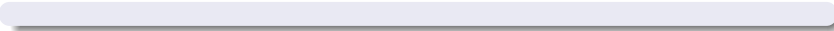


# Towards Verification of Domestic Robot Assistants-Part 1



Clare Dixon

Department of Computer Science  
University of Liverpool

`cldixon@liverpool.ac.uk`

`www.csc.liv.ac.uk/~clare`

[www.robosafe.org](http://www.robosafe.org)

# Collaborators

Farshid Amirabdollahian<sup>2,\*</sup>

Kerstin Dautenhahn<sup>2,\*</sup>

Louise Dennis<sup>1</sup>

Kerstin Eder<sup>3,\*</sup>

Michael Fisher<sup>1,\*</sup>

Paul Gainer<sup>1</sup>

Dejanira Araiza Illan<sup>3,\*</sup>

Kheng Lee Koay<sup>2,\*</sup>

Anthony Pipe<sup>3,\*</sup>

Maha Salem<sup>2,\*</sup>

Joe Saunders<sup>2,\*</sup>

Maarten Sierhuis<sup>5</sup>

Richard Stocker<sup>1,4</sup>

Matt Webster<sup>1,\*</sup>

David Western<sup>3,\*</sup>

<sup>1</sup> University of Liverpool (UoL)

<sup>2</sup> University of Hertfordshire (UoH)

<sup>3</sup> Bristol Robotics Lab (BRL)

<sup>4</sup> Nasa Ames Research Centre

<sup>5</sup> Nissan Research Centre

\* Trustworthy Robotic Assistants Project

# Talk Structure

- Part 1

- Introduction
- Tools and Techniques
- The Robot Scenario
- Modelling the Scenario
- Properties
- Discussion

- Part 2

- Introduction
- Tools and Techniques
- Brahms
- Formal Semantics of Brahms
- Brahms to Promela
- Properties
- Discussion
- Conclusions

# Robots in the Workplace and at Home

Currently many robots in use in industry or domestic use operate in limited physical space or have limited functionality. This helps assure their safety.

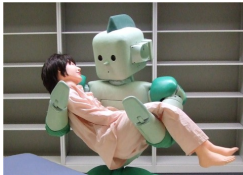
- Robots in industrial environments are limited so they can only move in a fixed area and have limited interactions with humans eg welding or paint spraying robots.
- Small or limited capability domestic robots, e.g., vacuum cleaning robots, robot lawn mowers, pool cleaning robots etc





# Robotic Assistants

- Robot assistants are being developed to help, or work closely with humans in industrial, domestic and health care environments.
- The robots will need to be able to act autonomously and make decisions to choose between a range of activities.
- How do we make sure they are trustworthy and safe?



RI-MAN

[rtc.nagoya.riken.jp/RI-MAN/](http://rtc.nagoya.riken.jp/RI-MAN/)



Pearl

[www.cs.cmu.edu/~nursebot/](http://www.cs.cmu.edu/~nursebot/)



Wakamaru

[www.mhi-global.com/products/detail/wakamaru.html](http://www.mhi-global.com/products/detail/wakamaru.html)

# What is Trustworthiness and Safety?

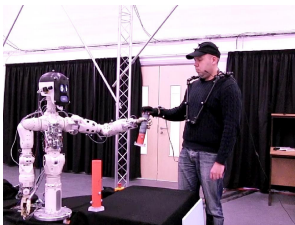
- Safety involves showing that the robot does nothing that (unnecessarily) endangers the person.
- The International Organization for Standardization (ISO) has been developing on ISO 13482, a standard relating to safety requirements for non-industrial, non-medical personal care robots.
- Trustworthiness involves social issues beyond pure safety.
- It is not just a question of whether the robots are safe but whether they are *perceived* to be safe, useful and reliable.
- There are also legal (and ethical) issues such as what happens when
  - the robot spills a hot drink on someone;
  - the robot doesn't remind the person to take their medicine;
  - the robot doesn't go to the kitchen when told?

# Verification and Validation

The EPSRC funded Trustworthy Robotic Assistants Project develops three different approaches to verification and validation of robotic assistants.

Each approach is aimed at increasing trust in robotic assistants.

- Formal Verification (Liverpool)
- Simulation-based Testing (Bristol Robotics Laboratory)
- End-user Validation (Hertfordshire)

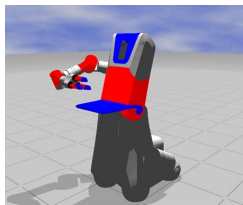


# Formal Verification

- A mathematical analysis of all behaviours using logics, and tools such as theorem provers or model checkers.
- Here we focus on temporal verification, i.e. we are interested in systems that change over time (dynamic rather than static).
- We focus on a fully automatic tools and techniques that do not require user interaction.
- Two main approaches to temporal verification are that of model checking and deductive techniques.

# Simulation Based Testing

- This is an exhaustive testing methodology widely used in the design of micro-electronic and avionics systems.
- These appeal to Monte-Carlo techniques and dynamic test refinement in order to cover a wide range or practical situations.
- Tools are used to automate the testing and analyse the coverage of the tests.

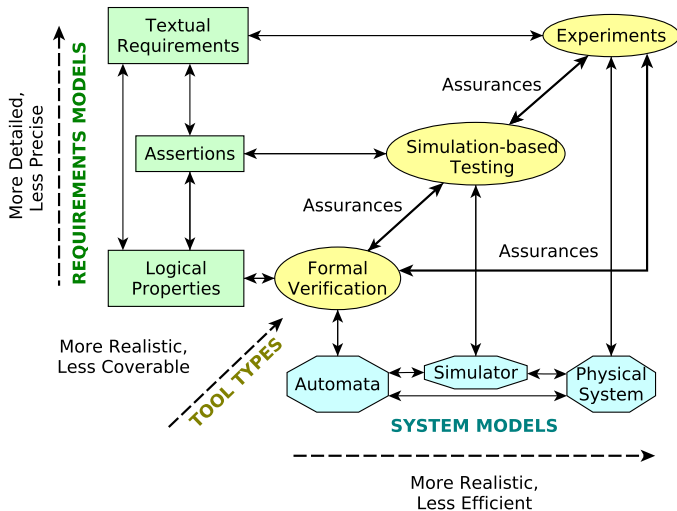


# End User Validation

- This approach involves experiments and user evaluations in practical robotic scenarios.
- Experiments take place in the UoH *Robot House* a domestic house equipped with equipment (sensors, video etc) to monitor the house and the interactions.
- Scenarios relating to robot human interaction are developed to test some hypothesis and experiments with users carried out.
- This helps establish whether the human participants indeed view the robotic assistants as safe and trustworthy.



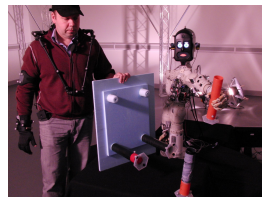
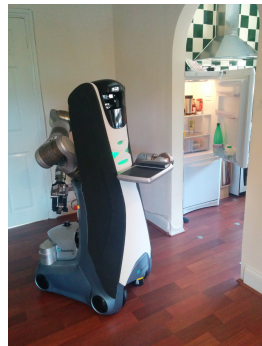
# Overall Approach



# Focus of the Talk

The focus of the rest of this talk is about formal verification, in particularly applying model checking, to a personal robot assistant (the Care-O-bot<sup>®</sup>), located in a domestic house in the University of Hertfordshire.

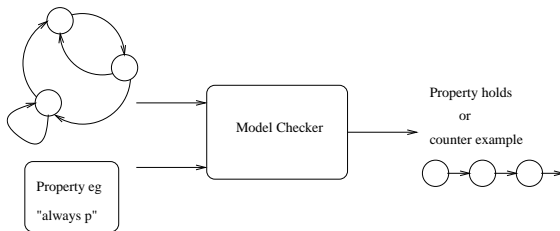
In another part of the project we consider the verification and validation of a co-operative manufacturing task with BERT a robot at Bristol Robotics Lab.





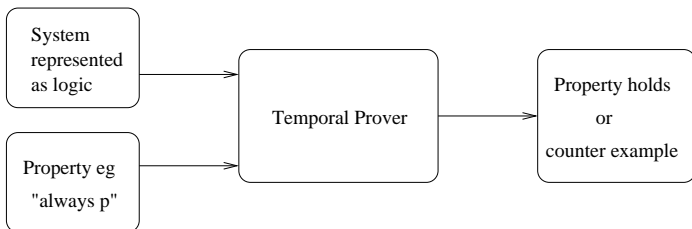
# Temporal Verification: Model Checking

- Model checking is a fully automatic, algorithmic technique for verifying the temporal properties of systems.
- Input to the model checker is a model of the system and a property to be checked on that model.
- Output is that the property is satisfied or a counter model.



# Temporal Verification: Deduction

- Deductive techniques involve the representation of both the system and the property as logical formulae and applying mathematical proof to these.
- The logics are some form of temporal logic which has operators that relate to time eg ' $\diamond$ ' (*sometime in the future*), or ' $\square$ ' (*always in the future*).



# Reactive Systems

- Reactive systems are systems that may need to interact with their environment frequently, i.e. they change over time and they may not terminate.
- Examples of reactive systems are air traffic control systems, programs that control cars or trains, or nuclear reactors etc.
- Reactive systems have *state* i.e. a snapshot of the values of the variables at that time.
- This state may change dependent upon the system or environment.
- The move from one state to another is known as a *transition*.

# Propositional Logic to Temporal Logic

Models for propositional logics have just one static world where formulae are evaluated. For example a proposition  $s$  meaning *it is sunny* evaluates to either true or false.

Models for temporal logics consider a set of worlds with a relation between them. So propositions, like  $s$ , may be true in some worlds and false in others.

Operators used in propositional logic are  $\wedge$ ,  $\vee$ ,  $\Rightarrow$ ,  $\neg$ . These are also used to construct temporal logic formulae.

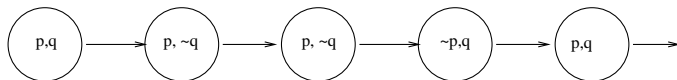
In temporal logics there are additional temporal operators dealing with time as well as the operators from propositional logic.

# Temporal Logic (PTL)

We consider propositional, linear, discrete time temporal logic with finite past and infinite future (PTL).

Propositional Temporal Logic can be viewed as propositional logic with some extra operators to deal with time.

For PTL, models are a linear sequence of states where propositions may be set to true or false in each state.



# Temporal Logic Operators

The usual range of future time temporal operators, are as follows eg. the following unary operators:

- $\bigcirc$  (X) 'in the next moment';
- $\Diamond$  (F) 'sometime in the future' or 'eventually in the future';
- $\Box$  (G) 'always in the future';

and the following binary operators:

- $\mathcal{W}$  'unless' or 'weak until'; and
- $\mathcal{U}$  'until'.

There are similar past time operators also.

# PTL–Syntax

PTL formulae are constructed from the following elements.

- A set,  $\mathcal{P}$ , of propositional symbols.
- Propositional connectives, **true**, **false**,  $\neg$ ,  $\vee$ ,  $\wedge$ , and  $\Rightarrow$ .
- Temporal connectives,  $\bigcirc$ ,  $\Diamond$ ,  $\Box$ ,  $\mathcal{U}$ , and  $\mathcal{W}$ .

The set of well-formed formulae of PTL (WFF), is defined as follows.

- Any element of  $\mathcal{P}$  is in WFF.
- **true** and **false** are in WFF.
- If  $A$  and  $B$  are in WFF then so are  $\neg A$     $A \vee B$     $A \wedge B$   
 $A \Rightarrow B$     $\Diamond A$     $\Box A$     $A \mathcal{U} B$     $A \mathcal{W} B$     $\bigcirc A$

# Semantics

PTL is interpreted over discrete, linear structures, for example the natural numbers,  $\mathbf{N}$ .

A model of PTL,  $\sigma$ , can be characterised as a sequence of *states*

$$\sigma = s_0, s_1, s_2, s_3, \dots$$

where each state,  $s_i$ , is a set of proposition symbols which are satisfied in the  $i^{th}$  moment in time.

As formulae in PTL are interpreted at a particular state in the sequence (i.e., at a particular moment in time), the notation

$$(\sigma, i) \models A$$

denotes the truth (or otherwise) of formula  $A$  in the model  $\sigma$  at state index  $i \in \mathbf{N}$ .



# Operators from Classical Logic

For propositions just check whether the proposition is in the relevant state.

$$(\sigma, i) \models p \text{ iff } p \in s_i \quad [\text{where } p \in \mathcal{P}]$$

For classical operators the semantics is the same as in classical logic but with reference to a particular world.

$$(\sigma, i) \models \mathbf{true}$$

$$(\sigma, i) \not\models \mathbf{false}$$

$$(\sigma, i) \models A \wedge B \text{ iff } (\sigma, i) \models A \text{ and } (\sigma, i) \models B$$

$$(\sigma, i) \models A \vee B \text{ iff } (\sigma, i) \models A \text{ or } (\sigma, i) \models B$$

$$(\sigma, i) \models A \Rightarrow B \text{ iff } (\sigma, i) \models \neg A \text{ or } (\sigma, i) \models B$$

$$(\sigma, i) \models \neg A \text{ iff } (\sigma, i) \not\models A$$

# Temporal Operators: Next

$$(\sigma, i) \models \bigcirc A \text{ iff } (\sigma, i + 1) \models A$$



This operator provides a constraint on the next moment in time.

## Examples:

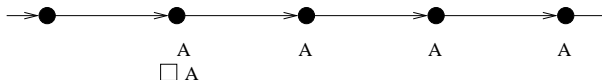
$$(is\_friday \vee is\_saturday) \Rightarrow \bigcirc \neg work$$

$$(at\_level\_crossing\_approach \wedge gate\_up) \Rightarrow \bigcirc on\_level\_crossing$$

$$ready \wedge \bigcirc steady \wedge \bigcirc \bigcirc go$$

# Temporal Operators: Always

$$(\sigma, i) \models \Box A \text{ iff for all } j \in \mathbf{N}, \text{ if } j \geq i \text{ then } (\sigma, j) \models A$$



The *always* operator provides a constraint now and on all future moments.

## Examples:

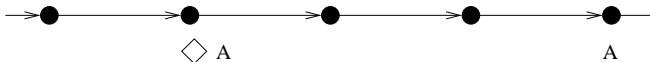
$$jane\_holds\_king\_spades \Rightarrow \Box jane\_holds\_king\_spades$$

$$\Box temp\_high$$

$$\Box \neg (process\_a\_write\_file \wedge process\_b\_write\_file)$$

# Temporal Operators: Sometimes

$(\sigma, i) \models \Diamond A$  iff there exists a  $k \in \mathbf{N}$  such that  $k \geq i$   
and  $(\sigma, k) \models A$



The *sometime* operator requires  $A$  to hold now or at some future moment *but we cannot specify when*.

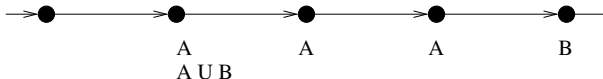
## Examples:

$request\_to\_print\_doc \Rightarrow \Diamond print\_doc$

$process\_a\_receives\_msg_1 \Rightarrow \Diamond process\_a\_sends\_msg_2$

# Temporal Operators: Until

$(\sigma, i) \models A \mathcal{U} B$  iff there exists a  $k \in \mathbf{N}$ , such that  $k \geq i$   
 and  $(\sigma, k) \models B$  and for all  $j \in \mathbf{N}$ , if  $i \leq j < k$   
 then  $(\sigma, j) \models A$



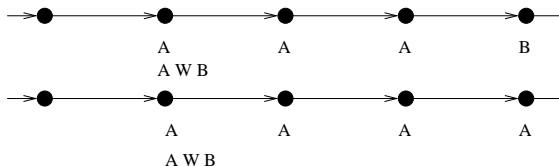
Similar to *sometime* as  $B$  must hold now or at some future moment *but we cannot specify when*. Also known as *strong until*.

## Examples:

*rainy*  $\mathcal{U}$  *monday*      *send\_bit*  $\mathcal{U}$  (*receive\_bit*  $\vee$  *abort*)

# Temporal Operators: Unless

$$(\sigma, i) \models A \mathcal{W} B \text{ iff } (\sigma, i) \models A \mathcal{U} B \text{ or } (\sigma, i) \models \Box A$$



The *unless* operator (or *weak until*) behaves similarly to  $\mathcal{U}$  except it allows the possibility that  $B$  never occurs.

## Examples:

*program\_executing*  $\mathcal{W}$  *program\_terminating*

# Safety, Liveness and Fairness

These can often be divided into one of the following.

**Safety:** something bad will not happen

- $\Box \neg (lower\_tray \wedge \neg tray\_empty)$
- $\Box \neg error\_state$

**Liveness:** something good will happen

- $\Diamond medicine\_reminder$
- $\Box \Diamond charging$

**Fairness:** independent processes will process/if something is requested infinitely often it will be allocated infinitely often.

- $\Box \Diamond turn_i$
- $\Box \Diamond doorbell\_ring \Rightarrow \Box \Diamond door\_answered$

# Temporal Logics

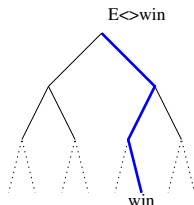
There are many different temporal logics,

- *linear* - each world has at most one successor world;
- *branching* - each world may have several successor worlds;
- *discrete* - worlds are isomorphic to the natural numbers (or integers);
- *dense* - worlds are isomorphic to the real numbers;
- *finite past* (future) there is a world with no predecessor (successor);
- *infinite past* (future) all worlds have a predecessor (successor);
- *propositional* temporal logics –interpretations for a world are like those in classical propositional logic.
- *first-order* temporal logics –interpretations for a world are like classical first-order logic.



# Common Temporal Logics Used in Model Checking I

- Branching-time temporal logics- models are trees.
- As well as PTL operators there are path operators **A** and **E**.
- Two main logics CTL and CTL\*.
- In CTL each path operator must be paired with a temporal operator eg  $\mathbf{E}\Diamond$  *win* - there is a possible future with a win.



- CTL\* allows formulae such as  $\mathbf{E}\Box\Diamond p$

# Common Temporal Logics Used in Model Checking

- Metric/timed temporal logics—allows constraints about when things occur eg the document will print within 5 time units

$$\diamond_{[0,5]} print$$

- Probabilistic temporal logics eg PCTL—an extension of the branching time temporal logic CTL with a probabilistic operator **P**

$$send \Rightarrow \mathbf{P}_{0.95} \diamond received$$

- Combinations with modal logics (temporal logics of knowledge) eg

$$\mathbf{K}_{jane} clare\_holds\_ace\_spades \Rightarrow$$

$$\Box \mathbf{K}_{jane} clare\_holds\_ace\_spades$$

# Expressivity versus Complexity

There is a trade off between the expressive power of these formalisms and the computational behaviour.

A decision procedure is an algorithm that given a decision problem terminates with a correct yes/no answer.

Here our decision procedures could be model checking or deductive procedures.

More expressive logics usually lead to decision procedures with higher complexity (or may be undecidable).

We often need to find a balance between what can be expressed and the difficulty of the decision procedure.

# A Simple Moving Robot

Consider the following description of a simple system where a robot is located either in the kitchen, or not, and a person can send a robot into the kitchen via touching some interface on the robot.

*The robot's location is either in the kitchen or not.*

*Initially the robot is not in the kitchen. If at some moment the person sends the robot to the kitchen send then in the next moment it will be in the kitchen.*

*If at some moment the person does not send the robot to the kitchen it could be either be in the kitchen or not in the next moment.*

We will specify this using PTL.

We identify *kitchen* and *send* as propositions in our specification.

# A Simple Moving Robot- Specification

- Initially the robot is not in the kitchen.

$$\neg kitchen$$

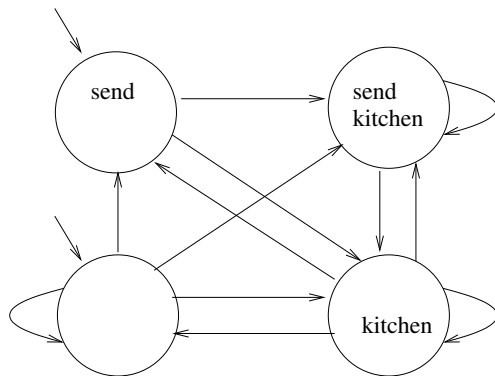
- If at some moment the person sends the robot to the kitchen *send* then in the next moment it will be in the kitchen.

$$\Box (send \Rightarrow \bigcirc kitchen)$$

- If at some moment the person does not send the robot to the kitchen it could be either be in the kitchen or not in the next moment.

$$\Box (\neg send \Rightarrow \bigcirc (kitchen \vee \neg kitchen))$$

# A Simple Moving Robot- State Transition System



$\neg kitchen$

$\square (send \Rightarrow \bigcirc kitchen)$

$\square (\neg send \Rightarrow \bigcirc (kitchen \vee \neg kitchen))$

# State Transition Systems

A state transition system  $\mathcal{T} = (S, R, L)$  consists of:-

- $S$  a set of states;
- $R \subseteq S \times S$  transition relation;
- $L : S \rightarrow \mathcal{P}(\text{props})$  a labelling function.

Usually we assume the transition relation is total, i.e. for each  $s \in S$  there is some  $s' \in S$  such that  $R(s, s')$ .

Here we consider *finite* state transition systems.

Sometimes we may also specify a set  $S' \subseteq S$  of initial states.

A *path* through a state transition system  $\mathcal{T} = (S, R, L)$  from a state  $s$  is a sequence  $s_0, s_1, \dots$  such that  $s_0 = s$  and for each  $i \geq 0$ ,  $(s_i, s_{i+1}) \in R$ .

# A Simple Moving Robot- Improved Specification

The following assumption implies that the robot can move around on its own which we may or may not want.

*If at some moment the person does not send the robot to the kitchen it could be either be in the kitchen or not in the next moment.*

$$\Box(\neg \text{send} \Rightarrow \bigcirc(\text{kitchen} \vee \neg \text{kitchen}))$$

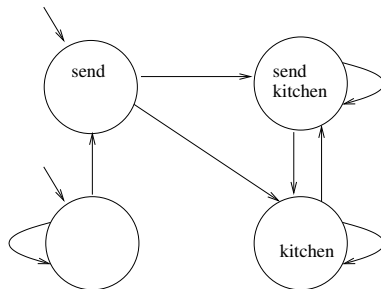
Also note that as the above formula is equivalent to **true** there is no real need to specify this at all.

Alternatively if we only want the robot to move when the person sends it somewhere and otherwise stays where it is we could replace the above by the following.

$$\begin{aligned} &\Box(\text{kitchen} \wedge \neg \text{send} \Rightarrow \bigcirc \text{kitchen}) \\ &\Box(\neg \text{kitchen} \wedge \neg \text{send} \Rightarrow \bigcirc \neg \text{kitchen}) \end{aligned}$$



# A Simple Moving Robot- Improved Specification



$\neg kitchen$

$\square (send \Rightarrow \bigcirc kitchen)$

$\square (kitchen \wedge \neg send \Rightarrow \bigcirc kitchen)$

$\square (\neg kitchen \wedge \neg send \Rightarrow \bigcirc \neg kitchen)$

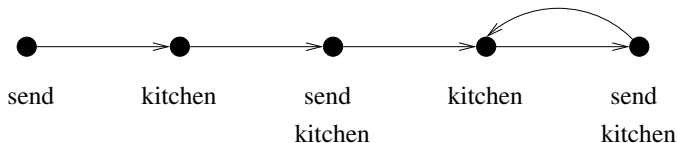
# Paths and Models

A path through the state transition system is a possible future for the system.

Previously we said a model of PTL,  $\sigma$ , was a sequence of *states*  $\sigma = s_0, s_1, s_2, s_3, \dots$  where each state,  $s_i$ , is a set of proposition symbols which are satisfied in the  $i^{th}$  moment in time.

Thus to construct a PTL model from a state transition system we take a path through the system, starting at an initial state, and the labelling of the sequence of states.

Here is a possible model for the improved robot specification.



# Model Checking

Thus for PTL a particular state transition system  $T$  every infinite path through the system, starting from an initial state, is a possible model of the system.

The *model checking problem* is given a state transition system  $T$  and a property  $\varphi$  expressed in PTL is to check whether *each path* (starting from an initial state) in  $T$  satisfies  $\varphi$ , i.e.  $T \models \varphi$ .

Alternatively, given a model  $T$ , a state  $s$  in  $T$ , and a PTL formula  $\varphi$  we can check whether  $\varphi$  is satisfied on each path starting from  $s$ , i.e.  $T, s \models \varphi$ .

# Model Checking Tools

Here we focus on a particular model checker NuSMV.

There are other model checkers such as:-

- SPIN: an on-the-fly model checker developed at Bell Labs;
- Java PathFinder: a model checker for Java programs developed at NASA;
- Mocha: a model checker for ATL;
- Prism: a probabilistic model checker;
- Uppaal: a model checker for real-time and timed systems; etc.

# NuSMV Background

- NuSMV stand for *New Symbolic Model Verifier*.
- It is an Open Source product, and is available from [nusmv.first.itc.it](http://nusmv.first.itc.it)
- NuSMV is a re-implementation of SMV which was written by K. McMillan at Carnegie Melon University.
- SMV and NuSMV have similar system description languages.
- NuSMV has a better user interface and more algorithms. It allows both PTL and CTL specifications (SMV only allowed CTL).

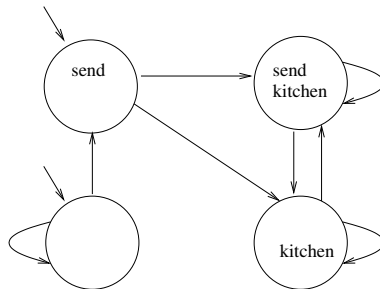
# Input File for the Simple Moving Robot

```
-- NuSMV input file for simple robot movement.
MODULE main
VAR
    send : boolean;
    kitchen : boolean;
ASSIGN
    init(kitchen) := FALSE;

    next(kitchen) := case
        send: TRUE;
        !send: kitchen;
    esac;

LTLSPEC
    G(send -> F kitchen);      --holds
LTLSPEC
    G(send -> X G kitchen);    --holds
LTLSPEC
    F kitchen;                  --doesn't hold
```

# A Simple Moving Robot- Checking Properties



$\square (send \Rightarrow \diamond kitchen)$

$\square (send \Rightarrow \bigcirc \square kitchen)$

$\diamond kitchen$

# Output from NuSMV

```
$ NuSMV robots.smv
*** This is NuSMV 2.5.4 (compiled on Fri Nov 23 21:36:06 UTC 2
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** Copyright (c) 2010, Fondazione Bruno Kessler

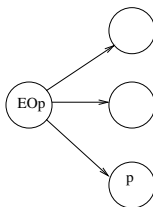
-- specification G (send -> F kitchen) is true
-- specification G (send -> X ( G kitchen)) is true
-- specification F kitchen is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 1.1 <-
    send = FALSE
    kitchen = FALSE
-> State: 1.2 <-
```



# How Do Model Checkers Work?

With CTL properties the algorithm labels the states transition system with subformulae of the property, where they are satisfied, starting from the smallest.

For example  $\mathbf{E}\bigcirc p$  will be added as a label to any state with a transition to a state labelled by  $p$



The output is a set states labelled (and so satisfying) the property itself.

# PTL Model Checking Algorithms: Idea

We take the property we are trying to check  $\varphi$ , negate it obtaining  $\neg\varphi$  and construct a structure representing  $\neg\varphi$ .

This structure is actually an automaton,  $\mathcal{A}_{\neg\varphi}$  and has the property that acceptable runs (paths) through the automaton satisfy  $\neg\varphi$ .

Combine this with the transition system (the model).

If there is a path (from an initial state) in the combined system the property is not satisfied, a counter model can be extracted from the path found.

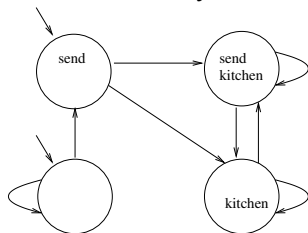
If there is no such path the property is satisfied.

# Example with our Simple Moving Robot

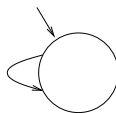
We consider the property  $\diamond kitchen$  which was false.

Negating it gives  $\neg \diamond kitchen \equiv \Box \neg kitchen$

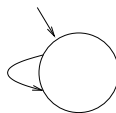
Transition System



Negation of the property



Combined Structure



There is a path from an initial state in the resulting structure so the property doesn't hold.

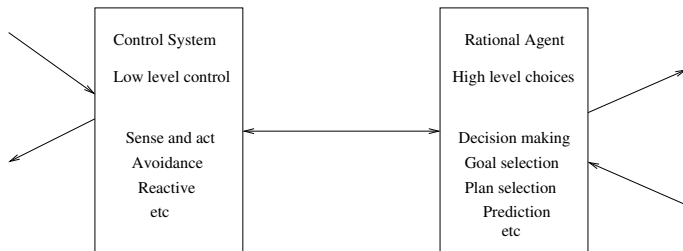
This matches the counter example output by the model checker.

# Issues Model Checking Robotic Systems

- What level of abstraction to use.
- We need a discrete and finite representation—for example how do we deal with real values relating to robot motion.
- Modelling explicit time constraints.
- The size of the state space.
- Expressivity versus computational behaviour - we may have to develop different models to deal with timed logics, probabilities, temporal aspects etc.
- What sort of concurrency do we need if there are multiple robots?
- How can we be sure the model is a suitable abstraction of the real situation?

# Robot Architectures and Our Approach

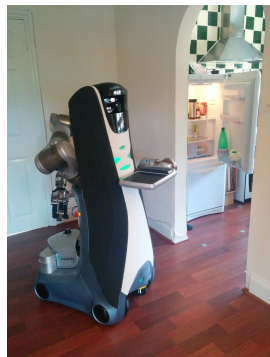
We assume an architecture where there is a separation between the high level decision making layer and the low level control layer.



We aim to represent and verify the decision making layer and we don't deal with low level control such as movement etc.

# Care-O-bot

- Here we apply model checking to the high level behaviours controlling the (commercially available) Care-O-bot<sup>®</sup>, manufactured by Fraunhofer IPA.
- It is based on the concept of a “robot butler” has been developed as a mobile robotic assistant to support people in domestic environments.
- It has a manipulator arm, an articulated torso, stereo sensors serving as “eyes”, LED lights, a graphical user interface, and a moveable tray.
- The robot’s sensors monitor its current location, the state of the arm, torso, eyes and tray.
- Its software is based on the Robot Operating System.



# Care-O-bot and Robot House

- This is deployed in a domestic-type house (the robot house) at the University of Hertfordshire.
- The robot house is equipped with sensors which provide information on the state of the house and its occupants, such as whether the fridge door is open and whether someone is seated on the sofa.
- Low-level robot actions such as movement, speech, light display, etc., are controlled by groups of high-level rules that together define particular behaviours.



# Care-O-bot Decision Making: Behaviours

- The Care-O-bot's high-level decision making is determined by a set of behaviours (each a sequence of rules).
- There are three types of rules relating to
  - **physical sensors:** from the robot or house eg the doorbell has been pressed, the TV is on, a sofa seat is occupied etc;
  - **semantic sensors:** to record what has happened in the physical world eg whether the person has been alerted that the fridge door is open, whether the person has indicated that they want the robot to move to the sofa etc;
  - **robot actions:** turn the lights to yellow, say something, move location, display something on the GUI etc.



# Care-O-bot Decision Making: Behaviours

- The rules are stored in a database.
- UoH have developed a number of rulesets. Here we focus on a set which together provide 31 default behaviours.
- Examples of high-level rules can take the form “lower tray”, “move to sofa area of the living room”, “say ‘The fridge door is open’ ”, set a flag, check a sensor etc.
- High-level rules are interpreted into low-level actions using the Robot Operating System (ROS).
- The rules are grouped together in a **precondition** → **action** ( IF a THEN b) structure to form behaviours.
- Some behaviours “raise tray” and “lower tray” are termed operators in that they only exist as support for other behaviours, i.e. can only be called by other behaviours.

# The S1-alertFridgeDoor Behaviour

Behaviours (a set of high level rules) take the form:

## Precondition-Rules -> Action-Rules

```
27  Fridge Freezer Is *ON* AND has been ON for more than 30 secs
31  ::514:: GOAL-fridgeUserAlerted is false
32  Turn light on ::0::Care-o-Bot 3.2 to yellow
34  move ::0::Care-o-Bot 3.2 to ::2:: Living Room and wait for
    completion
35  Turn light on ::0::Care-o-Bot 3.2 to white and wait for
    completion
36  ::0::Care-o-Bot 3.2 says 'The fridge door is open!' and
    wait for completion
37  SET ::506::GOAL-gotoCharger TO false
38  SET ::507::GOAL-gotoTable TO false
39  SET ::508::GOAL-gotoSofa TO false
40  ::0::Care-o-Bot 3.2 GUI, S1-Set-GoToKitchen, S1-Set-WaitHere
41  SET ::514::GOAL-fridgeUserAlerted TO true
```

# S1-Set-GoToKitchen and S1-goToKitchen

The S1-Set-GoToKitchen behaviour has no pre-conditions and just sets some flags.

S1-Set-GoToKitchen

```
3 SET ::505::GOAL-gotoKitchen TO true
4 SET ::512::GOAL-waitHere TO false
```

This then triggers the S1-goToKitchen behaviour.

S1-goToKitchen

```
31 ::505:: GOAL-gotoKitchen is true
32 Turn light on ::0::Care-O-Bot 3.2 to yellow
43 Execute sequence 'lowerTray' on ::0::Care-O-Bot 3.2
44 move ::0::Care-O-Bot 3.2 to ::7:: Kitchen Entrance in the
    Dining Room and wait for completion
45 Execute sequence 'raiseTray' on ::0::Care-O-Bot 3.2
46 Turn light on ::0::Care-O-Bot 3.2 to white
47 SET ::505::GOAL-gotoKitchen TO false
48 SET ::509::GOAL-waitAtKitchen TO true
```

# Care-O-bot Decision Making: Priorities

- In general only one behaviour executes at once.
- Each behaviour has a priority (integer between 0 and 90).
- When more than one behaviour was eligible for execution the higher priority behaviours are executed in preference to lower priority behaviours.
- If two behaviours of the same priority are eligible one will be selected randomly.

# Care-O-bot Decision Making: Interruptions

- Each behaviour is flagged as interruptible or not.
- If a behaviour is not interruptible once it has started executing it will execute to completion (even if a higher priority behaviour becomes eligible).
- If a behaviour is interruptible once it has started executing it may be interrupted by a behaviour with higher priority and may not be completed.
- This mechanism means critical behaviours can interrupt non-critical behaviours.
- Care needs to be taken when re-setting the flags that are part of the pre-conditions to the behaviours in the case of interruptible behaviours.

# Priority and Interruptibility

Name	Priority	Int
S1-Med-5PM-Reset	90	0
checkBell	80	0
unCheckBell	80	0
S1-remindFridgeDoor	80	0
answerDoorBell	70	0
S1-alertFridgeDoor	60	0
S1-Med-5PM	50	1
S1-Med-5PM-Remind	50	1
S1-gotoKitchen	40	1
S1-gotoSofa	40	1
S1-gotoTable	40	1
S1-kitchenAwaitCmd	40	1
S1-sofaAwaitCmd	40	1
S1-tableAwaitCmd	40	1
S1-WaitHere	40	1
S1-ReturnHome	40	1

Name	Priority	Int
S1-continueWatchTV	35	1
S1-watchTV	30	1
S1-sleep	10	1
lowerTray	0	0
raiseTray	0	0
S1-ResetAllGoals	0	0
S1-Set-Continue	0	0
S1-Set-GoToKitchen	0	0
S1-Set-GoToSofa	0	0
S1-Set-GoToTable	0	0
S1-Set-ReturnHome	0	0
S1-Set-WaitHere	0	0
S1-Set-Watch-TV	0	0
T-medicine	0	0
T-moveTo-person	0	0

# Modelling Behaviours

**Booleans from the Care-O-bot rules:** many of the Boolean values from the system can be used directly, for example the goal `GOAL-fridgeUserAlerted` or `GOAL-gotoSofa`.

**Robot actions:** involving its location, the robot torso position, speech, light colour, etc the orientation of the tray or providing alternatives on the Care-O-bot display for the person to select between are modelled as enumerated types e.g. `location` has values `livingroom`, `tv`, `sofa`, `table`, `kitchen`, `charging`.

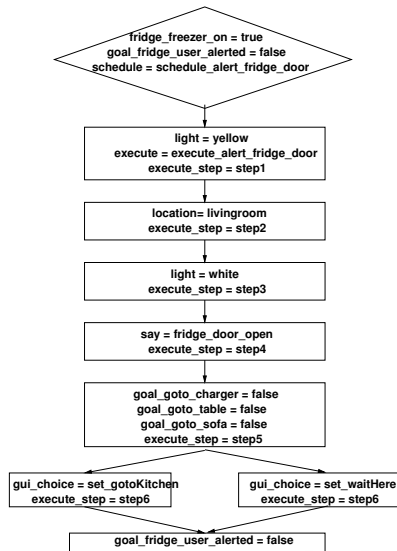
# Modelling Behaviours: Propositions

**Scheduling behaviours:** we use a variable `schedule` with an enumerated type for each behaviour, e.g. one of the values of `schedule` is `schedule_alert_fridge_door` if `schedule = schedule_alert_fridge_door` holds this denotes that the preconditions to the `S1-alertFridgeDoor` behaviour have been satisfied and this behaviour has been selected to run having the highest priority.

**Executing Behaviours:** we use a variable called `execute` with an enumerated type for each behaviour involving more than one step. eg `execute = execute_alert_fridge_door` denotes that the `S1-alertFridgeDoor` behaviour is executing. An enumerated type `execute_step` with values `step0`, `step1` etc keeps track of which part of the behaviour has been completed.



# Schema for S1-alertFridgeDoor



# Assumed Semantics of Behaviours

- The first box shows the preconditions that must hold (i.e. `fridge_freezer_on` and  $\neg$ `goal_fridge_userAlerted`) and that the behaviour must be scheduled (`schedule = schedule_alert_fridge_door`) before the other variables are set.
- This behaviour cannot be interrupted so once it is scheduled it will execute to completion.
- We assume that following each arrow in the previous diagram moves us into the next state, i.e. that moving to the living room, saying “The fridge door is open”, setting the three goal variables to false etc take one time step.

# Modelling Priorities I

- At each moment the next value of `schedule` is evaluated given what is currently executing `execute`, whether it is interruptible, its priority and the pre-conditions and priorities for other behaviours.
- To set the next value of `schedule` in the NuSMV input file, a list of cases are enumerated as follows

$$\begin{array}{ll} condition_1 : & schedule_1 \\ & \dots : \dots \\ condition_n : & schedule_n \end{array}$$

where  $condition_i$  represents the preconditions for the behaviour and  $schedule_i$  is the behaviour selected to execute.

# Modelling Priorities II

- The behaviours with higher priorities appear above behaviours with lower priorities and NuSMV selects the first case it encounters where the condition is satisfied.
- The scheduled behaviour is only executed, i.e. `execute` is assigned the scheduled behaviour if the previous behaviour being executed has completed or the current behaviour being executed is interruptible and the scheduled behaviour has a higher priority.
- Our modelling will *always* select a certain behaviour from those with the same priority that are eligible first. It will not be random (like the actual system).
- We could model the eligible behaviours using subsets of the behaviour with the same priority but this is exponential in the number of behaviours of the same priority so is infeasible for more than a small number of behaviours.

# Modelling The Environment

Some of the variables depend on sensors in the robot house

- `tv_on`,
- `sofa_occupied`,
- `fridge_freezer_on`, (denoting fridge door is open)
- `doorbell`

In the model we allow these to non-deterministically be set to true or false at any moment.

This means that the doorbell could be pressed at every moment, or the TV could be turned on and then off forever.

An improved version might also model the location of the person and only allow the `sofa_occupied` variable to be true if the person was in the living room near the sofa. Similarly with allowing `fridge_freezer_on` to change.

# Abstractions Relating to Timing Details

- We need to abstract away from some of the timing details included in the database to obtain a model that is discrete, for example, involving delays or timing constraints of 60 seconds or less.
- The behaviour `S1-watchTV` involves checking a goal has been false for 60 minutes. To achieve this we use an enumerated type `goal_watch_tv_time` with values for every 15 minutes `m0`, `m15`, `m30`, `m45`, `m60`.
- We could increase the number of values to represent 5 minute intervals (or even less), for example, but this would increase the size of the model.

# Requirements/Properties

Once the model has been encoded into the input language of the model checker it can be explored interactively to make sure it is as expected.

Next we need some requirements or properties of the system to check over the model.

Ideally these would come from a specification document about what is expected of the robot with respect to functionality, safety etc.

Some requirements might come from ISO standards documents.

Often it is hard to ascertain what properties should be checked.

# Sample Properties (1)

Here we focus on issues relating to the scheduling of behaviours, priorities and interruptions.

$$\Box((\text{fridge\_freezer\_on} \wedge \neg \text{goal\_fridge\_user\_alerted}) \Rightarrow \Diamond(\text{location} = \text{livingroom} \wedge \Diamond \text{say} = \text{fridge\_door\_open}))$$

We expect this to be false as, even though the preconditions to the behaviour `S1-alertFridgeDoor` are satisfied, preconditions to a behaviour with a higher priority, e.g.

`S1-answerDoorBell`, might hold and the other behaviour be executed instead of this.

Property	Output	Time (sec)
1	FALSE	11.1



## Sample Properties (2)

$$\square((\text{fridge\_freezer\_on} \wedge \neg \text{goal\_fridge\_user\_alerted} \wedge \\ \text{schedule} = \text{schedule\_alert\_fridge\_door}) \Rightarrow \\ \diamond(\text{location} = \text{livingroom} \wedge \diamond \text{say} = \text{fridge\_door\_open}))$$

We expect this to be true as the `S1-alertFridgeDoor` behaviour is not interruptible so once it is scheduled it should execute to conclusion.

Property	Output	Time (sec)
2	TRUE	12.3

# Other Properties and Model Checking Results

- 3  $\square (gui\_choice = gui\_set\_gotoKitchen \Rightarrow \Diamond location = kitchen)$
- 4  $\square ((gui\_choice = gui\_set\_gotoKitchen \wedge \Diamond schedule = schedule\_goto\_kitchen) \Rightarrow \Diamond (schedule = schedule\_goto\_kitchen \wedge \Diamond location = kitchen))$
- 5  $\square ((sofa\_occupied \wedge tv\_on \wedge \neg goal\_watch\_tv \wedge goal\_watch\_tv\_time = m60) \Rightarrow \Diamond (location = sofa \wedge say = shall\_we\_watch\_tv))$
- 6  $\square (execute\_raise\_tray \Rightarrow \Diamond (physical\_tray = raised \wedge \Diamond tray = raised))$
- 7  $\square ((execute = execute\_goto\_kitchen \wedge \neg move\_tray \wedge fridge\_freezer\_on \wedge \neg goal\_fridge\_user\_alerted) \Rightarrow \bigcirc (\neg (execute = execute\_goto\_kitchen)))$

Property	Output	Time (sec)
3	FALSE	7.7
4	FALSE	9.3
5	FALSE	11.6
6	TRUE	6.4
7	TRUE	6.9

The model had 130,593 reachable states.

# Automatic Translation from Behaviours I

- One issue is that a handwritten translation of the rules into input to a model checker is that it will need to be rewritten for a different rule-set, and may suffer from being error prone.
- A tool `CRuTON` was developed (by Gainer) that translates from the database rules/behaviours into input for a model checker.
- This is, in general, fully automatic but needs input to set parameters (eg in relation to timing) and in some cases may need user input to disambiguate between elements of the input.
- The rule database is parsed and output into intermediate data structures from which a translation into a number of model checkers could be developed.
- We have developed a translation into NuSMV.

# Automatic Translation from Behaviours II

- This is potentially useful as UoH has developed a number of rule-sets so we don't have to develop a hand translation for each set.
- But it assumes the same sort of structure as the two main rule-sets we have.
- Additionally UoH allow users to add their own personalised behaviours.
- These all have the priority zero.
- An example is "If it is 2pm remind me to watch my favourite TV programme."
- We would need a better treatment of timings but potentially these could be used to check for conflicts with existing behaviours online as the new behaviours are entered.
- For example "If it is 2pm remind me to take my medicine."

# Discussion I

- We have modelled the behaviours of a robotic assistant in the model-checker `NuSMV` and proved a number of properties relating to this.
- The priorities and interruptibility of the behaviours were modelled so that even if the satisfaction of the preconditions of behaviours became true these behaviours might not be fully executed because of higher priority behaviours being scheduled instead.
- The properties we checked were mainly related to the operation of the behaviours. It would be better to try properties relating to the requirements of the robot.
- We did find a small bug in the behaviours (a flag was wrongly set) but this was by inspection of the behaviours.

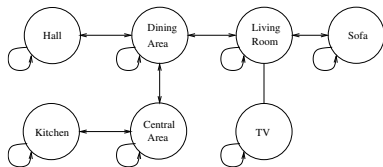
## Discussion II

- The model has been written by hand so is potentially prone to error.
- We tried to ensure the correctness of the models as far as we could by checking properties that were expected to hold (or not) and by using NuSMVs interactive model to explore models.
- Understanding the semantics of the robot execution cycle took a lot of close work and interaction with the developers at UoH.
- Recent discussions have indicated that behaviours with the same priority can interrupt each other which we had not realised.
- Behaviours calling behaviours need a more systematic treatment in the modelling.

# Discussion III

- The state explosion problem means we have to find a balance between the level of detail/abstraction and verification times.
- Many of the timing details were removed from the models but more detailed timings could be included at the expense of the size of models and verification times.
- The model of a person in the robot house was not represented but this could be incorporated showing their location for example.

# Modelling the Person's Location



TV, sofa and fridge door sensors relate to the person's location.





# Concluding Remarks

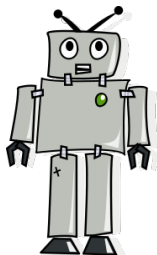
We discussed our experiences with applying formal verification to a robot assistant in the robot house at UoH.

We developed a by hand translation and an automatic translator from the sets of behaviours into input to a model checker.

However this approach is not easily reusable with different robot control mechanisms.

Next we consider another route to verification via a modelling/simulation language and briefly discuss temporal theorem proving.

These results should be used along with techniques such as simulation based testing and experiments with real people in the robot house to give more confidence in robotic assistants. ▶



# Papers

Webster, M., Dixon C., Fisher M., Salem M., Saunders J., Koay K.L., Dautenhahn, K and Saez-Pons, J.

Towards Reliable Autonomous Robotic Assistants Through Formal Verification: A Case Study

IEEE Transactions on Human-Machine Systems, to appear. 2015.

Dixon C., Webster, M., Saunders J., Fisher M. and Dautenhahn, K.

"The Fridge Door is Open"-Temporal Verification of a Robotic Assistant's Behaviours, Advances Autonomous Robotic Systems (TAROS), Springer, LNAI vol 8717, pages 97-108 2014. Paper won the Springer Best Paper Award.

Amirabdollahian, F., Dautenhahn, K., Dixon, C., Eder, K., Fisher, M., Koay, K.L., Magid, E., Pipe, A., Salem, S., Saunders J., and Webster, M.

Can You Trust Your Robotic Assistant?

In 5th International Conference in Social Robotics (ICSR 2013), LNAI vol 8239, pages 571-573. Springer, 2013.