Reasoning about High-Level Robot Behaviors by Model Checking and Local Validity Tests

Giuseppe De Giacomo & Riccardo Rosati

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
{degiacomo,rosati}@dis.uniroma1.it

Abstract

In this paper we explore a research direction in reasoning about actions stemming from the Robot-Tino Project at the University of Rome. We introduce a logical formalism that combines a very expressive logic of programs, the modal mu-calculus, with a special use of a minimal knowledge operator. Reasoning in such formalism can be done by integrating model checking for modal mu-calculus and propositional inference. This allows for exploiting existing model checking techniques and systems for reasoning about complex high-level robot behaviors, without renouncing to deal with incomplete information on the initial state and both the preconditions and the effects of actions.

Introduction

In this paper we explore a research direction stemming from the Robot-Tino Project at the University of Rome "La Sapienza" (De Giacomo *et al.* 1996; 1997). This project aims at integrating deliberative behaviors specified in a logical formalism with reactive behaviors provided by low-level mechanisms, which are not required to support reasoning. The robot adopted within the project is a Pioneer-1 equipped with reactive behaviors which are defined within the Saphira environment. The logical formalism adopted for representing and reasoning about actions at the deliberative level is a variant of Propositional Dynamic Logic (in the original proposal is a Description Logic corresponding to a PDL [1] (De Giacomo *et al.* 1996; 1997)).

The variant of PDL considered includes a minimal knowledge operator which strongly characterizes how the deliberative behavior of the robot is modeled: the robot may perform an action if it *knows* that the preconditions for that action hold, not simply if the preconditions are true. Similarly, the effects of an action of interest for the robot are only the ones the robot is aware of, i.e. the effects that change its epistemic state. This is obtained by specifying what a robot knows after an action, instead of specifying what is true after an action. In this approach, the robot follows the changes in the world through the changes in its epistemic state only. Obviously, it is responsibility of the designer to ensure that changes in the world are suitably reflected in changes in the robot epistemic state and vice-versa.

In (De Giacomo *et al.* 1996; 1997) it was shown that this approach allows for dealing with incomplete information about the initial state and about both preconditions and effects of actions, and moreover simplifies planning, which is the only form of reasoning about action considered in those papers. The simplification of the planning task is illustrated by the fact that the plan existence problem can be solved in PSPACE (as in the case of STRIPS (Bylander 1991)) instead of EXPTIME as required by a corresponding formulation in PDL (De Giacomo *et al.* 1996).[2] The simplification of the planning problem in this setting is due to the special use of the epistemic operator in the axioms that specify preconditions and effects of actions. Such axioms force a strong uniformity on the models. In general, PDL interpretation structures can be thought of as infinite trees labeled on both nodes and edges: each node represents a state and is labeled by the propositions that are true in that state, while each edge represents a state transition and is labeled by the action that causes the transition. The axioms mentioned above force their models so that, wrt the planning problem, all models can be seen as identical

[1] See (Schild 1991; De Giacomo & Lenzerini 1994) for an introduction to the correspondence between Description Logics and Propositional Dynamic Logics.

[2] There it was also shown that reasoning in the formalization based on the epistemic operator could be seen as a well-characterized weakening of reasoning in standard PDL. In addition, the proposed framework has the nice property wrt planning that the existence of a plan is inferred only if an "executable" plan exists, i.e. a sequence of actions which in every model reaches the goal, while a standard PDL framework would infer the existence of a plan also if such plan depends on the particular model.

except for the labeling of the states. In other words, the propositions that are true in the states are generally different in different models, however state transitions are the same in every model. This allows for building a single graph, representing all models, which reflects the common state transitions in each model, and whose nodes are the propositions that are true in all models –which correspond to the propositions that are known by the robot. The planning problem reduces to find a path on such a graph, executing validity tests on the nodes while traversing the graph.

In this paper we show that such an approach can be extended from planning to reasoning about actions in general. Specifically, we introduce and demonstrate a very general formalism to denote dynamic properties. Such a formalism is a variant of modal mu-calculus (Kozen 1983; Streett & Emerson 1989; Stirling 1996), a logic of programs that subsumes both propositional dynamic logics such as standard PDL, PDL enhanced with repeat constructs (Kozen & Tiuryn 1990), and branching time temporal logics such as CTL and CTL* (Emerson 1996). Modal mu-calculus is used in the verification of concurrent systems (Hoare 1985; Hennessy 1988; Milner 1989; Baeten & Weijland 1990), and for this task several automated model checking techniques and systems have been developed (Clarke, Emerson, & Sistla 1986; Emerson & Lei 1986; Winskel 1989; Cleaveland 1990; Stirling 1996; Burch *et al.* 1992; McMillan 1993; Cleaveland & Sims 1996). By exploiting the possibility of representing all models of a dynamic system specification as a single graph, it becomes possible to adapt model checking techniques for the modal mu-calculus to our setting. Essentially, such model checking techniques are used to visit the graph in a suitable fashion, checking validity (instead of truthness) of propositional formulae on single states, while traversing the graph.

The paper is structured as follows. In the next section, we introduce the logical formalism used for denoting dynamic properties. Then, we address representation of dynamic systems in such a formalism, and present techniques for reasoning about actions in this framework. Next, we present some examples of properties which can be expressed and computed in this setting, and we deal with the possibility of expressing and reasoning about complex actions. Finally, we briefly discuss the formalization of sensing actions.

Logical formalism

The technical background of our proposal is constituted by a logical formalism $\mathcal{L}$ that originates from a suitable integration of modal mu-calculus and epistemic decription logics (see (Kozen 1983; Stirling 1996; Emerson 1996) and (Donini *et al.* 1992; 1994; Donini, Nardi, & Rosati 1997; De Giacomo *et al.* 1996) respectively for an introduction to these formalisms). The basic elements of $\mathcal{L}$ are a finite set of actions Act, a countable set of propositions $Prop$, and a countable set of propositional variables Var.

Formulae of the formalism are divided in two layers:

- *state description formulae*:

$$p ::= A \mid p_1 \wedge p_2 \mid \neg p$$

  with $A \in Prop$.

- *dynamic formulae*:

$$\phi ::= \mathbf{k}p \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid [a]\phi \mid X \mid \mu X.\phi$$

  with p a state description formula, $a \in Act$ and $X \in Var$. The formula ϕ in $\mu X.\phi$ must be syntactically monotone in X, that is the variable X must be in the scope of an even number of negations.

We use the usual abbreviations $\vee, \supset$, tt, ff denoting tautology and contradiction, and also the abbreviations $\langle a\rangle\phi \doteq \neg[a]\neg\phi$ and $\nu X.\phi \doteq \neg\mu X.\neg\phi[X/\neg X]$ where $[X/\neg X]$ denotes the syntactic substitution of X by $\neg X$.

We give the semantics of $\mathcal{L}$ by first fixing once and for all a countable-infinite set $\mathcal{S}$ of *state names* which will constitute the interpretation domain of $\mathcal{L}$. We assume to have a set of constants $Const \subseteq \mathcal{S}$ that are used to univocally denote state names.

An $\mathcal{L}$ *pre-interpretation* $\mathcal{I}$ is a function over $\mathcal{S}$ which assigns to each constant in $Const$ the corresponding state name, i.e. $s^{\mathcal{I}} = s$; to each atomic proposition in $Prop$ a subset of $\mathcal{S}$, i.e. $A^{\mathcal{I}} \subseteq \mathcal{S}$; and to each action $a \in Act$ a functional relation over $\mathcal{S}$, i.e. $a^{\mathcal{I}} \subseteq \mathcal{S} \times \mathcal{S}$, with the restriction that for every $s, s', s'' \in \mathcal{S}$ if $(s, s') \in a^{\mathcal{I}}$ and $(s, s'') \in a^{\mathcal{I}}$ then $s' = s''$. In addition, the union of the relations interpreting the actions is backward functional, i.e. for every $s, s', s'' \in \mathcal{S}$ if $(s', s) \in \cup_{a\in Act} a^{\mathcal{I}}$ and $(s'', s) \in \cup_{a\in Act} a^{\mathcal{I}}$ then $s' = s''$. Pre-interpretations are extended to state description formulae as follows:

$$\begin{aligned} (p_1 \wedge p_2)^{\mathcal{I}} &= p_1^{\mathcal{I}} \cap p_2^{\mathcal{I}} \\ (\neg p)^{\mathcal{I}} &= \mathcal{S} - p^{\mathcal{I}} \end{aligned}$$

An $\mathcal{L}$ *valuation* ρ is a function from Var to a subset of $\mathcal{S}$ such that $\rho(X) \subseteq \mathcal{S}$ for every $X \in Var$. Given a valuation ρ and $\mathcal{E} \subseteq \mathcal{S}$,, we denote by $\rho[X/\mathcal{E}]$ the valuation obtained from ρ by changing to $\mathcal{E}$ the subset assigned to the variable X.

An $\mathcal{L}$ *interpretation* $\mathcal{W}$ is a set of pre-interpretations over $\mathcal{S}$. We define interpretations of state formulae and

actions respectively as:

$$\begin{aligned} p^{\mathcal{W}} &= \cap_{\mathcal{I}\in\mathcal{W}}\, p^{\mathcal{I}} \\ a^{\mathcal{W}} &= \cap_{\mathcal{I}\in\mathcal{W}}\, a^{\mathcal{I}} \end{aligned}$$

Interpretations and valuations are used to interpret dynamic formulae as follows:

$$\begin{aligned} (\mathbf{k}p)^{\mathcal{W}}_{\rho} &= p^{\mathcal{W}} \\ (\phi_1 \wedge \phi_2)^{\mathcal{W}}_{\rho} &= (\phi_1)^{\mathcal{W}}_{\rho} \cap (\phi_2)^{\mathcal{W}}_{\rho} \\ (\neg\phi)^{\mathcal{W}}_{\rho} &= \mathcal{S} - \phi^{\mathcal{W}}_{\rho} \\ ([a]\phi)^{\mathcal{W}}_{\rho} &= \{s \in \mathcal{S} \mid \forall s'.(s,s') \in a^{\mathcal{W}} \supset s' \in \phi^{\mathcal{W}}_{\rho}\} \\ X^{\mathcal{W}}_{\rho} &= \rho(X) \\ (\mu X.\phi)^{\mathcal{W}}_{\rho} &= \cap\{\mathcal{E} \subseteq \mathcal{S} \mid \phi^{\mathcal{W}}_{\rho[X/\mathcal{E}]} \subseteq \mathcal{E}\} \end{aligned}$$

In particular we will be interested in closed formulae (formulae with no free variables). Such formulae are interpreted independently from the valuation, hence we will interpret them using an interpretation $\mathcal{W}$ alone: $\phi^{\mathcal{W}}$.

An $\mathcal{L}$ knowledge base Σ is defined as a pair $\Sigma = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a finite set of state description formulae and (closed) dynamic formulae, and $\mathcal{A}$ is a finite set of assertions of the form $\psi(s)$ with ψ either a state description formula or a dynamic formula, and $s \in \mathit{Const}$.

An $\mathcal{L}$ interpretation $\mathcal{W}$ *satisfies* a formula $\psi \in \mathcal{T}$ iff $\psi^{\mathcal{W}} = \mathcal{S}$. $\mathcal{W}$ satisfies an assertion $p(s) \in \mathcal{A}$ iff $s \in p^{\mathcal{W}}$. $\mathcal{W}$ satisfies a knowledge base $\Sigma = (\mathcal{T}, \mathcal{A})$ iff $\mathcal{W}$ satisfies every formula from $\mathcal{T}$ and every assertion from $\mathcal{A}$.

An $\mathcal{L}$ interpretation $\mathcal{W}$ is a *model* for Σ iff $\mathcal{W}$ is a maximal set of $\mathcal{L}$ interpretations satisfying Σ, i.e., for each $\mathcal{L}$ interpretation $\mathcal{W}'$, if $\mathcal{W} \subset \mathcal{W}'$ then $\mathcal{W}'$ does not satisfy Σ. This corresponds to impose a "minimal knowledge" semantics on the epistemic states of the agent (De Giacomo *et al.* 1996). In fact, each $\mathcal{L}$ interpretation can be viewed as a Kripke structure in which each $\mathcal{L}$ pre-interpretation is a possible world, and each world is connected to all worlds in the structure: only structures satisfying Σ and having a maximal set of possible worlds are considered, which maximizes ignorance of the agent in its epistemic states.

Finally, Σ *logically implies* a formula or an assertion σ, written $\Sigma \models \sigma$, iff every model for Σ satisfies σ.

Dynamic system representation

In this section we present the framework for representing dynamic systems in the logic $\mathcal{L}$. Our framework essentially follows the one presented in (De Giacomo *et al.* 1996; 1997).

The formalization of a dynamic system is constituted by the following elements.

- *Initial state description* is formed by a finite set of assertions of the form

  $$p(s_{init})$$

  where p is a state description formula and s_{init} is a constant in *Const*. In fact we may assume that $\mathit{Const} = \{s_{init}\}$.

- *Static axioms* are a finite set of state description formulae p, which are assumed valid, defining invariance properties of states.

- *Precondition axioms* specify under which conditions an action can be executed. In our case such a condition depends on the epistemic state of the agent and not on what is true in the world. Precondition axioms are dynamic formulae of the the form:

  $$\mathbf{k}p \supset \langle a \rangle \mathbf{k}tt$$

- *Effect axioms* specify the effects of an action when executed under certain conditions. Again in our approach both effects and conditions concern the epistemic state of the agent. Effect axioms are dynamic formulae of the form:

  $$\mathbf{k}p_1 \supset [a]\mathbf{k}p_2$$

  No special treatment of the frame problem is considered here; we simply make use of frame axioms constituted by effect axioms of the form:

  $$\mathbf{k}p \supset [a]\mathbf{k}p$$

Let Σ be the knowledge base describing the dynamic system as above. We are interested in verifying if the system satisfies a certain dynamic property. Formally, we are interested in logical inference of the form

$$\Sigma \models \phi(s_{init}) \tag{1}$$

where ϕ can be any dynamic formula. As we shell see later, in this way we can deal with the *projection problem*, "given a sequence of actions, does a given state description formula hold in the resulting state?"; the *planning problem*, "is there a sequence of actions such that the goal (a state description formula) holds in the resulting state?"; but also very sophisticated dynamic properties such as *liveness*, *safeness*, etc. that are easily expressed using fixpoint formulae.

Reasoning

Let us now turn our attention to the problem of computing the logical implication (1).

First of all, an $\mathcal{L}$ knowledge base Σ corresponding to a dynamic system representation has in general many models, however it can be shown that all such models are isomorphic up to renaming of states. It is thus possible to reason on a single model, since it can be

ALGORITHM TG
INPUT: $\Sigma = (\Gamma_S \cup \Gamma_P \cup \Gamma_E, \{p(s_{init})\})$
OUTPUT: $TG(\Sigma)$
begin
  $S_{active} = \{s_{init}\}$;
  $S = \{s_{init}\}$;
  $L_S(s_{init}) = \{p(s_{init})\}$;
  repeat
    s = choose(S_{active});
    for each action a **do**
      if $(\mathbf{k}p \supset \langle a \rangle \mathbf{k}tt) \in \Gamma_P$ **and** $(\Gamma_S \cup L_S(s) \models p)$ **then**
      begin
        s' = NEW state name;
        $PROP(s') = \{q | (\mathbf{k}p' \supset [a]\mathbf{k}q \in \Gamma_E) \wedge (\Gamma_S \cup L_S(s) \models p')\}$;
        if there exists $s'' \in S$ **such that**
          $(\Gamma_S \cup L_S(s'))$ and $(\Gamma_S \cup L_S(s''))$ are logically equivalent
        then $L_A(s, a) = s''$
        else begin
            $S_{active} = S_{active} \cup \{s'\}$;
            $S = S \cup \{s'\}$;
            $L_S(s') = PROP(s')$
          end
      end;
    $S_{active} = S_{active} - \{s\}$
  until $S_{active} = \emptyset$;
  return (S, L_S, L_A)
end;

Figure 1: Algorithm computing $TG(\Sigma)$

shown that all the properties that are expressible in the right-hand side of (1) are independent of such state names.

We represent the models of Σ by means of the *transition graph* (TG) of Σ. Roughly speaking, the transition graph is a graph in which:

- each node corresponds to a state and is labeled with a propositional formula representing the properties which are known in such a state;
- each edge is labeled with an action name, and denotes the transition caused by the execution of the corresponding action.

Observe that what the robot knows in the initial state is the set of propositional formulae which are valid in s_{init}, i.e. the set of propositional formulae which are logically implied by $p(s_{init})$. Moreover, what the robot knows after executing an action is the set of propositional formulae which are logically implied by the postconditions representing the effects of the action execution. In this way it can be shown that it is possible in each state to verify whether an action can be executed (that is, whether the preconditions are known by the robot) by simply checking for the validity of the action precondition. This correspondence between the notions of robot's *knowledge* (about propositional properties) and propositional *validity* is exploited in the construction of the transition graph.

Formally, $TG(\Sigma) = (S, L_S, L_A)$, where $S \subseteq \mathcal{S}$ is the set of states which includes s_{init}, L_S is a function assigning a finite set of propositional formulae to each state in S, and L_A is a partial function assigning a state to a pair formed by a state and an action.

Let $\Sigma = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is the set of static axioms (Γ_S), precondition axioms (Γ_P), and effect axioms (Γ_E), and $\mathcal{A} = \{p(s_{init})\}$ is the initial state description, be the the dynamic system specification. The transition graph $TG(\Sigma)$ is computed by the algorithm shown in Fig. 1.

Informally, the algorithm, starting from the initial state s_{init}, iteratively proceeds as follows. First, it finds an action a which can be executed in the current state, by identifying in the set Γ_P a precondition axiom for a whose left-hand side is logically implied by the current knowledge base. Then, it propagates the effects of the action a, which again is based on checking whether the left-hand side of each effect axiom for a in the set Γ_E is logically implied by the properties holding in the current state. In this way, the set of properties corresponding to the effect of the execution of a in the current state is computed. A new state is then generated, unless a state with the same properties has already been created. This step is repeated until all actions executable in the current state have been considered. Then, a new current state is chosen among those previously created and the main iteration proceeds.

The transition graph is unique, that is, every order of extraction of the states from the set S_{active} produces the same set of assertions, up to the renaming of states. Moreover, the algorithm terminates, that is, the condition $S_{active} = \emptyset$ is eventually reached, since the number of states generated is bounded to the number of different subsets of the set $\mathcal{E} = \{q | \mathbf{k}p' \supset [a]\mathbf{k}q \in \Gamma_E\}$, i.e. 2^n, where n is the number of axioms in Γ_E. Finally, the condition

$$(\Gamma_S \cup L_S(s')) \text{ and } (\Gamma_S \cup L_S(s'')) \text{ are logically equivalent}$$

can be verified by a propositional validity check, as well as the propositional logical implication

$$\Gamma_s \cup L_s(s) \models p$$

Next let us define the *extension of a dynamic formula in* $TG(\Sigma)$ wrt an $\mathcal{L}$ valuation ρ as follows:

$$\begin{array}{lcl}
(\mathbf{k}p)_\rho^{TG(\Sigma)} & = & \{s \in S \mid \Gamma_s \cup L_s(s) \models p\} \\
(\phi_1 \wedge \phi_2)_\rho^{TG(\Sigma)} & = & (\phi_1)_\rho^{TG(\Sigma)} \cap (\phi_2)_\rho^{TG(\Sigma)} \\
(\neg\phi)_\rho^{TG(\Sigma)} & = & S - (\phi)_\rho^{TG(\Sigma)} \\
([a]\phi)_\rho^{TG(\Sigma)} & = & \{s \in S \mid \forall s'.(L_A(s,a) = s') \supset \\
 & & \qquad (s' \in \phi_\rho^{TG(\Sigma)})\} \\
X_\rho^{TG(\Sigma)} & = & \rho(X) \\
(\mu X.\phi)_\rho^{TG(\Sigma)} & = & \cap\{E \subseteq S \mid \phi_{\rho[X/E]}^{TG(\Sigma)} \subseteq E\}
\end{array}$$

In fact, we are interested in closed formulae ϕ, whose extension in $TG(\sigma)$ is independent of the valuation: each such formula will be denoted simply by $\phi^{TG(\Sigma)}$.

Now we can state the following result:

Theorem 1 *Let Σ be a specification of a dynamic system as above, and let ϕ be any closed dynamic formula in $\mathcal{L}$. Then, $\Sigma \models \phi(s_{init})$ if and only if $s_{init} \in \phi^{TG(\Sigma)}$.*

Being $TG(\Sigma)$ essentially a finite "transition system" whose nodes represent sets of valid propositional formulae, it is immediate to modify model checking algorithms for modal mu-calculus formulae for finite transition systems (Clarke, Emerson, & Sistla 1986; Emerson & Lei 1986; Winskel 1989; Cleaveland 1990; Stirling 1996; Burch *et al.* 1992; McMillan 1993; Cleaveland & Sims 1996), to verify whether s_{init} is in the extension of a formula in $TG(\Sigma)$, and hence, by the theorem above, to reason about actions in our setting.

Examples

We illustrate the expressiveness capabilities of the formalism proposed with some examples. Below, we informally say that a formula "holds" in a state if the

formula is "known" in the robot's corresponding epistemic state.

We start by expressing the *projection problem*: "does a proposition p hold in the state resulting from the execution of a given sequence of actions, say a_1, a_2, a_3?" This can be checked by verifying the following logical implication:

$$\Sigma \models [\langle a_1\rangle\langle a_2\rangle\langle a_3\rangle \mathbf{k}p](s_{init})$$

where $\langle a_1\rangle\langle a_2\rangle\langle a_3\rangle \mathbf{k}p$ expresses that the sequence of actions a_1, a_2, a_3 can be executed and that it leads to a state where p holds.

Let us now consider the *planning problem*: "is there a sequence of actions that leads to archiving a given goal p_{goal}?". This can be expressed by

$$\Sigma \models [\mu X.\, \mathbf{k}p_{goal} \vee \bigvee_{a \in Act} \langle a\rangle X](s_{init})$$

The dynamic formula on the right-hand side denotes the following inductive property: either p_{goal} holds in the current state, or there is an action a that leads to a state from which there exists a sequence of actions that leads to a state where p_{goal} holds. Notably our formalization guarantees that the existence of a sequence of actions can be inferred if and only if the *same* sequence of actions achieves the goal in every model. That is, unrealizable plans are discarded a priori (see also (De Giacomo *et al.* 1996)).

The planning problem can be more sophisticated than what shown above. For example we may want to do *planning with archiving and maintenance goals*: "is there a sequence of actions which achieves a certain goal p_{agoal} while another goal p_{mgoal} is always satisfied?". This can be expressed by modifying the formula expressing planning as follows:

$$\mu X.\, \mathbf{k}p_{mgoal} \wedge (\mathbf{k}p_{agoal} \vee \bigvee_{a \in Act} \langle a\rangle X)$$

expressing that, inductively, either both p_{mgoal} and p_{agoal} hold in the current state, or p_{mgoal} holds and there is an action a leading to a state where there exists a sequence achieving p_{agoal} while maintaining p_{mgoal}.

Next we consider *safeness properties*. These in general are properties that express that "something bad can never happen". For example, "it is not possible to reach a state from which there exists no plan to get the batteries charged"; in other words, in any reachable state the robot can formulate a plan to charge its battery. In $\mathcal{L}$, the existence of a plan to charge the batteries can be expressed, as shown above, by:

$$\phi_{pcb} \doteq \mu X.\mathbf{k}BttrChrgd \vee \bigvee_{a \in Act} \langle a\rangle X$$

the fact that this can always be done (a safeness property) is expressed as $\nu X.\, \phi_{pcb} \wedge \bigwedge_{a \in Act}[a]X$.

Invariance properties can be expressed in an analogous way, since they can be seen as safeness properties: the bad thing is the violation of the invariant.

Liveness properties, that in general express that "something good is eventually achieved", can also be captured. For example, "a given job eventually comes to an end" can be expressed as

$$\mu X.\, \mathbf{k}JobEnded \vee (\bigvee_{a \in Act} \langle a\rangle \mathbf{k}tt) \wedge (\bigwedge_{a \in Act} [a]X)$$

Liveness and safeness conditions can be used together to express complex properties as "whenever a job is started, the job is also terminated":

$$\nu X.\, [startjob]\psi \wedge \bigwedge_{a \in Act} [a]X$$

where

$$\psi = \mu Y.\, (\bigvee_{a \in Act} \langle a\rangle \mathbf{k}tt) \wedge (\bigwedge_{a \in Act \wedge a \neq endjob} [a]X)$$

Observe the use of ψ to express the well-foundedness of all sequences of actions not including *endjob*.

Programs

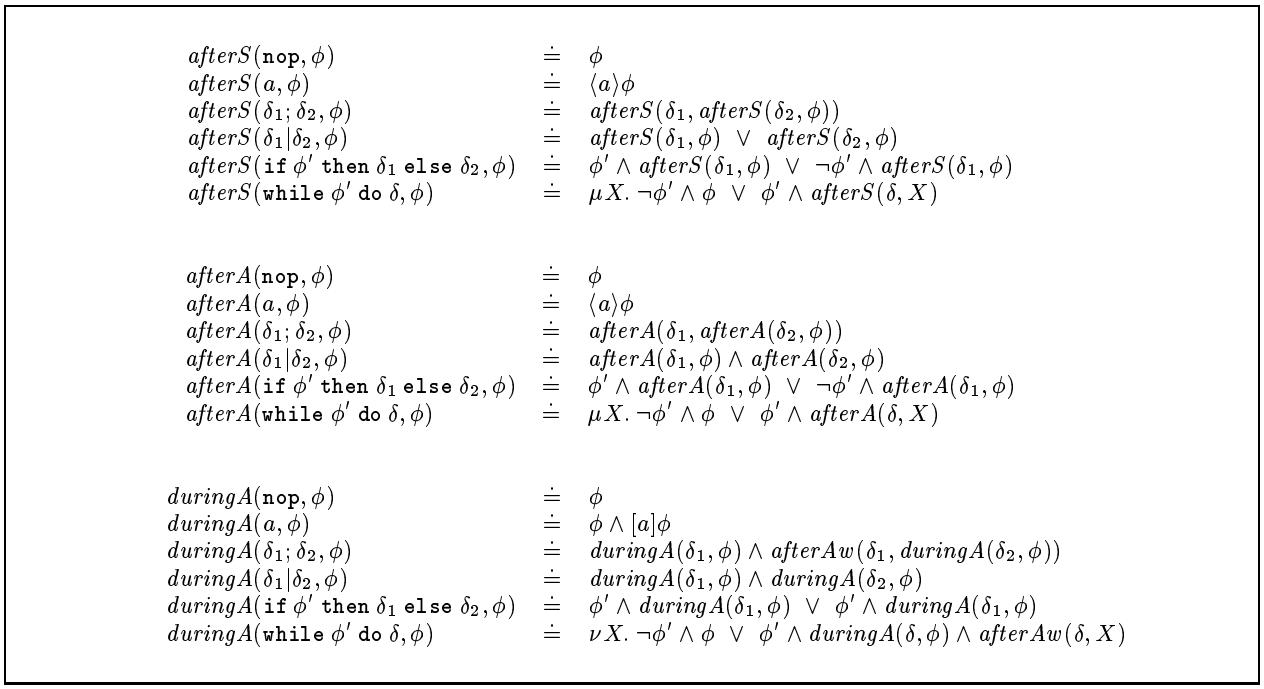
In this section we introduce a notion of robot program in order to enforce a control flow on actions. Robot programs are not part of the basic action theory specifying the general behavior of the robot; instead, they are used on top of the action theory to introduce a notion of control on the robot actions. This way to proceed mirrors that of the Toronto Cognitive Robotics Group in developing GOLOG (Levesque *et al.* 1997).

We consider a simple programming language that allows for building nondeterministic while-programs:

$$\delta \quad ::= \quad \texttt{nop} \mid a \mid \delta_1; \delta_2 \mid \delta_1 | \delta_2 \mid \texttt{if}\ \phi\ \texttt{then}\ \delta_1\ \texttt{else}\ \delta_2 \mid \texttt{while}\ \phi\ \texttt{do}\ \delta$$

where `nop` is a special instruction that does nothing, a is the command requiring the execution th action a, ";" is the sequential composition, "|" is nondeterministic choice, and `if` $\cdot$ `then` $\cdot$ `else` $\cdot$ and `while` $\cdot$ `do` $\cdot$ are the classical if-then-else and while constructs. The semantics of the various constructs is the usual one (see e.g. (H. R. Nielson 1992)), except for atomic actions, whose semantics is given by the basic action theory.

As in the case of GOLOG (Levesque *et al.* 1997), formally programs are not part of the formalism $\mathcal{L}$. They are used to define suitable macros that are translated into $\mathcal{L}$ dynamic formulae.



$$
\begin{array}{lcl}
\mathit{afterS}(\mathtt{nop},\phi) & \doteq & \phi \\
\mathit{afterS}(a,\phi) & \doteq & \langle a\rangle\phi \\
\mathit{afterS}(\delta_1;\delta_2,\phi) & \doteq & \mathit{afterS}(\delta_1,\mathit{afterS}(\delta_2,\phi)) \\
\mathit{afterS}(\delta_1|\delta_2,\phi) & \doteq & \mathit{afterS}(\delta_1,\phi) \ \vee \ \mathit{afterS}(\delta_2,\phi) \\
\mathit{afterS}(\mathtt{if}\ \phi'\ \mathtt{then}\ \delta_1\ \mathtt{else}\ \delta_2,\phi) & \doteq & \phi' \wedge \mathit{afterS}(\delta_1,\phi) \ \vee \ \neg\phi' \wedge \mathit{afterS}(\delta_1,\phi) \\
\mathit{afterS}(\mathtt{while}\ \phi'\ \mathtt{do}\ \delta,\phi) & \doteq & \mu X.\, \neg\phi' \wedge \phi \ \vee \ \phi' \wedge \mathit{afterS}(\delta, X)
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{afterA}(\mathtt{nop},\phi) & \doteq & \phi \\
\mathit{afterA}(a,\phi) & \doteq & \langle a\rangle\phi \\
\mathit{afterA}(\delta_1;\delta_2,\phi) & \doteq & \mathit{afterA}(\delta_1,\mathit{afterA}(\delta_2,\phi)) \\
\mathit{afterA}(\delta_1|\delta_2,\phi) & \doteq & \mathit{afterA}(\delta_1,\phi) \wedge \mathit{afterA}(\delta_2,\phi) \\
\mathit{afterA}(\mathtt{if}\ \phi'\ \mathtt{then}\ \delta_1\ \mathtt{else}\ \delta_2,\phi) & \doteq & \phi' \wedge \mathit{afterA}(\delta_1,\phi) \ \vee \ \neg\phi' \wedge \mathit{afterA}(\delta_1,\phi) \\
\mathit{afterA}(\mathtt{while}\ \phi'\ \mathtt{do}\ \delta,\phi) & \doteq & \mu X.\, \neg\phi' \wedge \phi \ \vee \ \phi' \wedge \mathit{afterA}(\delta, X)
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{duringA}(\mathtt{nop},\phi) & \doteq & \phi \\
\mathit{duringA}(a,\phi) & \doteq & \phi \wedge [a]\phi \\
\mathit{duringA}(\delta_1;\delta_2,\phi) & \doteq & \mathit{duringA}(\delta_1,\phi) \wedge \mathit{afterAw}(\delta_1,\mathit{duringA}(\delta_2,\phi)) \\
\mathit{duringA}(\delta_1|\delta_2,\phi) & \doteq & \mathit{duringA}(\delta_1,\phi) \wedge \mathit{duringA}(\delta_2,\phi) \\
\mathit{duringA}(\mathtt{if}\ \phi'\ \mathtt{then}\ \delta_1\ \mathtt{else}\ \delta_2,\phi) & \doteq & \phi' \wedge \mathit{duringA}(\delta_1,\phi) \ \vee \ \phi' \wedge \mathit{duringA}(\delta_1,\phi) \\
\mathit{duringA}(\mathtt{while}\ \phi'\ \mathtt{do}\ \delta,\phi) & \doteq & \nu X.\, \neg\phi' \wedge \phi \ \vee \ \phi' \wedge \mathit{duringA}(\delta,\phi) \wedge \mathit{afterAw}(\delta, X)
\end{array}
$$

Figure 2: Definitions of $\mathit{afterS}(\delta,\phi)$, $\mathit{afterA}(\delta,\phi)$, and $\mathit{duringA}(\delta,\phi)$

We illustrate this approach with some examples. First, we express the property "there exists a terminating execution of program δ that terminates in a state where ϕ holds". This is similar to the expression $\exists s'.DO(\delta,s,s') \wedge \Phi(s)$ used in GOLOG computations (Levesque *et al.* 1997). We define the corresponding $\mathcal{L}$ dynamic formula $\mathit{afterS}(\delta,\phi)$ by induction on the structure of the program as in Fig.2 (we define $\langle\mathtt{nop}\rangle\phi = [\mathtt{nop}]\phi = \phi$). The formula $\mathit{afterS}(\delta,\phi)$ is particularly meaningful if we assume that at the various choice points of the program the robot can do the choice, choosing the execution that eventually leads to termination in a state where ϕ holds (exactly as assumed by GOLOG computations).

Similarly, we express the property "all executions of program δ terminate in states where ϕ holds", as the $\mathcal{L}$ formula $\mathit{afterA}(\delta,\phi)$ defined in Fig.2.[3] The basic difference between the definitions of $\mathit{afterA}(\delta,\phi)$ and $\mathit{afterS}(\delta,\phi)$ is in the treatment of the choice construct: in the case of $\mathit{afterA}(\delta,\phi)$ we require that, independently of the choices made, the program terminates in a state satisfying ϕ, while in the case of $\mathit{afterA}(\delta,\phi)$ only one such choice has to do so. That is, $\mathit{afterA}(\delta,\phi)$ is especially meaningful if the robot has no control on the choice points of the program, so we require that the program "does the right thing" independently of the choices made.[4]

Typical *total correctness* conditions, usually written as $[\phi_1]\delta[\phi_2]$, are expressible by $\phi_1 \supset \mathit{afterA}(\delta,\phi_2)$. Instead, *partial correctness* conditions (correctness for terminating executions only), usually written as $\{\phi_1\}\delta\{\phi_2\}$, are expressible by $\phi_1 \supset \mathit{afterAw}(\delta,\phi_2)$, where $\mathit{afterAw}(\delta,\phi)$ is the formula obtained from $\mathit{afterA}(\delta,\phi)$ replacing $\langle a\rangle\phi$ in the first equation by $[a]\phi$, and the least fixpoint μ in the last equation by a greatest fixpoint ν.

As a final example, we express the property "during every execution of the program δ a given property ϕ always holds". Again, we define the corresponding $\mathcal{L}$ dynamic formula $\mathit{duringA}(\delta,\phi)$ by induction on the structure of the program, as shown in Fig.2.

Sensing actions

In this section we sketch the formalization of sensing actions in our framework. Such a formalization follows the line of (De Giacomo *et al.* 1997).

A sensing action is an action which allows the robot to know the truth value of a certain property. We assume that a sensing action changes the epistemic state of the robot only wrt the value of the sensed property

[3] Observe that $\langle a\rangle\phi \equiv \langle a\rangle\mathbf{k}tt \wedge [a]\phi$, since actions are assumed to be deterministic.

[4] Notice that $\mathit{afterS}(\delta,\phi)$ is expressible in PDL (leaving aside the $\mathbf{k}$ operator), while $\mathit{afterA}(\delta,\phi)$ is not.

(and only if such value was not known by the robot before the action execution). An example of this kind of actions is the action *sense-door-open*, which requires the robot to check whether the door is open or closed.

Suppose a is a generic sensing action whose effect is to let the agent know the truth value of the property q, where q is any state formula. Also, suppose p is the precondition for the execution of a. Such a sensing action is represented in our framework by an usual action precondition axiom $\mathbf{k}p \supset \langle a \rangle \mathbf{k}tt$, plus the effect axiom

$$\mathbf{k}p \supset [a](\mathbf{k}q \vee \mathbf{k}\neg q)$$

which formalizes the fact that, after the execution of a, the robot knows whether q holds or not. Notably, for each sensing action a, we enforce a frame axiom schema of the form:

$$\mathbf{k}\varphi \supset [a]\mathbf{k}\varphi$$

which formalizes the fact that all the properties known by the robot before the execution of the sensing action are still known after executing it. Observe that as a consequence of the frame axiom schema if the robot already knows the truth-value of q then the sensing action a does not have any effect, in the sense if the robot knows q ($\neg q$) then after executing a the robot will still know q ($\neg q$). It is possible to show that the above axiom schema can be represented, without loss of generality, through a finite (linear) number of instances, by replacing φ in the schema with the initial state description and with each effect appearing in effect axioms.

The reasoning technique presented here can be straightforwardly adapted to deal with sensing actions. Roughly, adding sensing actions gives rise to logical theories with multiple models, however it is possible to suitably modify the algorithm reported in Fig.1 in order to correctly represent all such models by means of a unique transition graph.

Conclusions

In this paper we have shown a research direction in reasoning about actions stemming from the Robot-Tino Project(De Giacomo *et al.* 1996; 1997). The basic idea is to combine model checking for a very expressive logic of programs with propositional inference in order to exploit the model checking techniques and systems for reasoning about complex high-level robot behaviors.

The work presented is related to several proposals in reasoning about actions and cognitive robotics. There are clear connections with the work done by the Cognitive Robotics Group in Toronto. In particular our formal treatment of programs as a means to define formulae macros is inspired by (Levesque *et al.* 1997). Recently that group has also developed a "validity/provability based Golog" which shares, in fact, some of the ideas behind our transition graph construction (Lesperance & Tremaine 1998).

There are also some similarities with $\mathcal{A}$-like action languages (Gelfond & Lifschitz 1993; Baral & Gelfond 1993; Lifshitz & Karta 1994), indeed the semantics of such language is based on a single transition function, and this allows for building a single transition graph. States in such graph are characterized by the formulae that are *true* (vs. *valid*), while the initial state is replaced by a set of possible initial states. Notably, model checking techniques could be adopted in that setting as well (see (Chen & De Giacomo 1998) for a simple treatment of multiple possible initial states) and some work towards that direction has already started (Cimatti *et al.* 1997).

Model checking is the basic reasoning technique adopted in (De Giacomo & Chen 1996; Chen & De Giacomo 1998), where a process algebra is introduced to specify the behavior of the dynamic system, and a suitable variant of modal mu-calculus is adopted as verification formalism. Interestingly, programs (processes) in that work have a somewhat different role, since they are used for specifying basic behavior of the robot and are not considered in the verification formalism.

References

Baeten, J., and Weijland, W. 1990. *Process Algebra.* Cambrige, UK: Cambrige University Press.

Baral, C., and Gelfond, M. 1993. Representing concurrent actions in extended logic programming. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93)*, 866–871.

Burch, J. R.; Clarke, E. M.; McMillan, K. L.; Dill, D. L.; and Hwang, L. J. 1992. Symbolic model checking: 10^20 states and beyond. *Information and Computation* 98:142–170.

Bylander, T. 1991. Complexity results for planning. *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91).*

Chen, X. J., and De Giacomo, G. 1998. Reasoning about nondeterministic and concurrent actions: A process algebra approach. Submitted.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via model checking. In *Proceedings of the Fourth European Conference on Planning (ECP-97).*

Clarke, E.; Emerson, E.; and Sistla, A. 1986. Automatic verification of finite state concurrent systems

using temporal logic specifications. *ACM TOPLAS* 8(2):24–37.

Cleaveland, R., and Sims, S. 1996. The NCSU concurrency workbench. In *Computer-Aided Verification (CAV-96)*, number 1102 in Lecture Notes in Computer Science, 394–397. Springer-Verlag.

Cleaveland, R. 1990. Tableaux-based model checking in the propositional mu-calculus. *Acta Informatica* 27:725–747.

De Giacomo, G., and Chen, X. 1996. Reasoning about nondeterministic and concurrent actions: a process algebra approach. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI-96)*, 658–663. AAAI-Press/The MIT-Press.

De Giacomo, G., and Lenzerini, M. 1994. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, 205–212. AAAI Press/The MIT Press.

De Giacomo, G.; Iocchi, L.; Nardi, D.; and Rosati, R. 1996. Moving a robot: the KR&R approach at work. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-96)*, 198–209.

De Giacomo, G.; Iocchi, L.; Nardi, D.; and Rosati, R. 1997. Planning with sensing for a mobile robot. In *Proceedings of the Fourth European Conference on Planning (ECP-97)*.

Donini, F. M.; Lenzerini, M.; Nardi, D.; Nutt, W.; and Schaerf, A. 1992. Adding epistemic operators to concept languages. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-92)*, 342–353. Morgan Kaufmann, Los Altos.

Donini, F. M.; Lenzerini, M.; Nardi, D.; Nutt, W.; and Schaerf, A. 1994. Queries, rules and definitions. In *Foundations of Knowledge Representation and Reasoning*. Springer-Verlag.

Donini, F. M.; Nardi, D.; and Rosati, R. 1997. Autoepistemic description logics. In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI-97)*, 136–141.

Emerson, E. A., and Lei, C.-L. 1986. Efficient model checking in fragments of the mu-calculus. In *Proceeding of 1st IEEE Symposium on Logics in Computer Science (LICS'86*, 267–278.

Emerson, E. A. 1996. Automated temporal reasoning about reactive systems. In *Logics for Concurrency: Structure versus Automata*, number 1043 in Lecture Notes in Computer Science. Springer-Verlag. 41–101.

Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. *Journal of Logic Programming* 17:301–322.

H. R. Nielson, F. N. 1992. *Semantics with Applications*. Wiley.

Hennessy. 1988. *An Algebraic Theory of Processes*. Cambrige, MA: MIT Press.

Hoare, C. 1985. *Communicating Sequential Processes*. London: Prentice Hall Int.

Kozen, D., and Tiuryn, J. 1990. Logics of programs. In Leeuwen, J. V., ed., *Handbook of Theoretical Computer Science – Formal Models and Semantics*, 789–840. Elsevier Science Publishers (North-Holland), Amsterdam.

Kozen, D. 1983. Results on the propositional mu-calculus. *Theoretical Computer Science* 27:333–355.

Lesperance, Y., and Tremaine, D. 1998. A procedural approach to belief update for agent programming. Unpublished Manuscript, Department of Computer Science York University.

Levesque, H.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31:59–84.

Lifshitz, V., and Karta, G. 1994. Actions with indirect effects (preliminary report). In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, 341–350.

McMillan, K. L. 1993. *Symbolic Model Checking*. Kluwer Academic Publishers.

Milner, M. 1989. *Communication and Concurrency*. Prentice-Hall.

Schild, K. 1991. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, 466–471.

Stirling, C. 1996. Modal and temporal logics for processes. In *Logics for Concurrency: Structure versus Automata*, number 1043 in Lecture Notes in Computer Science. Springer-Verlag. 149–237.

Streett, R. S., and Emerson, E. A. 1989. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Control* 81:249–264.

Winskel, G. 1989. A note on model checking the modal ν-calculus. In *Proceedings of the 11th International Colloquium on Automata, Languages and Programming*, number 372 in Lecture Notes in Computer Science, 761–772. Springer-Verlag.