Extending $\mathcal{C}+$ with Composite Actions for Robotic Task Planning

Xiaoping Chen[1], Guoqiang Jin[1], and Fangkai Yang[2]

1 School of Computer Science, University of Science and Technology of China
2 Department of Computer Science, University of Texas at Austin

Abstract

This paper extends action language $\mathcal{C}+$ by introducing *composite actions* as sequential execution of other actions, leading to a more intuitive and flexible way to represent action domains, better exploit a general-purpose formalization, and improve the reasoning efficiency for large domains. Our experiments show that the composite actions can be seen as a method of knowledge acquisition for intelligent robots.

1998 ACM Subject Classification I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases Reasoning about Actions, Action Languages, Robotic Task Planning

Digital Object Identifier 10.4230/LIPIcs.ICLP.2012.404

1 Introduction

The problem of describing changes caused by the execution of actions plays an important role in knowledge representation. Actions may be described

1. by specifying their preconditions and effects, as in STRIPS [5], PDDL-like languages, action languages such as $\mathcal{B}$ and $\mathcal{C}$ [7], $\mathcal{C}+$ [8], situation calculus [14];
2. in terms of execution of primitive actions, such as programs in GoLog [10], ASP [17], extended event calculus [16], ABStrips [15] and HTN [4]; or
3. as a special case of actions of more general kind, as in MAD [11] and $\mathcal{ALM}$ [6].

Actions formalized in the first and third approach are used to automate planning, and more generally, to automate commonsense reasoning tasks such as temporal projection and postdiction, with an emphasis on addressing the problem of generality in AI [12]. However, actions formalized in the second approach are usually used for complementary purposes: they are abstractions or aggregates that characterize the hierarchical structure of the domain and improve search efficiency. This paper extends action language $\mathcal{C}+$ with *composite actions* defined as sequential execution of other actions, and shows that these composite actions can be used for the purposes of the first and third approaches as well.

The extended $\mathcal{C}+$ has three advantages. First, it provides one more way to formalize actions in $\mathcal{C}+$. Second, composite actions can be defined by exploiting the general purpose formalization of actions, a step of addressing the problem of generality in AI, or by exploiting natural language information for knowledge acquisition. Third, composite actions can be used to characterize the hierarchical structure of problem and improve planning efficiency.

To achieve this goal, we add a new construct to $\mathcal{C}+$ that defines *composite actions* as sequential executions of actions $a_0, \dots a_k$ under conditions (written as formulas) $E_0, \dots E_k$. For instance, consider a domain of a robot with a hand which can deliver small objects from one place to another. The primitive actions represent the basic functions of the robot such

Technical Communications of the 28th International Conference on Logic Programming (ICLP'12).
Editors: A. Dovier and V. Santos Costa; pp. 404–414
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

as `move`, `pickup`, `putdown`. We can define a composite action $\mathit{Fetch}(s, l)$ as the consecutive executing of the actions $\mathit{Move}(l_1)$ and $\mathit{Bring}(s, l)$

$$\mathit{Fetch}(s, l) \textbf{ is } \mathit{Move}(l_1) \textbf{ if } \mathit{Loc}(s) = l_1 \wedge \mathit{Loc}(\mathit{Robot}) \neq l_1; \mathit{Bring}(s, l).$$

However, to define the semantics of the construct is not a trivial task. In $\mathcal{C}+$, all actions are assumed to be executed over 1 time interval. This assumption affects the design of both description language and query language of CCalc[1]: descriptions of action domains don't involve formalizing time, leading to a concise representation; when formulating queries, time instances can be named explicitly and conveniently based on the assumption. In presence of composite actions, it is natural to talk about their lengths and how their lengths affect the design of the language. It happens that defining the length of a composite action in terms of the number of primitive actions it involves leads to a cumbersome query language, since the number is not fixed for a composite action: the length of $\mathit{Fetch}(s, l)$ depends on the location of the robot and s. Therefore, when formulating queries, the user may need to explicitly name the indefinite lengths of actions, which becomes complicated when doing distant projection, postdiction or planning. For simplicity of the syntax, we extend the assumption to composite actions so that it is fully compatible with CCalc input, but use a notion of *subintervals* in their semantics to characterize execution trajectories of composite actions.

The new language is implemented by modifying the software cplus2asp [1], which translates the input into an incremental answer set program and calls the solver iclingo[2]. We formalize a version of a service robot domain with composite actions, and show that composite actions can be used for knowledge acquisition, and improve planning efficiency for large problems.

The work presented in this paper is somewhat similar to [9] but composite actions defined there have fixed and explicitly specified length.

2 Preliminaries

The review of action language $\mathcal{C}+$ follows [8]. A (multi-valued) signature is a set σ of symbols, called (multi-valued) constants, along with a non-emtpy finite set $\mathit{Dom}(c)$ of symbols, disjoint from σ, assigned to each constant c. Each constant belongs to one of the three groups: *action* constants, *simple fluent* constants and *statically determined fluent* constants.

Consider a fixed multi-valued signature σ. An *atom* is an expression of the form $c = v$ ("the value of c is v") where $c \in \sigma$ and $v \in \mathit{Dom}(c)$. A *formula* is a propositional combination of atoms. An interpretation maps every constant in σ to an element of its domain. A formula is called *fluent formula* if it does not contain action constants, and *action formula* if it contains at least one action constant and no fluent constants.

An *action description* consists of a set of *causal laws* of the form

$$\textbf{caused } F \textbf{ if } G \tag{1}$$

where F and G are formulas. The rule is called *static law* if F and G are fluent formulas, or *action dynamic law* if F is an action formula; and rules of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{2}$$

where F and G are fluent formulas, and H is a formula, called *fluent dynamic law*.

Many useful constructs are defined as abbreviations for the basic forms (1) and (2) shown above. For instance, the law

[1] `http://www.cs.utexas.edu/users/tag/cc/`
[2] `http://potassco.sourceforge.net/`

$$\textbf{a causes F if G}, \qquad \text{for an action constant } a, \tag{3}$$

stands for **caused** F **if** $\top$ **after** $a \wedge G$;

$$\textbf{inertial } c, \qquad \text{for a fluent constant } c, \tag{4}$$

stands for **caused** c **if** c **after** c;

$$\textbf{exogenous } a, \qquad \text{for an action constant } a, \tag{5}$$

stands for **caused** a **if** a and **caused** $\neg a$ **if** $\neg a$;

$$\textbf{default } a, \qquad \text{for an action constant } a, \tag{6}$$

stands for **caused** a **if** a; and

$$\textbf{nonexecutable H if F}, \qquad \text{for an action formula } H, \tag{7}$$

stands for **caused** $\bot$ **after** $H \wedge F$.

A *causal theory* contains a finite set of *causal rules* of the form $F \Leftarrow G$ where F and G are formulas. Following [8], the semantics of an action description D is defined by a translation to the union of an infinite sequence of causal theories D_m $(m \geq 0)$. The signature of D_m consists of pairs of form $i : c$ such that $i \in \{0, \ldots, m\}$ and c is a fluent constant of D, or $i \in \{0, \ldots, m-1\}$ and c is an action constant of D. The rules of D_m are

- $i : F \Leftarrow i : G$, for static law (1) in D and $i \in \{0, \ldots, m\}$, and action dynamic law (1) in D and $i \in \{0, \ldots, m-1\}$;
- $i{+}1 : F \Leftarrow (i{+}1 : G) \wedge (i : H)$, for every fluent dynamic law (2) and $i \in \{0, \ldots, m-1\}$;
- $0 : c = v \Leftarrow 0 : c = v$, for simple fluent constant c and $v \in Dom(c)$.

A model of causal theory D_m can be seen as a path of length m in the transition diagram, as described in proposition 8 of [8].

Example 1. Consider a robot that uses a manipulator to transfer small objects from one place to another. It can perform actions $Move(l)$, $Pickup(s)$, $Putdown(s)$ which affects inertial fluents $Loc(o)$, $Hold(s)$, where l denotes the places in the domain, o the objects, s the small objects which can be grasped by the robot. The action description is

$$\textbf{inertial } Loc(o) = l \quad \textbf{inertial } Hold(s)$$
$$\textbf{exogenous } Move(l) \quad \textbf{exogenous } Pickup(s) \quad \textbf{exogenous } Putdown(s)$$
$$\textbf{caused } Loc(s) = l \textbf{ if } Hold(s) \wedge Loc(Robot) = l$$
$$Move(l) \textbf{ causes } Loc(Robot) = l \quad \textbf{nonexecutable } Move(l) \textbf{ if } Loc(Robot) = l$$
$$Pickup(s) \textbf{ causes } Hold(s) \quad \textbf{nonexecutable } Pickup(s) \textbf{ if } \neg Hold(Nothing)$$
$$\textbf{nonexecutable } Pickup(s) \textbf{ if } Loc(Robot) \neq Loc(s)$$
$$Putdown(s) \textbf{ causes } Hold(Nothing) \quad \textbf{nonexecutable } Putdown(s) \textbf{ if } \neg Hold(s)$$

The action desription D^0 is obtained by setting the variables $l \in \{L_1, L_2\}, o \in \{Robot, S\}, s \in \{S\}$. A model of D^0_4 can be represented as a path of length 4 in the transition diagram of D^0.

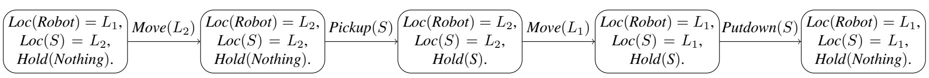

Figure 1 One path in the transition diagram D^0.

3 Defining Composite Actions

3.1 Syntax

We consider a fragment of general action descriptions in $\mathcal{C}+$ containing static laws of the form (1), action dynamic laws of the form (5) and (6), and fluent dynamic laws of the form (3), (4) and (7).

Given an action description D with a set of fluent constants σ^{fl} and a set of action constants σ^{act}, an *extended action description* D^+ introduces a set of *composite action constants* σ^{comp} and *composite action definition laws* of the form

$$b \textbf{ is } (a_0 \textbf{ if } E_0); \ldots; (a_k \textbf{ if } E_k) \tag{8}$$

where $b \in \sigma^{comp}$ is the *head* of the law, called a *composite action constant.* $a_0, \ldots, a_k \in \sigma^{act} \cup \sigma^{comp}$, and $E_0, \ldots, E_k$ are fluent formulas. Intuitively, this law means executing composite actions b is defined as executing a_0 if E_0 holds, then executing a_1 if E_1 holds, ..., then executing a_k if E_k holds. If E_i does not hold, action a_i will be skipped.

A composite action defined in (8) is *acyclic* if there exists a mapping $\lambda : \sigma^{act} \cup \sigma^{comp} \rightarrow \{0, 1, 2, \ldots\}$, such that $\lambda(b) > \lambda(a_i)$ for every $i \in \{0, \ldots, k\}$. In the following we assume composite actions are acyclic to forbid infinite recursion such as b **is** $b; a$.

An action description is acyclic if there exists one mapping λ such that every composite action definition law in the action description is acyclic.

Example 1, continued. We would like to extend the action description D^0 by introducing two composite actions $Fetch(s, l)$, and $Bring(s, l)$:

$$\begin{array}{l} Fetch(s,l) \ \textbf{is} \ Move(l_1) \ \textbf{if} \ Loc(s) = l_1 \wedge Loc(Robot) \neq l_1; Bring(s,l). \\ Bring(s,l) \ \textbf{is} \ Pickup(s); Move(l); Putdown(s). \end{array} \tag{9}$$

Intuitively, $Fetch(s, l)$ means "fetch the object s from some other location to l", and $Bring(s, l)$ means "bring the object s from here to location l".

3.2 Semantics

Given an acyclic action description D^+, let S be the set of composite action definition laws in D^+. For each $r \in S$, an *associate action tuple* $t(r)$ is a pair $\langle b, A\rangle$ where b is the head of r, A is an ordered list over $\sigma^{act} \cup \{\epsilon\}$.each $t(r)$ is defined sequentially on the ordered list of $[r_1, r_2, \ldots, r_m]$, where $r_i \in S$ and $\lambda(head(r_i)) \leq \lambda(head(r_j))$ for $i < j$ such that

- $t(r) = \langle b, [a_0, \ldots, a_k]\rangle$ if for every $i \in \{0, \ldots, k\}$, $a_i \in \sigma^{act}$ of r.
- otherwise, $t(r) = \alpha(\langle b, [a_0, \ldots, a_k]\rangle)$. For all $\alpha(\langle b, A\rangle)$ of the form $\langle b, A'\rangle$, A' is a list obtained from replacing every $a_i \in \sigma^{comp}$ in A with all elements of an corresponding ordered list B_i such that
  - for every $a_i \in \sigma^{comp}$ in A, there is an associate action tuple $t' = \langle a_i, A_i\rangle$ which is already defined for some $r \in S$, and
  - B_i is an ordered list of the same length as A_i, with the first element a_i and the remaining elements ϵ.

For example, the associate action tuples of the two rules in (9) are:

$$t(r_1) = \langle Bring, [Pickup, Move, Putdown]\rangle\,,$$
$$t(r_2) = \alpha(\langle Fetch, [Move, Bring]\rangle) = \langle Fetch, [Move, Bring, \epsilon, \epsilon]\rangle\,.$$

For a composite action definition law r and its associate action tuple $\langle b, [a_0, a_1, \ldots, a_{k'}]\rangle$, $index(b, a_i) = i$ if $a_i \neq \epsilon$.

For instance, in (9), we have

$$index(Fetch, Move) = 0, index(Fetch, Bring) = 1,$$
$$index(Bring, Pickup) = 0, index(Bring, Move) = 1, index(Bring, Putdown) = 2.$$

The intuitive meaning of $index(b, a) = t$ is that a is the t-th action that defines b.

Let k^* be the maximal length of A in the associate action tuples of S, σ_0 be the set of all the actions that defines the composite actions. Intuitively, it is the maximal number of primitive actions expanded by a composite action. e.g $k^* = 4$ for (9), the action $Fetch(s, l)$ can be expanded to 4 primitive actions at most. Since we specify that a composite action is executed in 1 time interval as well as a primitive action, we can only represent its executing trajectory in a different dimension to specify time. As a result, a time interval $(i, i+1)$ is divided by subtime points $i = i.0, \ldots, i.k^* = i+1$ and into k^* subintervals $(i, i.1), (i.1, i.2) \ldots, (i.k^*-1, i+1)$, and fluents have values in all subtime points.

Formally, an extended action description D^+ can be translated into an infinite sequence of causal theories D_m^+ $(m \geq 0)$.

The signature of D_m^+ contains all the symbols occurring in the signature of D_m, and in addition, for each composite action definition law (8), the triples:

- $i.j : a_t$, where $i \in \{0, \ldots, m-1\}$, $j \in \{0, \ldots, k^*-1\}$ and $a_t \in \sigma_0$, and
- $i.j : c$, where $i \in \{0, \ldots, m-1\}$, $j \in \{0, \ldots, k^*\}$ and c is a fluent constant.

The causal theory translated by D_m^+ contains rules of the following parts (assuming $i \in \{0, \ldots m-1\}$, $j \in \{0, \ldots, k^*-1\}$ unless stated otherwise):

1. all rules in D_m except rules obtained from (4). That means the primitive actions are executed in 1 time interval.
2. for every fluent dynamic law (4) and $v \in Dom(c)$, rules
   $i.j{+}1 : c = v \Leftarrow (i.j{+}1 : c = v) \wedge (i.j : c = v)$.
   The rules state that the original inertial laws form (4) are replaced by a group of inertial laws specifying the values of fluents at subtime points.
3. for every $v \in Dom(c)$, the synonymity rules
   $i.0 : c = v \leftrightarrow i : c = v \Leftarrow \top, \qquad i{+}1 : c = v \leftrightarrow i.k^* : c = v \Leftarrow \top.$
   These rules states that every simple fluent has the same value at time point i and $i.0$, as well as $i.k^*$ and $i{+}1$.
4. for each static law (1) and $t \in \{1, \ldots, k^*-1\}$, rules
   $i.t : F \Leftarrow i.t : G.$
   The rules mean that the static laws defining the relationship between fluents at time points are also used for subtime points.
5. for every law (3), rules
   $i.j{+}1 : F \Leftarrow (i.j{+}1 : G) \wedge (i.j : H).$
   These rules say that the action a_j leads to the same effect in the subinterval.
6. for every law (7) where H contains only one action symbol, rules
   $\bot \Leftarrow (i.j : H \wedge F).$
   The rules state that when an action is nonexecutable at some timepoint, it is also nonexecutable at the subtime point with the same condition.
7. for each law (8),

   a. for each fluent dynamic rule (7) and there is at least one action symbol other than a_0 occurs in H, rules
      $\bot \Leftarrow (i : H_{a_0}^b \wedge F),$
      where $H_{a_0}^b$ means to replace every occurrance of a_0 with b in H. The rules say that any action that can not be concurrently executed with the first action of the composite action can also not be executed concurrently with the composite action itself.

b. for $0 \leq n \leq k$, set of rules

$$i : b \Leftarrow i : b \qquad i : \neg b \Leftarrow i : \neg b \qquad i.j : \neg a_t \Leftarrow i.j : \neg a_n$$
$$i.t : b_j \Leftarrow (i : b) \wedge (i.t : E_j) \wedge \mathit{index}(b, b_j) = t$$
$$i.j{+}t : b_j \Leftarrow (i.j : b) \wedge (i.j{+}t : E_j) \wedge \mathit{index}(b, b_j) = t$$
$$\bot \Leftarrow i : a_n \wedge i : b.$$

These rules say that any composite action is exogenous, and its primitive actions can only be "triggered" when the condition E_j is true at the shifted subtime point, which is determined by the value of *index* over the action pair. Also, we state that the composite action can not be executed concurrently with its primitive actions.

8. for $b_m, b_n \in \sigma^{comp}$, rules

$$\bot \Leftarrow i : b_m \wedge i : b_n.$$

The rules state that composite actions cannot be concurrently executed.

4 Properties of Extended Action Description

In this section we investigate the properties of the semantics of extended action descriptions by generalizing the notion of using a transition diagram to characterize the model of an action description proposed in [8]. We will identify an interpretation I of a causal theory with the set of atoms that are satisfied by this interpretation, that is to say, with the set of atoms of the form $c = I(c)$. Such a convention allows us to represent a model of an extended action description D_m^+ as

$$\bigcup_{0 \leq i \leq m} i : s_i \cup \bigcup_{0 \leq i \leq m-1} i : e_i \cup \bigcup_{0 \leq i \leq m-1} \Big(\bigcup_{0 \leq j \leq k^*} i.j : s_{i.j} \cup \bigcup_{0 \leq j \leq k^*-1} i.j : \widehat{e}_{i.j} \Big) \qquad (10)$$

where $e_0, \ldots, e_{m-1}$ are interpretations of $\sigma^{act} \cup \sigma^{comp}$, $s_0, \ldots s_m, s_{i.1}, \ldots, s_{i.k}$ are interpretations of σ^{fl}, and $\widehat{e}_{i.0}, \ldots, \widehat{e}_{i.k^*}$ are interpretations of σ_0.

A *state* is an interpretation s of σ^{fl} such that $0 : s$ is a model of D_0^+. States are vertexes of the transition diagram represented by D^+.

The transitions are defined by models of D_1^+, a model of D_1^+ can be represented in (10) with $m = 1$.

An *explicit transition* is a triple $\langle s, e, s' \rangle$ where s and s' are interpretations of σ^{fl} and e is an interpretation of $\sigma^{act} \cup \sigma^{comp}$ such that $(0 : s) \cup (0 : e) \cup (1 : s')$ belongs to a model of D_1^+. If for some $b \in \sigma^{comp}$, $e(b) = \mathbf{t}$, then $\langle s, e, s' \rangle$ is called a *composite transition*, otherwise it is called a *simple transition*.

An *elaboration* is a tuple of the form $\langle s, \widehat{e}_0, s_1, \ldots, s_{k^*}, \widehat{e}_{k^*}, s' \rangle$, where $\widehat{e}_i$ is an interpretation of σ_0 and s_i is an interpretation of σ^{fl}, such that

$$(0 : s) \cup (0.0 : \widehat{e}_0) \cup (0.1 : s_1) \cup \ldots \cup (0.k^*{-}1 : s_{k^*-1}) \cup (0.k^*{-}1 : \widehat{e}_{k^*-1}) \cup (1 : s')$$

belongs to a model of D_1^+. An elaboration can be seen as a list of k^* triples $\langle s, \widehat{e}_0, s_1 \rangle, \ldots, \langle s_{k^*-1}, \widehat{e}_{k^*-1}, s' \rangle$. Each of the triples is called an *implicit transition*. If $\widehat{e}_j(a_j) = \mathbf{f}$ for any a_j occurring in (8) for $j \in \{0, \ldots, k\}$, the elaboration is called a *trivial elaboration for* b. The edge of the transition diagram of D^+ are the transitions in the models of D_1^+.

The above definition implicitly relies on the following properties of transitions.

▶ Proposition 1. For any explicit transition $\langle s, e, s' \rangle$ or implicit transition $\langle s, \widehat{e}_i, s' \rangle$, s and s' are states.

This proposition is a generalization of Proposition 7 in [8]. Again, the validity of this proposition depends on the fact that statically determined fluents are not allow to occur in the head of a fluent dynamic law (2).

To relate the model of the causal theory obtained from an extended action description, Proposition 8 of [8] is generalized to include composite transitions and elaborations.

▶ Proposition 2. For any $m > 0$, an interpretation (10) on the signature of D^+_m is a model of D^+_m iff for $0 \le i \le m-1$ each triple $\langle s_i, e, s_{i+1}\rangle$ is an explicit transition, and each tuple $\langle s_i, \widehat{e}_i, s_{i.1}, \ldots, s_{i.k^*-1}, \widehat{e}_{i.k^*-1}, s_{i+1}\rangle$ is an elaboration.

Proposition 1 and Proposition 2 allow us to represent an extended action description as a transition graph.

Now we investigate the soundness of the new language. Following [3], for action description D and D' such that the signature of D is a part of the signature of D', D is a *residue* of D' if restricting the states and events of the transition system for D' to the signature of D establishes an isomorphism between the transition systems for D' and D.

▶ Proposition 3. Let D be an action description of a signature σ and b be a constant such that $b \notin \sigma$. If D' is an action description of the signature $\sigma \cup \{b\}$ obtained from D by adding a composite action definition law of b in terms of σ, then D is a residue of D'.

For instance, in the simple robotic domain, the transition system represented by $(D^0)^+$ is isomorphic to the transition system represented by D^0, by restricting the events of the transition system for $(D^0)^+$ to the action constants other than $Fetch(s,l)$, $Bring(s,l)$.

In addition to showing that an extended action description inherits all "good" things from the original action description, we also show that it doesn't introduce anything "bad": a primitive action a_j, if executed at subtime point, is the exact simulation of the action a_j executed at some time point as a primitive action, their transitions are in 1-1 correspondence.

▶ Proposition 4. Each implicit transition $\langle s, \widehat{e}, s'\rangle$ of D^+ corresponds to a transition $\langle s, e, s'\rangle$ of D.

Based on this proposition, it is easy to see that an elaboration in D^+ corresponds to a path of length k^* in the transition diagram of D. Figure 2 shows the transitions of a model of $(D^0_1)^+$, where the implicit transitions are represented as dashed arrows. It can be seen that every implicit transition corresponds to a transition in D^0, as shown in Figure 1.

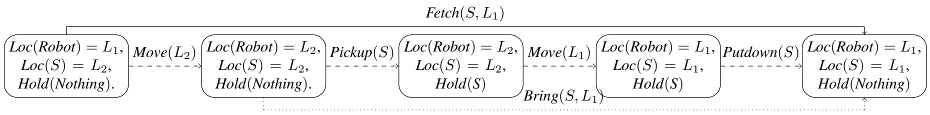

Figure 2 A model of $(D^0_1)^+$ represented as transitions.

5 Experiments—KeJia's Domain

5.1 Formalizing and Reasoning with Composite Actions

In this section, we use composite actions to formalize the domain of the robot KeJia [2]. The robot has a manipulator that can operate various kinds of appliances. The actions that he can perform include $Move(l)$, $Pickup(s)$, $Putdown(s)$, $Open(m)$, $Close(m)$, $Putin(s,m)$, $Takeout(s,m)$, $Start(m)$. Typical scenarios include fetching objects from different places according to the requests of humans and doing other housework such as heating the food with the microwave oven. In addition to do usual task planning, KeJia can acquire knowledge from either human user or textual materials to enrich its knowledge base and planning abilities. For instance, when KeJia is asked to heat the food with microwave oven while he doesn't know how to use the appliance, he can either try to download microwave manuals from internet, did textual analysis to extract instructions, or ask help from humans.[3]

[3] A video of using microwave is at `http://wrighteagle.org/en/demo/ServiceRobot_oven.php`

The instructions of using many household appliances is usually acquired from either textual manuals or humans. The structure of these instructions are usually quite similar, such as "first put the object into the machine, then close the door of the machine, and start the machine, after a while, open the door, finally take out the object from the machine". Instructions of this kind can be converted to composite action definition law by KeJia's natural language understanding module:

$$Use(o,m) \textbf{ is } Putin(o,m); Close(m); Start(m); Open(m); Takeout(o,m).$$

Therefore, heating food with a microwave oven and washing clothes with a washer can be formalized by refering to the knowledge of using the machine as:

$$Heat(f) \textbf{ is } Move(l) \textbf{ if } Loc(Microwave) = l \wedge Loc(Robot) \neq l; Use(f, Microwave).$$

$$Wash(c) \textbf{ is } Move(l) \textbf{ if } Loc(Washer) = l \wedge Loc(Robot) \neq l; Use(c, Washer).$$

These laws are added into the knowledge base incrementally without modifying any other parts in the knowledge base, due to the feature of elaboration tolerance of the formalism. Composite actions make it easier for a robot to gain useful procedural knowledge in many ways, such as oral instructions, or information from internet. More generally, the actions can be defined by referring to actions in a general-purpose library.

A complete formalization of the domain is available at `http://wrighteagle.org/kejiaexp/`. In the following we assume four places (l_1, l_2, l_3, l_4).

Prediction. *Initially, the robot is at l_2, the popcorn is at l_3 and not heated, the microwave oven is at l_2 with the door open, the washer is at l_4 and the door is closed, and the milk is in the robot's hand. The robot heats the milk with the microwave oven, and then put to milk into her plate. Does it follow that in the resulting state, the robot, the milk and the microwave oven are at the same location?*

To solve this problem, we add the following query rules into the causal theory

```
:- query
maxstep :: 2;
0:loc(robot)=l2, loc(microwave)=l2, loc(popcorn)=l3, -heated(popcorn),
  -heated(milk), dooropen(microwave), loc(washer)=l4, doorclosed(washer),
  inside(hand)=milk, heat(milk,microwave);
1:putintoplate(milk).
2:loc(robot) \= loc(milk) ++ loc(milk) \= loc(microwave).
```

The extended cplus2asp return "`UNSATISFIABLE`", indicating that at time 2, the robot is at the same location with the milk and the microwave.

Planning. *Given the same initial state as above, find a plan within 10 steps so that the milk and the popcorn are both heated by the robot.*

When the corresponding query is specified, one of the answer sets returned by the extended cplus2asp contains atoms:

```
0:heat(milk), 0.1:use(milk,microwave), 0.1:putin(milk,microwave),
0.2:close(microwave), 0.3:operate(microwave), 0.4:open(microwave),
0.5:takeout(milk,microwave), 1:toplate(milk), 2:move(l3), 3:pickup(popcorn),
4:heat(popcorn), 4.0:move(l2), 4.1:use(popcorn,microwave),
4.1:putin(popcorn,microwave), 4.2:close(microwave), 4.3:operate(microwave),
4.4:open(microwave), 4.5:takeout(popcorn,microwave).
```

We have three observations. First, composite actions occur as building blocks of the plan, for example, we see `0:heat(milk)`, `0.0:use(milk,microwave)`, etc in the result. Second, when a composite action is executed, all details about the executions of the primitive actions in the composite action are also included, for instance, when `0:heat(milk)` is executed, we also have the details `0.0:use(milk,microwave)`, . . . , `0.4:takeout(milk,microwave)`.

Table 1 The results of the *KeJia* domain.

Length of Plans	#Instances	#Time-Outs	Time ratio
≤20	35	0	0.138
21–25	13	0	0.274
26–30	24	0	1.505
31–35	23	3	1.553
36–40	6	2	2.096
41–45	1	1	–

Third, composite actions can have different kinds of execution trajectory, for instance, the execution trajectory of the action `4:heat(popcorn)` has the action `4.0:move(l2)` more than that of `0:heat(milk)`.

5.2 Performance

We test planning performance by two representations of the domain KeJia: a traditional representation $KeJia_1$ without any composite actions, and an extended representation $KeJia_2$ by adding some composite actions into $KeJia_1$. We consider 120 different instances, for every instance, the numbers of locations and objects, the initial states and the goal states are randomly generated. We set the longest acceptable length of a plan for a instance using $KeJia_1$ to 50 and time limit for computing to be 30min[4].

The result is shown in Table 1. There are 18 problems which can be solved by neither representations. We classify the other instances into 6 categories by the length of the plans generated using $KeJia_1$. For each category, the third column shows the number of instances that cannot be computed using $KeJia_1$. The last column shows the average ratio of times on computing a instance using $KeJia_1$ and $KeJia_2$ where time-out instances are excluded. There are no time-outs using $KeJia_2$.

In Table 1, we notice that when the plan length increases from ≤20 to 36–40, the ratio increases simultaneously, especially, when the length of a plan is up to 26-30, the time ratio is always > 1, indicating that the composite actions help improve the efficiency as the complexity of domain tasks increases. The reason the time ratio is < 1 is that there are more rules introduced by composite actions, which may also become overhead of computation. For large domains, the composite actions in the plan contain a lot of consecutive executions of the primitive actions. Making use of composite actions allows the solver iClingo to find the "cumulative effects" at earlier stages of grounding.

Therefore, when the task domain has a "hierarchical structure" such that its plan consists of many consecutive executions of primitive actions which can compose to an action in a different abstraction space, composite actions may be worthwhile and can improve the efficiency.

6 Conclusion

In this paper we introduce composite actions into a fragment of $\mathcal{C}+$. Action description equipped with composite actions leads to a more intuitive and flexible way to formalize action domains by exploiting general-purpose formalization, a step to address the problem of

[4] The detailed representation, instances and logs, as well as the extended Cplus2ASP system can be found at `http://wrighteagle.org/kejiaexp/`

generality, and improve efficiency of reasoning and planning by characterizing the hierarchical structure of the problem domain. Extended action descriptions can be processed by the extended CPLUS2ASP system.

A direct next step is to apply CPLUS2ASP on robot KeJia to solve the real-life problems for real-time computation. In the future, we would like to introduce composite action definition to MAD, where modular actions can be defined as special case or sequential executions of actions, by referring to a general-purpose library. Composite actions should also be defined on $\mathcal{C}+$ in its full generality.

Acknowledgements. This work is supported by the National Hi-Tech Project of China under grant 2008AA01Z150 and the Natural Science Foundation of China under grant 60745002 and 61175057, as well as the USTC Key Direction Project and the USTC 985 Project. The authors are grateful to Vladimir Lifschitz, Michael Gelfond, Alfredo Gabaldon, Daniela Inclezan and the anonymous reviewers for their constructive comments and suggestions.

References

1 Michael Casolary and Joohyung Lee. Representing the language of the causal calculator in answer set programming. In *Technical Communications of the 27th International Conference on Logic Programming (ICLP 2011)*, pages 51–61, 2011.

2 X. Chen, J. Ji, J. Jiang, G. Jin, F. Wang, and J. Xie. Developing high-level cognitive functions for service robots. In *Proc. of 9th Int. Conf. on Autonomous Agents and Multi-agent Systems (AAMAS 2010)*, 2010.

3 Selim T. Erdoğan and Vladimir Lifschitz. Actions as special cases. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 377–387, 2006.

4 Kutluhan Erol, James A. Hendler, and Dana S. Nau. Htn planning: Complexity and expressivity. In *AAAI*, pages 1123–1128, 1994.

5 Richard Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.

6 Michael Gelfond and Daniela Inclezan. Yet another modular action language. In *Proceedings of the Second International Workshop on Software Engineering for Answer Set Programming*, pages 64–78, 2009.

7 Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 3:195–210, 1998.

8 Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1–2):49–104, 2004.

9 Daniela Inclezan and Michael Gelfond. Representing Biological Processes in Modular Action Language ALM. In *Proceedings of the 2011 AAAI Spring Symposium on Formalizing Commonsense*, pages 49–55. AAAI Press, 2011.

10 Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *J. Log. Program.*, 31(1-3):59–83, 1997.

11 Vladimir Lifschitz and Wanwan Ren. A modular action description language. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 853–859, 2006.

12 John McCarthy. Generality in Artificial Intelligence. *Communications of the ACM*, 30(12):1030–1035, 1987. Reproduced in [13].

13 John McCarthy. *Formalizing Common Sense: Papers by John McCarthy.* Ablex, Norwood, NJ, 1990.

14 John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

15 Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. In *Proceedings of the 3rd international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., 1973.

16 Murray Shanahan. Event calculus planning revisited. In *Proceedings 4th European Conference on Planning (ECP 97), Springer Lecture Notes in Artificial Intelligence no. 1348*, pages 390–402. Springer, 1997.

17 Tran Cao Son, Chitta Baral, and Sheila A. McIlraith. Planning with different forms of domain-dependent control knowledge - an answer set programming approach. In *LPNMR*, pages 226–239, 2001.