.2
      GUI,S1-Set-ReturnHome,S1-Set-WaitHere,S1-Set-Continue
132 S1-tableAwaitCmd  40  1 1 11  A 0 0 SET ::511::GOAL-waitAtTable TO  false
133 S1-WaitHere 40  1 1 14  R 0 0 ::512:: GOAL-waitHere is true
134 S1-WaitHere 40  1 1 15  A 0 0 Execute sequence 'S1-sleep' on
      ::0::Care-O-Bot 3.2
135 S1-WaitHere 40  1 1 16  A 0 0 ::0::Care-O-Bot 3.2
      GUI,S1-Set-WaitHere,S1-Set-ReturnHome,S1-Set-Continue
136 S1-watchTV  30  1 1 32  R 2 0 Living room sofa seat 1 is occupied
137 S1-watchTV  30  1 1 33  R 2 0 Living room sofa seat 2 is occupied
138 S1-watchTV  30  1 1 34  R 2 0 Living room sofa seat 3 is occupied
139 S1-watchTV  30  1 1 35  R 2 0 Living room sofa seat 4 is occupied
140 S1-watchTV  30  1 1 36  R 0 0 Living room sofa seat 5 is occupied
141 S1-watchTV  30  1 1 37  R 0 0 Television Wattage > 10
142 S1-watchTV  30  1 1 44  R 0 0 ::513:: GOAL-watchTV is false AND has been
      in this state for more than 3600 seconds
143 S1-watchTV  30  1 1 45  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  yellow
144 S1-watchTV  30  1 1 47  A 0 0 Execute sequence 'lowerTray' on
      ::0::Care-O-Bot 3.2
145 S1-watchTV  30  1 1 48  A 0 0 move ::0::Care-O-Bot 3.2 to ::14:: Living
      Room Sofa Area in the Living Room and wait for completion
146 S1-watchTV  30  1 1 49  A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  white
147 S1-watchTV  30  1 1 50  A 0 0 ::0::Care-O-Bot 3.2 says 'Shall we watch TV
      together?' and wait for completion
148 S1-watchTV  30  1 1 51  A 0 0 SET ::513::GOAL-watchTV TO  true
149 S1-watchTV  30  1 1 52  A 0 0 ::0::Care-O-Bot 3.2
      GUI,S1-Set-Watch-TV,S1-Set-ReturnHome,S1-Set-Continue
150 T-medicine  0 0 0 0 R 0 0 Time is on or after 17:00:00
```

```
151 T-medicine  0 0 0 1 A 0 0 Execute sequence 'T-moveTo-person' on
      ::0::Care-O-Bot 3.2
152 T-moveTo-person 0 0 0 0 A 0 0 Turn light on ::0::Care-O-Bot 3.2 to
      yellow and wait for completion
153 T-moveTo-person 0 0 0 1 A 0 0 move ::0::Care-O-Bot 3.2 to ::999:: Current
      user Location
154 T-moveTo-person 0 0 0 2 A 0 0 Turn light on ::0::Care-O-Bot 3.2 to  white
      and wait for completion
155 unCheckBell 80  0 1 0 R 0 0 ::515:: GOAL-AnserDoorBell is true AND has
      been in this state for more than 10 seconds
156 unCheckBell 80  0 1 1 A 0 0 SET ::515::GOAL-AnserDoorBell TO  false
```

12.1.2 Grammar Rule File

```
 1 <pvc1> ::= +<any_text> 'is' <any_text>
 2 <pvc2> ::= +<any_text> <relational_operator> <integer>
 3 <pvc3> ::= +<any_text> 'In' <any_text>
 4 <evc1> ::= '::0::Care-O-Bot' '3.2' 'location' 'is' +<any_text>
 5 <evc2> ::= 'Living' 'room' 'sofa' 'seat' <integer> 'is' 'occupied'
 6 <tc1> ::= 'Time' 'is' 'on' 'or' 'after' <time>
 7 <tc2> ::= 'Time' 'is' 'between' <time> 'and' <time>
 8 <sns1> ::= 'has' 'been' 'in' 'this' 'state' 'for' 'more' 'than' <integer>
      'seconds'
 9 <sns2> ::= 'was' 'in' 'this' 'state' 'within' 'the' 'last' <integer>
      'seconds'
10 <pva1> ::= 'SET' +<any_text> 'TO' <any_text>
11 <eva1> ::= 'Turn' 'light' 'on' '::0::Care-O-Bot' '3.2' 'to' <any_text>
12 <eva2> ::= '::0::Care-O-Bot' '3.2' 'says' +<any_text>
13 <eva3> ::= 'move' '::0::Care-O-Bot' '3.2' 'to' +<any_text>
14 <eva4> ::= 'move' 'tray' 'on' '::0::Care-O-Bot' '3.2' 'to' <any_text>
15 <eva5> ::= 'move' 'torso' 'on' '::0::Care-O-Bot' '3.2' 'to' +<any_text>
16 <exbnd1> ::= '::0::Care-O-Bot' '3.2' +<any_text>
17 <exb1> ::= 'Execute' 'sequence' <any_text> 'on' '::0::Care-O-Bot' '3.2'
18 <del1> ::= 'Wait' 'for' <integer> 'seconds' 'on' '::0::Care-O-Bot' '3.2'
```

12.1.3 Data Extraction File

```
 1 pvc1; prop_name = [1]; truth_value = [3]; true = "true"; false = "false";
      non_deterministic = false;
 2 pvc2; prop_name = [1,2,3]; truth_value = null; true = null; false = null;
      non_deterministic = true;
 3 pvc3; prop_name = [1,2,3]; truth_value = null; true = null; false = null;
      non_deterministic = true;
 4 evc1; enum_name = [3]; enum_value = [5]; has_none_value = false;
      non_deterministic = false;
 5 evc2; enum_name = [4, 7]; enum_value = [4,5]; has_none_value = true;
      non_deterministic = true;
 6 tc1; start_time = [6]; end_time = null;
 7 tc2; start_time = [4]; end_time = [6];
 8 pva1; prop_name = [2]; truth_value = [4]; true = "true"; false = "false";
 9 eva1; enum_name = [2]; enum_value = [7]; resets = false;
10 eva2; enum_name = [3]; enum_value = [4]; resets = true;
11 eva3; enum_name = "location"; enum_value = [5]; resets = false;
12 eva4; enum_name = [2]; enum_value = [7]; resets = false;
13 eva5; enum_name = [2]; enum_value = [7]; resets = true;
14 exbnd1; behaviour_values = [3]; split_character = ","; ignore_entries =
      [1];
15 exb1; behaviour_name = [3];
```

```
del1; seconds = [3];
sns1; seconds = [9]; been_in_state = true; was_in_state = false;
sns2; seconds = [8]; been_in_state = false; was_in_state = true;
```

12.1.4 Intermediate Form

```
================================================================================
Propositional Variables
================================================================================
::515::GOAL-AnserDoorBell [deterministic]
Doorbell Last Wattage_>_1 [non-deterministic]
::500::TrayIsRaised [deterministic]
::504::TrayIsEmpty [deterministic]
::501::TrayIsLowered [deterministic]
Fridge Freezer_In_*ON* [non-deterministic]
::514::GOAL-fridgeUserAlerted [deterministic]
::506::GOAL-gotoCharger [deterministic]
::507::GOAL-gotoTable [deterministic]
::508::GOAL-gotoSofa [deterministic]
::513::GOAL-watchTV [deterministic]
Television Wattage_>_10 [non-deterministic]
::505::GOAL-gotoKitchen [deterministic]
::509::GOAL-waitAtKitchen [deterministic]
::510::GOAL-waitAtSofa [deterministic]
::511::GOAL-waitAtTable [deterministic]
::502::5PM-MedicineDue [deterministic]
::503::5PM-MedicineReminder [deterministic]
::512::GOAL-waitHere [deterministic]
================================================================================
Enumerated Type Variables
================================================================================
says [does reset][values: none, 'Doorbell', 'The fridge door is open!',
    'Its time for your medicine', 'Have you taken your medicine', 'Shall
    we watch TV together?'][deterministic]
light [does not reset][values: yellow, white][deterministic]
tray [does not reset][values: Lowered, Raised][deterministic]
location [does not reset][values: ::2:: Living Room, ::31:: TV location
    in the Living Room, ::7:: Kitchen Entrance in the Dining Room, ::14::
    Living Room Sofa Area in the Living Room, ::23:: Living Room Table in
    the Living Room Sofa Area of the Living Room, ::5:: ChargingStation
    Area in the Dining Room, ::999:: Current user Location][deterministic]
torso [does reset][values: none, the right, the back
    position][deterministic]
seat_occupied [does not reset][values: no_value, seat_1, seat_2, seat_3,
    seat_4, seat_5][non-deterministic]
================================================================================
Behaviours
================================================================================
answerDoorBell [not interruptible][schedulable][priority: 70]
  [preconditions]
    prop_value_check: [variable: ::515::GOAL-AnserDoorBell][value: true]
  [actions]
  enum_value_assignment: [order: 1][variable: says][value: 'Doorbell']
  delay: [order: 2][wait_seconds: 5]

checkBell [not interruptible][schedulable][priority: 80]
  [preconditions]
```

```
44     prop_value_check: [variable: Doorbell Last Wattage_>_1][value:
          true](was_in_state_within: 10)
45   [actions]
46   prop_value_assignment: [order: 2][variable:
       ::515::GOAL-AnserDoorBell][value: true]
47
48 lowerTray [not interruptible][not schedulable][priority: 0]
49   [preconditions]
50         prop_value_check: [variable: ::504::TrayIsEmpty][value: true]
51     &&--+
52         prop_value_check: [variable: ::500::TrayIsRaised][value: true]
53   [actions]
54   enum_value_assignment: [order: 11][variable: light][value: yellow]
55   enum_value_assignment: [order: 12][variable: tray][value: Lowered]
56   enum_value_assignment: [order: 13][variable: light][value: white]
57   prop_value_assignment: [order: 14][variable:
       ::500::TrayIsRaised][value: false]
58   prop_value_assignment: [order: 15][variable:
       ::501::TrayIsLowered][value: true]
59
60 raiseTray [not interruptible][not schedulable][priority: 0]
61   [preconditions]
62     prop_value_check: [variable: ::501::TrayIsLowered][value: true]
63   [actions]
64   enum_value_assignment: [order: 14][variable: light][value: yellow]
65   enum_value_assignment: [order: 15][variable: tray][value: Raised]
66   enum_value_assignment: [order: 16][variable: light][value: white]
67   prop_value_assignment: [order: 17][variable:
       ::500::TrayIsRaised][value: true]
68   prop_value_assignment: [order: 18][variable:
       ::501::TrayIsLowered][value: false]
69
70 S1-alertFridgeDoor [not interruptible][schedulable][priority: 60]
71   [preconditions]
72         prop_value_check: [variable:
              ::514::GOAL-fridgeUserAlerted][value: false]
73     &&--+
74         prop_value_check: [variable: Fridge Freezer_In_*ON*][value:
              true](been_in_state_for: 30)
75   [actions]
76   enum_value_assignment: [order: 32][variable: light][value: yellow]
77   enum_value_assignment: [order: 34][variable: location][value: ::2::
       Living Room]
78   enum_value_assignment: [order: 35][variable: light][value: white]
79   enum_value_assignment: [order: 36][variable: says][value: 'The fridge
       door is open!']
80   prop_value_assignment: [order: 37][variable:
       ::506::GOAL-gotoCharger][value: false]
81   prop_value_assignment: [order: 38][variable:
       ::507::GOAL-gotoTable][value: false]
82   prop_value_assignment: [order: 39][variable:
       ::508::GOAL-gotoSofa][value: false]
83   execute_behaviour_non_d: [order: 40][behaviours: S1-Set-GoToKitchen,
       S1-Set-WaitHere]
84   prop_value_assignment: [order: 41][variable:
       ::514::GOAL-fridgeUserAlerted][value: true]
```

```
85
86 S1-continueWatchTV [interruptible][schedulable][priority: 35]
87   [preconditions]
88         prop_value_check: [variable: Television Wattage_>_10][value: true]
89     &&--+
90         prop_value_check: [variable: ::513::GOAL-watchTV][value: true]
91   [actions]
92   enum_value_assignment: [order: 13][variable: light][value: yellow]
93   execute_behaviour: [order: 21][behaviour: lowerTray]
94   enum_value_assignment: [order: 22][variable: location][value: ::31:: TV
        location in the Living Room]
95   enum_value_assignment: [order: 23][variable: light][value: white]
96   enum_value_assignment: [order: 24][variable: torso][value: the right]
97   enum_value_assignment: [order: 25][variable: torso][value: the back
        position]
98   execute_behaviour_non_d: [order: 26][behaviours: S1-Set-ReturnHome,
        S1-Set-Continue]
99
100 S1-goToKitchen [interruptible][schedulable][priority: 40]
101   [preconditions]
102     prop_value_check: [variable: ::505::GOAL-gotoKitchen][value: true]
103   [actions]
104   enum_value_assignment: [order: 32][variable: light][value: yellow]
105   execute_behaviour: [order: 43][behaviour: lowerTray]
106   enum_value_assignment: [order: 44][variable: location][value: ::7::
        Kitchen Entrance in the Dining Room]
107   execute_behaviour: [order: 45][behaviour: raiseTray]
108   enum_value_assignment: [order: 46][variable: light][value: white]
109   prop_value_assignment: [order: 47][variable:
        ::505::GOAL-gotoKitchen][value: false]
110   prop_value_assignment: [order: 48][variable:
        ::509::GOAL-waitAtKitchen][value: true]
111
112 S1-gotoSofa [interruptible][schedulable][priority: 40]
113   [preconditions]
114     prop_value_check: [variable: ::508::GOAL-gotoSofa][value: true]
115   [actions]
116   enum_value_assignment: [order: 13][variable: light][value: yellow]
117   execute_behaviour: [order: 14][behaviour: lowerTray]
118   enum_value_assignment: [order: 15][variable: location][value: ::14::
        Living Room Sofa Area in the Living Room]
119   enum_value_assignment: [order: 16][variable: light][value: white]
120   prop_value_assignment: [order: 17][variable:
        ::508::GOAL-gotoSofa][value: false]
121   prop_value_assignment: [order: 18][variable:
        ::510::GOAL-waitAtSofa][value: true]
122
123 S1-gotoTable [interruptible][schedulable][priority: 40]
124   [preconditions]
125     prop_value_check: [variable: ::507::GOAL-gotoTable][value: true]
126   [actions]
127   enum_value_assignment: [order: 9][variable: light][value: yellow]
128   execute_behaviour: [order: 10][behaviour: lowerTray]
129   enum_value_assignment: [order: 11][variable: location][value: ::23::
        Living Room Table in the Living Room Sofa Area of the Living Room]
130   enum_value_assignment: [order: 12][variable: light][value: white]
```

```
131   prop_value_assignment: [order: 13][variable:
          ::507::GOAL-gotoTable][value: false]
132   prop_value_assignment: [order: 14][variable:
          ::511::GOAL-waitAtTable][value: true]
133
134 S1-kitchenAwaitCmd [interruptible][schedulable][priority: 40]
135   [preconditions]
136         prop_value_check: [variable: ::509::GOAL-waitAtKitchen][value:
              true]
137     &&--+
138         enum_value_check: [variable: location][value: ::7:: Kitchen
              Entrance in the Dining Room]
139   [actions]
140   execute_behaviour_non_d: [order: 16][behaviours: S1-Set-GoToSofa,
          S1-Set-GoToTable, S1-Set-Continue, S1-Set-WaitHere]
141   prop_value_assignment: [order: 17][variable:
          ::509::GOAL-waitAtKitchen][value: false]
142
143 S1-Med-5PM [interruptible][not schedulable][priority: 50]
144   [preconditions]
145         prop_value_check: [variable: ::502::5PM-MedicineDue][value: true]
146     &&--+
147         timing_constraint: [start_time: 17:00:00][end time: 23:59:00]
148   [actions]
149   enum_value_assignment: [order: 44][variable: light][value: yellow]
150   enum_value_assignment: [order: 45][variable: location][value: ::14::
          Living Room Sofa Area in the Living Room]
151   enum_value_assignment: [order: 46][variable: light][value: white]
152   enum_value_assignment: [order: 47][variable: says][value: 'Its time for
          your medicine']
153   execute_behaviour_non_d: [order: 48][behaviours: S1-Set-GoToKitchen,
          S1-Set-ReturnHome, S1-Set-WaitHere]
154   prop_value_assignment: [order: 49][variable:
          ::502::5PM-MedicineDue][value: false]
155   prop_value_assignment: [order: 50][variable:
          ::503::5PM-MedicineReminder][value: true]
156
157 S1-Med-5PM-Remind [interruptible][schedulable][priority: 50]
158   [preconditions]
159     prop_value_check: [variable: ::503::5PM-MedicineReminder][value:
            true](been_in_state_for: 60)
160   [actions]
161   enum_value_assignment: [order: 12][variable: light][value: yellow]
162   enum_value_assignment: [order: 13][variable: location][value: ::14::
          Living Room Sofa Area in the Living Room]
163   enum_value_assignment: [order: 14][variable: light][value: white]
164   enum_value_assignment: [order: 15][variable: says][value: 'Have you
          taken your medicine']
165   prop_value_assignment: [order: 16][variable:
          ::503::5PM-MedicineReminder][value: false]
166
167 S1-Med-5PM-Reset [not interruptible][schedulable][priority: 90]
168   [preconditions]
169         prop_value_check: [variable: ::502::5PM-MedicineDue][value: false]
170     &&--+
171         timing_constraint: [start_time: 00:00:00][end time: 16:59:00]
```

```
172   [actions]
173   prop_value_assignment: [order: 11][variable:
          ::502::5PM-MedicineDue][value: true]
174   prop_value_assignment: [order: 12][variable:
          ::503::5PM-MedicineReminder][value: false]
175
176 S1-remindFridgeDoor [not interruptible][schedulable][priority: 80]
177   [preconditions]
178     prop_value_check: [variable: ::514::GOAL-fridgeUserAlerted][value:
            true](been_in_state_for: 300)
179   [actions]
180   prop_value_assignment: [order: 1][variable:
          ::514::GOAL-fridgeUserAlerted][value: false]
181
182 S1-ResetAllGoals [not interruptible][not schedulable][priority: 0]
183   [actions]
184   prop_value_assignment: [order: 19][variable:
          ::500::TrayIsRaised][value: false]
185   prop_value_assignment: [order: 20][variable:
          ::501::TrayIsLowered][value: true]
186   prop_value_assignment: [order: 21][variable:
          ::502::5PM-MedicineDue][value: true]
187   prop_value_assignment: [order: 22][variable:
          ::503::5PM-MedicineReminder][value: false]
188   prop_value_assignment: [order: 23][variable: ::504::TrayIsEmpty][value:
          true]
189   prop_value_assignment: [order: 24][variable:
          ::505::GOAL-gotoKitchen][value: false]
190   prop_value_assignment: [order: 25][variable:
          ::506::GOAL-gotoCharger][value: false]
191   prop_value_assignment: [order: 26][variable:
          ::507::GOAL-gotoTable][value: false]
192   prop_value_assignment: [order: 27][variable:
          ::508::GOAL-gotoSofa][value: false]
193   prop_value_assignment: [order: 28][variable:
          ::509::GOAL-waitAtKitchen][value: false]
194   prop_value_assignment: [order: 29][variable:
          ::510::GOAL-waitAtSofa][value: false]
195   prop_value_assignment: [order: 30][variable:
          ::511::GOAL-waitAtTable][value: false]
196   prop_value_assignment: [order: 31][variable:
          ::512::GOAL-waitHere][value: false]
197   prop_value_assignment: [order: 32][variable:
          ::513::GOAL-watchTV][value: false]
198   prop_value_assignment: [order: 33][variable:
          ::514::GOAL-fridgeUserAlerted][value: false]
199
200 S1-ReturnHome [interruptible][schedulable][priority: 40]
201   [preconditions]
202     prop_value_check: [variable: ::506::GOAL-gotoCharger][value: true]
203   [actions]
204   enum_value_assignment: [order: 16][variable: light][value: yellow]
205   execute_behaviour: [order: 17][behaviour: lowerTray]
206   enum_value_assignment: [order: 18][variable: location][value: ::5::
          ChargingStation Area in the Dining Room]
207   enum_value_assignment: [order: 19][variable: light][value: white]
```

```
208   prop_value_assignment: [order: 20][variable:
          ::506::GOAL-gotoCharger][value: false]
209
210 S1-Set-Continue [not interruptible][not schedulable][priority: 0]
211   [actions]
212   prop_value_assignment: [order: 1][variable:
          ::512::GOAL-waitHere][value: false]
213   prop_value_assignment: [order: 2][variable: ::513::GOAL-watchTV][value:
          false]
214
215 S1-Set-GoToKitchen [not interruptible][not schedulable][priority: 0]
216   [actions]
217   prop_value_assignment: [order: 3][variable:
          ::505::GOAL-gotoKitchen][value: true]
218   prop_value_assignment: [order: 4][variable:
          ::512::GOAL-waitHere][value: false]
219
220 S1-Set-GoToSofa [not interruptible][not schedulable][priority: 0]
221   [actions]
222   prop_value_assignment: [order: 2][variable:
          ::508::GOAL-gotoSofa][value: true]
223
224 S1-Set-GoToTable [not interruptible][not schedulable][priority: 0]
225   [actions]
226   prop_value_assignment: [order: 2][variable:
          ::507::GOAL-gotoTable][value: true]
227
228 S1-Set-ReturnHome [not interruptible][not schedulable][priority: 0]
229   [actions]
230   prop_value_assignment: [order: 6][variable:
          ::506::GOAL-gotoCharger][value: true]
231   prop_value_assignment: [order: 7][variable:
          ::512::GOAL-waitHere][value: false]
232   prop_value_assignment: [order: 8][variable: ::513::GOAL-watchTV][value:
          false]
233
234 S1-Set-WaitHere [not interruptible][not schedulable][priority: 0]
235   [actions]
236   prop_value_assignment: [order: 1][variable:
          ::512::GOAL-waitHere][value: true]
237
238 S1-Set-Watch-TV [not interruptible][not schedulable][priority: 0]
239   [actions]
240   prop_value_assignment: [order: 2][variable: ::513::GOAL-watchTV][value:
          true]
241   prop_value_assignment: [order: 3][variable:
          ::512::GOAL-waitHere][value: false]
242
243 S1-sleep [interruptible][schedulable][priority: 10]
244   [actions]
245   enum_value_assignment: [order: 0][variable: light][value: white]
246   delay: [order: 5][wait_seconds: 1]
247   enum_value_assignment: [order: 6][variable: light][value: yellow]
248   delay: [order: 7][wait_seconds: 1]
249   enum_value_assignment: [order: 8][variable: light][value: white]
250
```

```
S1-sofaAwaitCmd [interruptible][schedulable][priority: 40]
  [preconditions]
        prop_value_check: [variable: ::510::GOAL-waitAtSofa][value: true]
    &&--+
        enum_value_check: [variable: location][value: ::14:: Living Room
           Sofa Area in the Living Room]
  [actions]
  execute_behaviour_non_d: [order: 17][behaviours: S1-Set-ReturnHome,
     S1-Set-WaitHere, S1-Set-Continue]
  prop_value_assignment: [order: 18][variable:
     ::510::GOAL-waitAtSofa][value: false]

S1-tableAwaitCmd [interruptible][schedulable][priority: 40]
  [preconditions]
        prop_value_check: [variable: ::511::GOAL-waitAtTable][value: true]
    &&--+
        enum_value_check: [variable: location][value: ::23:: Living Room
           Table in the Living Room Sofa Area of the Living Room]
  [actions]
  execute_behaviour_non_d: [order: 10][behaviours: S1-Set-ReturnHome,
     S1-Set-WaitHere, S1-Set-Continue]
  prop_value_assignment: [order: 11][variable:
     ::511::GOAL-waitAtTable][value: false]

S1-WaitHere [interruptible][schedulable][priority: 40]
  [preconditions]
    prop_value_check: [variable: ::512::GOAL-waitHere][value: true]
  [actions]
  enum_value_assignment: [order: 15][variable: light][value: white]
  delay: [order: 16][wait_seconds: 1]
  enum_value_assignment: [order: 17][variable: light][value: yellow]
  delay: [order: 18][wait_seconds: 1]
  enum_value_assignment: [order: 19][variable: light][value: white]
  execute_behaviour_non_d: [order: 21][behaviours: S1-Set-WaitHere,
     S1-Set-ReturnHome, S1-Set-Continue]

S1-watchTV [interruptible][schedulable][priority: 30]
  [preconditions]
        prop_value_check: [variable: ::513::GOAL-watchTV][value:
           false](been_in_state_for: 3600)
    &&--+
        |   prop_value_check: [variable: Television Wattage_>_10][value:
           true]
        &&--+
            |   enum_value_check: [variable: seat_occupied][value: seat_5]
            ||--+
                |   enum_value_check: [variable: seat_occupied][value:
                   seat_4]
                ||--+
                    |   enum_value_check: [variable:
                       seat_occupied][value: seat_3]
                    ||--+
                        |   enum_value_check: [variable:
                           seat_occupied][value: seat_2]
                        ||--+
                            enum_value_check: [variable:
```

```
                                seat_occupied][value: seat_1]
295   [actions]
296   enum_value_assignment: [order: 45][variable: light][value: yellow]
297   execute_behaviour: [order: 47][behaviour: lowerTray]
298   enum_value_assignment: [order: 48][variable: location][value: ::14::
          Living Room Sofa Area in the Living Room]
299   enum_value_assignment: [order: 49][variable: light][value: white]
300   enum_value_assignment: [order: 50][variable: says][value: 'Shall we
          watch TV together?']
301   prop_value_assignment: [order: 51][variable:
          ::513::GOAL-watchTV][value: true]
302   execute_behaviour_non_d: [order: 52][behaviours: S1-Set-Watch-TV,
          S1-Set-ReturnHome, S1-Set-Continue]
303
304 T-medicine [not interruptible][not schedulable][priority: 0]
305   [preconditions]
306     timing_constraint: [start_time: 17:00:00][end time: 23:59:00]
307   [actions]
308   enum_value_assignment: [order: 1][variable: light][value: yellow]
309   enum_value_assignment: [order: 2][variable: location][value: ::999::
          Current user Location]
310   enum_value_assignment: [order: 3][variable: light][value: white]
311
312 T-moveTo-person [not interruptible][not schedulable][priority: 0]
313   [actions]
314   enum_value_assignment: [order: 0][variable: light][value: yellow]
315   enum_value_assignment: [order: 1][variable: location][value: ::999::
          Current user Location]
316   enum_value_assignment: [order: 2][variable: light][value: white]
317
318 unCheckBell [not interruptible][schedulable][priority: 80]
319   [preconditions]
320     prop_value_check: [variable: ::515::GOAL-AnserDoorBell][value:
            true](been_in_state_for: 10)
321   [actions]
322   prop_value_assignment: [order: 1][variable:
          ::515::GOAL-AnserDoorBell][value: false]
```

12.2 NuSMV Input Translation Results

12.2.1 Generated NuSMV Input

```
1 -----------------------------------------------------------------------------
2 --  file generated Tue May 01 12:14:39 2015 using CRuToN v1.0.0
3 --
4 --  true non-determinism: no
5 --  minimum one state for state_n_seconds: yes
6 --  seconds per state: 600
7 --  max seconds for state_n_seconds: 5000
8 -----------------------------------------------------------------------------
9 --  behaviours:
10 --
11 --    answerDoorBell     [priority: 70][not interruptible][schedulable]
12 --    checkBell    [priority: 80][not interruptible][schedulable]
13 --    lowerTray    [priority: 0][not interruptible][not schedulable]
14 --    raiseTray    [priority: 0][not interruptible][not schedulable]
15 --    S1_alertFridgeDoor     [priority: 60][not
       interruptible][schedulable]
16 --    S1_continueWatchTV     [priority: 35][interruptible][schedulable]
17 --    S1_goToKitchen     [priority: 40][interruptible][schedulable]
18 --    S1_gotoSofa    [priority: 40][interruptible][schedulable]
19 --    S1_gotoTable     [priority: 40][interruptible][schedulable]
20 --    S1_kitchenAwaitCmd     [priority: 40][interruptible][schedulable]
21 --    S1_Med_5PM     [priority: 50][interruptible][not schedulable]
22 --    S1_Med_5PM_Remind    [priority: 50][interruptible][schedulable]
23 --    S1_Med_5PM_Reset     [priority: 90][not interruptible][schedulable]
24 --    S1_remindFridgeDoor    [priority: 80][not
       interruptible][schedulable]
25 --    S1_ResetAllGoals     [priority: 0][not interruptible][not
       schedulable]
26 --    S1_ReturnHome    [priority: 40][interruptible][schedulable]
27 --    S1_Set_Continue    [priority: 0][not interruptible][not schedulable]
28 --    S1_Set_GoToKitchen     [priority: 0][not interruptible][not
       schedulable]
29 --    S1_Set_GoToSofa    [priority: 0][not interruptible][not schedulable]
30 --    S1_Set_GoToTable     [priority: 0][not interruptible][not
       schedulable]
31 --    S1_Set_ReturnHome    [priority: 0][not interruptible][not
       schedulable]
32 --    S1_Set_WaitHere    [priority: 0][not interruptible][not schedulable]
33 --    S1_Set_Watch_TV    [priority: 0][not interruptible][not schedulable]
34 --    S1_sleep     [priority: 10][interruptible][schedulable]
35 --    S1_sofaAwaitCmd    [priority: 40][interruptible][schedulable]
36 --    S1_tableAwaitCmd     [priority: 40][interruptible][schedulable]
37 --    S1_WaitHere    [priority: 40][interruptible][schedulable]
38 --    S1_watchTV     [priority: 30][interruptible][schedulable]
39 --    T_medicine     [priority: 0][not interruptible][not schedulable]
40 --    T_moveTo_person    [priority: 0][not interruptible][not schedulable]
41 --    unCheckBell    [priority: 80][not interruptible][schedulable]
42 -----------------------------------------------------------------------------
43 MODULE main
44     ---------------------------------------------------------------------------
45     --  Variables
46     ---------------------------------------------------------------------------
47     VAR
```

```
48          time: {_00_00_00_to_16_59_00, _17_00_00_to_23_59_00};
49          step: {step_none, step_1, step_2, step_3, step_4, step_5, step_6,
                step_7, step_8, step_9, step_10, step_11, step_12, step_13,
                step_14, step_15};
50          schedule: {schedule_none, schedule_answerDoorBell,
                schedule_checkBell, schedule_lowerTray, schedule_raiseTray,
                schedule_S1_alertFridgeDoor, schedule_S1_continueWatchTV,
                schedule_S1_goToKitchen, schedule_S1_gotoSofa,
                schedule_S1_gotoTable, schedule_S1_kitchenAwaitCmd,
                schedule_S1_Med_5PM, schedule_S1_Med_5PM_Remind,
                schedule_S1_Med_5PM_Reset, schedule_S1_remindFridgeDoor,
                schedule_S1_ResetAllGoals, schedule_S1_ReturnHome,
                schedule_S1_Set_Continue, schedule_S1_Set_GoToKitchen,
                schedule_S1_Set_GoToSofa, schedule_S1_Set_GoToTable,
                schedule_S1_Set_ReturnHome, schedule_S1_Set_WaitHere,
                schedule_S1_Set_Watch_TV, schedule_S1_sleep,
                schedule_S1_sofaAwaitCmd, schedule_S1_tableAwaitCmd,
                schedule_S1_WaitHere, schedule_S1_watchTV,
                schedule_T_medicine, schedule_T_moveTo_person,
                schedule_unCheckBell};
51          last_schedule: {schedule_none, schedule_answerDoorBell,
                schedule_checkBell, schedule_lowerTray, schedule_raiseTray,
                schedule_S1_alertFridgeDoor, schedule_S1_continueWatchTV,
                schedule_S1_goToKitchen, schedule_S1_gotoSofa,
                schedule_S1_gotoTable, schedule_S1_kitchenAwaitCmd,
                schedule_S1_Med_5PM, schedule_S1_Med_5PM_Remind,
                schedule_S1_Med_5PM_Reset, schedule_S1_remindFridgeDoor,
                schedule_S1_ResetAllGoals, schedule_S1_ReturnHome,
                schedule_S1_Set_Continue, schedule_S1_Set_GoToKitchen,
                schedule_S1_Set_GoToSofa, schedule_S1_Set_GoToTable,
                schedule_S1_Set_ReturnHome, schedule_S1_Set_WaitHere,
                schedule_S1_Set_Watch_TV, schedule_S1_sleep,
                schedule_S1_sofaAwaitCmd, schedule_S1_tableAwaitCmd,
                schedule_S1_WaitHere, schedule_S1_watchTV,
                schedule_T_medicine, schedule_T_moveTo_person,
                schedule_unCheckBell};
52
53          ------------------------------------------------------------------------
54          --  Behaviour Module Instances
55          ------------------------------------------------------------------------
56          b_answerDoorBell: behaviour(pre_answerDoorBell, can_interrupt_70,
                FALSE, schedule, schedule_answerDoorBell, step, step_2);
57          b_checkBell: behaviour(pre_checkBell, can_interrupt_80, FALSE,
                schedule, schedule_checkBell, step, step_1);
58          b_lowerTray: behaviour(pre_lowerTray, FALSE, FALSE, schedule,
                schedule_lowerTray, step, step_5);
59          b_raiseTray: behaviour(pre_raiseTray, FALSE, FALSE, schedule,
                schedule_raiseTray, step, step_5);
60          b_S1_alertFridgeDoor: behaviour(pre_S1_alertFridgeDoor,
                can_interrupt_60, FALSE, schedule,
                schedule_S1_alertFridgeDoor, step, step_9);
61          b_S1_continueWatchTV: behaviour(pre_S1_continueWatchTV,
                can_interrupt_35, (b_S1_Med_5PM_Reset.can_be_scheduled |
                b_unCheckBell.can_be_scheduled |
                b_S1_remindFridgeDoor.can_be_scheduled |
                b_checkBell.can_be_scheduled |
```

```
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled |
            b_S1_goToKitchen.can_be_scheduled |
            b_S1_gotoSofa.can_be_scheduled |
            b_S1_gotoTable.can_be_scheduled |
            b_S1_kitchenAwaitCmd.can_be_scheduled |
            b_S1_ReturnHome.can_be_scheduled |
            b_S1_sofaAwaitCmd.can_be_scheduled |
            b_S1_tableAwaitCmd.can_be_scheduled |
            b_S1_WaitHere.can_be_scheduled), schedule,
            schedule_S1_continueWatchTV, step, step_7);
62      b_S1_goToKitchen: behaviour(pre_S1_goToKitchen, can_interrupt_40,
            (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled), schedule,
            schedule_S1_goToKitchen, step, step_7);
63      b_S1_gotoSofa: behaviour(pre_S1_gotoSofa, can_interrupt_40,
            (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled), schedule,
            schedule_S1_gotoSofa, step, step_6);
64      b_S1_gotoTable: behaviour(pre_S1_gotoTable, can_interrupt_40,
            (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled), schedule,
            schedule_S1_gotoTable, step, step_6);
65      b_S1_kitchenAwaitCmd: behaviour(pre_S1_kitchenAwaitCmd,
            can_interrupt_40, (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled), schedule,
            schedule_S1_kitchenAwaitCmd, step, step_2);
66      b_S1_Med_5PM: behaviour(pre_S1_Med_5PM, FALSE,
            (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled), schedule,
            schedule_S1_Med_5PM, step, step_7);
```

```
67      b_S1_Med_5PM_Remind: behaviour(pre_S1_Med_5PM_Remind,
            can_interrupt_50, (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled), schedule,
            schedule_S1_Med_5PM_Remind, step, step_5);
68      b_S1_Med_5PM_Reset: behaviour(pre_S1_Med_5PM_Reset,
            can_interrupt_90, FALSE, schedule, schedule_S1_Med_5PM_Reset,
            step, step_2);
69      b_S1_remindFridgeDoor: behaviour(pre_S1_remindFridgeDoor,
            can_interrupt_80, FALSE, schedule,
            schedule_S1_remindFridgeDoor, step, step_1);
70      b_S1_ResetAllGoals: behaviour(pre_S1_ResetAllGoals, FALSE, FALSE,
            schedule, schedule_S1_ResetAllGoals, step, step_15);
71      b_S1_ReturnHome: behaviour(pre_S1_ReturnHome, can_interrupt_40,
            (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled), schedule,
            schedule_S1_ReturnHome, step, step_5);
72      b_S1_Set_Continue: behaviour(pre_S1_Set_Continue, FALSE, FALSE,
            schedule, schedule_S1_Set_Continue, step, step_2);
73      b_S1_Set_GoToKitchen: behaviour(pre_S1_Set_GoToKitchen, FALSE,
            FALSE, schedule, schedule_S1_Set_GoToKitchen, step, step_2);
74      b_S1_Set_GoToSofa: behaviour(pre_S1_Set_GoToSofa, FALSE, FALSE,
            schedule, schedule_S1_Set_GoToSofa, step, step_1);
75      b_S1_Set_GoToTable: behaviour(pre_S1_Set_GoToTable, FALSE, FALSE,
            schedule, schedule_S1_Set_GoToTable, step, step_1);
76      b_S1_Set_ReturnHome: behaviour(pre_S1_Set_ReturnHome, FALSE,
            FALSE, schedule, schedule_S1_Set_ReturnHome, step, step_3);
77      b_S1_Set_WaitHere: behaviour(pre_S1_Set_WaitHere, FALSE, FALSE,
            schedule, schedule_S1_Set_WaitHere, step, step_1);
78      b_S1_Set_Watch_TV: behaviour(pre_S1_Set_Watch_TV, FALSE, FALSE,
            schedule, schedule_S1_Set_Watch_TV, step, step_2);
79      b_S1_sleep: behaviour(pre_S1_sleep, can_interrupt_10,
            (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled |
            b_S1_goToKitchen.can_be_scheduled |
            b_S1_gotoSofa.can_be_scheduled |
            b_S1_gotoTable.can_be_scheduled |
            b_S1_kitchenAwaitCmd.can_be_scheduled |
            b_S1_ReturnHome.can_be_scheduled |
            b_S1_sofaAwaitCmd.can_be_scheduled |
            b_S1_tableAwaitCmd.can_be_scheduled |
            b_S1_WaitHere.can_be_scheduled |
            b_S1_continueWatchTV.can_be_scheduled |
```

```
            b_S1_watchTV.can_be_scheduled), schedule, schedule_S1_sleep,
            step, step_5);
80          b_S1_sofaAwaitCmd: behaviour(pre_S1_sofaAwaitCmd,
            can_interrupt_40, (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled), schedule,
            schedule_S1_sofaAwaitCmd, step, step_2);
81          b_S1_tableAwaitCmd: behaviour(pre_S1_tableAwaitCmd,
            can_interrupt_40, (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled), schedule,
            schedule_S1_tableAwaitCmd, step, step_2);
82          b_S1_WaitHere: behaviour(pre_S1_WaitHere, can_interrupt_40,
            (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled), schedule,
            schedule_S1_WaitHere, step, step_6);
83          b_S1_watchTV: behaviour(pre_S1_watchTV, can_interrupt_30,
            (b_S1_Med_5PM_Reset.can_be_scheduled |
            b_unCheckBell.can_be_scheduled |
            b_S1_remindFridgeDoor.can_be_scheduled |
            b_checkBell.can_be_scheduled |
            b_answerDoorBell.can_be_scheduled |
            b_S1_alertFridgeDoor.can_be_scheduled |
            b_S1_Med_5PM_Remind.can_be_scheduled |
            b_S1_goToKitchen.can_be_scheduled |
            b_S1_gotoSofa.can_be_scheduled |
            b_S1_gotoTable.can_be_scheduled |
            b_S1_kitchenAwaitCmd.can_be_scheduled |
            b_S1_ReturnHome.can_be_scheduled |
            b_S1_sofaAwaitCmd.can_be_scheduled |
            b_S1_tableAwaitCmd.can_be_scheduled |
            b_S1_WaitHere.can_be_scheduled |
            b_S1_continueWatchTV.can_be_scheduled), schedule,
            schedule_S1_watchTV, step, step_7);
84          b_T_medicine: behaviour(pre_T_medicine, FALSE, FALSE, schedule,
            schedule_T_medicine, step, step_3);
85          b_T_moveTo_person: behaviour(pre_T_moveTo_person, FALSE, FALSE,
            schedule, schedule_T_moveTo_person, step, step_3);
86          b_unCheckBell: behaviour(pre_unCheckBell, can_interrupt_80,
            FALSE, schedule, schedule_unCheckBell, step, step_1);
87
88          ------------------------------------------------------------------------
89          --  Boolean Variables
```

```
90          ------------------------------------------------------------------------------
91          __515__GOAL_AnserDoorBell: boolean;
92          Doorbell_Last_Wattage___1: boolean;
93          __500__TrayIsRaised: boolean;
94          __504__TrayIsEmpty: boolean;
95          __501__TrayIsLowered: boolean;
96          Fridge_Freezer_In__ON_: boolean;
97          __514__GOAL_fridgeUserAlerted: boolean;
98          __506__GOAL_gotoCharger: boolean;
99          __507__GOAL_gotoTable: boolean;
100         __508__GOAL_gotoSofa: boolean;
101         __513__GOAL_watchTV: boolean;
102         Television_Wattage___10: boolean;
103         __505__GOAL_gotoKitchen: boolean;
104         __509__GOAL_waitAtKitchen: boolean;
105         __510__GOAL_waitAtSofa: boolean;
106         __511__GOAL_waitAtTable: boolean;
107         __502__5PM_MedicineDue: boolean;
108         __503__5PM_MedicineReminder: boolean;
109         __512__GOAL_waitHere: boolean;
110
111         ------------------------------------------------------------------------------
112         --  Enumerated Types
113         ------------------------------------------------------------------------------
114         says: {none, _Doorbell_, _The_fridge_door_is_open__,
                _Its_time_for_your_medicine_, _Have_you_taken_your_medicine_,
                _Shall_we_watch_TV_together__};
115         light: {yellow, white};
116         tray: {Lowered, Raised};
117         location: {__2___Living_Room,
                __31___TV_location_in_the_Living_Room,
                __7___Kitchen_Entrance_in_the_Dining_Room,
                __14___Living_Room_Sofa_Area_in_the_Living_Room,
                __23___Living_Room_Table_in_the_Living_Room_Sofa_Area_of_the_Living_Room,
                __5___ChargingStation_Area_in_the_Dining_Room,
                __999___Current_user_Location};
118         torso: {none, the_right, the_back_position};
119         seat_occupied: {no_value, seat_1, seat_2, seat_3, seat_4, seat_5};
120
121         ------------------------------------------------------------------------------
122         --  Been in State/Was in State Counters
123         ------------------------------------------------------------------------------
124         Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE: {s_0, s_final};
125         __503__5PM_MedicineReminder_TRUE_BEEN_IN_STATE: {s_0, s_final};
126         __514__GOAL_fridgeUserAlerted_TRUE_BEEN_IN_STATE: {s_0, s_final};
127         __513__GOAL_watchTV_FALSE_BEEN_IN_STATE: {s_0, s_600, s_1200,
                s_1800, s_2400, s_3000, s_final};
128         __515__GOAL_AnserDoorBell_TRUE_BEEN_IN_STATE: {s_0, s_final};
129         Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE: {s_0, s_final};
130
131     ------------------------------------------------------------------------------
132     --  Assignments
133     ------------------------------------------------------------------------------
134     ASSIGN
135         init(time):= {_00_00_00_to_16_59_00, _17_00_00_to_23_59_00};
136
```

```
137         init(step):= step_none;
138         next(step):=
139             case
140                 a_behaviour_can_be_scheduled: step_1;
141
142                 (b_S1_alertFridgeDoor.is_scheduled & step = step_8):
                        step_1;
143                 (b_S1_continueWatchTV.is_scheduled & step = step_2 &
                        b_lowerTray.preconditions_hold): step_1;
144                 (b_S1_continueWatchTV.is_scheduled & step = step_7):
                        step_1;
145                 (b_S1_goToKitchen.is_scheduled & step = step_2 &
                        b_lowerTray.preconditions_hold): step_1;
146                 (b_S1_goToKitchen.is_scheduled & step = step_4 &
                        b_raiseTray.preconditions_hold): step_1;
147                 (b_S1_gotoSofa.is_scheduled & step = step_2 &
                        b_lowerTray.preconditions_hold): step_1;
148                 (b_S1_gotoTable.is_scheduled & step = step_2 &
                        b_lowerTray.preconditions_hold): step_1;
149                 (b_S1_kitchenAwaitCmd.is_scheduled & step = step_1):
                        step_1;
150                 (b_S1_Med_5PM.is_scheduled & step = step_5): step_1;
151                 (b_S1_ReturnHome.is_scheduled & step = step_2 &
                        b_lowerTray.preconditions_hold): step_1;
152                 (b_S1_sofaAwaitCmd.is_scheduled & step = step_1): step_1;
153                 (b_S1_tableAwaitCmd.is_scheduled & step = step_1): step_1;
154                 (b_S1_WaitHere.is_scheduled & step = step_6): step_1;
155                 (b_S1_watchTV.is_scheduled & step = step_2 &
                        b_lowerTray.preconditions_hold): step_1;
156                 (b_S1_watchTV.is_scheduled & step = step_7): step_1;
157                 (b_S1_Set_GoToKitchen.is_last_step & last_schedule =
                        schedule_S1_alertFridgeDoor): step_9;
158                 (b_S1_Set_WaitHere.is_last_step & last_schedule =
                        schedule_S1_alertFridgeDoor): step_9;
159                 (b_lowerTray.is_last_step & last_schedule =
                        schedule_S1_continueWatchTV): step_3;
160                 (b_lowerTray.is_last_step & last_schedule =
                        schedule_S1_goToKitchen): step_3;
161                 (b_raiseTray.is_last_step & last_schedule =
                        schedule_S1_goToKitchen): step_5;
162                 (b_lowerTray.is_last_step & last_schedule =
                        schedule_S1_gotoSofa): step_3;
163                 (b_lowerTray.is_last_step & last_schedule =
                        schedule_S1_gotoTable): step_3;
164                 (b_S1_Set_GoToSofa.is_last_step & last_schedule =
                        schedule_S1_kitchenAwaitCmd): step_2;
165                 (b_S1_Set_GoToTable.is_last_step & last_schedule =
                        schedule_S1_kitchenAwaitCmd): step_2;
166                 (b_S1_Set_Continue.is_last_step & last_schedule =
                        schedule_S1_kitchenAwaitCmd): step_2;
167                 (b_S1_Set_WaitHere.is_last_step & last_schedule =
                        schedule_S1_kitchenAwaitCmd): step_2;
168                 (b_S1_Set_GoToKitchen.is_last_step & last_schedule =
                        schedule_S1_Med_5PM): step_6;
169                 (b_S1_Set_ReturnHome.is_last_step & last_schedule =
                        schedule_S1_Med_5PM): step_6;
```

```
170                 (b_S1_Set_WaitHere.is_last_step & last_schedule =
                        schedule_S1_Med_5PM): step_6;
171                 (b_lowerTray.is_last_step & last_schedule =
                        schedule_S1_ReturnHome): step_3;
172                 (b_S1_Set_ReturnHome.is_last_step & last_schedule =
                        schedule_S1_sofaAwaitCmd): step_2;
173                 (b_S1_Set_WaitHere.is_last_step & last_schedule =
                        schedule_S1_sofaAwaitCmd): step_2;
174                 (b_S1_Set_Continue.is_last_step & last_schedule =
                        schedule_S1_sofaAwaitCmd): step_2;
175                 (b_S1_Set_ReturnHome.is_last_step & last_schedule =
                        schedule_S1_tableAwaitCmd): step_2;
176                 (b_S1_Set_WaitHere.is_last_step & last_schedule =
                        schedule_S1_tableAwaitCmd): step_2;
177                 (b_S1_Set_Continue.is_last_step & last_schedule =
                        schedule_S1_tableAwaitCmd): step_2;
178                 (b_lowerTray.is_last_step & last_schedule =
                        schedule_S1_watchTV): step_3;
179
180                 a_behaviour_is_ending: step_none;
181                 an_executed_behaviour_is_ending_as_a_last_action:
                        step_none;
182
183                 step = step_1: step_2;
184                 step = step_2: step_3;
185                 step = step_3: step_4;
186                 step = step_4: step_5;
187                 step = step_5: step_6;
188                 step = step_6: step_7;
189                 step = step_7: step_8;
190                 step = step_8: step_9;
191                 step = step_9: step_10;
192                 step = step_10: step_11;
193                 step = step_11: step_12;
194                 step = step_12: step_13;
195                 step = step_13: step_14;
196                 step = step_14: step_15;
197                 step = step_15: step_none;
198
199                 TRUE: step_none;
200             esac;
201
202         init(last_schedule):= schedule_none;
203         next(last_schedule):=
204             case
205                 executed_behaviour_execute_next: last_schedule;
206                 an_executed_behaviour_is_scheduled: last_schedule;
207                 TRUE: schedule;
208             esac;
209
210         init(schedule):= schedule_none;
211         next(schedule):=
212             case
213                 b_S1_Med_5PM_Reset.can_be_scheduled:
                        schedule_S1_Med_5PM_Reset;
214                 b_unCheckBell.can_be_scheduled: schedule_unCheckBell;
```

```
215             b_S1_remindFridgeDoor.can_be_scheduled:
                    schedule_S1_remindFridgeDoor;
216             b_checkBell.can_be_scheduled: schedule_checkBell;
217             b_answerDoorBell.can_be_scheduled:
                    schedule_answerDoorBell;
218             b_S1_alertFridgeDoor.can_be_scheduled:
                    schedule_S1_alertFridgeDoor;
219             b_S1_Med_5PM_Remind.can_be_scheduled:
                    schedule_S1_Med_5PM_Remind;
220             b_S1_gotoTable.can_be_scheduled: schedule_S1_gotoTable;
221             b_S1_WaitHere.can_be_scheduled: schedule_S1_WaitHere;
222             b_S1_tableAwaitCmd.can_be_scheduled:
                    schedule_S1_tableAwaitCmd;
223             b_S1_sofaAwaitCmd.can_be_scheduled:
                    schedule_S1_sofaAwaitCmd;
224             b_S1_ReturnHome.can_be_scheduled: schedule_S1_ReturnHome;
225             b_S1_kitchenAwaitCmd.can_be_scheduled:
                    schedule_S1_kitchenAwaitCmd;
226             b_S1_gotoSofa.can_be_scheduled: schedule_S1_gotoSofa;
227             b_S1_goToKitchen.can_be_scheduled:
                    schedule_S1_goToKitchen;
228             b_S1_continueWatchTV.can_be_scheduled:
                    schedule_S1_continueWatchTV;
229             b_S1_watchTV.can_be_scheduled: schedule_S1_watchTV;
230             b_S1_sleep.can_be_scheduled: schedule_S1_sleep;
231             (b_S1_alertFridgeDoor.is_scheduled & step = step_8):
                    {schedule_S1_Set_GoToKitchen,
                    schedule_S1_Set_WaitHere};
232             (b_S1_continueWatchTV.is_scheduled & step = step_2 &
                    b_lowerTray.preconditions_hold): schedule_lowerTray;
233             (b_S1_continueWatchTV.is_scheduled & step = step_7):
                    {schedule_S1_Set_ReturnHome,
                    schedule_S1_Set_Continue};
234             (b_S1_goToKitchen.is_scheduled & step = step_2 &
                    b_lowerTray.preconditions_hold): schedule_lowerTray;
235             (b_S1_goToKitchen.is_scheduled & step = step_4 &
                    b_raiseTray.preconditions_hold): schedule_raiseTray;
236             (b_S1_gotoSofa.is_scheduled & step = step_2 &
                    b_lowerTray.preconditions_hold): schedule_lowerTray;
237             (b_S1_gotoTable.is_scheduled & step = step_2 &
                    b_lowerTray.preconditions_hold): schedule_lowerTray;
238             (b_S1_kitchenAwaitCmd.is_scheduled & step = step_1):
                    {schedule_S1_Set_GoToSofa, schedule_S1_Set_GoToTable,
                    schedule_S1_Set_Continue, schedule_S1_Set_WaitHere};
239             (b_S1_Med_5PM.is_scheduled & step = step_5):
                    {schedule_S1_Set_GoToKitchen,
                    schedule_S1_Set_ReturnHome, schedule_S1_Set_WaitHere};
240             (b_S1_ReturnHome.is_scheduled & step = step_2 &
                    b_lowerTray.preconditions_hold): schedule_lowerTray;
241             (b_S1_sofaAwaitCmd.is_scheduled & step = step_1):
                    {schedule_S1_Set_ReturnHome,
                    schedule_S1_Set_WaitHere, schedule_S1_Set_Continue};
242             (b_S1_tableAwaitCmd.is_scheduled & step = step_1):
                    {schedule_S1_Set_ReturnHome,
                    schedule_S1_Set_WaitHere, schedule_S1_Set_Continue};
243             (b_S1_WaitHere.is_scheduled & step = step_6):
```

```
                    {schedule_S1_Set_WaitHere,
                    schedule_S1_Set_ReturnHome, schedule_S1_Set_Continue};
244                 (b_S1_watchTV.is_scheduled & step = step_2 &
                    b_lowerTray.preconditions_hold): schedule_lowerTray;
245                 (b_S1_watchTV.is_scheduled & step = step_7):
                    {schedule_S1_Set_Watch_TV,
                    schedule_S1_Set_ReturnHome, schedule_S1_Set_Continue};
246
247                 a_behaviour_is_ending: schedule_none;
248                 an_executed_behaviour_is_ending_as_a_last_action:
                    schedule_none;
249                 (an_executed_behaviour_is_ending & last_schedule !=
                    schedule): last_schedule;
250                 an_executed_behaviour_is_ending: schedule_none;
251
252                 TRUE: schedule;
253             esac;
254
255         init(Doorbell_Last_Wattage___1):= {TRUE, FALSE};
256         next(Doorbell_Last_Wattage___1):= {TRUE, FALSE};
257
258         init(Fridge_Freezer_In__ON_):= {TRUE, FALSE};
259         next(Fridge_Freezer_In__ON_):= {TRUE, FALSE};
260
261         init(Television_Wattage___10):= {TRUE, FALSE};
262         next(Television_Wattage___10):= {TRUE, FALSE};
263
264         init(seat_occupied):= {no_value, seat_1, seat_2, seat_3, seat_4,
                seat_5};
265         next(seat_occupied):= {no_value, seat_1, seat_2, seat_3, seat_4,
                seat_5};
266
267         init(__500__TrayIsRaised):= FALSE;
268         next(__500__TrayIsRaised):=
269             case
270                 (b_lowerTray.is_scheduled & step = step_4): FALSE;
271                 (b_raiseTray.is_scheduled & step = step_4): TRUE;
272                 (b_S1_ResetAllGoals.is_scheduled & step = step_1): FALSE;
273
274                 TRUE: __500__TrayIsRaised;
275             esac;
276
277         init(__501__TrayIsLowered):= TRUE;
278         next(__501__TrayIsLowered):=
279             case
280                 (b_lowerTray.is_scheduled & step = step_5): TRUE;
281                 (b_raiseTray.is_scheduled & step = step_5): FALSE;
282                 (b_S1_ResetAllGoals.is_scheduled & step = step_2): TRUE;
283
284                 TRUE: __501__TrayIsLowered;
285             esac;
286
287         init(__502__5PM_MedicineDue):= TRUE;
288         next(__502__5PM_MedicineDue):=
289             case
290                 (b_S1_Med_5PM.is_scheduled & step = step_6): FALSE;
```

```
291                 (b_S1_Med_5PM_Reset.is_scheduled & step = step_1): TRUE;
292                 (b_S1_ResetAllGoals.is_scheduled & step = step_3): TRUE;
293
294                 TRUE: __502__5PM_MedicineDue;
295             esac;
296
297         init(__503__5PM_MedicineReminder):= FALSE;
298         next(__503__5PM_MedicineReminder):=
299             case
300                 (b_S1_Med_5PM.is_scheduled & step = step_7): TRUE;
301                 (b_S1_Med_5PM_Remind.is_scheduled & step = step_5): FALSE;
302                 (b_S1_Med_5PM_Reset.is_scheduled & step = step_2): FALSE;
303                 (b_S1_ResetAllGoals.is_scheduled & step = step_4): FALSE;
304
305                 TRUE: __503__5PM_MedicineReminder;
306             esac;
307
308         init(__504__TrayIsEmpty):= TRUE;
309         next(__504__TrayIsEmpty):=
310             case
311                 (b_S1_ResetAllGoals.is_scheduled & step = step_5): TRUE;
312
313                 TRUE: __504__TrayIsEmpty;
314             esac;
315
316         init(__505__GOAL_gotoKitchen):= FALSE;
317         next(__505__GOAL_gotoKitchen):=
318             case
319                 (b_S1_goToKitchen.is_scheduled & step = step_6): FALSE;
320                 (b_S1_ResetAllGoals.is_scheduled & step = step_6): FALSE;
321                 (b_S1_Set_GoToKitchen.is_scheduled & step = step_1): TRUE;
322
323                 TRUE: __505__GOAL_gotoKitchen;
324             esac;
325
326         init(__506__GOAL_gotoCharger):= FALSE;
327         next(__506__GOAL_gotoCharger):=
328             case
329                 (b_S1_alertFridgeDoor.is_scheduled & step = step_5):
                        FALSE;
330                 (b_S1_ResetAllGoals.is_scheduled & step = step_7): FALSE;
331                 (b_S1_ReturnHome.is_scheduled & step = step_5): FALSE;
332                 (b_S1_Set_ReturnHome.is_scheduled & step = step_1): TRUE;
333
334                 TRUE: __506__GOAL_gotoCharger;
335             esac;
336
337         init(__507__GOAL_gotoTable):= FALSE;
338         next(__507__GOAL_gotoTable):=
339             case
340                 (b_S1_alertFridgeDoor.is_scheduled & step = step_6):
                        FALSE;
341                 (b_S1_gotoTable.is_scheduled & step = step_5): FALSE;
342                 (b_S1_ResetAllGoals.is_scheduled & step = step_8): FALSE;
343                 (b_S1_Set_GoToTable.is_scheduled & step = step_1): TRUE;
344
```

```
345                 TRUE: __507__GOAL_gotoTable;
346             esac;
347
348         init(__508__GOAL_gotoSofa):= FALSE;
349         next(__508__GOAL_gotoSofa):=
350             case
351                 (b_S1_alertFridgeDoor.is_scheduled & step = step_7):
                        FALSE;
352                 (b_S1_gotoSofa.is_scheduled & step = step_5): FALSE;
353                 (b_S1_ResetAllGoals.is_scheduled & step = step_9): FALSE;
354                 (b_S1_Set_GoToSofa.is_scheduled & step = step_1): TRUE;
355
356                 TRUE: __508__GOAL_gotoSofa;
357             esac;
358
359         init(__509__GOAL_waitAtKitchen):= FALSE;
360         next(__509__GOAL_waitAtKitchen):=
361             case
362                 (b_S1_goToKitchen.is_scheduled & step = step_7): TRUE;
363                 (b_S1_kitchenAwaitCmd.is_scheduled & step = step_2):
                        FALSE;
364                 (b_S1_ResetAllGoals.is_scheduled & step = step_10): FALSE;
365
366                 TRUE: __509__GOAL_waitAtKitchen;
367             esac;
368
369         init(__510__GOAL_waitAtSofa):= FALSE;
370         next(__510__GOAL_waitAtSofa):=
371             case
372                 (b_S1_gotoSofa.is_scheduled & step = step_6): TRUE;
373                 (b_S1_ResetAllGoals.is_scheduled & step = step_11): FALSE;
374                 (b_S1_sofaAwaitCmd.is_scheduled & step = step_2): FALSE;
375
376                 TRUE: __510__GOAL_waitAtSofa;
377             esac;
378
379         init(__511__GOAL_waitAtTable):= FALSE;
380         next(__511__GOAL_waitAtTable):=
381             case
382                 (b_S1_gotoTable.is_scheduled & step = step_6): TRUE;
383                 (b_S1_ResetAllGoals.is_scheduled & step = step_12): FALSE;
384                 (b_S1_tableAwaitCmd.is_scheduled & step = step_2): FALSE;
385
386                 TRUE: __511__GOAL_waitAtTable;
387             esac;
388
389         init(__512__GOAL_waitHere):= FALSE;
390         next(__512__GOAL_waitHere):=
391             case
392                 (b_S1_ResetAllGoals.is_scheduled & step = step_13): FALSE;
393                 (b_S1_Set_Continue.is_scheduled & step = step_1): FALSE;
394                 (b_S1_Set_GoToKitchen.is_scheduled & step = step_2):
                        FALSE;
395                 (b_S1_Set_ReturnHome.is_scheduled & step = step_2): FALSE;
396                 (b_S1_Set_WaitHere.is_scheduled & step = step_1): TRUE;
397                 (b_S1_Set_Watch_TV.is_scheduled & step = step_2): FALSE;
```

```
398
399             TRUE: __512__GOAL_waitHere;
400         esac;
401
402     init(__513__GOAL_watchTV):= FALSE;
403     next(__513__GOAL_watchTV):=
404         case
405             (b_S1_ResetAllGoals.is_scheduled & step = step_14): FALSE;
406             (b_S1_Set_Continue.is_scheduled & step = step_2): FALSE;
407             (b_S1_Set_ReturnHome.is_scheduled & step = step_3): FALSE;
408             (b_S1_Set_Watch_TV.is_scheduled & step = step_1): TRUE;
409             (b_S1_watchTV.is_scheduled & step = step_6): TRUE;
410
411             TRUE: __513__GOAL_watchTV;
412         esac;
413
414     init(__514__GOAL_fridgeUserAlerted):= FALSE;
415     next(__514__GOAL_fridgeUserAlerted):=
416         case
417             (b_S1_alertFridgeDoor.is_scheduled & step = step_9): TRUE;
418             (b_S1_remindFridgeDoor.is_scheduled & step = step_1):
                    FALSE;
419             (b_S1_ResetAllGoals.is_scheduled & step = step_15): FALSE;
420
421             TRUE: __514__GOAL_fridgeUserAlerted;
422         esac;
423
424     init(__515__GOAL_AnserDoorBell):= FALSE;
425     next(__515__GOAL_AnserDoorBell):=
426         case
427             (b_checkBell.is_scheduled & step = step_1): TRUE;
428             (b_unCheckBell.is_scheduled & step = step_1): FALSE;
429
430             TRUE: __515__GOAL_AnserDoorBell;
431         esac;
432
433     init(light):= white;
434     next(light):=
435         case
436             (b_lowerTray.is_scheduled & step = step_1): yellow;
437             (b_lowerTray.is_scheduled & step = step_3): white;
438             (b_raiseTray.is_scheduled & step = step_1): yellow;
439             (b_raiseTray.is_scheduled & step = step_3): white;
440             (b_S1_alertFridgeDoor.is_scheduled & step = step_1):
                    yellow;
441             (b_S1_alertFridgeDoor.is_scheduled & step = step_3):
                    white;
442             (b_S1_continueWatchTV.is_scheduled & step = step_1):
                    yellow;
443             (b_S1_continueWatchTV.is_scheduled & step = step_4):
                    white;
444             (b_S1_goToKitchen.is_scheduled & step = step_1): yellow;
445             (b_S1_goToKitchen.is_scheduled & step = step_5): white;
446             (b_S1_gotoSofa.is_scheduled & step = step_1): yellow;
447             (b_S1_gotoSofa.is_scheduled & step = step_4): white;
448             (b_S1_gotoTable.is_scheduled & step = step_1): yellow;
```

```
449                 (b_S1_gotoTable.is_scheduled & step = step_4): white;
450                 (b_S1_Med_5PM.is_scheduled & step = step_1): yellow;
451                 (b_S1_Med_5PM.is_scheduled & step = step_3): white;
452                 (b_S1_Med_5PM_Remind.is_scheduled & step = step_1):
                       yellow;
453                 (b_S1_Med_5PM_Remind.is_scheduled & step = step_3): white;
454                 (b_S1_ReturnHome.is_scheduled & step = step_1): yellow;
455                 (b_S1_ReturnHome.is_scheduled & step = step_4): white;
456                 (b_S1_sleep.is_scheduled & step = step_1): white;
457                 (b_S1_sleep.is_scheduled & step = step_3): yellow;
458                 (b_S1_sleep.is_scheduled & step = step_5): white;
459                 (b_S1_WaitHere.is_scheduled & step = step_1): white;
460                 (b_S1_WaitHere.is_scheduled & step = step_3): yellow;
461                 (b_S1_WaitHere.is_scheduled & step = step_5): white;
462                 (b_S1_watchTV.is_scheduled & step = step_1): yellow;
463                 (b_S1_watchTV.is_scheduled & step = step_4): white;
464                 (b_T_medicine.is_scheduled & step = step_1): yellow;
465                 (b_T_medicine.is_scheduled & step = step_3): white;
466                 (b_T_moveTo_person.is_scheduled & step = step_1): yellow;
467                 (b_T_moveTo_person.is_scheduled & step = step_3): white;
468
469                 TRUE: light;
470             esac;
471
472         init(location):= __5___ChargingStation_Area_in_the_Dining_Room;
473         next(location):=
474             case
475                 (b_S1_alertFridgeDoor.is_scheduled & step = step_2):
                       __2___Living_Room;
476                 (b_S1_continueWatchTV.is_scheduled & step = step_3):
                       __31___TV_location_in_the_Living_Room;
477                 (b_S1_goToKitchen.is_scheduled & step = step_3):
                       __7___Kitchen_Entrance_in_the_Dining_Room;
478                 (b_S1_gotoSofa.is_scheduled & step = step_3):
                       __14___Living_Room_Sofa_Area_in_the_Living_Room;
479                 (b_S1_gotoTable.is_scheduled & step = step_3):
                       __23___Living_Room_Table_in_the_Living_Room_Sofa_Area_of_the_Living_Room
480                 (b_S1_Med_5PM.is_scheduled & step = step_2):
                       __14___Living_Room_Sofa_Area_in_the_Living_Room;
481                 (b_S1_Med_5PM_Remind.is_scheduled & step = step_2):
                       __14___Living_Room_Sofa_Area_in_the_Living_Room;
482                 (b_S1_ReturnHome.is_scheduled & step = step_3):
                       __5___ChargingStation_Area_in_the_Dining_Room;
483                 (b_S1_watchTV.is_scheduled & step = step_3):
                       __14___Living_Room_Sofa_Area_in_the_Living_Room;
484                 (b_T_medicine.is_scheduled & step = step_2):
                       __999___Current_user_Location;
485                 (b_T_moveTo_person.is_scheduled & step = step_2):
                       __999___Current_user_Location;
486
487                 TRUE: location;
488             esac;
489
490         init(says):= none;
491         next(says):=
492             case
```

```
493                 (b_answerDoorBell.is_scheduled & step = step_1):
                       _Doorbell_;
494                 (b_S1_alertFridgeDoor.is_scheduled & step = step_4):
                       _The_fridge_door_is_open__;
495                 (b_S1_Med_5PM.is_scheduled & step = step_4):
                       _Its_time_for_your_medicine_;
496                 (b_S1_Med_5PM_Remind.is_scheduled & step = step_4):
                       _Have_you_taken_your_medicine_;
497                 (b_S1_watchTV.is_scheduled & step = step_5):
                       _Shall_we_watch_TV_together__;
498
499                 TRUE: none;
500             esac;
501
502         init(torso):= none;
503         next(torso):=
504             case
505                 (b_S1_continueWatchTV.is_scheduled & step = step_5):
                       the_right;
506                 (b_S1_continueWatchTV.is_scheduled & step = step_6):
                       the_back_position;
507
508                 TRUE: none;
509             esac;
510
511         init(tray):= Lowered;
512         next(tray):=
513             case
514                 (b_lowerTray.is_scheduled & step = step_2): Lowered;
515                 (b_raiseTray.is_scheduled & step = step_2): Raised;
516
517                 TRUE: tray;
518             esac;
519
520
521         init(Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE):= s_0;
522         next(Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE):=
523             case
524                 Fridge_Freezer_In__ON_ = FALSE: s_0;
525                 (Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE = s_0 &
                       Fridge_Freezer_In__ON_ = TRUE): s_final;
526                 Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE = s_final:
                       s_final;
527
528                 TRUE: Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE;
529             esac;
530
531         init(__503__5PM_MedicineReminder_TRUE_BEEN_IN_STATE):= s_0;
532         next(__503__5PM_MedicineReminder_TRUE_BEEN_IN_STATE):=
533             case
534                 __503__5PM_MedicineReminder = FALSE: s_0;
535                 (__503__5PM_MedicineReminder_TRUE_BEEN_IN_STATE = s_0 &
                       __503__5PM_MedicineReminder = TRUE): s_final;
536                 __503__5PM_MedicineReminder_TRUE_BEEN_IN_STATE = s_final:
                       s_final;
537
```

```
538             TRUE: __503__5PM_MedicineReminder_TRUE_BEEN_IN_STATE;
539         esac;
540
541     init(__514__GOAL_fridgeUserAlerted_TRUE_BEEN_IN_STATE):= s_0;
542     next(__514__GOAL_fridgeUserAlerted_TRUE_BEEN_IN_STATE):=
543         case
544             __514__GOAL_fridgeUserAlerted = FALSE: s_0;
545             (__514__GOAL_fridgeUserAlerted_TRUE_BEEN_IN_STATE = s_0 &
                    __514__GOAL_fridgeUserAlerted = TRUE): s_final;
546             __514__GOAL_fridgeUserAlerted_TRUE_BEEN_IN_STATE =
                    s_final: s_final;
547
548             TRUE: __514__GOAL_fridgeUserAlerted_TRUE_BEEN_IN_STATE;
549         esac;
550
551     init(__513__GOAL_watchTV_FALSE_BEEN_IN_STATE):= s_0;
552     next(__513__GOAL_watchTV_FALSE_BEEN_IN_STATE):=
553         case
554             __513__GOAL_watchTV = TRUE: s_0;
555             (__513__GOAL_watchTV_FALSE_BEEN_IN_STATE = s_0 &
                    __513__GOAL_watchTV = FALSE): s_600;
556             __513__GOAL_watchTV_FALSE_BEEN_IN_STATE = s_600: s_1200;
557             __513__GOAL_watchTV_FALSE_BEEN_IN_STATE = s_1200: s_1800;
558             __513__GOAL_watchTV_FALSE_BEEN_IN_STATE = s_1800: s_2400;
559             __513__GOAL_watchTV_FALSE_BEEN_IN_STATE = s_2400: s_3000;
560             __513__GOAL_watchTV_FALSE_BEEN_IN_STATE = s_3000: s_final;
561             __513__GOAL_watchTV_FALSE_BEEN_IN_STATE = s_final:
                    s_final;
562
563             TRUE: __513__GOAL_watchTV_FALSE_BEEN_IN_STATE;
564         esac;
565
566     init(__515__GOAL_AnserDoorBell_TRUE_BEEN_IN_STATE):= s_0;
567     next(__515__GOAL_AnserDoorBell_TRUE_BEEN_IN_STATE):=
568         case
569             __515__GOAL_AnserDoorBell = FALSE: s_0;
570             (__515__GOAL_AnserDoorBell_TRUE_BEEN_IN_STATE = s_0 &
                    __515__GOAL_AnserDoorBell = TRUE): s_final;
571             __515__GOAL_AnserDoorBell_TRUE_BEEN_IN_STATE = s_final:
                    s_final;
572
573             TRUE: __515__GOAL_AnserDoorBell_TRUE_BEEN_IN_STATE;
574         esac;
575
576
577     init(Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE):= s_0;
578     next(Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE):=
579         case
580             Doorbell_Last_Wattage___1 = TRUE: s_final;
581             Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE = s_final:
                    s_0;
582
583             TRUE: Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE;
584         esac;
585
586
```

```
587
588     ------------------------------------------------------------------------------
589     --  Definitions
590     ------------------------------------------------------------------------------
591     DEFINE
592         pre_answerDoorBell:= __515__GOAL_AnserDoorBell;
593         pre_checkBell:= Doorbell_Last_Wattage___1 &
                Doorbell_Last_Wattage___1_TRUE_WAS_IN_STATE != s_0;
594         pre_lowerTray:= (__504__TrayIsEmpty & __500__TrayIsRaised);
595         pre_raiseTray:= __501__TrayIsLowered;
596         pre_S1_alertFridgeDoor:= (!__514__GOAL_fridgeUserAlerted &
                Fridge_Freezer_In__ON_ &
                Fridge_Freezer_In__ON__TRUE_BEEN_IN_STATE = s_final);
597         pre_S1_continueWatchTV:= (Television_Wattage___10 &
                __513__GOAL_watchTV);
598         pre_S1_goToKitchen:= __505__GOAL_gotoKitchen;
599         pre_S1_gotoSofa:= __508__GOAL_gotoSofa;
600         pre_S1_gotoTable:= __507__GOAL_gotoTable;
601         pre_S1_kitchenAwaitCmd:= (__509__GOAL_waitAtKitchen & location =
                __7___Kitchen_Entrance_in_the_Dining_Room);
602         pre_S1_Med_5PM:= (__502__5PM_MedicineDue & time =
                _17_00_00_to_23_59_00);
603         pre_S1_Med_5PM_Remind:= __503__5PM_MedicineReminder &
                __503__5PM_MedicineReminder_TRUE_BEEN_IN_STATE = s_final;
604         pre_S1_Med_5PM_Reset:= (!__502__5PM_MedicineDue & time =
                _00_00_00_to_16_59_00);
605         pre_S1_remindFridgeDoor:= __514__GOAL_fridgeUserAlerted &
                __514__GOAL_fridgeUserAlerted_TRUE_BEEN_IN_STATE = s_final;
606         pre_S1_ResetAllGoals:= TRUE;
607         pre_S1_ReturnHome:= __506__GOAL_gotoCharger;
608         pre_S1_Set_Continue:= TRUE;
609         pre_S1_Set_GoToKitchen:= TRUE;
610         pre_S1_Set_GoToSofa:= TRUE;
611         pre_S1_Set_GoToTable:= TRUE;
612         pre_S1_Set_ReturnHome:= TRUE;
613         pre_S1_Set_WaitHere:= TRUE;
614         pre_S1_Set_Watch_TV:= TRUE;
615         pre_S1_sleep:= TRUE;
616         pre_S1_sofaAwaitCmd:= (__510__GOAL_waitAtSofa & location =
                __14___Living_Room_Sofa_Area_in_the_Living_Room);
617         pre_S1_tableAwaitCmd:= (__511__GOAL_waitAtTable & location =
                __23___Living_Room_Table_in_the_Living_Room_Sofa_Area_of_the_Living_Room);
618         pre_S1_WaitHere:= __512__GOAL_waitHere;
619         pre_S1_watchTV:= (!__513__GOAL_watchTV &
                __513__GOAL_watchTV_FALSE_BEEN_IN_STATE = s_final &
                (Television_Wattage___10 & (seat_occupied = seat_5 |
                (seat_occupied = seat_4 | (seat_occupied = seat_3 |
                (seat_occupied = seat_2 | seat_occupied = seat_1))))));
620         pre_T_medicine:= time = _17_00_00_to_23_59_00;
621         pre_T_moveTo_person:= TRUE;
622         pre_unCheckBell:= __515__GOAL_AnserDoorBell &
                __515__GOAL_AnserDoorBell_TRUE_BEEN_IN_STATE = s_final;
623
624         can_interrupt_10:= FALSE;
625         can_interrupt_30:= (can_interrupt_10 | (schedule =
                schedule_S1_sleep));
```

```
626         can_interrupt_35:= (can_interrupt_30 | (schedule =
               schedule_S1_watchTV));
627         can_interrupt_40:= (can_interrupt_35 | (schedule =
               schedule_S1_continueWatchTV));
628         can_interrupt_50:= (can_interrupt_40 | (schedule =
               schedule_S1_WaitHere | schedule = schedule_S1_tableAwaitCmd |
               schedule = schedule_S1_sofaAwaitCmd | schedule =
               schedule_S1_ReturnHome | schedule =
               schedule_S1_kitchenAwaitCmd | schedule =
               schedule_S1_gotoTable | schedule = schedule_S1_gotoSofa |
               schedule = schedule_S1_goToKitchen));
629         can_interrupt_60:= (can_interrupt_50 | (schedule =
               schedule_S1_Med_5PM_Remind));
630         can_interrupt_70:= (can_interrupt_60);
631         can_interrupt_80:= (can_interrupt_70);
632         can_interrupt_90:= (can_interrupt_80);
633
634         executed_behaviour_execute_next:=
               ((!b_S1_alertFridgeDoor.can_be_interrupted &
               b_S1_alertFridgeDoor.is_scheduled & step = step_8) |
               (!b_S1_continueWatchTV.can_be_interrupted &
               b_S1_continueWatchTV.is_scheduled & step = step_2) |
               (!b_S1_continueWatchTV.can_be_interrupted &
               b_S1_continueWatchTV.is_scheduled & step = step_7) |
               (!b_S1_goToKitchen.can_be_interrupted &
               b_S1_goToKitchen.is_scheduled & step = step_2) |
               (!b_S1_goToKitchen.can_be_interrupted &
               b_S1_goToKitchen.is_scheduled & step = step_4) |
               (!b_S1_gotoSofa.can_be_interrupted &
               b_S1_gotoSofa.is_scheduled & step = step_2) |
               (!b_S1_gotoTable.can_be_interrupted &
               b_S1_gotoTable.is_scheduled & step = step_2) |
               (!b_S1_kitchenAwaitCmd.can_be_interrupted &
               b_S1_kitchenAwaitCmd.is_scheduled & step = step_1) |
               (!b_S1_Med_5PM.can_be_interrupted & b_S1_Med_5PM.is_scheduled
               & step = step_5) | (!b_S1_ReturnHome.can_be_interrupted &
               b_S1_ReturnHome.is_scheduled & step = step_2) |
               (!b_S1_sofaAwaitCmd.can_be_interrupted &
               b_S1_sofaAwaitCmd.is_scheduled & step = step_1) |
               (!b_S1_tableAwaitCmd.can_be_interrupted &
               b_S1_tableAwaitCmd.is_scheduled & step = step_1) |
               (!b_S1_WaitHere.can_be_interrupted &
               b_S1_WaitHere.is_scheduled & step = step_6) |
               (!b_S1_watchTV.can_be_interrupted & b_S1_watchTV.is_scheduled
               & step = step_2) | (!b_S1_watchTV.can_be_interrupted &
               b_S1_watchTV.is_scheduled & step = step_7));
635         a_behaviour_can_be_scheduled:=
               (b_S1_Med_5PM_Reset.can_be_scheduled |
               b_unCheckBell.can_be_scheduled |
               b_S1_remindFridgeDoor.can_be_scheduled |
               b_checkBell.can_be_scheduled |
               b_answerDoorBell.can_be_scheduled |
               b_S1_alertFridgeDoor.can_be_scheduled |
               b_S1_Med_5PM_Remind.can_be_scheduled |
               b_S1_goToKitchen.can_be_scheduled |
               b_S1_gotoSofa.can_be_scheduled |
```

```
            b_S1_gotoTable.can_be_scheduled |
            b_S1_kitchenAwaitCmd.can_be_scheduled |
            b_S1_ReturnHome.can_be_scheduled |
            b_S1_sofaAwaitCmd.can_be_scheduled |
            b_S1_tableAwaitCmd.can_be_scheduled |
            b_S1_WaitHere.can_be_scheduled |
            b_S1_continueWatchTV.can_be_scheduled |
            b_S1_watchTV.can_be_scheduled | b_S1_sleep.can_be_scheduled);
636         a_behaviour_is_ending:= (b_S1_Med_5PM_Reset.is_last_step |
            b_unCheckBell.is_last_step |
            b_S1_remindFridgeDoor.is_last_step | b_checkBell.is_last_step
            | b_answerDoorBell.is_last_step |
            b_S1_alertFridgeDoor.is_last_step |
            b_S1_Med_5PM_Remind.is_last_step |
            b_S1_goToKitchen.is_last_step | b_S1_gotoSofa.is_last_step |
            b_S1_gotoTable.is_last_step |
            b_S1_kitchenAwaitCmd.is_last_step |
            b_S1_ReturnHome.is_last_step | b_S1_sofaAwaitCmd.is_last_step
            | b_S1_tableAwaitCmd.is_last_step |
            b_S1_WaitHere.is_last_step |
            b_S1_continueWatchTV.is_last_step | b_S1_watchTV.is_last_step
            | b_S1_sleep.is_last_step);
637         an_executed_behaviour_is_ending_as_a_last_action:=
            ((b_S1_Set_ReturnHome.is_last_step & last_schedule =
            schedule_S1_continueWatchTV) |
            (b_S1_Set_Continue.is_last_step & last_schedule =
            schedule_S1_continueWatchTV) |
            (b_S1_Set_WaitHere.is_last_step & last_schedule =
            schedule_S1_WaitHere) | (b_S1_Set_ReturnHome.is_last_step &
            last_schedule = schedule_S1_WaitHere) |
            (b_S1_Set_Continue.is_last_step & last_schedule =
            schedule_S1_WaitHere) | (b_S1_Set_Watch_TV.is_last_step &
            last_schedule = schedule_S1_watchTV) |
            (b_S1_Set_ReturnHome.is_last_step & last_schedule =
            schedule_S1_watchTV) | (b_S1_Set_Continue.is_last_step &
            last_schedule = schedule_S1_watchTV));
638         an_executed_behaviour_is_ending:= (b_S1_Set_WaitHere.is_last_step
            | b_raiseTray.is_last_step | b_lowerTray.is_last_step |
            b_S1_Set_GoToKitchen.is_last_step |
            b_S1_Set_Continue.is_last_step |
            b_S1_Set_GoToSofa.is_last_step |
            b_S1_Set_GoToTable.is_last_step |
            b_S1_Set_ReturnHome.is_last_step |
            b_S1_Set_Watch_TV.is_last_step);
639         an_executed_behaviour_is_scheduled:=
            (b_S1_Set_WaitHere.is_scheduled | b_raiseTray.is_scheduled |
            b_lowerTray.is_scheduled | b_S1_Set_GoToKitchen.is_scheduled
            | b_S1_Set_Continue.is_scheduled |
            b_S1_Set_GoToSofa.is_scheduled |
            b_S1_Set_GoToTable.is_scheduled |
            b_S1_Set_ReturnHome.is_scheduled |
            b_S1_Set_Watch_TV.is_scheduled);
640
641
642 ------------------------------------------------------------------------
643 --  Behaviour Module
```

```
644 ------------------------------------------------------------------------------
645 MODULE behaviour(preconditions, can_interrupt, can_be_int, schedule,
      this_schedule, step, last_step)
646
647     DEFINE
648         preconditions_hold:= preconditions;
649         can_be_scheduled:= ((schedule = schedule_none | can_interrupt) &
              preconditions_hold);
650         can_be_interrupted:= can_be_int;
651         is_last_step:= (is_scheduled & step = last_step);
652         is_scheduled:= (schedule = this_schedule);
```

12.2.2 Testing of Expected Properties

Variable Assignments For any behaviour named N having as its k^{th} action a propositional variable assignment or enumerated variable assignment, where some value v is assigned to some variable var we would expected the following property to hold as var should have the value v in the next moment in time:

$$\Box\ ((\text{schedule} = \text{schedule\_N} \wedge \text{step} = \text{step\_k}) \Rightarrow \bigcirc(var = v))$$

All of the following properties are expected to be true in the model.

Test 1 If the 1^{st} action of the *answerDoorBell* behaviour is being executed then in the next step the robot should say 'Doorbell'.

Property

$\Box\ ((\text{schedule} = \text{schedule\_answerDoorBell} \wedge \text{step} = \text{step\_1}) \Rightarrow \bigcirc(\text{says} = \text{'Doorbell'}))$

Result

```
-- specification  G ((schedule = schedule_answerDoorBell & step = step_1) ->  X
says = _Doorbell_)  is true
```

Test 2 If the 3^{rd} action of the *S1-alertFridgeDoor* behaviour is being executed then in the next step the robot's light should be white.

Property

$\Box\ ((\text{schedule} = \text{schedule\_S1\_alertFridgeDoor} \wedge \text{step} = \text{step\_3}) \Rightarrow \bigcirc(\text{light} = \text{white}))$

Result

```
-- specification  G ((schedule = schedule_S1_alertFridgeDoor & step = step_3) ->
  X light = white)  is true
```

Test 3 If the 1^{st} action of the *uncheckBell* behaviour is being executed then in the robot's internal flag *::515::GOAL_AnserDoorBell* should be set to false.

Property

$\Box\ ((\text{schedule} = \text{schedule\_uncheckBell} \wedge \text{step} = \text{step\_1}) \Rightarrow \bigcirc(\text{::515::GOAL-AnserDoorBell} = \text{false}))$

Result

```
-- specification  G ((schedule = schedule_unCheckBell & step = step_1) ->  X __5
15__GOAL_AnserDoorBell = FALSE)  is true
```

Test 4 If the 6^{th} action of the *S1-watchTV* behaviour is being executed then in the next step the robot's internal flag *::513::GOAL-watchTV* should be set to true.

Property

$\Box$ ((schedule = schedule_S1-watchTV $\wedge$ step = step_6) $\Rightarrow$ $\bigcirc$(::513::GOAL-watchTV = true))

Result

```
-- specification  G ((schedule = schedule_S1_watchTV & step = step_6) ->  X __51
3__GOAL_watchTV = TRUE)  is true
```

Property 3 Test Results These tests correspond to property 3 defined in Section 10.7:

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

Test 1 It is always the case that if no behaviour is currently scheduled, the preconditions of the *S1_remindFridgeDoor* behaviour hold, and the preconditions of the *checkBell* and *uncheckBell* behaviours do not hold, then in the next moment in time the *S1_remindFridgeDoor* behaviour should be scheduled and should be executing it first action. This is expected to be true as all three behaviours have equal priority, and have the highest priority of all schedulable behaviours, therefore if no behaviour is currently scheduled and the preconditions to only one of these behaviours holds, then in the next moment in time that behaviour should be scheduled.

Property

$\Box$((schedule = schedule_none $\wedge$ b_S1_remindFridgeDoor.preconditions_hold
$\wedge$ $\neg$b_checkBell.preconditions_hold $\wedge$ $\neg$b_unCheckBell.preconditions_hold)
$\Rightarrow$ $\bigcirc$(schedule = schedule_S1_remindFridgeDoor $\wedge$ step = step_1))

Result

```
-- specification  G ((((schedule = schedule_none & b_S1_remindFridgeDoor.precond
itions_hold) & !b_checkBell.preconditions_hold) & !b_unCheckBell.preconditions_h
old) ->  X (schedule = schedule_S1_remindFridgeDoor & step = step_1))  is true
```

Test 2 It is always the case that if no behaviour is currently scheduled, the preconditions of the *answerDoorBell* behaviour hold, and the preconditions of the *S1-remindFridgeDoor*,*checkBell* and *uncheckBell* behaviours do not hold, then in the next moment in time the *S1-answerDoorBell* behaviour should be scheduled and should be executing its first action. This is expected to be true as the preconditions of all schedulable behaviours having a higher priority than *answerDoorBell* do not hold, therefore it should be scheduled in the next moment in time.

Property

$\Box$((schedule = schedule_none $\wedge$ b_answerDoorBell.preconditions_hold
$\wedge$ $\neg$b_checkBell.preconditions_hold $\wedge$ $\neg$b_unCheckBell.preconditions_hold)
$\wedge$ $\neg$b_S1_remindFridgeDoor.preconditions_hold
$\Rightarrow$ $\bigcirc$(schedule = schedule_answerDoorBell $\wedge$ step = step_1))

Result

```
-- specification  G (((((schedule = schedule_none & b_answerDoorBell.preconditio
ns_hold) & !b_checkBell.preconditions_hold) & !b_unCheckBell.preconditions_hold)
 & !b_S1_remindFridgeDoor.preconditions_hold) ->  X (schedule = schedule_answerD
oorBell & step = step_1))  is true
```

Test 3 It is always the case that if no behaviour is currently scheduled, the preconditions of the *S1-alertFridgeDoor* behaviour hold, and the preconditions of the *alertFridgeDoor*, *S1-remindFridgeDoor*,*checkBell* and *uncheckBell* behaviours do not hold, then in the next moment in time the *S1-alertFridgeDoor* behaviour should be scheduled and should be executing its first action. This is expected to be true as the preconditions of all schedulable behaviours having a higher priority than *S1-alertFridgeDoor* do not hold, therefore it should be scheduled in the next moment in time.

Property

$\Box$((schedule = schedule_none $\wedge$ b_S1_alertFridgeDoor.preconditions_hold
$\wedge$ $\neg$b_answerDoorBell.preconditions_hold $\wedge$ $\neg$b_checkBell.preconditions_hold
$\wedge$ $\neg$b_unCheckBell.preconditions_hold) $\wedge$ $\neg$b_S1_remindFridgeDoor.preconditions_hold
$\Rightarrow$ $\bigcirc$(schedule = schedule_S1_alertFridgeDoor $\wedge$ step = step_1))

Result

```
-- specification  G ((((((schedule = schedule_none & b_S1_alertFridgeDoor.precon
ditions_hold) & !b_answerDoorBell.preconditions_hold) & !b_checkBell.preconditio
ns_hold) & !b_unCheckBell.preconditions_hold) & !b_S1_remindFridgeDoor.precondit
ions_hold) ->  X (schedule = schedule_S1_alertFridgeDoor & step = step_1))  is t
rue
```

Property 4 Test Results These tests correspond to property 4 defined in Section 10.7:

"If a scheduled behaviour is interruptible, and the preconditions to one or more schedulable behaviours with a higher priority hold, then in the next moment in time the schedulable behaviour having the highest priority of all these behaviours will interrupt the currently scheduled behaviour and will be executing its first action."

Test 1 It is always the case that if the *S1-gotoTable* behaviour is scheduled and the preconditions to the *S1-remindFridgeDoor* behaviour hold, and the preconditions to the *uncheckBell* and *checkBell* behaviours do not hold, then in the next moment in time the *S1-remindFridgeDoor* behaviour should be scheduled and executing its first action. This is expected to be true as the *S1-gotoTable* behaviour is interruptible, the priority of the *S1-remindFridgeDoor* behaviour is greater than the priority of the *S1-gotoTable* behaviour, and no other schedulable behaviour has a priority greater than or equal to the priority of the *S1-remindFridgeDoor* behaviour with the exception of the *uncheckBell* and *checkBell* behaviours.

Property

$\Box$((schedule = schedule_S1_gotoTable $\wedge$ b_S1_remindFridgeDoor.preconditions_hold
$\wedge$ $\neg$b_checkBell.preconditions_hold $\wedge$ $\neg$b_unCheckBell.preconditions_hold
$\Rightarrow$ $\bigcirc$(schedule = schedule_S1_remindFridgeDoor $\wedge$ step = step_1))

Result

```
-- specification  G ((((schedule = schedule_S1_gotoTable & b_S1_remindFridgeDoor
.preconditions_hold) & !b_checkBell.preconditions_hold) & !b_unCheckBell.precond
itions_hold) ->  X (schedule = schedule_S1_remindFridgeDoor & step = step_1))  i
s true
```

Test 2 It is always the case that if the *S1-gotoTable* behaviour is scheduled and the preconditions to the *S1-alertFridgeDoor* behaviour hold, then in the next moment in time the *S1-remindFridgeDoor* behaviour should be scheduled and executing its first action. This is expected to be false as the there are other behaviours with a higher priority than the *S1-alertFridgeDoor* behaviour that could interrupt the *S1-gotoTable* behaviour instead.

Property
$\Box$((schedule = schedule_S1_gotoTable)
$\Rightarrow \bigcirc$(schedule = schedule_S1_remindFridgeDoor $\wedge$ step = step_1))

Result

```
-- specification  G ((schedule = schedule_S1_gotoTable & b_S1_alertFridgeDoor.pr
econditions_hold) ->  X (schedule = schedule_S1_alertFridgeDoor & step = step_1)
)  is false
```

Property 5 Test Results These tests correspond to property 5 defined in Section 10.7:

"If a behaviour scheduled and is executing its k^{th} action, this action does not execute another behaviour and is not the final action, and in the next moment in time the scheduled behaviour will not be interrupted by another behaviour, then in the next moment in time the scheduled behaviour will be executing its $(k+1)^{th}$ action."

All of the following properties are expected to be true in the model.

Test 1 It is always the case that if the *S1-gotoSofa* behaviour is scheduled and is executing its first action and it is not the case that this behaviour can be interrupted in the next moment in time, then in the next moment in time this behaviour will be executing its second action.

Property
$\Box$((schedule = schedule_S1_gotoSofa $\wedge$ step = step_1
$\wedge$ $\neg$b_S1_gotoSofa.can_be_interrupted) $\Rightarrow \bigcirc$(schedule = schedule_S1_gotoSofa
$\wedge$ step = step_2))

Result

```
-- specification  G (((schedule = schedule_S1_gotoSofa & step = step_1) & !b_S1_
gotoSofa.can_be_interrupted) ->  X (schedule = schedule_S1_gotoSofa & step = ste
p_2))  is true
```

Test 2 It is always the case that if the *T-moveTo-person* behaviour is scheduled and is executing its second action and it is not the case that this behaviour can be interrupted in the next moment in time, then in the next moment in time this behaviour will be executing its third action.

Property
$\Box$((schedule = schedule_T_moveTo_person $\wedge$ step = step_2
$\wedge$ $\neg$b_T_moveTo_person.can_be_interrupted)
$\Rightarrow \bigcirc$(schedule = schedule_T_moveTo_person $\wedge$ step = step_3))

Result

```
-- specification  G (((schedule = schedule_T_moveTo_person & step = step_2) & !b
_T_moveTo_person.can_be_interrupted) ->  X (schedule = schedule_T_moveTo_person
& step = step_3))  is true
```

Property 6 Test Results These tests correspond to property 6 defined in Section 10.7:

"If a behaviour scheduled and is executing another behaviour whose precondition holds, and in the next moment in time the scheduled behaviour will not be interrupted by another behaviour, then in the next moment in time the executed behaviour will be scheduled and will be executing its first action."

All of the following properties are expected to be true in the model.

Test 1 It is always the case that if the *S1-continueWatchTV* behaviour is scheduled and is executing its second action, and in the next moment in time the *S1-continueWatchTV* behaviour cannot be interrupted, and the preconditions for the *lowerTray* behaviour hold, then in the next moment in time the *lowerTray* behaviour will be scheduled and executing its first action

Property

$\Box$((schedule = schedule_S1_continueWatchTV $\wedge$ step = step_2
$\wedge$ $\neg$b_S1_continueWatchTV.can_be_interrupted $\wedge$ b_lowerTray.preconditions_hold)
$\Rightarrow$ $\bigcirc$(schedule = schedule_lowerTray $\wedge$ step = step_1))

Result

```
-- specification  G ((((schedule = schedule_S1_continueWatchTV & step = step_2)
& !b_S1_continueWatchTV.can_be_interrupted) & b_lowerTray.preconditions_hold) ->
  X (schedule = schedule_lowerTray & step = step_1))  is true
```

Test 2 It is always the case that if the *S1-goToKitchen* behaviour is scheduled and is executing its fourth action, and in the next moment in time the *S1-goToKitchen* behaviour cannot be interrupted, and the preconditions for the *raiseTray* behaviour hold, then in the next moment in time the *raiseTray* behaviour will be scheduled and executing its first action

Property

$\Box$((schedule = schedule_S1_goToKitchen $\wedge$ step = step_2
$\wedge$ $\neg$b_S1_goToKitchen.can_be_interrupted $\wedge$ b_raiseTray.preconditions_hold)
$\Rightarrow$ $\bigcirc$(schedule = schedule_raiseTray $\wedge$ step = step_1))

Result

```
-- specification  G ((((schedule = schedule_S1_goToKitchen & step = step_4) & !b
_S1_goToKitchen.can_be_interrupted) & b_raiseTray.preconditions_hold) ->  X (sch
edule = schedule_raiseTray & step = step_1))  is true
```

Property 7 Test Results These tests correspond to property 7 defined in Section 10.7:

"If a scheduled behaviour is executing its k^{th} action, this action executes another behaviour whose preconditions hold, this action is not the final action, and in the next moment in time the scheduled behaviour will not be interrupted by another behaviour, then in the next moment in time the executed behaviour will be scheduled and will be executing its first action and at some time after the executing behaviour will again be scheduled and will be performing its $(k+1)^{th}$ action."

All of the following properties are expected to be true in the model.

Test 1 It is always the case that if the *S1-continueWatchTV* behaviour is scheduled and is executing its second action, and in the next moment in time the *S1-continueWatchTV* behaviour cannot be interrupted, and the preconditions for the *lowerTray* behaviour hold, then in the next moment in time the *lowerTray* behaviour will be scheduled and executing its first action and at some time after that the *S1-continueWatchTV* behaviour will again be scheduled and will be executing its third action.

Property
$\Box$((schedule = schedule_S1_continueWatchTV $\wedge$ step = step_2
$\wedge$ $\neg$b_S1_continueWatchTV.can_be_interrupted $\wedge$ b_lowerTray.preconditions_hold)
$\Rightarrow$ $\bigcirc$(schedule = schedule_lowerTray $\wedge$ step = step_1
$\wedge$ $\Diamond$(schedule = schedule_S1_continueWatchTV $\wedge$ step = step 3)))

Result

```
-- specification  G ((((schedule = schedule_S1_continueWatchTV & step = step_2)
& !b_S1_continueWatchTV.can_be_interrupted) & b_lowerTray.preconditions_hold) ->
  X ((schedule = schedule_lowerTray & step = step_1) &  F (schedule = schedule_S
1_continueWatchTV & step = step_3)))  is true
```

Test 2 It is always the case that if the *S1-goToKitchen* behaviour is scheduled and is executing its fourth action, and in the next moment in time the *S1-goToKitchen* behaviour cannot be interrupted, and the preconditions for the *raiseTray* behaviour hold, then in the next moment in time the *raiseTray* behaviour will be scheduled and executing its first action and at some time after that the *S1-goToKitchen* behaviour will again be scheduled and will be executing its fifth action.

Property
$\Box$((schedule = schedule_S1_goToKitchen $\wedge$ step = step_4
$\wedge$ $\neg$b_S1_goToKitchen.can_be_interrupted $\wedge$ b_raiseTray.preconditions_hold)
$\Rightarrow$ $\bigcirc$(schedule = schedule_raiseTray $\wedge$ step = step_1
$\wedge$ $\Diamond$(schedule = schedule_S1_goToKitchen $\wedge$ step = step 5)))

Result

```
-- specification  G ((((schedule = schedule_S1_goToKitchen & step = step_4) & !b
_S1_goToKitchen.can_be_interrupted) & b_raiseTray.preconditions_hold) ->  X ((sc
hedule = schedule_raiseTray & step = step_1) &  F (schedule = schedule_S1_goToKi
tchen & step = step_5)))  is true
```

Property 9 Test Results These tests correspond to property 9 defined in Section 10.7:

"When an uninterruptible behaviour has been scheduled it is expected to execute all of its actions."

Test 1 It is always the case that if the *raiseTray* behaviour is scheduled and is executing its first action then at some future point the *raiseTray* behaviour will be scheduled and will be executing its last (fifth) action. This is expected to be true as this behaviour is uninterruptible.

Property
$\Box$((schedule = schedule_raiseTray $\wedge$ step = step_1)
$\Rightarrow$ $\bigcirc$(schedule = schedule_raiseTray $\wedge$ step = step_5))

Result

```
-- specification  G ((schedule = schedule_raiseTray & step = step_1) ->  F (sche
dule = schedule_raiseTray & step = step_5))  is true
```

Test 2 It is always the case that if the *S1-alertFridgeDoor* behaviour is scheduled and is executing its first action then at some future point the *S1-alertFridgeDoor* behaviour will be scheduled and will be executing its last (ninth) action. This is expected to be true as this behaviour is uninterruptible.

Property
$\Box$((schedule = schedule_S1_alertFridgeDoor $\wedge$ step = step_1)
$\Rightarrow$ $\bigcirc$(schedule = schedule_S1_alertFridgeDoor $\wedge$ step = step_9))

Result

```
-- specification  G ((schedule = schedule_S1_alertFridgeDoor & step = step_1) ->
  F (schedule = schedule_S1_alertFridgeDoor & step = step_9))  is true
```

Test 3 It is always the case that if the *S1-sleep* behaviour is scheduled and is executing its first action then at some future point the *S1-sleep* behaviour will be scheduled and will be executing its last (fifth) action. This is expected to be false as this behaviour is interruptible.

Property
$\Box$((schedule = schedule_S1_sleep $\wedge$ step = step_1)
$\Rightarrow$ $\bigcirc$(schedule = schedule_S1_sleep $\wedge$ step = step_5))

Result

```
-- specification  G ((schedule = schedule_S1_sleep & step = step_1) ->  F (sched
ule = schedule_S1_sleep & step = step_5))  is false
```

13 Appendix E - Selected Source Code Listings

Some important functions are listed here and referred to in the main text. Full code listings have been submitted electronically along with this document. Instructions on how to build the software using the source code, and both sets of input files used during the development of the software, are also included.

13.1 The *getByName* Procedure

```
/*-----------------------------------------------------------------------------
  getByName

  Finds and returns a precondition, action or behaviour with the given
  name. If automatic identifier matching is disabled then the user is
  prompted when ambiguity or case-insensitive matches are encountered.
.............................................................................
  @param  variable    the list of propositional variables/enumerated
                variables/behaviour
  @param  name      the name of the propositional variable/
                enumerated variable/behaviour to retrieve
  @param  type      text describing the type being disambiguated
                (either propositional variable, enumerated
                variable, or behaviour)
  @param  info      some information to display to the user
  @return         the propositional variable/enumerated variable/
                behaviour, or nullptr if none is found
-----------------------------------------------------------------------------*/
template<typename T>
T* IntermediateForm::getByName(std::list<T*>& instances, const
    std::string& name,
  const std::string& type, const std::string& info)
{
  T* case_insensitive_match = nullptr;
  std::list<std::pair<T*, const int>> similarity_matches;
  std::for_each(instances.begin(), instances.end(),
    [&](T* variable)
    {
      if(caseInsensitiveCompare(variable->getName(), name))
      {
        // a case-insensitive match has been found
        case_insensitive_match = variable;
      }
      else
      {
        auto pair = compareStrings(variable->getName(), name);
        if(pair.second >= g_string_matching_threshhold)
        {
          // a match has been found
          similarity_matches.push_back(std::pair<T*,
            const int>(variable, pair.second));
        }
      }
    });

  if(case_insensitive_match != nullptr)
  {
```

```
47      // a case insensitive match was found
48      if(case_insensitive_match->getName() != name)
49      {
50        if(!g_no_prompt_case_insensitivity)
51        {
52          /*  prompt the user to choose whether to replace the
53            existing identifier, to use the existing identifier,
54            or to simply ignore the match and treat both
55            identifiers as being distinct */
56          displayTitle(DIVIDER_DISAMBIGUATION, DIVIDER_2, info);
57          switch(disambiguationPrompt(case_insensitive_match->getName(),
58            type))
59          {
60          case DISAMBIGUATION_REPLACE_EXISTING:
61            case_insensitive_match->setName(name);
62            return case_insensitive_match;
63            break;
64
65          case DISAMBIGUATION_USE_EXISTING:
66            return case_insensitive_match;
67            break;
68
69          case DISAMBIGUATION_IGNORE:
70            return nullptr;
71            break;
72          }
73        }
74        else
75        {
76          // automatically match
77          displayTitle(DIVIDER_DISAMBIGUATION, DIVIDER_2,
78            "automatic case-insensitive match");
79          std::cout << "matched " << type << " \'" << name
80            << "\'\nwith existing " << type << " \'"
81            <<  case_insensitive_match->getName() << "\'\n";
82          return case_insensitive_match;
83        }
84      }
85      else
86      {
87        return case_insensitive_match;
88      }
89    }
90    else
91    {
92      if(similarity_matches.empty())
93      {
94        // no matches were found
95        return nullptr;
96      }
97      else
98      {
99        // sort the matches by similarity
100       similarity_matches.sort(
101         [&](const std::pair<T*, const int>& this_pair,
102           const std::pair<T*, const int>& that_pair)
```

```
103         {
104           return this_pair.second > that_pair.second;
105         });
106 
107       if(!g_no_prompt_identifier_matching)
108       {
109         /*  prompt the user to choose whether to replace the
110           existing identifier, to use the existing identifier,
111           or to simply ignore the match and treat both
112           identifiers as being distinct */
113         displayTitle(DIVIDER_DISAMBIGUATION, DIVIDER_2, info);
114         auto it = similarity_matches.begin();
115         auto end = similarity_matches.end();
116         int count = 1;
117         while(it != end)
118         {
119           std::pair<T*, const int> matched_pair = *it;
120           std::cout << "match " << count << " of "
121             << similarity_matches.size() << " ("
122             << matched_pair.second << "% similarity)\n";
123           switch(disambiguationPrompt(matched_pair.first->getName(),
124             type))
125           {
126           case DISAMBIGUATION_REPLACE_EXISTING:
127             matched_pair.first->setName(name);
128             return matched_pair.first;
129             break;
130 
131           case DISAMBIGUATION_USE_EXISTING:
132             return matched_pair.first;
133             break;
134 
135           case DISAMBIGUATION_IGNORE:
136             break;
137           }
138           count++;
139           it++;
140         }
141         return nullptr;
142       }
143       else
144       {
145         // automatically use the most similar match
146         displayTitle(DIVIDER_DISAMBIGUATION, DIVIDER_2,
147           "automatic similarity match");
148         std::locale loc;
149         auto first_pair = *similarity_matches.begin();
150         std::string matched_name = first_pair.first->getName();
151         int matched_name_spc = 0;
152         for(char c : matched_name)
153         {
154           if(std::isspace(c, loc))
155           {
156             matched_name_spc++;
157           }
158         }
```

```
159         int name_spc = 0;
160         for(char c : name)
161         {
162           if(std::isspace(c, loc))
163           {
164             name_spc++;
165           }
166         }
167         std::cout << "matched " << type << " \'" << name
168           << "\'\nwith existing " << type << " \'"
169           <<  first_pair.first->getName()
170           << "\'\nhaving " << first_pair.second << "% similarity\n";
171         if(name_spc < matched_name_spc)
172         {
173           std::cout << "new identifier contains less whitespace: "
174             << "replacing original identifier\n";
175           first_pair.first->setName(name);
176         }
177         return first_pair.first;
178       }
179     }
180   }
181   return nullptr;
182 }
```

13.2 The *parseGrammarFile* Procedure

```
/*-----------------------------------------------------------------------------
    parseGrammarFile

  Parses the given grammar file. Throws an error if parsing was
  unsuccessful.
.....................................................................
  @param  filename    the name of the file containing the grammar
                definitions
  @throw          an error message if parsing failed
-----------------------------------------------------------------------------*/
void IntermediateFormParser::parseGrammarFile(
  const std::string filename) throw (std::string)
{
  std::ifstream ifstream(filename);

  try
  {
    if(!ifstream)
    {
      throw("error loading file \'" + filename + "\'\n");
    }

    // read the file in as a string
    std::string input_file_string = std::string(
      std::istreambuf_iterator<char>(ifstream),
      std::istreambuf_iterator<char>());

    // do this while there is more to read
    while(hasNextToken(input_file_string))
    {
      std::string line = getNextLine(input_file_string);
      // get the left hand side non-terminal symbol
      std::string non_t = getNextToken(line);

      NonTerminalSymbol* new_non_terminal;
      if(!isNonTerminalSymbol(non_t))
      {
        throw("\'" + non_t + "\' bad non_terminal name on left\n");
      }
      if(getNextToken(line) != "::=")
      {
        throw("expected '::=' in file \'" + filename + "\'\n");
      }
      // create a new empty non-terminal symbol
      new_non_terminal =
        new NonTerminalSymbol(getSymbolText(non_t),
          Automaton::REPEAT_NONE);
      while(hasNextToken(line))
      {
        std::string token = getNextToken(line);
        bool is_non_terminal = false;
        bool is_terminal = false;
        bool is_predefined = false;

```

```
55          if(isPredefinedAutomaton(token))
56          {
57            is_predefined = true;
58          }
59          else if(isNonTerminalSymbol(token))
60          {
61            is_non_terminal = true;
62          }
63          else if(isTerminalSymbol(token))
64          {
65            is_terminal = true;
66          }
67
68          if(!is_non_terminal && !is_terminal && !is_predefined)
69          {
70            // this is an invalid token
71            delete new_non_terminal;
72            throw("\'" + token
73              + "\' is not a terminal or non-terminal\n");
74          }
75
76          int repeat;
77          std::string prefix = getSymbolPrefix(token);
78          if(prefix == "+")
79          {
80            repeat = Automaton::REPEAT_ONE_OR_MORE;
81          }
82          else if(isInteger(prefix))
83          {
84            std::istringstream stream(prefix);
85            stream >> repeat;
86          }
87          else
88          {
89            repeat = Automaton::REPEAT_NONE;
90          }
91
92          std::string symbol_name = getSymbolText(token);
93
94          if(is_terminal)
95          {
96            // create a new terminal symbol...
97            TerminalSymbol* new_terminal =
98              new TerminalSymbol(symbol_name,
99                  repeat);
100           // ...and add it to the list of automata in the non-terminal
101           new_non_terminal->addAutomaton(new_terminal);
102         }
103         else if(is_predefined)
104         {
105           // get a copy of the predefined automaton
106           Automaton* predefined_symbol =
107             getPredefinedAutomatonByName(symbol_name);
108           predefined_symbol->setRepeat(repeat);
109           // ...and add it to the list of automata in the non-terminal
110           new_non_terminal->addAutomaton(predefined_symbol);
```

```
111         }
112         else
113         {
114           // has this non-terminal symbol already been defined?
115           auto it = std::find_if(
116             non_terminal_symbols.begin(),
117             non_terminal_symbols.end(),
118             [&](NonTerminalSymbol* n)
119             {
120               return symbol_name == n->getName();
121             });
122           if(it == non_terminal_symbols.end())
123           {
124             // it was not defined, throw an error
125             delete new_non_terminal;
126             throw("non-terminal symbol " + symbol_name
127               + " was not already defined\n");
128           }
129           // it was already defined, make a copy of it
130           NonTerminalSymbol* n = (NonTerminalSymbol*)(*it)->getCopy();
131           n->setRepeat(repeat);
132           // ...and add it to the list of automata in the non-terminal
133           new_non_terminal->addAutomaton(n);
134         }
135       }
136       // check to see if the new non-terminal has already been defined
137       auto it = std::find_if(
138         non_terminal_symbols.begin(),
139         non_terminal_symbols.end(),
140         [&](NonTerminalSymbol* n)
141         {
142           return getSymbolText(non_t) == n->getName();
143         });
144       if(it != non_terminal_symbols.end())
145       {
146         // it was already defined, throw an error
147         delete new_non_terminal;
148         throw("non_terminal " + non_t + " is already defined\n");
149       }
150       int type;
151       if((type = getAutomatonType(getSymbolText(non_t)))
152         != NO_TYPE)
153       {
154         // this matched a type, add the automaton to the corresponding
155         // list
156         non_terminal_symbols_by_type[type - 1].push_back(
157           new_non_terminal);
158       }
159       // add the new non-terminal to the list of non-terminals
160       non_terminal_symbols.push_back(new_non_terminal);
161     }
162   }
163   catch(std::string& error)
164   {
165     throw(std::string("[parseGrammarDefinitions]->\n")
166       + "error in file \'" + filename + "\'\n"
```

```
167         + error);
168     }
169     ifstream.close();
170 }
```

13.3 The *buildBehaviourLists* Procedure

```
/*-----------------------------------------------------------------------------
  buildBehaviourLists

  Builds the lists of schedulable, executing, and executable behaviours.
-----------------------------------------------------------------------------*/
void NuSMVTranslator::buildBehaviourLists()
{
  std::list<Behaviour*> behaviours = intermediate_form->getBehaviours();
  if(!behaviours.empty())
  {
    auto it = behaviours.begin();
    auto end = behaviours.end();
    while(it != end)
    {
      Behaviour* behaviour = *it;
      if(behaviour->isSchedulable())
      {
        // this behaviour is schedulable so add it to the list of
        // schedulable behaviours
        schedulable_behaviours.push_back(behaviour);
      }
      bool executes_another = false;
      std::list<Action*> actions = behaviour->getActions();
      auto action_it = actions.begin();
      auto action_end = actions.end();
      while(action_it != action_end)
      {
        Action* action = *action_it;
        if(action->getActionType() == ActionType::EXECUTE)
        {
          ActionExecute* action_ex = (ActionExecute*)action;
          // add the executed behaviour to the list of
          // executable behaviours
          executable_behaviours.insert(
            intermediate_form->getBehaviourByName(
              action_ex->getBehaviour(), ""));
          executes_another = true;
        }
        else if(action->getActionType() == ActionType::EXECUTE_NON_D)
        {
          ActionExecuteNonDeterministic* action_ex_non_d =
            (ActionExecuteNonDeterministic*)action;
          // add the executable behaviours to the list of
          // executable behaviours
          for(std::string behaviour_name :
            action_ex_non_d->getBehaviourValues())
          {
            executable_behaviours.insert(
              intermediate_form->getBehaviourByName(
                behaviour_name, ""));
          }
          executes_another = true;
        }
        action_it++;
```

```
55        }
56        if(executes_another)
57        {
58          // this behaviour executes another behaviour, add it
59          // to the list of executing behaviours
60          executing_behaviours.push_back(behaviour);
61        }
62        it++;
63      }
64    }
65 }
```

13.4 The *buildTimingConstraintMap* Procedure

```
/*---------------------------------------------------------------------------
  buildTimingConstraintMap

  Builds the timing constraint map.
---------------------------------------------------------------------------*/
void NuSMVTranslator::buildTimingConstraintMap()
{
  std::list<Behaviour*> behaviours = intermediate_form->getBehaviours();
  std::list<PreconditionTimingConstraint*> timing_constraints;
  if(!behaviours.empty())
  {
    auto it = behaviours.begin();
    auto end = behaviours.end();
    while(it != end)
    {
      Behaviour* behaviour = *it;
      std::list<Precondition*> preconditions =
        behaviour->getPreconditions();
      auto precondition_it = preconditions.begin();
      auto precondition_end = preconditions.end();
      while(precondition_it != precondition_end)
      {
        Precondition* precondition = *precondition_it;
        if(precondition->getPreconditionType()
          == PreconditionType::TIMING_CONSTRAINT)
        {
          // this precondition is a timing constraint, add it
          // to the map
          PreconditionTimingConstraint* precondition_t =
            (PreconditionTimingConstraint*)precondition;
          timing_constraints.push_back(precondition_t);
          timing_constraint_map.insert(
            std::pair<PreconditionTimingConstraint*,
            std::list<std::string>*>(precondition_t,
              new std::list<std::string>()));
        }
        precondition_it++;
      }
      it++;
    }
  }
  if(timing_constraints.size() == 1)
  {
    // there is only one timing constraint, build a value for it
    // and add a no_time_constraints_hold value
    PreconditionTimingConstraint* p = timing_constraints.front();
    std::string value = validateIdentifier("_" + p->getStartTime() +
        "_to_"
      + p->getEndTime());
    auto pair = *(timing_constraint_map.find(p));
    pair.second->push_back(value);
    time_intervals.push_back(value);
    time_intervals.push_back("no_time_constraints_hold");
  }
```

```
54  else if(!timing_constraints.empty())
55  {
56    std::string time = "00:00:00";
57    bool added_none_accepting_interval = false;
58    std::list<PreconditionTimingConstraint*> accepting;
59    // remove any time constraints from the list that hold
60    // at the current time, and add them to the accepting list
61    timing_constraints.remove_if(
62      [&](PreconditionTimingConstraint* p)
63      {
64        bool remove_if = timeConstraintHoldsAtTime(p, time);
65        if(remove_if)
66        {
67          accepting.push_back(p);
68        }
69        return remove_if;
70      });
71    // do this while there are still some time constraints that
72    // are yet to be accepting, or are still accepting
73    while(!timing_constraints.empty() || !accepting.empty())
74    {
75      PreconditionTimingConstraint* earliest_starting = nullptr;
76      PreconditionTimingConstraint* earliest_finishing = nullptr;
77      std::for_each(timing_constraints.begin(), timing_constraints.end(),
78        [&](PreconditionTimingConstraint* p)
79        {
80          // get the timing constraint with the earliest
81          // start time
82          if(earliest_starting == nullptr ||
83            timeIsBefore(p->getStartTime(),
84              earliest_starting->getStartTime()))
85          {
86            earliest_starting = p;
87          }
88        });
89      std::for_each(accepting.begin(), accepting.end(),
90        [&](PreconditionTimingConstraint* p)
91        {
92          // get the timing constraint with the earliest
93          // finishing time
94          if(earliest_finishing == nullptr ||
95            timeIsBefore(p->getEndTime(),
96              earliest_finishing->getEndTime()))
97          {
98            earliest_finishing = p;
99          }
100       });
101     if(earliest_starting == nullptr || (!(earliest_finishing == nullptr)
102       && timeIsBefore(earliest_finishing->getEndTime(),
103         earliest_starting->getStartTime())))
104     {
105       std::string value = validateIdentifier("_" + time + "_to_" +
106         earliest_finishing->getEndTime());
107       if(accepting.empty())
108       {
109         if(!added_none_accepting_interval)
```

```
110           {
111             time_intervals.push_back(
112               "no_time_constraints_hold");
113             added_none_accepting_interval = true;
114           }
115         }
116         else
117         {
118           time_intervals.push_back(value);
119         }
120         accepting.remove_if(
121           [&](PreconditionTimingConstraint* p)
122           {
123             bool remove_if = p->getEndTime() ==
124               earliest_finishing->getEndTime();
125             auto pair = *(timing_constraint_map.find(p));
126             pair.second->push_back(value);
127             return remove_if;
128           });
129         time = momentAfter(earliest_finishing->getEndTime());
130       }
131       else
132       {
133         if(timeIsBefore(time, earliest_starting->getStartTime()))
134         {
135           std::string value = validateIdentifier("_" + time + "_to_"
136             + momentBefore(earliest_starting->getStartTime()));
137           if(accepting.empty())
138           {
139             if(!added_none_accepting_interval)
140             {
141               time_intervals.push_back(
142                 "no_time_constraints_hold");
143               added_none_accepting_interval = true;
144             }
145           }
146           else
147           {
148             time_intervals.push_back(value);
149           }
150           for(PreconditionTimingConstraint* p : accepting)
151           {
152             auto pair = *(timing_constraint_map.find(p));
153             pair.second->push_back(value);
154           }
155         }
156         timing_constraints.remove_if(
157           [&](PreconditionTimingConstraint* p)
158           {
159             bool remove_if = p->getStartTime() ==
160               earliest_starting->getStartTime();
161             if(remove_if)
162             {
163               accepting.push_back(p);
164             }
165             return remove_if;
```

```
166             });
167           time = earliest_starting->getStartTime();
168         }
169       }
170     }
171 }
```

13.5 The *vectorPowerSet* Procedure

```
1 /*-------------------------------------------------------------------------
2   vectorPowerSet
3
4   Given a vector v of type T, returns a vector of vectors of type T
5   corresponding to the power set of the vector v.
6 .........................................................................
7   @param  T       the type of elements in the vector
8   @param  v       a vector of elements of type T
9   @param  sort_increasing_size  true if the powerset of vectors should
10              be sorted into increasing size, or false if the
11              set should be sorted into decreasing size.
12  @return         the powerset of v as a vector of vectors of type
13              T
14 -------------------------------------------------------------------------*/
15 template<typename T>
16 std::vector<std::vector<T>>
       NuSMVTranslator::vectorPowerSet(std::vector<T>& v,
17   const bool sort_increasing_size)
18 {
19   std::vector<std::vector<T>> power_set;
20   int num_items = v.size();
21   int num_subsets = static_cast<int>(std::pow(2.0, num_items));
22   for(int characteristic_vector = 0; characteristic_vector < num_subsets;
23     characteristic_vector++)
24   {
25     power_set.push_back(filterVectorByCharacteristicVector(v,
26       characteristic_vector));
27   }
28
29   if(sort_increasing_size)
30   {
31     std::sort(power_set.begin(), power_set.end(),
32       [&](std::vector<T> this_v, std::vector<T> that_v)
33       {
34         return that_v.size() < this_v.size();
35       });
36   }
37   else
38   {
39     std::sort(power_set.begin(), power_set.end(),
40       [&](std::vector<T> this_v, std::vector<T> that_v)
41       {
42         return this_v.size() > that_v.size();
43       });
44   }
45
46   return power_set;
47 }
48
49 /*-------------------------------------------------------------------------
50   filterVectorByCharacteristicVector
51
52   Returns a vector containing the elements of the vector v, filtered by
53   the given characteristic vector.
```

```
54 ..............................................................................
55   @param  T       the type of elements in the vector
56   @param  v       a vector of elements of type T
57   @param  characteristic_vector a characteristic vector, as an integer
58               from 0 to (|v|^2)-1, determining which elements
59               of v should be included in the returned vector
60   @return         the vector v filtered by the characteristic
61               vector
62 ------------------------------------------------------------------------------*/
63 template<typename T>
64 std::vector<T> NuSMVTranslator::filterVectorByCharacteristicVector(
65   std::vector<T>& v, int characteristic_vector)
66 {
67   std::vector<T> subset;
68   int index = 0;
69   while(characteristic_vector)
70   {
71     if(characteristic_vector & 1)
72     {
73       subset.push_back(v[index]);
74     }
75     index++;
76     characteristic_vector >>= 1;
77   }
78   return subset;
79 }
```

14 Appendix F - Feedback

The following questionnaire was completed by Dr.Matt Webster, a postdoctoral Research Associate in the department of Computer Science at the University of Liverpool, who will be using the software in further work to be conducted as part of the Trustworthy Robotic Assistants project.

Question 1 *From your initial experience of using the software, how do you rate the level of automation of the transformation process into both the Intermediate Form and into NuSMV input?*

9.5 out of 10. The software does the vast majority of the work, and only minimal user input is needed. For example, when the software finds a rule without a precondition it raises a warning and asks the user what to do, It would be nice if the software continued with the translation without user input, and just mentioned the warnings at the end of the translation process (i.e., like a compiler). I imagine this would be quite straightforward to implement within the software though.

Question 2 *Does the software given appropriate notifications when it encounters errors or ambiguity in the parsed input files?*

Yes. All errors and warnings are given during the translation process.

Question 3 *Does the software provide a flexible solution for parsing new sets of control* rules?

Yes. The software has been able to translate everything I've given it.

Question 4 *How might the software be used in future work relating to the Trustworthy Robot Assistants research project?*

In previous work on the project we did a manual translation of the Care-O-bot's behaviours into Brahms and NuSMV. However this was laborious (it took several weeks) and error-prone, and by the time we had done the translation the rule set had been expanded upon by the team at Hertfordshire. We would like to use the translator software to automate this process so that it can be done much faster and with fewer errors.

Another potential application is in the next healthcare scenario on the TRA project, in which the person living in the Robot House will be able to input new rules for the Care-O-bot to follow. In this case the rule set would need to be formally-verified again. Clearly the translator software would be very useful here as we could automatically translate the new rule set into NuSMV and check that all the requirements of the system still hold, and if not warn the user that there is a problem.

Question 5 *Are there any specific features of the software that will be of noteworthy use?*

I think the intermediate form is very useful, as it will allow translation into Promela, PRISM, UPPAAL or other languages so that different kinds of model checking can be done, e.g., probabilistic model checking or real-time model checking.

Question 6 *Is it likely that the softwares functionality will be extended, and if so what extra functionality might be added?*

If the software is used in the healthcare scenario it will need to be modified so that no user input is required, i.e., the software will have to make a "best guess" at what to do during the translation process if something seems awry. Also, the software may be modified to translate to other languages for model checking (see response to last question).