The Description Logic of Tasks[1]

Zhang Hui Li Sikun

College of Computer Science,
National University of Defense Technology,
Changsha, China
qd_zhanghui@163.com

Abstract. The logic of tasks can be used in AI as an alternative logic of planning and action, its main appeal is that it is immune to the frame problem and the knowledge preconditions problem other plan logics confront. A drawback the present logic of tasks has is that it is nondecidable (semidecidable only). This paper extends the logic of tasks to enable the task description by adapting description structure into it. A formal system DTL, which is sound, complete and decidable, for agent abilities specification and accomplishablity of tasks judgment is proposed.

1 Introduction

The semantic used in the logic of tasks can claim to be a formalization of the resource philosophy associated with linear logic[4,5,6], if resources are understood as agents carrying out tasks. The formalism may also have a potential to be used in AI as an alternative logic of planning and action and its main appeal is that it is immune to the frame problem and the knowledge preconditions problem other plan logics confront [2,10]. A drawback the present logic of tasks has is that it is nondecidable (semidecidable only).

Motivated by the success of description logic[7.8.9], in this paper we present a decidable logic system, the description logic of tasks that enable task description by adapting the description structure into the logic of tasks. In our paper the task may have parameter e.g. we can use C(x) denoting the task to clean x, x can be either room or lawn (room and lawn are constant). The expression $\alpha\rightarrow\beta$ in our paper means that the accomplishment of task α is the condition to accomplish the task β, such as F(rake)$\rightarrow$C(lawn) and F(mop)$\rightarrow$C(room) express that the cleaner can clean the room if be given a mop and can clean the lawn if be given a rake respectively.

A characteristic feature of description languages is their ability to represent other kinds of relationships, expressed by role, beyond IS-A relationships. The role has what is called a "value restriction," denoted by the label $\forall$R. which expresses a limitation on the range of types of objects that can fill that role. In this paper we use role to express the relation between objects. For example R(room,mop) means that

[1] Supported by the National Grand Fundamental Research 973 Program (2002CB312105) and 863 program (2004AA115130) of China.

© A. Gelbukh, R. Monroy. (Eds.)
Advances in Artificial Intelligence Theory
Research on Computing Science 16, 2005, pp. 13-22

there is certain relation between room and mop (a mop is the necessity tool to clean a room). The role has what is called "value restriction" denoted by the label ∀R. too, which also expresses the limitation on the range of object that can fill the role. For example, the expression ∀R(room,y).F(y)→C(room) means that if be given all necessity tools for cleaning a room the cleaner can accomplish the task of cleaning it.

In addition, we use predict to express the limitation on the range of value of the parameter of tasks. For example, if we used predict P to express whether the object is in charged of by the cleaner or not, then ∀P(x).(∀R(x,y).F(y)→C(x)) expresses that for every object *x* that the cleaner in charge of, if be given all necessity tools the cleaner can accomplish the task of cleaning *x*.

In remain of the paper we will give out the syntax and the semantic of the description logic of tasks.

2 Syntax and Semantic of the Description Logic of Tasks

2.1. Syntax

We fix a set of expressions that we call atom task names $\{A,A_1,A_2\ldots\}$, with each of which is associated a natural number called its arity, a set of predict name$\{P,P_1, P_{2,}\ldots\}$, with each of which also is associated a natural number called its arity, a set of role names $\{ R,R_1,R_2,\ldots\}$.

We also fix an infinite set $C=\{c_0, c_1, \ldots\}$ of constants and an infinite set $X=\{x_0, x_1, x_2, \ldots\}$ of variables. We refer to elements of $C \cup X$ as terms.

Definition 2.1 (domain knowledge axioms**)** Domain knowledge axioms are defined as follows:

1. Let *a, b* be two constants and R a role name, then R(a,b) is a domain knowledge axiom;
2. Let $c_1, c_2, \ldots, c_n$ be constants and P an n-ary predict then $P(c_1,c_2,\ldots,c_n)$ is a domain knowledge axiom.

Definition2.2 (task formula) Task formulas are elements of the smallest class of expressions such that:

1. If A is a n-ary atom task name, $t_1,t_2,\ldots,t_n$ are terms, then $A(t_1,t_2,\ldots,t_n)$ is task formula, be called an atomic task;
2. if P is a n-ary predict, α is a formula, $t_1,t_2,\ldots,t_n$ are terms, then $\forall P(t_1,t_2,\ldots,t_n).\alpha$ and $\exists P(t_1,t_2,\ldots,t_n).\ \alpha$ are both task formulas;
3. If α is a formula, R is an role, *t* is a term and *y* is a variable, then $\forall R(t,y).\alpha$ and $\exists R(t,y).\alpha$ are both task formulas;
4. if α and β are task formulas, then so are $\alpha\wedge\beta,\alpha\vee\beta$ and $\alpha\rightarrow\beta$;
5. if α and β are task formulas, then so is $\alpha\sqcap\beta$;
6. if α is a formula, *x* is a variable, then $\Pi x\alpha$ is a task formula.

$\sqcap$ and Π are called additive operators, or additives. The additive complexity of a formula is the number of occurrences of additives in that formula. The task formula whose complexity is zero is called primitive task formula and task formula that does not contain free variable is call to be closed.

Except the sets mentioned above, the description logic of tasks also have two other sets, the domain knowledge and the capability specification, defined as follows:

Definition 2.3 (domain knowledge and capability specification) The Domain knowledge is a finite set of domain knowledge axioms. The capability specification is a finite set of primitive task formulas.

2.2 Semantic

Let $t_1,t_2,\ldots,t_n$ be terms, an assignment of $(t_1,t_2,\ldots,t_n)$ is a n-tuple $(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)$ such that $c_i \in C$ and if t_i is constant then $c_i=t_i$ for all i($1\le i\le n$). Let α be a primitive task formula, $\alpha(t_1/c_1,t_2/c_2,\ldots, t_n/c_n)$ is the result of replacing all free occurrences of t_i in α by c_i respectively (i=1,2,…,n) if t_i is a variable.

We consider interpretation I that consist of a non-empty set Δ^I (the domain of the interpretation) and an interpretation function $\cdot^I$, which assigns to every predict P a set $P^I \subseteq (\Delta^I)^n$, to every role R a binary relation $R^I \subseteq \Delta^I \times \Delta^I$, to every closed atomic task an element of $\{0,1\}$. To give out the value of all closed primitive task formulas, $\cdot^I$ is extended as follows:

1. $(\neg\alpha)^I=\begin{cases}0 & \text{if } \alpha^I=1\\ 1 & \text{else}\end{cases}$;

2. $\forall P(t_1,t_2,\ldots,t_n).\alpha=$

$\begin{cases}1 & \text{if } (\alpha(t_1/c_1,t_2/c_2,\ldots,t_n/c_n))^I=1 \text{ for every assignment } (t_1/c_1,t_2/c_2,\ldots,t_n/c_n) \text{ such that } (c_1,c_2,\ldots,c_n)\in P^I\\ 0 & \text{else}\end{cases}$;

3. $\exists P(t_1,t_2,\ldots,t_n).\alpha=$

$\begin{cases}1 & \text{if there is an assignment } (t_1/c_1,t_2/c_2,\ldots,t_n/c_n) \text{ such that } (c_1,c_2,\ldots,c_n)\in P^I \text{ and } (\alpha(t_1/c_1,t_2/c_2,\ldots,t_n/c_n))^I=1\\ 0 & \text{else}\end{cases}$;

4. $(\forall R(a,y).\alpha)^I=\begin{cases}1 & \text{if } (\alpha(y/b))^I=1 \text{ for every } b \text{ such that } (a,b)\in R^I\\ 0 & \text{else}\end{cases}$;

5. $(\exists R(a,y).\alpha)^I=\begin{cases}1 & \text{if there exist a constant } b \text{ such that } (a,b)\in R^I \text{ and } (\alpha(y/b))^I=1\\ 0 & \text{else}\end{cases}$;

6. $(\alpha\wedge\beta)^I=\begin{cases}1 & \text{if } \alpha^I=1 \text{ and } \beta^I=1\\ 0 & \text{else}\end{cases}$;

7. $(\alpha\vee\beta)^I=\begin{cases}1 & \text{if } \alpha^I=1 \text{ or } \beta^I=1\\ 0 & \text{else}\end{cases}$;

8. $(\alpha\rightarrow\beta)^I=\begin{cases}0 & \text{if } \alpha^I=1 \text{ and } \beta^I=0\\ 1 & \text{else}\end{cases}$;

We say an interpretation I is coincident with the domain knowledge if it satisfies conditions follows:

1. For every n-tuple of constants $(c_1,c_2,\ldots,c_n)$, $(c_1,c_2,\ldots,c_n)\in P^I$ if and only if $P(c_1,c_2,\ldots,c_n)$ is in the domain knowledge;
2. for every 2-tuple of constants (a,b), $(a,b)\in R^I$ if and only if $R(a,b)$ is in the domain knowledge.

Let Γ be a set of primitive task formulas, $\{x_1,x_2,\ldots,x_n\}$ be the set of all free variable that occur in the task formulas in Γ. An interpretation I and an assignment $(x_1/c_1,x_2/c_2,\ldots,x_n/c_n)$ is said satisfy Γ if $(\alpha(x_1/c_1,x_2/c_2,\ldots,x_n/c_n))^I=1$ for every formula α in Γ. A task formula set Γ is said be not satisfiable if there is no interpretation I and assignment $(x_1/c_1,x_2/c_2,\ldots,x_n/c_n)$ satisfy it, else be satisfiable.

In remain of the paper we only consider the interpretation that is coincident with the domain knowledge and assumes that the ability specification is satisfiable.

Definition 2.5(accomplishability of primitive tasks) Let Γ be the ability specification and α a primitive task. Let $\{x_1,x_2,\ldots,x_n\}$ be the set of all free variables that appear in the task formula in the set $\Gamma\cup\{\alpha\}$. We say that α is accomplishable if for every interpretation I and every assignment $(x_1/c_1,x_2/c_2,\ldots,x_n/c_n)$ of $(x_1,x_2,\ldots,x_n)$ that satisfy Γ, we have $(\alpha(x_1/c_1,x_2/c_2,\ldots,x_n/c_n))^I=1$.

Now we will give out the concept of accomplishable task. The concepts of strategy and realization used are same as those in [2].

Observe that development preserves the basic structure of the formula. I.e.

1. assume $\alpha_0=\forall P(t_1,t_2,\ldots,t_n).\beta$ (or $\alpha_0=\exists P(t_1,t_2,\ldots,t_n).\beta$), then for every realization $\mathcal{R}=<\alpha_0,\alpha_1,\alpha_2,\ldots,\alpha_m>$ of α_0, α_i must has the form of $\forall P(t_1,t_2,\ldots,t_n).\beta_i$ (or $\exists P(t_1,t_2,\ldots,t_n).\beta_i$)) $(1\le i\le n)$. For every assignment $(t_1/c_1,\ t_2/c_2,\ldots,t_n/c_n)$ the sequence of $\beta_i(t_1/c_1,\ t_2/c_2,\ldots,t_n/c_n)$, denoted by $[P:\ t_1/c_1,\ t_2/c_2,\ldots,t_n/c_n]\ \mathcal{R}$, is an realization of $\beta(\ t_1/c_1,\ t_2/c_2,\ldots,t_n/c_n)$.
2. Assume $\alpha_0=\forall R(t,y).\beta$ (or $\alpha_0=\exists R(t,y).\beta$), then for every realization $\mathcal{R}=<\alpha_0,\alpha_1,\alpha_2,\ldots,\alpha_m>$ of α_0, α_i must has the form of $\forall R(t,y).\beta_i$ (or $\exists R(t,y).\beta_i$) $(1\le i\le n)$ and for every constant b the sequence of $\beta_i(y/b)$, denoted by $[R:y/b]\mathcal{R}$, is an realization of $\beta(y/b)$.
3. Assume $\alpha_0=\beta\wedge\gamma$(or $\alpha_0=\beta\vee\gamma$), then for every realization $\mathcal{R}=<\alpha_0,\alpha_1,\alpha_2,\ldots,\alpha_m>$ of α_0, α_i can be expressed as $\beta_i\wedge\gamma_i$ (or $\beta_i\vee\gamma_i$) $(1\le i\le n)$ such that $<\beta_0,\beta_1,\beta_2,\ldots,\beta_m>$ and $<\gamma_0,\gamma_1,\gamma_2,\ldots,\gamma_m>$, denoted by $p(\mathcal{R})$ and $r(\mathcal{R})$, are realizations of β and γ respectively.
4. Assume $\alpha_0=\beta\to\gamma$, then for every realization $\mathcal{R}=<\alpha_0,\alpha_1,\alpha_2,\ldots,\alpha_m>$ of α_0, α_i must has the form of $\beta_i\to\gamma_i$ $(1\le i\le n)$. $<\beta_0,\beta_1,\beta_2,\ldots,\beta_m>$ and $<\gamma_0,\gamma_1,\gamma_2,\ldots,\gamma_m>$ denoted by $a(\mathcal{R})$ and $c(\mathcal{R})$, are realizations of β and γ respectively.

Definition 2.6 We say that an realization $\mathcal{R}=<\alpha_0,\ \alpha_1,\ \alpha_2,\ldots,\ \alpha_m>$ of a task formula α_0 is successful if one of the following conditions holds:

1. If α_0 is an atomic task formula, or $\alpha_0=\neg\alpha$ and α is an atomic task formula(both imply m=0), and α_0 is accomplishable;
2. $\alpha_0=\forall P(t_1,t_2,\ldots,t_n).\beta$ and $[P:\ t_1/c_1,\ t_2/c_2,\ldots,t_n/c_n]\ \mathcal{R}$ is successful for every assignment $(t_1/c_1,t_2/c_2,\ldots,t_n/c_n)$ such that $P(c_1,c_2,\ldots,c_n)$.
3. $\alpha_0=\exists P(t_1,t_2,\ldots,t_n).\beta$ and there is an assignment $(t_1/c_1,t_2/c_2,\ldots,t_n/c_n)$ such that $P(c_1,c_2,\ldots,c_n)$ and $[P:t_1/c_1,t_2/c_2,\ldots,t_n/c_n]\ \mathcal{R}$ is successful;

4. $\alpha_0=\forall R(t,y).\beta$ and $[R:y/b]\ \mathcal{R}$ is successful for every constant b such that $R(t,b)$;
5. $\alpha_0=\exists R(t,y).\beta$ and there is a constant b such that $R(t,b)$ and $[R:y/b]\ \mathcal{R}$ is successful ;
6. $\alpha_0=\beta\wedge\gamma$ and $p(\mathcal{R})$ and $r(\mathcal{R})$ both are successful;
7. $\alpha_0=\beta\vee\gamma$ and either $p(\mathcal{R})$ or $r(\mathcal{R})$ is successful;
8. $\alpha_0=\beta\rightarrow\gamma$ and $c(\mathcal{R})$ if successful if $a(\mathcal{R})$ is successful.
9. α_0 is an additive formula and either $m=0$ or $m\geq 1$ and $<\alpha_1, \alpha_2,\ldots, \alpha_m>$ is successful.

Definition 2.7 (accomplishability of tasks) Let α be a task formula. If there is an action strategy *f* such that every realization of α with *f* is successful then we say that α is accomplishable.

3 Accomplishablity judgment of primitive tasks

In this section the method for accomplishablity judgment of primitive tasks is presented. The work in this section is inspired and highly related to F. Baader and P. Hanschke's work for the consistent judgment of description logic formula set [9].

For a task formula α, it is accomplishable if and only if that $\Gamma\cup\{\neg\alpha\}$ is not satisfiable. So here we need only give out the method for satisfiability judgment of finite set of primitive task formulas.

First, we assume that the task formula in the task formula set, denoted by S_1, does not has R(1) type free variable. A variable of task formula α is said to be R(1) type if *x* is a free variable and α has sub formula has the form $\forall R(x,y).\beta$ or $\exists R(x,y).\beta$. In fact if there is a R(1) type free variables *x* in S_1 , the number of constant *a* with a constant *b* such that R(a,b) is in domain knowledge is finite, assume $\{a_1,a_2,\ldots,a_n\}$ is the set of all such *a,* then S_1 is satisfiable if and only if at least one of the sets $S_1(x/a_i)$ is satisfiable.

Furthermore, we assume that every formula in S_1 is in negation normal form, i.e. $\neg$ occurs only immediately before the atom task name, in fact if a task formula in S_1 is not in negation normal form it can be transformed in to an equivalent one.

Definition 3.1(transformation rules) Let M be a finite set of finite primitive task formula sets. The following rule will replace one of elements S of M by another set (or several other sets) of primitive task formulas.

Rule 1: If $\alpha(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)\in S$ for every assignment $(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)$ such that $P(c_1,c_2,\ldots,c_n)$, $\forall P(t_1,t_2,\ldots,t_n).\alpha$ is a sub formula of one element of S and $\forall P(t_1,t_2,\ldots,t_n).\alpha$ is not in S, then replace S by $S'=S\cup\{\forall P(t_1,t_2,\ldots,t_n).\alpha\}$;

Rule 1′: if $P(c_1,c_2,\ldots,c_n)$, $\forall P(t_1,t_2,\ldots,t_n).\alpha\in S$ and $\alpha(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)$ is not in S, then replace S by $S'=S\cup\{\alpha(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)\}$;

Rule 2: if there is an assignment $(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)$ of $(t_1,t_2,\ldots,t_n)$ such that $P(c_1,c_2,\ldots,c_n)$ and $\alpha(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)\in S$, $\exists P(t_1,t_2,\ldots,t_n).\alpha$ is a sub formula of one element of S but $\exists P(t_1,t_2,\ldots,t_n).\alpha$ is not in S, then replace S by $S'=S\cup\{\exists P(t_1,t_2,\ldots,t_n).\alpha\}$

Rule 2′: if $\exists P(t_1,t_2,\ldots,t_n).\alpha\in S$, but there is no assignments $(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)$ such that $P(c_1,c_2,\ldots,c_n)$ and $\alpha(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)\in S$, then replace S by sets $S\cup$

$\{\alpha(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)\}$ where $(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)$ are all assignments of $(t_1,t_2,\ldots,t_n)$ such that $P(c_1,c_2,\ldots,c_n)$;

Rule 3: if $\alpha(y/b)\in$ S for every constant b such that $R(a,b)$, $\forall R(a,y).\alpha$ is a sub formula of one element of S and $\forall R(a,y).\ \alpha$ does not in S, then replace S by $S'=S\cup\{\forall R(a,y).\alpha\ \}$;

Rule 3′: if $\forall R(a,y).\alpha \in$ S, but there is a constant b such that $R(a,b)$ and $\alpha(y/b)$ is not in S, then replace S by $S'=S\cup\{\alpha(y/b)\}$;

Rule 4: if there is a constant b such that $R(a,b)$ and $\alpha(y/b)\in$ S, $\exists R(a,y).\alpha$ is a sub formula of one element of S but $\exists R(a,y).\alpha$ is not in S, then replace S by $S'=S\cup\{\exists R(a,y).\alpha\}$;

Rule 4′: if $\exists R(a,y).\alpha \in$ S, but there is not a constant b such that $R(a,b)$ and $\alpha(y/b)\in$ S, assume the set of all constant b such that $R(a,b)$ is $\{b_1,b_2,\ldots,b_n\}$, then replace S by n sets $S_i=S\cup\{\alpha(y/b_i)\}(i=1,2,\ldots,n)$;

Rule 5: if $\alpha\in$ S and $\beta\in$ S, $\alpha\wedge\beta$ is an sub formula of one element of S, but $\alpha\wedge\beta$ is not in S, then replace S by $S'=S\cup\{\alpha\wedge\beta\}$;

Rule 5′: if $\alpha\wedge\beta\in$ S but α and β are not both in S, then replace S by $S'=S\cup\{\alpha,\beta\}$;

Rule 6:if $\alpha\in$ S or $\beta\in$ S, $\alpha\vee\beta$ is an sub formula of one element of S and $\alpha\vee\beta$ is not in S, then replace S by $S'=S\cup\{\alpha\vee\beta\}$;

Rule 6′: if $\alpha\vee\beta\in$ S but none of the α and β is in S, then replace S by $S'=S\cup\{\alpha\}$ and $S''=S\cup\{\beta\}$;

Rule 7: if$\alpha\rightarrow\beta\in$ S and $\alpha\in$ S but β in not in S, then replace S by $S'=S\cup\{\beta\}$;

Definition 3.2 (clash) We say that a primitive formula set S has a clash if α and $\neg\alpha$ are both in S for certain task formula α.

To test whether a finite set of primitive task formulas S_1 is satisfiable or not, we set $M_1=\{S_1\}$ and apply the transformation rule in the definition 3.3 to M as long as possible then we finally end up with a complete Set M_r i.e. a set to which no rule are applicable. The initial set S_1 is satisfiable if and only if there is a set in M_r does not contain a clash (see the follow part of this section for a proof). The test procedure can be defined in pseudo programming language as follows:

Algorithm 3.1 (satisfiability judgment) The following procedure takes a finite set of primitive formulas as an argument and checks whether it is satisfiable or not.

```
Define   procedure check-satisfiable(S1)
    r:=1;
    M1:={S1}
    While 'a transformation rule is applicable to Mr'
        do
            r:=r+1
            Mr:=apply-a-transformation rule(Mr-1)
        od
    if 'there is an S ∈ Mr that does not contain a clash'
        then return YES;
        else return NO.
```

For example, let {P(room), R(room, mop)} be the domain knowledge. P(room) means that the cleaner is in charge of the room. R(room, mop) means that mop is the necessary tool to clean the room. If the ability specification is

{$\forall P(x).(\forall R(x,y).F(y)\rightarrow C(x))$, F(mop)}, where $\forall P(x).(\forall R(x,y).F(y)\rightarrow C(x))$ means that for every object x that the cleaner is in charge of, if be given all necessity tools the cleaner can accomplish the task of cleaning it. F(mop) means that the keeper can give the cleaner a mop. To judge whether C(room) is accomplishable or not, i.e. whether the cleaner can clean the room or not, we need to judge whether the set S_1={$\forall P(x).(\forall R(x,y).F(y)\rightarrow C(x))$, F(mop),$\neg$C(room)} is satisfiable or not. The set M_i generated in the testing process are:

M_1={{$\forall P(x).(\forall R(x,y).F(y)\rightarrow C(x))$, F(mop), $\neg$ C(room)}};

M_2={{$\forall P(x).(\forall R(x,y).F(y)\rightarrow C(x))$, F(mop), $\neg$C(room),$\forall R(room,y).F(y)\rightarrow C(room)$}}

(by Rule 1 , $\forall P(x).(\forall R(x,y).F(y)\rightarrow C(x)$)and P(room));

M_3={{$\forall P(x).(\forall R(x,y)F(y)\rightarrow C(x))$, F(mop), $\neg$C(room)}, $\forall R(room,y).F(y)\rightarrow C(room)$, $\forall R$ (room,y).F(y) }

(by Rule 3 , R(room, mop) and F(mop));

M_4={{$\forall P(x).(\forall R(x,y).F(y)\rightarrow C(x))$, F(mop), $\neg$C(room)}, $\forall R(room,y).F(y)\rightarrow C(room)$, $\forall R(room,y).F(y)$, C(room)}}

(by Rule 7, $\forall R(room,y).F(y)\rightarrow C(room)$ and $\forall R(room,y).F(y)$).

Then S_1 is not satisfiable because there is not a set in M_4 that does not contain a clash. So we know that C(room) is accomplishable.

For a primitive task formula β the length of β, denoted by $|\beta|$, is inductively defined as:

1. If β is an atomic task formula or $\beta=\neg\alpha$ and α is an atomic task formula, then $|\beta|=1$;
2. if $\beta=\forall P(t_1,t_2,\ldots,t_n).\alpha$ or $\beta=\exists P(t_1,t_2,\ldots,t_n).\alpha$ then $|\beta|=|\alpha|+1$;
3. if $\beta=\forall R(t,y).\alpha$ or $\beta=\exists R(t,y).\alpha$, then $|\beta|=|\alpha|+1$;
4. if $\beta=\alpha\wedge\gamma$ or $\beta=\alpha\vee\gamma$, then $|\beta|=|\alpha|+|\gamma|$;
5. if $\beta=\alpha\rightarrow\gamma$, then $|\beta|=|\alpha|+|\gamma|$.

We call a task formula α that has the form $\alpha=\beta\wedge\gamma$ is a $\wedge$-task. The maximal $\wedge$-expression of a $\wedge$-task is an express $\alpha_1\wedge\alpha_2\ldots\wedge\alpha_n$ such that α_i is no long a $\wedge$-task for every i($1\leq i\leq n$). If α is a $\wedge$-task and its maximal $\wedge$-expression is $\alpha_1\wedge\alpha_2\ldots\wedge\alpha_n$, then the $\wedge$-length of α is n. The concept of $\vee$-task and the $\vee$-length of a $\vee$-task, the concept of $\rightarrow$-task and the $\rightarrow$-length of a $\rightarrow$-task all can be defined analogously.

Proposition 3.1 The algorithm 3.1 can always compute a complete set M_r in finite time and the initial set S_1 is not satisfiable if and only all set in M_r contain a clash.

Proof: Because the number of different assignments $(c_1/t_1,c_2/t_2,\ldots,c_n/t_n)$ of $(x_1,x_2,\ldots,x_n)$ such that $P(c_1,c_2,\ldots,c_n)$ for every predict P that occurs in domain knowledge, the number of different sub formulas that has the form $\forall P(t_1,t_2,\ldots,t_n).\alpha$ of elements of S_1, the number of different sub formulas that has the form $\exists P(t_1,t_2,\ldots,t_n).\alpha$ of elements of S_1, the number of different constant b which satisfies $R(a,b)$ for each pair (a,R) of constant and role that occurs in S_1, the number of the sub formulas that has the form $\forall R(t,y).\alpha$ of elements of S_1, the number the sub formulas that has the form $\exists R(t,y).\alpha$ of elements of S_1, the number of the different sub formulas that has the form of $\alpha\wedge\beta$ of elements of S_1 and the maximal $\wedge$-length of them, the number of different sub formula that has the form $\alpha\vee\beta$ of elements of S_1 and the maximal $\vee$-length of them, the number of the different formula that has the form $\alpha\rightarrow\beta$ of the elements of S_1 and the maximal $\rightarrow$-length of them are all finite, then it

can be proved that the rules in definition 3.1 can only be applied for finite times, so the computation process will terminate in finite time.

The second part of the proposition is a consequence of lemma 3.1 and lemma 3.2 bellows, where the notion of contradictory formula set which is syntactic equivalent of not satisfiable formula set is defined by induction on the relation of "descendant". A formula set S occurring in the computation is contradictory with respect to the computation if and only if

1. S does not have a descendant and contains a clash or
2. all descendants of S are contradictory.

Lemma 3.1 If the initial formula set is contradictory with respect to a given computation then it is not satisfiable.

Proof: The proof is by induction on the definition of contradictory with a case analysis according to the transformation rule applied. Assume S_1 is a given set of formulas which is contradictory with respect to a given computation, we will show that it is not satisfiable.

If S_1 does not have a descendant, then it must have a clash. Obviously a set of formulas that have a clash is not satisfiable. For the induction step, assume S is satiafiable, we have to show that the descendant (resp. one of the descendant in the case of rule 2′, rule 4′, and rule 6′) of S is satisfiable too, this will be contradiction to the induction hypothesis, because all descendants of contradictory set are contradictory.

We shall only demonstrate the case of rule 5. The other cases can be treated similarly. Assume that rule 5 is applied to a set, denoted by S_{r-1}, means that there are two formula α and β such that $\alpha\in S_{r-1}$ and $\beta\in S_{r-1}$ and the descendant of S_{r-1}, denoted by S_r, is equal to $S_{r-1} \cup \{\alpha\wedge\beta\}$. If the interpretation I and the assignment $(x_1/c_1,x_2/c_2\ldots,x_n/c_n)$ satisfy S_{r-1}, then we have $(\alpha(x_1/c_1,x_2/c_2\ldots,x_n/c_n))^I=1$ and $(\beta(x_1/c_1,x_2/c_2\ldots,x_n/c_n))^I=1$, thus $((\alpha\wedge\beta)(x_1/c_1,x_2/c_2\ldots,x_n/c_n))^I=1$ by the definition of the semantic of closed task formulas, so I and $(x_1/c_1,x_2/c_2\ldots,x_n/c_n)$ satisfy S_r too.

Lemma 3.2 If the initial formula set is not contradictory with respect to a given computation then it is satisfiable.

Proof: If S_1 is not contradictory then there is a primitive formula set $S\supseteq S_1$ in the complete set M_r such that there is no clash in S. Assume the set of all different free variable occur in S is $\{x_1,x_2,\ldots,x_m\}$, we define an interpretation $I=(\Delta^I,\cdot^I)$ as follows:

Δ^I is the set of all constants. $\cdot^I$ assigns to each constant itself, to each predict P a set $P^I=\{(a_1, a_2, \ldots, a_n)|a_i\in\Delta^I\ (1\le i\le n)$ and $P(a_1, a_2,\ldots, a_n)\}$, to each role R a set $R^I=\{(a,b)|\ a,b\in\Delta^I$ and $R(a,b)\}$. Let $(c_1,c_2,\ldots,c_m)$ be an arbitrary m-tuple of constants, $\cdot^I$ assigns each atomic task $A(a_1,a_2,\ldots,a_j)(j=1,2,3,\ldots)$ an element of $\{0,1\}$ according to following rule:

$$A(a_1,a_2,\ldots,a_j)^I=\begin{cases}0 & \text{if } \neg A(a_1,a_2,\ldots,a_j)\in S(x_1/c_1,x_2/c_2,\ldots,x_m/c_m)\\ 1 & \text{else}\end{cases}$$

We will prove that the interpretation I and the assignment $(c_1/t_1,c_2/t_2,\ldots,c_m/t_m)$ satisfy S, so satisfy S_1 also. We use induction on the length of α.

If $|\alpha|=1$, then α is an atomic task or $\alpha=\neg\beta$ and β is an atomic task. Then $(\alpha(x_1/c_1,x_2/c_2\ldots,x_m/c_m))^I=1$ by the definition of I.

Assume $|\alpha|=k$ ($k>1$), we prove that $(\alpha(x_1/c_1,x_2/c_2\ldots,x_m/c_m))^I=1$. In the case of $\alpha=\forall P(t_1,t_2,\ldots,t_n).\beta$. M_r is complete then we have $\beta(t_1/a_1,t_2/a_2,\ldots,t_n/a_n)\in S$ for every assignment $(t_1/a_1,t_2/a_2,\ldots,t_n/a_n)$ such that $P(a_1,a_2,\ldots,a_n)$, so $(\beta(t_1/a_1,t_2/a_2,\ldots,t_n/a_n)(x_1/c_1,x_2/c_2\ldots,x_m/c_m))^I=1$ by the induction hypothesis, so we have $(\alpha(x_1/c_1,x_2/c_2\ldots,x_m/c_m))^I=1$. The other cases can be proved analogously.

4 Logic DTL

The logic *DTL* (Description Logic of Tasks) that we are going to define in this section is intended to axiomatize the set of accomplishable task formulas. It will be mentioned that the concept of quasiaction and quasireaction of a task formula is same as those in [2].

Definition 4.1 (*DTL*). The axioms of *DTL* are all the primitive formulas that are accomplishable.

The rules of inference are

A-rule:

$\frac{\pi}{\alpha}$, where π is an elementary quasiaction for α.

R-rule：

$\frac{\overline{\alpha},\pi_1,\pi_2,\cdots,\pi_e}{\alpha}$, where $e\geq 1$ and $\pi_1,\pi_2,\cdots,\pi_e$ are all quasireaction for α, $\overline{\alpha}$ is the primitivization of α.

Theorem 4.1(soundness) Let α be a task formula, if DTL $\vdash\alpha$ then α is accomplishable.

Theorem 4.2 (completeness) Let α be a task formula, if α is accomplishable, then DTL $\vdash\alpha$.

The only different between the logic DTL and the logic L proposed in [2] is the different of their axioms. Axioms of DTL are primitive task formulas that are accomplishable while the axioms of L are formulas provable in classical first order logic. The rules of inference in them are same. Keep in mind that the concept of strategy, quasiaction and quasireaction used in this paper are same to those in [2]. So, the proof of the soundness, completeness of DTL can be carried out analogically as the proof of the corresponding properties of the logic L.

Theorem 4.3 (decidability) DTL is decidable.

Proof: Here is an informal description of a decision procedure for DTL $\vdash\alpha$, together with a proof, by induction on the additive complexity ofα, that the procedure takes a finite time. Given a formula α

(a) If α is primitive, then we can judge whether it is an axiom, i.e. whether it is accomplishable using algorithm 3.1 in finite time by Proposition 3.1.

(b) If α is not primitive, then the only way it can be proved in DTL is if either one of the elementary quasiactions for it is provable, or all of the elementary quasireactions for it, together with its primitivization, are provable in DTL. Whether the primitivization is provable can be checked in a finite time. Also, as we noted, the

number all the elementary quasiactions and quasireactions for α is finite. So, check each of them for provability in DTL. If it turns out that either one of the elementary quasiactions, or all of the elementary quasireactions together with the primitivization of α are provable in DTL, then output "yes", otherwise output "no". The additive complexities of those elementary quasiactions and quasireactions are lower than the additive complexity of α and, by the induction hypothesis, their provability in DTL can be checked in a finite time. So that this step, too, can be completed in a finite time.

5 Conclusion

The description logic of tasks enable tasks description and have more reasoning power, it can be used to structure the cooperation plan system for multi agent system. For example, for behavior modeling of large-scale battlefield simulation, we use it to describe the ability knowledge of military entities and the relations among them, then we can use the reasoning power of it for cooperation actions plan and verification.

References

1. V.Dignum, J.J. Meyer, F. Dignum, H. Weigand. Formal Specification of Interaction in Agent Societies. In: M. Hinchey, J. Rash, W. Truszkowski, C. Rouff, D. Gordon-Spears (Eds.): Formal Approaches to Agent-Based Systems (FAABS), Lecture Notes in Artificial Intelligence, Springer-Verlag, Volume 2699/2003.
2. Giorgi Japaridze, The logic of tasks, Annals of Pure and Applied Logic 117 (2002).
3. Peep Küngas Analysing AI Planning Problems in Linear Logic – A Partial Deduction Approach Lecture Notes in Computer Science Volume 3171/2004.
4. Peep Küngas, Mihhail Matskin Linear Logic, Partial Deduction and Cooperative Problem Solving Lecture Notes in Computer Science Volume 2990/2004.
5. G.Japaridze, Introduction to computability logic. Annals of Pure and Applied Logic, vol. 123 (2003).
6. A. Blass, A game semantics for linear logic, Ann. Pure Appl. Logic 56 /1992.
7. F. Baader etc. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge, Cambridge University Press 2002.
8. F. Baader and U. Sattler. Tableau Algorithms for Description Logics ， Lecture Notes In Computer Science， Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods ,2000.
9. F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. DFKI Research Report RR-91-10, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, 1991.
10. S. Russel, P. Norwig, Artificial Intelligence: A Modern Approach, Prentice-Hall, Englewood Cli9s, NJ,1995.