Reasoning about Actions in a Multiagent Domain

Laura Giordano[1], Alberto Martelli[2], and Camilla Schwind[3]

[1] DISTA, Università del Piemonte Orientale "A. Avogadro"
laura@mfn.unipmn.it
[2] Dipartimento di Informatica - Università di Torino
mrt@di.unito.it
[3] Faculté des Sciences de Luminy, LIM-CNRS
schwind@lim.univ-mrs.fr

Abstract. In this paper we present a theory for reasoning about actions which is based on the Product Version of Dynamic Linear Time Temporal Logic (denoted $DLTL^{\otimes}$) and allows to describe the behaviour of a network of sequential agents which coordinate their activities by performing common actions together. $DLTL^{\otimes}$ extends LTL, the propositional linear time temporal logic, by strengthening the until operator by indexing it with the regular programs of dynamic logic. Moreover, it allows the formulas of the logic to be decorated with the names of sequential agents, taken from a finite set.
The action theory we propose is an extension of the theory presented in [8], which is based on the logic DLTL, and allows reasoning with incomplete initial states and dealing with postdiction, ramifications as well as with nondeterministic actions. Here we extend this theory to cope with multiple agents synchronizing on common actions.

1 Introduction

An approach to reasoning about actions that recently gained renewed attention is the one based on the use of dynamic logic and of temporal logic. The suitability of modal logic for reasoning about actions has been pointed out by several authors in the last years [3,7]. On the one hand, Dynamic Logic [11] adopts essentially the same ontology as McCarthy's situation calculus, by taking the state of the world as primary, and encoding actions as transformations on states: actions can be represented in a natural way by modalities, and states as sequences of modalities. On the other hand, the adoption of a temporal logic for action theories allows general goals, like achievement and maintenance goals to be specified through temporal modalities.

The need of temporally extended goals has been motivated by Bacchus and Kabanza [1] and by Kabanza et al. [14], who proposed an approach to planning based on a linear time temporal logic. The formalization of properties of planning domains as temporal formulas in CTL has also been proposed in [10], where the idea of planning as model checking in a temporal logic has been explored. Other

F. Esposito (Ed.): AI*IA 2001, LNAI 2175, pp. 237–248, 2001.

authors have made use of the μ-calculus for reasoning about actions [21,4,5]. The μ-calculus allows complex goals to be formalized in a planning context, including achievement and maintenance goals [21].

In [8] the modal action theory developed in [7] has been enhanced by moving to the setting of temporal logics. More precisely, an action theory has been proposed based on the linear time temporal logic, DLTL (Dynamic Linear Time Temporal Logic [13]), which is, essentially, a dynamic logic equipped with a linear time semantics. It provides a simple way of constraining the (possibly infinite) evolutions of the system by making use of regular programs. The temporal projection problem and the planning problem can be modelled as satisfiability problems in DLTL.

In this paper we want to further exploit the expressiveness of temporal logic to extend our action theory for modelling a collection of interacting agents. To this purpose, we will make use of the Product Version of DLTL ($DLTL^{\otimes}$) to reason about a fixed number of finite state sequential agents, that coordinate their activities by performing their actions together. In [12] $DLTL^{\otimes}$ has been shown to be expressively equivalent to the regular product languages and to admit an exponential time decision procedure. In particular, the satisfiability and model checking problems for $DLTL^{\otimes}$ can be solved by product Büchi automata. A nice property of $DLTL^{\otimes}$ is that every formula of $DLTL^{\otimes}$ is trace consistent, so that properties defined by $DLTL^{\otimes}$ formulas can be verified efficiently by making use of partial order based reduction techniques.

$DLTL^{\otimes}$ does not allow to describe global properties of a system of agents, since the truth of a formula is evaluated at a local state (that is, a state local to an agent) and the temporal modalities define causal relationships among local states. However, it allows the specification of the dynamic of the system to be given through the separate specification of the different agents in the domain description.

Our action theory allows reasoning with incomplete initial states, and dealing with postdiction, ramifications and nondeterministic actions, which are captured by possibly alternative extensions (temporal models). Moreover, it provides a formalization of complex actions through the regular programs of dynamic logic and, in particular, it allows modelling the behaviour of different agents, which interact by executing common actions.

2 The Logic

We first shortly recall the definition of the logic $DLTL$ and, then, the definition of its product version.

2.1 *DLTL*

In this section we shortly define the syntax and semantics of DLTL as introduced in [13]. In such a linear time temporal logic the next state modality is indexed

by actions. Moreover, (and this is the extension to LTL) the until operator is indexed by programs in Propositional Dynamic Logic (PDL) [11].

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ, where $\omega = \{0, 1, 2, \ldots\}$. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω and by τ, τ' the words over Σ^*. Moreover, we denote by $\leq$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, we denote by $prf(u)$ the set of finite prefixes of u.

We define the set of programs (regular expressions) $Prg(\Sigma)$ generated by Σ as follows:

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*$$

where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$. A set of finite words is associated with each program by the mapping $[[]] : Prg(\Sigma) \to 2^{\Sigma^*}$, which is defined in the standard way, as follows:

- $[[a]] = \{a\}$;
- $[[\pi_1 + \pi_2]] = [[\pi_1]] \cup [[\pi_2]]$;
- $[[\pi_1; \pi_2]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi_1]] \text{ and } \tau_2 \in [[\pi_2]]\}$;
- $[[\pi^*]] = \bigcup [[\pi^i]]$, where
  - $[[\pi^0]] = \{\varepsilon\}$
  - $[[\pi^{i+1}]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi]] \text{ and } \tau_2 \in [[\pi^i]]\}$, for every $i \in \omega$.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions. The set of formulas of DLTL(Σ) si defined as follows:

$$\text{DLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \mathcal{U}^\pi \beta$$

where $p \in \mathcal{P}$ and α, β range over DLTL(Σ).

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \to 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula α, the satisfiability of a formula α at τ in M, written $M, \tau \models \alpha$, is defined as follows:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \alpha \mathcal{U}^\pi \beta$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \leq \tau'' < \tau'$[1], $M, \tau\tau'' \models \alpha$.

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in prf(\sigma)$ such that $M, \tau \models \alpha$.

The formula $\alpha \mathcal{U}^\pi \beta$ is true at τ if "α until β" is true on a finite stretch of behaviour which is in the linear time behaviour of the program π.

The derived modalities $\langle\pi\rangle$ and $[\pi]$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top\mathcal{U}^\pi\alpha$ and $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$. It is easy to see that $M, \tau \models \langle\pi\rangle\alpha$ iff there exists $\tau' \in [[\pi]]$

[1] We define $\tau \leq \tau'$ iff $\exists\tau''$ such that $\tau\tau'' = \tau'$. Moreover, $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$.

such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \alpha$. Also, $M, \tau \models [\pi]\alpha$ iff for all $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ it holds that $M, \tau\tau' \models \alpha$.

Furthermore, if we let $\Sigma = \{a_1, \ldots, a_n\}$, the $\mathcal{U}$, O (next), $\Diamond$ and $\Box$ of LTL can be defined as follows: $O\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box \equiv \neg\Diamond\neg\alpha$, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Hence both LTL(Σ) and PDL are fragments of DLTL(Σ). As shown in [13], DLTL(Σ) is strictly more expressive than LTL(Σ).

2.2 A Product Version of *DLTL*

We shortly recall the definition of $DLTL^{\otimes}$ from [12]. Let $Loc = \{1, \ldots, K\}$ be a set of locations, the names of the agents synchronizing on common actions. A *distributed alphabet* $\tilde{\Sigma} = \{\Sigma_i\}_{i=1}^K$ is a family of (possibly non-disjoint) alphabets, with each Σ_i a non-empty, finite set of actions (Σ_i is the set of actions which require the participation of agent i). Let $\Sigma = \bigcup_{i=1}^K \Sigma_i$. For $\sigma \in \Sigma^\infty$, we denote by $\sigma \uparrow i$ the projection of σ down to Σ_i. Moreover, we define $Loc(a) = \{i \mid a \in \Sigma_i\}$, the set of agents which participate in each occurrence of action a.

Atomic propositions are introduced in a local fashion, by introducing a non-empty set of atomic propositions $\mathcal{P}$. For each proposition $p \in \mathcal{P}$ and agent $i \in Loc$, p_i represents the "local" view of the proposition p at i, and is evaluated in the local state of agent i.

Let us define the set of formulas of $DLTL^{\otimes}(\tilde{\Sigma})$ and their locations:

- $\top$ is a formula and loc($\top$)=$\emptyset$;
- if $p \in P$ and $i \in Loc$, p_i is a formula and $loc(p_i) = \{i\}$;
- if α and β are formulas, then $\neg\alpha$ and $\alpha \vee \beta$ are formulas and $loc(\neg\alpha) = loc(\alpha)$ and $loc(\alpha \vee \beta) = loc(\alpha) \cup loc(\beta)$;
- if α and β are formulas and $loc(\alpha), loc(\beta) \subseteq \{i\}$ and $\pi \in Prg(\Sigma_i)$, then $\alpha\mathcal{U}_i^\pi\beta$ is a formula and $loc(\alpha\mathcal{U}_i^\pi\beta) = \{i\}$.

Notice that no nesting of modalities $\mathcal{U}_i$ and $\mathcal{U}_j$ (for $i \neq j$) is allowed, and the formulas in $DLTL^{\otimes}(\tilde{\Sigma})$ are boolean combinations of formulas from the set $\bigcup_i DLTL_i^{\otimes}(\tilde{\Sigma})$, where

$$DLTL_i^{\otimes}(\tilde{\Sigma}) = \{\alpha \mid \alpha \in DLTL^{\otimes}(\tilde{\Sigma}) \textit{ and } loc(\alpha) \subseteq \{i\}\}.$$

A model of $DLTL^{\otimes}(\tilde{\Sigma})$ is a pair $M = (\sigma, V)$, Where $\sigma \in \Sigma^\infty$ and $V = \{V_i\}_{i=1}^K$ is a family of functions V_i, where $V_i : prf(\sigma \uparrow i) \to 2^{\mathcal{P}}$ is the valuation function for agent i.

The satisfiability of formulas in a model is defined as above, except that propositions are evaluated locally. In particular, for all $\tau \in prf(\sigma)$:

- $M, \tau \models p_i$ iff $p \in V_i(\tau \uparrow i)$;
- $M, \tau \models \alpha\mathcal{U}_i^\pi\beta$ iff there exists a τ' such that $\tau' \uparrow i \in [[\pi]]$, $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for all τ'' such that $\varepsilon \leq \tau'' < \tau'$, $M, \tau\tau'' \models \alpha$.

Satisfiability in $DLTL^{\otimes}$ is defined as above. In [12] $DLTL^{\otimes}$ has been shown to be expressively equivalent to the regular product languages and to admit an exponential time decision procedure.

3 Action Theories

In this section we extend to a multiagent setting the action theory developed in [8]. In the following the behaviour of the global system will emerge as the product of the behaviours of the single agents which interact by synchronizing on common actions. The behaviour of each agent i is specified by a domain description D_i, which describes the atomic actions the agent may perform by means of a set of action laws, causal rules, precondition laws and a set of general constraints. Such constraints also provide a description of the complex behaviour of the agent, given by means of regular programs of dynamic logic. The agents interact by performing common actions together, where each agent participating in the action execution has its own local description of the action determining the action effects on its local state.

In our action theories we call *fluent names* the atomic propositions in $\mathcal{P}$ indexed by an agent name i. Namely, for each $p \in P$ and $i \in Loc$, p_i is a fluent name local to i. A *fluent literal* l is a fluent name f or its negation $\neg f$. Given a fluent literal l, such that $l = f$ or $l = \neg f$, we define $|l| = f$. We will denote by Lit_i the set of all fluent literals local to i.

A *(distributed) domain description* D is a family of domain descriptions D_i, one for each agent i. A *domain description* D_i *for agent* i is defined as a tuple $(\Pi_i, Frame_i, \mathcal{C}_i)$, where Π_i is a set of *action laws* and *causal laws*; $\mathcal{C}_i$ is a set of *constraints*; $Frame_i$ provides a classification of fluents as frame fluents and nonframe fluents as we will define below.

Action laws in Π_i have the form: $\Box_i(\alpha \rightarrow [a]_i\beta)$, with $a \in \Sigma_i$ and $\alpha, \beta \in DLTL_i^{\otimes}$, meaning that executing action a in a local i state where precondition α holds causes the effect β to hold.

Causal laws in Π_i have the form: $\Box_i(\bigwedge_{a\in\Sigma}([a]_i\alpha \rightarrow [a]_i\beta))$, with $a \in \Sigma_i$ and $\alpha, \beta \in DLTL_i^{\otimes}$, meaning that, for all actions a, if α holds (in the local i state) after the execution of a, then β also holds after its execution. Such laws are intended to expresses "causal" dependencies among fluents (see [18,17,22,7]), and, intuitively, their directionality makes them similar to inference rules: if we are able to derive α then we can conclude β.

Constraints in $\mathcal{C}_i$ are arbitrary formulas of $DLTL^{\otimes}$. Constraints not only put conditions on the value of fluents at the different states, but they also determine which are the possible behaviours of the agent in a state. We do not put any restrictions on the kind of constraints that we allow in $\mathcal{C}_i$, which include both safety and liveness constraints. Let us only mention some constraints that are usually discussed in reasoning about actions literature. *Domain constraints* of the form **always** α, as introduced in [15], which enforce the condition α to hold on all possible states, are safety constraints which can be formalized by the formula $\Box_i\alpha$. *Precondition laws*, which put conditions on the executability of actions in a state, can be formalized, in a linear time temporal logic, by formulas of the form $\Box_i(\alpha \rightarrow [a]_i\bot)$, meaning that action a cannot be executed in all reachable states

in which α holds[2]. *Observations* about the value of fluents in different states can be formalized as state constraints of the form $[a_1; \ldots; a_j]_i\alpha$, meaning that $\alpha \in DLTL^{\otimes}$ holds in the state obtained after executing the action sequence $a_1, \ldots, a_j$ (and, in particular, in the initial state when the action sequence is empty).

Constraints in $\mathcal{C}_i$ can be used to describe the sequences of actions which are possible for the agent. For instance, the liveness constraint $\langle\pi\rangle_i\top$ constrains all the executions of agent i in the system to start with a finite sequence of actions which is a possible behaviour of the program π.

We assume that for each causal law $\Box_i(\bigwedge_{a\in\Sigma}([a]_i\alpha \rightarrow [a]_i\beta)$ in Π, there is a corresponding constraint formula $\alpha \rightarrow \beta$ in the set $\mathcal{C}_i$, which assures that also in the initial state if α holds, β holds too.

$Frame_i$ is a set of pairs (p_i, a), where p_i is a fluent for agent i, and $a \in \Sigma_i$ is an action to which agent i participates. $(p_i, a) \in Frame_i$ means that, for agent i, p_i is a frame fluent for action a, that is, p_i is a fluent to which persistency applies when action a is executed. Those fluents which are not frame with respect to a do not persist and may change value in a nondeterministic way, when executing a. Note that it may occur that $(p_i, a) \in Frame_i$ while $(p_j, a) \notin Frame_j$ for $i \neq j$.

As $DLTL^{\otimes}$ does not include test actions, we introduce them in the language as atomic actions in the same way as done in [8]. Test actions allow the choice among different behaviours to be controlled. To allow agent i to test the value of a proposition ϕ in its local state, we introduce the modality $[\phi?]_i$ (regarded as an atomic action in Σ_i), ruled by the following laws:

$\Box_i(\neg\phi \rightarrow [\phi?]_i\bot)$
$\Box_i(< \phi? >_i \top \rightarrow (L \leftrightarrow [\phi?]_i L))$, for all fluent literals $L \in Lit_i$.

The first law is a precondition law, saying that action $\phi?$ is only executable in a state in which ϕ holds. The second law describes the effects of the action on the state: the execution of the action $\phi?$ leaves the state unchanged[3]. In the following we will assume that, for all test actions occurring in the examples, the corresponding action laws are implicitly added (as constraints) to the domain description.

We remark that the global state of the system can be regarded as a set of local states, one for each agent i. The action laws and causal rules in D_i describe how the local i state changes when an action $a \in \Sigma_i$ is executed. An action a which is common to agents i and j is executed synchronously by the two agents, which update their local states separately, according to their action specification given in D_i and D_j. Moreover, the behaviour of each agent i, a subsequence σ_i of the system behaviour σ, must satisfy the constraints in $\mathcal{C}_i$. In particular, the action must be executable for both the agents i and j.

[2] In a branching time logic, precondition laws are usually formalized by laws of the form $\Box_i\ (\alpha \rightarrow \langle a\rangle_i\top)$. However, this formalization is not well suited for a linear time logic, in which only a single action can be executed at each state.

[3] Note that, as a difference with PDL, the execution of test actions causes the state to change, though the value of propositions in the state is kept unaltered.

Though action laws and causal laws can themselves be regarded as special kinds of constraints, we have distinguished them from all other constraints as they are given a special treatment when dealing with the frame problem. As in [8], to deal with the frame problem we make use of a completion construction which, given a domain description, introduces frame axioms for all the frame fluents in the style of the successor state axioms introduced by Reiter [20] in the context of the situation calculus. The completion construction is applied only to the action laws and causal laws and not to the constraints. In fact, we assume that frame fluents only change values according to the immediate and indirect effects of actions described by the action laws and causal laws. For lack of space we refer to [8] for the details on the completion construction. We just mention that, in this multiagent setting, the action laws and causal rules in the Π_i's have to be considered separately for each i, by introducing different frame axioms for the different agents. Given a domain description D, we call $Comp(D)$ be the set of formulas including all frame axioms as well as the constraints $\mathcal{C}_i$ for all agents i. The extensions of the domain description D are defined as the models of $Comp(D)$.

The temporal projection problem and the planning problem can be modelled as satisfiability problems in $DLTL^{\otimes}$, as it has been done in [8] in the single agent context. In particular, the *planning problem* "is there a sequence of actions leading to a state where a given formula α_i holds for agent i ?" can be solved by checking if the query $\Diamond_i \alpha_i$ has a solution in D, that is, if there is a model $M = (\sigma, V)$ satisfying the query and the completion of the domain description $Comp(D)$. The plan which makes α_i true can be extracted from the model M[4]. It is a "concurrent plan" and it contains the actions which have to be performed by the different agents to reach the goal. Some of the actions are common to different agents and must be executed by the different agents together. Other actions are local to a single agent. Though the plan which is extracted from a model is a linear plan, it represents a class of equivalent plans which can be obtained by permutations of adjacent independent actions (where two actions a and b are independent when there is no agent participating in both of them, i.e. $Loc(a) \cap Loc(b) = \emptyset$)[5]. Thus a linear plan can be regarded as a plan in which the actions of the different agents are partially ordered.

Example 1. Let us first consider an example of two robots which must move a set of blocks from a room to another using a table [2]. The robots can put individually the blocks on the table, but they must lift and move the table simultaneously, otherwise the blocks will fall off.

[4] In general, one may want to find a plan in which the different agents achieve different goals. Hence, the query to be satisfied may have the form $\Diamond_i \alpha_i \wedge \ldots \wedge \Diamond_K \alpha_K$, where α_i is the goal of agent i.

[5] In [12] it has been shown that the behaviours described by the temporal logic $DLTL^{\otimes}$ (i.e. regular product languages) lie within the domain of regular Mazurkiewics trace languages. Thus $DLTL^{\otimes}$ provides a flexible and powerful means for specifying trace consistent properties of distributed programs, which can be verified efficiently by making use of partial order based reduction techniques.

Each robot has a private state consisting of a set of fluents, which specify what the robot knows about the world. In a simplified formulation we assume that there are only two blocks $Block_1$ and $Block_2$, and each robot knows the position of one of them, besides knowing its own position and that of the table. In the following, p, q, r will range in the set $\{Room_1, Room_2\}$, while b will range in the set $\{Block_1, Block_2\}$.

The set of fluents of the i-th robot is[6]: $P_i = \{at_i(Block_i, p), at_i(Robot_i, p), at_i(Table, p), table\_up_i, on\_table_i(Block_i)\}$ for all values of p. Each one of the two robots has the following alphabet of actions: $\Sigma_i = \{put\_on\_table_i(Block_i), lift\_table, move\_table(p)\}$, for all p and $i = 1, 2$, where $put\_on\_table_i(Block_i)$, the action of putting block $Block_i$ on the table, is private of each robot, whereas the two other actions of lifting the table and moving it to a position p are shared by them and must be executed simultaneously.

We introduce the following precondition and action laws (for all p, q, r, b):

$\Box_i((at_i(Table, p) \wedge at_i(Robot_i, q) \wedge at_i(b, r)) \vee on\_table_i(b)) \rightarrow$
$[put\_on\_table_i(b)]_i \bot)$ with $p \neq q$ or $p \neq r$.
$\Box_i([put\_on\_table_i(b)]_i on\_table_i(b))$
$\Box_i((at_i(Table, p) \wedge at_i(Robot_i, q)) \rightarrow [lift\_table]_i \bot)$ with $p \neq q$.
$\Box_i([lift\_table]_i table\_up_i)$
$\Box_i((at_i(Table, p) \vee \neg table\_up_i) \rightarrow [move\_table(p)]_i \bot$
$\Box_i([move\_table(r)]_i at_i(Table, r))$

As we have pointed out before, precondition laws specify when actions *cannot* be executed. Note that when executing a shared action each robot will update its state separately according to its action law. Furthermore we need some constraints. For all possible values of x and for $p \neq q$, $\Box_i(\neg(at_i(x, p) \wedge at_i(x, q))$ Finally, the fact that a box on the table moves with it can be described by the causal law: $\Box_i([a]_i at_i(Table, p) \wedge on\_table_i(b)) \rightarrow [a]_i at_i(b, p))$, for each action $a \in \Sigma_i$.

We assume that all fluents are frame fluents. Let us now add formulas describing the initial state of $Robot_1$ and $Robot_2$:

$at_1(Robot_1, Room_1) \wedge at_1(Block_1, Room_1) \wedge at_1(Table, Room_1) \wedge \neg table\_up_1 \wedge$
$\neg on\_table_1(Block_1)$
$at_2(Robot_2, Room_1) \wedge at_2(Block_2, Room_1) \wedge at_2(Table, Room_1) \wedge \neg table\_up_2 \wedge$
$\neg on\_table_2(Block_2)$

Assume now that we want the robots to move both boxes to the other room. This can be expressed as a satisfiability problem: Given the query:

$$\Diamond_1 at_1(Block_1, Room_2) \wedge \Diamond_2 at_2(Block_2, Room_2)$$

find a model (of the domain description $Comp(D)$) satisfying it. For instance the model beginning with

$put\_on\_table_1(Block_1), put\_on\_table_2(Block_2), lift\_table, move\_table(Room_2)$

[6] Identifiers beginning with a capital letter denote constants.

is a possible solution. From this solution we can extract the two plans for the two robots, by projecting the solution on the two alphabets. Note that, as we have pointed out before, the relative order among private actions of different agents (as, for instance, $put\_on\_table_1(Block_1)$ and $put\_on\_table_2(Block_2)$), is not important so that they can also be executed in a different order and the above plan can be regarded as only specifying a partial ordering of actions.

In this example each robot can only reason about the effects of its own actions and cannot know what the other robot is doing. In a more realistic setting the actions of the robots might interfere and thus a robot should be able to acquire knowledge about the effects of the actions of other robots. This can be modeled by introducing an environment which keeps track of the changes caused by all robots, and by providing the robots with *sensing* actions allowing the value of a fluent to be asked to the environment. In the next example, reasoning about actions allows properties of a system of agents to be verified.

Example 2. Two robots have to lift a table [6] and their actions must be synchronized so that the table does not tip so much that objects on it fall off. Differently from the previous example there are no shared actions between the two robots, and they can lift the table separately.

We model the two robots, $robot_1$ and $robot_2$ and their interaction with the environment E. Here we have three locations $\{1, 2, E\}$ and a distributed alphabet $\tilde{E} = \{\Sigma_1, \Sigma_2, \Sigma_E\}$ where $\Sigma_1 = \{ grab\_left, vmove\_left, sense\_vpos\_right\}$ contains the actions which $robot_1$ executes to lift the left part of the table; Σ_2 $=\{grab\_right, vmove\_right, sense\_vpos\_left\}$ contains the actions which $robot_2$ executes to lift the right part of the table; Σ_E $=\{vmove\_right, vmove\_left, sense\_vpos\_right, sense\_vpos\_left\}$ contains the actions which the environment executes, to record the modifications produced by the robot actions. The two actions $sense\_vpos\_right$ and $sense\_vpos\_left$ are informative actions (usually called "sensing actions") which allow the robots to acquire up to date knowledge about the part of the environment which is not under their control. For instance, we can expect that $robot_1$ has to do a sensing action to know the vertical position of the right end of the table which is under the control of $robot_2$.

The set of atomic propositions $\mathcal{P}$ is the following $\{holding\_right$,$holding\_left$, $vpos\_right(d)$, $vpos\_right(d)$, up_right, $up\_left\}$, for all the possible (finite and discrete) values d of the vertical position. In the following we will assume that there are 10 different vertical positions possible for the right end of the table $vpos\_right(1), \ldots, vpos\_right(10)$, and the same for the left one.

The set Π contains the following action laws for $robot_1$ (for all $d \in \{1 \ldots 10\}$ and d' successors of d):

$\Box_1([grab\_left]_1 holding\_left_1)$
$\Box_1(vpos\_left(d)_1 \rightarrow [vmove\_left]_1 vpos\_left(d')_1)$
$\Box_1([sense\_vpos\_right(d)]_1 vpos\_right(d)_1)$.

The $grab\_left$ action has the effect of $robot_1$ holding the left end of the table. The $vmove\_left$ action increases the vertical position of the left end. The $sense\_vpos\_right(d)$ action provides the position d of the right end.

The set $\mathcal{C}$ contains the following precondition laws and constraints:

$\Box_1(holding\_left_1 \rightarrow [grab\_left]_1\bot)$
$\Box_1(\neg holding\_left_1 \rightarrow [vmove\_left]_1\bot)$
$\Box_1(\neg[sense\_vpos\_right(d)]_1\bot)$
$\Box_1(up\_left_1 \leftrightarrow vpos\_left(10)_1)$
$\Box_1(safetolift_1 \leftrightarrow vpos\_left(d)_1 \wedge vpos\_right(f)_1 \wedge d \leq f + Tot)$

The $grab\_left$ action is possible only if $robot_1$ is not holding the left end of the table. The $vmove\_left$ action is possible only if $robot_1$ is holding the left end of the table. As far as $robot_1$ is concerned the $sense\_vpos\_right(d)$ action is always executable. The left end of the table is up if its vertical position is 10. It is safe for $robot_1$ to lift the table if the difference between the vertical positions of the left and the right ends of the table is less then a certain value.

The following program π_1 describes the possible behaviours of $robot_1$:

$$\pi_1 = grab\_left; (\neg up\_left_1?; (sense\_vpos\_right(1) + \ldots + sense\_vpos\_right(10)); \\ ((safetolift?; vmove\_left) + \neg safetolift?))^*; up\_left_1?$$

Analogous action laws and constraints are provided for the robot 2 (just exchange left and right and replace 1 with 2). Concerning the environment, we have the following actions laws Π (for all $d \in \{1 \ldots 10\}$ and d' successors of d):

$\Box_E(vpos\_right(d)_E \rightarrow [vmove\_right]_E vpos\_right(d')_E)$
$\Box_E(vpos\_left(d)_E \rightarrow [vmove\_left]_E vpos\_left(d')_E)$

and constraints $\mathcal{C}$:

$\Box_E(\neg[vmove\_right]_E\bot)$
$\Box_E(\neg[vmove\_left]_E\bot)$
$\Box_E(\neg vpos\_right(d)_E \rightarrow [sense\_vpos\_right(d)]_E\bot)$
$\Box_E(\neg vpos\_left(d)_E \rightarrow [sense\_vpos\_left(d)]_E\bot)$
$\Box_E(up\_right_E \leftrightarrow vpos\_right(10)_E)$
$\Box_E(up\_left_E \leftrightarrow vpos\_left(10)_E)$

The $sense\_vpos\_right(d)$ action is executable if d is the vertical position of the right end of the table. It has no effect on the environment, but it is how the environment provides information to $robot_1$ about the value of $vpos\_right$. The $vmove\_right$ action is always executable by the environment. When it is executed by $robot_2$, it has the effect of changing the value of $vpos\_right$ in the environment.

By adding the constraints $< \pi_1 >_1 \top$ and $< \pi_2 >_2 \top$ we specify that we will accept only sequences of actions where the programs of both robots terminate.

Assuming that in the initial state both ends of the table are down, we can formalize the fact that in any execution of the system the ends of the table are always at the same height to within a threshold Tot until both ends are up, by the following formula: $Level\ \mathcal{U}_E(up\_left_E \wedge up\_right_E)$, where $Level \equiv_{def} vpos\_right(d)_E \wedge vpos\_left(f)_E \wedge \mid d - f \mid \leq Tot$. The formula has to be true in all models of $Comp(D)$. While in the previous example we wanted to find a

plan and the problem was formalized as a satisfiability problem, here we want to verify some property α of the system of agents, which can be formalized as the problem of verifying the validity of a formula (namely, $Comp(D) \rightarrow \alpha$).

4 Conclusions

In this paper we have proposed a temporal logic approach for reasoning about actions and change in a multiagent context. Our action theory extends the one presented in [8], which is based on Dynamic Linear Time Temporal Logic (DLTL), by moving to the Product Version of DLTL: the behaviours of the system are generated by a network of sequential agents that coordinate their activities by performing common actions together. In our action theory the frame and the ramification problem are both addressed by lifting to the multiagent case the solution developed in [8]. More precisely, (modal) successor state axioms are introduced which are obtained by a completion construction and which are similar to those introduced by Sheila McIlraith in [19] for dealing with ramifications in the situation calculus. A similar kind of transformation has also been used by Enrico Giunchiglia in [9] to translate action theories to first order logic theory to model planning as satisfiability in first order logic. As a difference with these proposals, our action theory deals with the multiagent case and it allows constraints (which are arbitrary temporal formulas) and queries to include program expressions. Under this respect, the language is related to the language ConGolog [6], an extended version of the language Golog [16] that incorporates a rich account of concurrency, in which complex actions (plans) can be formalized as Algol-like programs in the situation calculus. A substantial difference with ConGolog, apart from the different logical foundation, is that here we model agents with their own local states, while in Congolog the agents share a common global environment and all the properties are referred to a global state.

In [2] Boutilier and Brafman address the problem of planning with concurrent interacting actions. They show that, the STRIPS action representation can be augmented to handle concurrent interacting actions and they develop an extension of the UCPOP algorithm to solve a multiagent planning problem. Also in this case, differently from our approach, all action affect the same global state. Though the presence of a global environment is very natural in a robotic context (and, in fact, in our second example we had to explicitly introduce a model of the environment), this is not always the case. For instance, when modelling the interaction between software agents, it is reasonable to assume that each of them has its own local state on which actions have effect.

References

1. F. Bacchus and F. Kabanza. Planning for temporally extended goals. in *Annals of Mathematics and AI*, 22:5–27, 1998.
2. C. Boutilier and R.I. Brafman. Planning with Concurrent Interacting Actions. in *AAAI-97*, Providence, August 1997.

3. G. De Giacomo, M. Lenzerini. PDL-based framework for reasoning about actions. In *LNAI 992*, pages 103–114, 1995.
4. G. De Giacomo and X.J.Chen. Reasoning about nondeterministic and concurrent actions: A process algebra approach. In Artificial Intelligence, 107:63-98,1999.
5. G. De Giacomo and R. Rosati. Minimal knowledge approach to reasoning about actions and sensing. In *Proc. of the 3rd Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC'99)*, Stockholm, Sweden, August 1999.
6. G. De Giacomo, Y. Lespérance, H. J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(2000), pp.109-169.
7. L. Giordano, A. Martelli, and C. Schwind. Ramification and causality in a modal action logic. In *Journal of Logic and Computation*, 10(5):625-662, 2000.
8. L. Giordano, A. Martelli, and C. Schwind. Reasoning About Actions in Dynamic Linear Time Temporal Logic. *Proceedings FAPR'2000, Int. Conf. on Pure and Applied Practical Reasoning*, London, September 2000. To appear on the Logic Journal of the IGPL.
9. E. Giunchiglia. Planning as satisfiability with expressive action languages: Concurrency, Constraints and Nondeterminism. In Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'00). Breckenridge, Colorado, USA 12-15 April 2000.
10. F. Giunchiglia and P. Traverso. Planning as Model Checking. In Proc. The 5th European Conf. on Planning (ECP'99), pp.1–20, Durham (UK), 1999.
11. D. Harel. First order dynamic logic in *Extensions of Classical Logic, Handbook of Philosophical Logic II*, pp. 497–604, 1984.
12. J.G. Henriksen and P.S. Thiagarajan A product Version of Dynamic Linear Time Temporal Logic. in *CONCUR'97*, 1997.
13. J.G. Henriksen and P.S. Thiagarajan Dynamic Linear Time Temporal Logic. in *Annals of Pure and Applied logic*, vol.96, n.1-3, pp.187–207, 1999
14. F. Kabanza, M. Barbeau and R.St-Denis Planning control rules for reactive agents. In *Artificial Intelligence*, 95(1997) 67-113.
15. G.N. Kartha and V. Lifschitz. Actions with Indirect Effects (Preliminary Report). In *Proc. KR'94* , pages 341–350, 1994.
16. H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *J. of Logic Prog.*, 31, 1997.
17. F. Lin. Embracing Causality in Specifying the Indirect Effects of Actions. In *Proc. IJCAI'95*, pages 1985–1991, 1995.
18. N. McCain and H. Turner. A Causal Theory of Ramifications and Qualifications. In *Proc. IJCAI'95*, pages 1978–1984, 1995.
19. S. Mc Ilraith Representing Actions and State Constraints in Model-Based Diagnosis. AAAI'97, pp. 43–49, 1997.
20. R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, V. Lifschitz, ed.,pages 359–380, Academic Press, 1991.
21. Munindar P. Singh. Applying the Mu-Calculus in Planning and Reasoning about Actions. In *Journal of Logic and Computation*, 1998.
22. M. Thielscher. Ramification and Causality. Artificial Intelligence Journal, vol. 89, No. 1-2, pp. 317-364, 1997.