# Golog+HTN$^{TI}$: Adding time and intervals to procedural and hierarchical control knowledge

**Chitta Baral**
Dept. of Computer Sc. & Eng.
Arizona State University
Tempe, AZ 85287, USA
chitta@asu.edu

**Tran Cao Son**
Department of Computer Sc.
New Mexico State University
Las Cruces, NM 88003, USA
tson@cs.nmsu.edu

**Le-Chi Tuan**
Dept of Computer Sc. and Eng.
Arizona State University
Tempe, AZ 85287, USA
lctuan@asu.edu

**Abstract**

In this paper we introduce a language for expressing procedural and HTN-based domain constraints. Our language starts with features from GOLOG and HTN and extends them so that we can deal with actions with duration by being able to specify time intervals between the start (or end) of an action (or a program) and the start of another action (or program). We then discuss a planner based on the answer set planning paradigm that can exploit such domain knowledge.

## Introduction and Motivation

GOLOG (Levesque *et al.* 1997) is an Algol-like logic programming language for agent programming, and control and execution. It is based on a situation calculus theory of actions (Reiter 2000). GOLOG has been primarily used as a programming language for high-level agent control in dynamical environments (see e.g. (Burgard *et al.* 1998)). Although a planner can be written as a GOLOG program (See Chapter 10 of (Reiter 2000)), in (Son, Baral, & McIlraith 2001) an alternative view of GOLOG programs is presented. There it is viewed as an incompletely speci£ed plan (or a form of procedural knowledge) that includes nondeterministic choice points that are £lled in by the planner. For example, the GOLOG program $a_1; a_2; (a_3|a_4|a_5); f$, when viewed as a procedural knowledge tells a planner that the £rst action of the plan should be $a_1$, the second action should be $a_2$, and the 3rd action should be one of $a_3, a_4$ and $a_5$ such that $f$ holds afterward. A planner, when given this procedural knowledge needs only to decide which one of $a_3$, $a_4$, or $a_5$ it should choose as its third action. In (Son, Baral, & McIlraith 2001) it is shown how such procedural knowledge can be exploited to speed up planning. There it is also shown how to combine constructs from HTN-planning with the GOLOG constructs.

Now suppose that actions have duration. In that case we may want to say that start $a_1$ and after 2 unit time of execution of $a_1$ start executing $a_2$ and so on. Similarly in an HTN-based domain constraint instead of the standard $a_1 \prec a_2$ – which means that $a_1$ should be executed before $a_2$ – we might want to say that $a_1$ should start at least 3 units of time before the start of $a_2$ and so on. To the best of our knowledge current extensions of GOLOG and HTN do not have such features. *Thus our main goal in this paper is to develop a language that allows the expression of domain knowledge (procedural and HTN-based) of the above kind.* We will refer to this language as Golog+HTN$^{TI}$, meaning that we add time and intervals to a language that has features from Golog and HTN.

To characterize Golog+HTN$^{TI}$ we need an action theory that allows actions to have durations. For that we chose a simple extension of the language $\mathcal{A}$, which we refer to as $\mathcal{AD}$. We give the semantics of $\mathcal{AD}$ using logic programming with answer sets. This (the semantics of $\mathcal{AD}$) allows us to de£ne the notion of a trajectory. We use the notion of a trajectory to de£ne the notion of a trace of a Golog+HTN$^{TI}$ speci£cation. We also use logic programming with answer sets (by adding additional rules to the logic program that de£nes a trajectory) to give an alternative de£nition of trace and show the two notions to be equivalent. Finally we try to show that using Golog+HTN$^{TI}$ speeds up planning.

## Background: LPASS and LP$_{smodels}$

We now brie¤y introduce LPASS and its extension LP$_{smodels}$. An LPASS program is a collection of rules of the form

$$a_0 \leftarrow a_1, \ldots, a_m,\ not\ a_{m+1}, \ldots,\ not\ a_n \qquad (1)$$

where $a_i$'s are atoms. For an atom $a$, "*not a*" is referred to as a naf-literal. Intuitively, the above LPASS rule means that if $a_1 \ldots a_m$ are true and $a_{m+1} \ldots a_n$ can be assumed to be false then $a_0$ must be true.

The semantics of an LPASS program is de£ned using answer sets. LPASS programs whose rules do not have $not$ in the body – referred to as de£nite programs – have unique answer sets, which are the least models of the theory obtained by treating rules of the form $a_0 \leftarrow a_1, \ldots, a_m$ as the classical formula $a_1 \wedge \ldots \wedge a_m \supset a_0$. Given an LPASS program $P$ and a set of atoms $S$, the Gelfond-Lifschitz transformation $P^S$ (Gelfond & Lifschitz 1990) is de£ned as the set of rules obtained from $P$ by removing all rules from $P$ whose body contains $not\ b$ such that $b \in S$, and then removing the naf-literals from the rest of the rules. *A set $S$ of atoms is said to be an answer set of an LPASS program $P$ if $S$ is* *the answer set of the de£nite program* $P^S$. Answer sets of propositional LPASS programs can be computed using answer set solvers such as **smodels** (Niemelä & Simons 1997) and **dlv** (Citrigno *et al.* 1997). By LP$_{smodels}$ we refer to the extension of LPASS used in (Niemelä, Simons, & Soininen 1999) where rules of the following form are also allowed:

$$l\{b_1,\ldots,b_k\}u \leftarrow a_1,\ldots,a_m, not\ a_{m+1},\ldots, not\ a_n$$

where $a_i$ and $b_j$ are atoms and $l$ and $u$ are two integers, and $l \leq u$. Intuitively, such a rule enforces the constraint that if the body is true then at least $l$ and at most $u$ atoms from the head are also true.

## Reasoning about durative actions using LPASS

As we mentioned earlier, to characterize domain constraints, we need to £rst describe an action description language. The action description language that we plan to use is a simple extension of the language $\mathcal{A}$ (Gelfond & Lifschitz 1998). In our extension, which we will refer to as $\mathcal{AD}$, we will allow actions to have duration and this will be suf£cient to help us to justify and illustrate our language for domain constraints with new connectives.

**Syntax of $\mathcal{AD}$.** An action theory consists of two £nite, disjoint sets of names **A** and **F**, called *actions* and *¤uents*, respectively, and a set of propositions of the following form:

(2) **causes**$(a, f)$ **initially**$(f)$ (4)

(3) **executable**$(a, \{p_1,\ldots,p_n\})$ **duration**$(a, v)$ (5)

where $f$ and $p_i$'s are ¤uent literals (a *¤uent literal* is either a ¤uent $g$ or its negation $\neg g$) and $a$ is an action. (2) is called a *dynamic causal law* and represents the effect of $a$ while (3) states an executability condition of $a$. Intuitively, a proposition of the form (2) states that $f$ is guaranteed to be true after the execution of $a$. An executability condition of $a$ says that $a$ is executable in a state in which $p_1,\ldots,p_n$ hold. Propositions of the form (4) are used to describe the initial state. It states that $f$ holds in the initial state. Finally, a proposition of the form (5) is used to say that duration of action $a$ is $v$.

An action theory is a set of propositions of the form (2)-(5). We will assume that each action $a$ appears in one and only one proposition of the form (5) and $v$ is a non-negative integer expression. We will often conveniently write $d(a)$ to denote the value $v$ if **duration**$(a, v) \in D$ and for a set of actions $A$, $d(A) = \max\{d(a) \mid a \in A\}$.

**Example 1** Consider an action theory with the set of ¤uents $\{f, g, h\}$ and the set of actions $\{a, b, c, d\}$ and the following propositions:

**causes**$(a, f)$ **duration**$(a, 3)$ **executable**$(a, \{g, h\})$
**causes**$(b, h)$ **duration**$(b, 2)$ **executable**$(b, \{\})$
**causes**$(c, g)$ **duration**$(c, 2)$ **executable**$(c, \{\})$
**causes**$(d, \neg g)$ **duration**$(d, 1)$ **executable**$(d, \{\})$
**initially**$(\neg f)$ **initially**$(\neg g)$ **initially**$(\neg h)$

The propositions about $a$ (the £rst line) say that $a$ will cause the ¤uent $f$ to be true after 3 units of time and is executable only if $g$ and $h$ are true. The propositions for other actions have similar meaning.

Since the characterization of $\mathcal{AD}$ is not the aim of this paper, we do not present an independent characterization of it. (Recall that our goal is to use $\mathcal{AD}$ to show how to plan using a proposed domain constraint language Golog+HTN$^{TI}$.) Instead we give a LPASS encoding of prediction and planning using $\mathcal{AD}$. In another work, we present a transition function based characterization of $\mathcal{AD}$ and prove that it is equivalent to the LPASS characterization in this paper.

**Semantics: Prediction in $\mathcal{AD}$.** Given a a set of propositions $D$ we construct an LPASS program $\pi_{\mathbf{D}}$ as follows:

1. For each v-proposition (4) in $D$, $\pi_D$ contains the following rule:
$$h(f, 1). \quad (7)$$
This describes the initial state (which ¤uents hold in time point 1) as speci£ed by the set of v-propositions in $D$.

2. For each ex-proposition (3) in $D$, $\pi_D$ contains the following rules:
$$exec(a, T) \leftarrow not\ not\_exec(a, T). \quad (8)$$
$$not\_exec(a, T) \leftarrow not\ h(p_1, T). \quad (9)$$
$$\ldots$$
$$not\_exec(a, T) \leftarrow not\ h(p_n, T). \quad (10)$$
These rules de£ne when $a$ is executable at a time point $T$, based only on the truth of ¤uents. Rules (9)-(10) say that $a$ is not executable if any of the ¤uent literals in its executability conditions does not hold.

3. For each d-proposition (5) in $D$, we add the following rules to $\pi_D$,
$$ends(a, T+v) \leftarrow init(a, T). \quad (11)$$
$$in\_exec(a, T) \leftarrow init(a, T'), T' \leq T < T+v. \quad (12)$$
These rules de£ne when an action ends and when it is under execution.

4. For each action $a$ and an ef-proposition (2), the following rules are added to $\pi_D$,
$$h(f, T) \leftarrow ends(a, T). \quad (13)$$
$$ab(f, T+1) \leftarrow init(a, T). \quad (14)$$
These rules are used to reason about truth value of ¤uents at different time points.

**Encoding the frame axiom.** $\pi_D$ contains the following rules that encode the frame axiom. They are slightly different from the normal LPASS encoding of the frame axiom so as to take into account action duration.

$$h(F, T+1) \leftarrow h(F, T),\ not\ ab(\neg F, T+1). \quad (15)$$
$$h(\neg F, T+1) \leftarrow h(\neg F, T),\ not\ ab(F, T+1). \quad (16)$$

If we now want to £nd out if $f$ would be true at time point $t$ after starting the execution of actions $a_1$ at time point $t_1$, $a_2$ at time point $t_2$, ... and $a_n$ at time $t_n$ all we need to do is to add

$$\{init(a_i, t_i) \mid i \in \{t_1,\ldots,t_n\}\}$$

and the constraints

$$\leftarrow init(a_i, t_i), not\ exec(a_i, t_i)$$

(for $i = 1, \ldots, n$) to $\pi_D$, set the limits for the various variables, and ask if the resulting program entails $h(f, t)$.

One assumption we made in our characterization is that we assume that the effect of an action takes into effect only after its execution ends, and the ¤uents, whose value changes due to an action execution, are in a unknown state during the execution. This of course can be changed by appropriately modifying $\pi_D$, in particular the rule (14).

## Answer Set Planning with $\mathcal{AD}$ Action Theories

We now show how the idea of answer set planning (Lifschitz 1999) can be extended to $\mathcal{AD}$ action theories. Our LPASS planner for an action theory $D$, denoted by $\Pi(D)$, will consist of the program representing and reasoning about actions of $D$, $\pi_D$, the rules representing the goal, and the rules that generate action occurrences. Besides, we will need to set the limit on the maximal number of steps (the length) of the plan. We will call it $plan\_size$. From now on, whenever we refer to a time point $t$, we mean that $1 \leq t \leq plan\_size$.

**Representing goal.** Assume that we have a goal that is a conjunction of ¤uent literals $g_1 \wedge \ldots \wedge g_m$. We represent this by a set of atoms $\{finally(g_i) \mid i = 1, \ldots, m\}$. The following rules encode when the goal – as described by the $finally$ facts – is satis£ed at a time point $T$.

$$not\_goal(T) \leftarrow finally(X),\ not\ h(X,T). \quad (17)$$
$$goal(T) \leftarrow not\ not\_goal(T). \quad (18)$$

The following constraint eliminates otherwise possible answer sets where the goal is not satis£ed at the time point $plan\_size$.

$$\leftarrow not\ goal(plan\_size). \quad (19)$$

We now de£ne the notion of a *plan*.

**De£nition 1** Given an action theory $D$, a goal $G$, and a plan size *plan_size*, we say that a sequence of sets of grounded actions $A_1, \ldots, A_n$ is a *plan* achieving $G$ if $goal(plan\_size)$ is true in every answer set of the program the program $\pi^{PVer}(D, G)$[1], which consists of

- the rules of $\Pi(D)$ in which the time variable is less than or equal *plan_size*;
- the rules representing $G$;
- the set of action occurrences
$$\bigcup_{i=1}^{n} \{init(a, i) \mid a \in A_i\};$$
- the rules that disallow actions with contradictory conclusions to overlap (rules (23) and (24), below.

We say that a plan $p = A_1, \ldots, A_n$ is a *concurrent plan* if there exists a pair $i$ and $j$ and an action $a \in A_i \cap A_j$ such that $i + d(a) > j$, i.e., $p$ contains an overlapping of two instantiations of a same actions. $p$ is said to be *non-concurrent* if it is not a concurrent plan.

**Generating Action Occurrences.** The following rules enumerate action initiations. To decrease the number of answer sets we have made the assumption that two action instantiations corresponding to the same action can not overlap each other, i.e., we consider only non-concurrent plans. This need not be the case in general. Our point here is that LPASS allows us to express such restrictions very easily. For each action $a$ with the duration $v$ (i.e., **duration**$(a, v)$ belongs to $D$), the following rule will be added to $\Pi(D)$:

$$occ\_before(a,T) \leftarrow init(a,T_1), T_1{<}T{<}T_1{+}v. \quad (20)$$
$$init(a,T) \leftarrow exec(a,T),\ not\ occ\_before(a,T),\ not\ not\_init(a,T). \quad (21)$$
$$not\_init(a,T) \leftarrow not\ init(a,T). \quad (22)$$

The next rules disallow actions with contradictory effects to overlap: for every pair of $a$ and $b$ such that **causes**$(a, f)$ and **causes**$(b, \neg f)$ belong to $D$, $\Pi_D$ contains the two rules:

$$overlap(a,b,T) \leftarrow in\_exec(a,T), in\_exec(b,T). \quad (23)$$
$$\leftarrow overlap(a,b,T). \quad (24)$$

Let $\pi^{PGen}(D,G)$ be the set of rules of $\Pi(D)$ with the goal $G$ and $plan\_size{=}n$. For an answer set $M$ of $\pi^{PGen}(D,G)$, let $s_i(M) = \{f \mid h(f,i) \in M\}$ and $A_i(M) = \{a \mid init(a,i) \in M\}$. We can prove that

**Theorem 1** For an action theory $D$ and a goal $G$, $B_1, \ldots, B_n$ is a non-concurrent plan that achieves $G$ iff there exists an answer set $M$ of $\pi^{PGen}(D,G)$ such that $A_i(M) = B_i$.

## Golog+HTN$^{TI}$: Using durations in Procedural and Hierarchical domain Constraints

We begin with an informal discussion on the new construct in Golog+HTN$^{TI}$. Consider the domain from Example 1. It is easy to see that the program $b; c; a$ is a plan achieving $f$ from any state and the time needed to execute this plan is the sum of the actions's durations (Figure 1, Case (a)). Observe that $b$ and $c$ are two actions that achieve the condition for $a$ to be executable and can be executed in parallel. Hence it should be obvious that any plan that allows $b$ and $c$ to execute in parallel will have a shorter execution time. For the moment, let us represent this by the program $p_1 = \{b, c\}; a$. The execution of this program is depicted in Figure 1, Case (b).

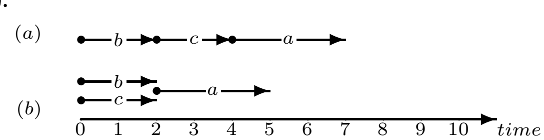


Figure 1: A pictorial view of program execution (the dot shows when an action starts and the arrow shows when an action stops)

Now consider a modi£cation of the domain in Example 1, in which the executable propositions of $c$ changes to **executable**$(c, \{\neg g\})$. A program achieving $f$ would be to execute $b$ and $d$ in parallel, then $c$, and lastly $a$. We cannot execute $b$, $c$, and $d$ in parallel all the time because $c$ is not

[1]The $^{PVer}$ stands for *plan veri£cation*.

executable until $\neg g$ holds, and hence, it might need to wait for $d$ to £nish. It is easy to see, however, that it is better if $c$ starts whenever $d$ £nishes. To account for this, we introduce a new construct that allows programs to start even if the preceding program has not £nished. We write $(\{b, d\};^{s}_{[1,1]} c); a$ to indicate that $c$ should start its execution 1 time unit after $\{b, d\}$ and then $a$ and denote this program by $p_2$. The execution of this program can be illustrated as follows.

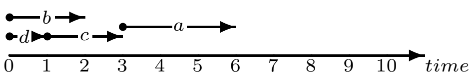


Figure 2: Execution of $(\{b, d\};^{s}_{[1,1]} c); a$

We now de£ne programs in Golog+HTN$^{TI}$ that express domain knowledge to be exploited by a planner.

**De£nition 2 (Program)** For an action theory $D$,

1. an action $a$ is a program;
2. a temporal constraint $\phi[t_1, t_2]$ is a program;
3. if $p_1$ and $p_2$ are programs and $0 \leq t_1 \leq t_2$ are two time non-negative integers then so are $(p_1|p_2)$, $(p_1;^{s}_{[t_1,t_2]} p_2)$, and $(p_1;^{e}_{[t_1,t_2]} p_2)$;
4. if $p_1$ and $p_2$ are programs and $\phi$ is a ¤uent formula then so are "**if** $\phi$ **then** $p_1$ **else** $p_2$" and "**while** $\phi$ **do** $p$";
5. if $X_1, \ldots, X_n$ are variables of sort $s_1, \ldots, s_n$, respectively, $p(X_1, \ldots, X_n)$ is a program, and $f(X_1, \ldots, X_n)$ is a formula, then **pick**$(\vec{X}, f(\vec{X}), p(\vec{X}))$ is a program where $\vec{X}$ stands for $X_1, \ldots, X_n$;
6. if $p_1, \ldots, p_n$ are programs then a pair $(S, C)$ is a program where $S = \{p_1, \ldots, p_n\}$ and $C$ is a set of constraints over $S$ of the following form:
   i) $p_1 \prec^{s}_{[t_1,t_2]} p_2$ (or $p_1 \prec^{e}_{[t_1,t_2]} p_2$), (ii) $(p, \phi[t_1, t_2])$, (iii) $(\phi[t_1, t_2], p)$, and (iv) $(p_1, \phi[t_1, t_2], p_2)$ where $p, p_1, p_2$ are programs, $\phi$ is a ¤uent formula and $0 \leq t_1 \leq t_2$.

The constructs 1-5 in the above de£nitions are generalizations of constructs in GOLOG (Levesque *et al.* 1997) and the construct 6 is a generalization of hierarchical task networks (HTN) (Erol, Nau, & Subrahmanian 1995).

Intuitively, $\phi[t_1, t_2]$ expresses the constraint that $\phi$ holds during the time interval $[t_1, t_2]$ (from the current moment) and $(p_1;^{s}_{[t_1,t_2]} p_2)$ states that $p_2$ should start its execution at least $t_1$ and at most $t_2$ units of time after $p_1$ *starts* whereas $(p_1;^{e}_{[t_1,t_2]} p_2)$ forces $p_2$ to wait for $t$ ($t_1 \leq t \leq t_2$) units of time after $p_1$ *£nishes*. It is easy to see that $(p_1;^{s}_{[0,0]} p_2)$ requires that $p_1$ and $p_2$ be executed in parallel whereas $(p_1;^{e}_{[0,0]} p_2)$ requires that $p_2$ starts its execution at the time $p_1$ £nishes. Note that $(p_1;^{e}_{[0,0]} p_2)$ corresponds to the original notation $p_1; p_2$.

The constraints in item 6 above are similar to truth constraints and ordering constraints over tasks in HTN. Intuitively, $p_1 \prec^{s}_{[t_1,t_2]}$ says that $p_2$ can start its execution after $p_1$ is in execution (or after $p_1$ £nishes its execution) for $t_1$ time units but no later than $t_2$ time units of time. Similarly, $(p, \phi[t_1, t_2])$ (resp. $(\phi[t_1, t_2], p)$) means that $\phi$ must hold from $t_1$ to $t_2$ immediately after (resp. before) $p$'s execution. $(p_1, \phi[t_1, t_2], p_2)$ states that $p_1$ must start before $p_2$ and $\phi$ must hold $t_1$ units of time after $p_1$ starts until $t_2$ units of time before $p_2$ starts.

**Example 2** In our notation, $p_1$ and $p_2$ (from the discussion before Figure 1) are represented by $((b;^{s}_{[0,0]} c);^{e}_{[0,0]} a)$ and $(((b;^{s}_{[0,0]} d);^{s}_{[1,1]} c);^{e}_{[0,0]} a)$, respectively.

We will now de£ne the notion of *a trace of a program*, which describe what actions are done when. But £rst we need to de£ne the notion of a trajectory. For an action theory $D$ and an integer $n$, let $\pi^{Gen}(D)$ be the program consisting of the set of rules of $\Pi(D)$ whose time variable belongs to the set $1, \ldots, n$ (recall that $\Pi(D)$ consists of the rules (7)-(16) and (20)-(24)).

**De£nition 3 (Trajectory)** For an action theory $D$ and an answer set $M$ of $\pi^{PGen}(D)$, let $s_i = \{f \mid h(f, i) \in M\}$ and $A_i = \{a \mid init(a, i) \in M\}$. We say that the sequence $\alpha = s_1 A_1 \ldots s_n A_n$ is a *trajectory* of $D$.

Intuitively, a trajectory is an alternating sequence of states and action occurrences $s_1 A_1, \ldots, s_n A_n$, where $s_i$ is a state at time point $i$ and $A_i$ is the set of actions that are supposed to have occurred at time point $i$. We are now ready to de£ne what is a trace of a program. Since the de£nition of a trace is somewhat complicated we divide it into several small de£nitions and illustrate the complex ones with examples. First, we begin with the primitive cases.

**De£nition 4 (Trace - Primitive Cases)** A trajectory $\alpha = s_1 A_1 \ldots s_n A_n$ is a trace of a program $p$ if
• $p = a$, $n = d(a)$ and $A_1 = \{a\}$ and $A_i = \emptyset$ for $i > 1$; or
• $p = \phi[t_1, t_2]$, $n = t_2$, $A_i = \emptyset$ for every $i$, and $\phi$ holds in $s_t$ for $t_1 \leq t \leq t_2$.

The next de£nition deals with programs that are constructed using GOLOG-constructs ((Levesque *et al.* 1997)).

**De£nition 5 (Trace – Programs with GOLOG-Constructs)** A trajectory $\alpha = s_1 A_1 \ldots s_n A_n$ is a trace of a program $p$ if one of the following is satis£ed.
• $p = p_1 \mid p_2$, $\alpha$ is a trace of $p_1$ or $\alpha$ is a trace of $p_2$,
• $p =$ **if** $\phi$ **then** $p_1$ **else** $p_2$, $\alpha$ is a trace of $p_1$ and $\phi$ holds in $s_1$ or $\alpha$ is a trace of $p_2$ and $\neg\phi$ holds in $s_1$,
• $p =$ **while** $\phi$ **do** $p_1$, $n = 1$ and $\neg\phi$ holds in $s_1$ or $\phi$ holds in $s_1$ and there exists some $i$ such that $s_1 A_1 \ldots A_i$ is a trace of $p_1$ and $s_{i+1} A_{i+1} \ldots A_n$ is a trace of $p$, or
• $p =$ **pick**$(\vec{X}, f(\vec{X}), q(\vec{X}))$, then there exists a constant $\vec{x}$ of the sort of $\vec{X}$ such that $f(\vec{x})$ holds in $s_1$ and $\alpha$ is a trace of $q(\vec{x})$.

The trace of each program is de£ned based on its structure, i.e., how it is built. We next deal with the new connectives $;^{s}_{[t_1,t_2]}$ and $;^{e}_{[t_1,t_2]}$.

**De£nition 6 (Trace – Parallel and Overlapping Programs)** A trajectory $\alpha = s_1 A_1 \ldots s_n A_n$ is a trace of a program $p$ if
• $p = p_1;^{s}_{[t_1,t_2]} p_2$, there exists two numbers $t_3$ and $t_4$ such that $t_1 + 1 \leq t_3 \leq t_2 + 1$ and $t_4 \leq n$ (because the index of the trace starts from 1) and either (i) there exists a trace $s_1 B_1 \ldots s_{t_4} B_{t_4}$ of $p_1$ and a trace $s_{t_3} C_{t_3} \ldots s_n C_n$ of $p_2$ such

that $A_i = B_i \cup C_i$ for every $i$; or (ii) $t_3 \leq t_4$ and there exists a trace $s_1B_1 \ldots s_nB_n$ of $p_1$ and a trace $s_{t_3}C_{t_3} \ldots s_{t_4}C_{t_4}$ of $p_2$ such that $A_i = B_i \cup C_i$ (we write $B_j = \emptyset$ or $C_j = \emptyset$ for indexes that do not belong to the trace of $p_1$ or $p_2$); or

• $p{=}p_1;^e_{[t_1,t_2]}\, p_2$, there exists two numbers $t_3$ and $t_4$ such that $t_1 + t_3 \leq t_4 \leq t_2 + t_3$ and $t_4 \leq n$ and $s_1A_1 \ldots s_{t_3}A_{t_3}$ is a trace of $p_1$ and a trace $s_{t_4}A_{t_4} \ldots s_nA_n$ is a trace of $p_2$ and $A_i = \emptyset$ for every $t_3 \leq i < t_4$.

This de£nition is best illustrated using a picture.

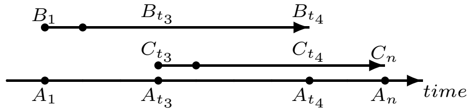


Figure 3: $A_i = B_i \cup C_i$ – First Item, Case 1 (De£nition 6)

**Example 3** • For $p_1 = ((b;^s_{[0,0]}\, c);^e_{[0,0]}\, a)$ (wrt. the action theory in Example 1), we can easily check that

$$\mathbf{s_1^1}\ \{\mathbf{b},\mathbf{c}\}\ \mathbf{s_2^1}\ \emptyset\ \mathbf{s_3^1}\{\mathbf{a}\}\ \mathbf{s_4^1}\ \emptyset\ \mathbf{s_5^1}\ \emptyset\ \mathbf{s_6^1}\ \emptyset$$

where
– $s_1^1 = \{\neg f, \neg g, \neg h\}$,
– $s_2^1 = \{\neg f\}$,
– $s_3^1 = \{\neg f, g, h\}$,
– $s_4^1 = \{g, h\}$,
– $s_5^1 = \{g, h\}$,
– $s_6^1 = \{f, g, h\}$,

is a trace of $p_1$. On the other hand,

$$\mathbf{s_1^1}\ \{\mathbf{b},\mathbf{c},\mathbf{d}\}\ \mathbf{s_2^1}\ \emptyset\ \mathbf{s_3^1}\{\mathbf{a}\}\ \mathbf{s_4^1}\ \emptyset\ \mathbf{s_5^1}\ \emptyset\ \mathbf{s_6^1}\ \emptyset$$

although it contains a trace of $p_1$.

• For $p_2 = (((b;^s_{[0,0]}\, d);^s_{[1,1]}\, c);^e_{[0,0]}\, a)$. (wrt. the modi£ed action theory), we can easily check that

$$\mathbf{s_1^2}\ \{\mathbf{b},\mathbf{d}\}\ \mathbf{s_2^2}\ \{\mathbf{c}\}\ \mathbf{s_3^2}\ \emptyset\ \mathbf{s_4^2}\{\mathbf{a}\}\ \mathbf{s_5^2}\ \emptyset\ \mathbf{s_6^2}\ \emptyset\ \mathbf{s_7^2}\ \emptyset$$

where
– $s_1^2 = \{\neg f, \neg g, \neg h\}$,
– $s_2^2 = \{\neg f, \neg g\}$,
– $s_3^2 = \{\neg f, h\}$,
– $s_4^2 = \{\neg f, g, h\}$,
– $s_5^2 = \{g, h\}$,
– $s_6^2 = \{g, h\}$,
– $s_7^2 = \{f, g, h\}$,

is a trace of $p_2$. but

$$\mathbf{s_1^2}\ \{\mathbf{b},\mathbf{d}\}\ \mathbf{s_2^2}\ \emptyset\ \mathbf{s_2^2}\ \{\mathbf{c}\}\ \mathbf{s_3^2}\ \emptyset\ \mathbf{s_4^2}\{\mathbf{a}\}\ \mathbf{s_5^2}\ \emptyset\ \mathbf{s_6^2}\ \emptyset\ \mathbf{s_7^2}\ \emptyset$$

is not a trace of $p_2$ because $c$ should start at the time moment 2.

In the next de£nition we deal with HTN-programs.

**De£nition 7 (Trace - HTN Programs)** A trajectory $\alpha = s_1A_1 \ldots s_nA_n$ is a trace of a program $p = (S, C)$ with $S = \{p_1, \ldots, p_k\}$ if there exists two sequences of numbers $b_1, \ldots, b_k$ and $e_1, \ldots, e_k$ with $b_j \leq e_j$, a permutation $(i_1, \ldots, i_k)$ of $(1, \ldots, k)$, and a sequence of traces $\alpha_j = s_{b_j}A^j_{b_j} \ldots s_{e_j}A^j_{e_j}$ that satisfy the following conditions:
• for each $l$, $1 \leq l \leq k$, $\alpha_l$ is a trace of $p_{i_l}$,
• if $p_t \prec p_l \in C$ then $i_t < i_l$,
• if $p_t \prec^s_{[q_1,q_2]} p_l \in C$ then $i_t < i_l$ and $b_{i_t} + q_1 \leq b_{i_l} \leq b_{i_t} + q_2$,
• if $p_t \prec^e_{[q_1,q_2]} p_l \in C$ then $i_t < i_l$ and $e_{i_t} + q_1 \leq b_{i_l} \leq e_{i_t} + q_2$,
• if $(\phi[t_1, t_2], p_l) \in C$ (or $(p_l, \phi[t_1, t_2]) \in C$) then $\phi$ holds in the state $s_{b_{i_l}-t_2}, \ldots, s_{b_{i_l}-t_1}$ (or $s_{e_{i_l}+t_1}, \ldots, s_{e_{i_l}+t_2}$), and
• if $(p_t, \phi[t_1, t_2], p_l) \in C$ then $\phi$ holds in $s_{b_{i_t}+t_1}, \ldots, s_{b_{i_l}-t_2}$.
• $A_i = \cup_{j=1}^k A_i^j$ for every $i = 1, \ldots, n$ where we assume that $A_i^j = \emptyset$ for $i < b_j$ or $i > e_j$.

The intuition of the above de£nition is as follows. First, each program starts ($b_i$'s) and ends ($e_i$'s) at some time point and it cannot £nish before it even starts, hence, the requirement $b_i \leq e_i$. The order of the execution is speci£ed by the ordering constraints and not by the program's number. The permutation $(i_1, \ldots, i_k)$ and $j$'s record the starting time of the programs. The conditions on the trajectories make sure that the constraints are satis£ed (£rst four items) and they indeed create $A_1, \ldots, A_n$ (last item).

## An LPASS interpreter

We now present an LPASS interpreter for programs. We will adopt the way to encode formulas and programs in (Son, Baral, & McIlraith 2001) for use with answer set solvers. In short, each program $p$ (resp. formula $\phi$) will be associated with a unique name $n_p$ (resp. $n_\phi$) and will be replaced by a set of rules and facts, denoted by $r(n_p)$ (resp. $r(n_\phi)$). The formal de£nition of $r(n_p)$ and $r(n_\phi)$ can be found in the aforementioned paper. Here, we demonstrate it by an example.

**Example 4** $p_1{=}((b;^s_{[0,0]}\, c);^e_{[0,0]}\, a)$ is encoded by the two atoms $proc(p_1, p_1^1, a, start, 0, 0)$ and $proc(p_1^1, b, c, start, 0, 0)$.

$p_2{=}(((b;^s_{[0,0]}\, d);^s_{[1,1]}\, c);^e_{[0,0]}\, a)$ is encoded by the atoms $proc(p_2, p_2^1, a, end, 0, 0)$, $proc(p_2^1, p_2^2, c, start, 1, 1)$, and $proc(p_2^2, b, c, start, 0, 0)$.

In the above, a program is speci£ed by the predicate $proc$ with 5 arguments: the name, the £rst sub-program (the head), the rest of the program (the tail), whether the tail should start relative to the start or to the end of the head (*start/end*), and the minimal or the maximal time the tail needs to wait. Programs constructed using other constructs are encoded similarly. For example, the program $p$ = **if** $\phi$ **then** $p_1$ **else** $p_2$ is encoded by the atom $if(p, n_\phi, p_1, p_2)$ and the set of atoms encoding $p_1$ and $p_2$, and $\phi[t_1, t_2]$ by $formula(n_\phi, t_1, t_2)$ etc.

For a program $p$ of an action theory $D$, we de£ne a logic program $\Pi$ that consists of the rules encoding the domain, $\pi_D$, the rules describing the program $r(n_p)$, the set of rules for generating action occurrences (20)-(24), and a set of rules that realizes the operational semantics of programs. We follow the approach in (Son, Baral, & McIlraith 2001) and de£ne a predicate $trans(p, t_1, t_2)$ which holds in an answer set $M$ iff $s_{t_1}(M)A_{t_1}(M)\ldots s_{t_2}(M)A_{t_2}(M)$ is a trace of $p$[2]. We will concentrate on describing the ideas behind the rules and their meaning rather than presenting the rules in great detail. We will present only a few representative rules. Readers interested in the source code of the program can obtain it from our web site[3].

We will begin with an informal discussion on the ideas behind the rules de£ning $trans(p, t_1, t_2)$. Intuitively, because of the rules (20)-(24), each answer set $M$ of the program $\Pi$ will contain a sequence of sets of actions $\alpha = A_1, \ldots, A_n$ where $A_i = \{a \mid init(a, i) \in M\}$. The encoding of the action theory, $\pi_D$, makes sure that whenever an action $a$ is initiated it is executable. Thus, the sequence $\alpha$ is a trajectory of $D$. So, it remains to be veri£ed that $\alpha$ is indeed a trace of the program $p$. We will do this in two steps. First, we check if $\alpha$ contains a trace of $p$, i.e., we make sure that there is a trace $s_1B_1\ldots s_nB_n$ of $p$ such that $B_i \subseteq A_i$. Second, we make sure that no action is initiated when it is not needed. To do so, we de£ne two predicates:

- $tr(p, t_1, t_2)$ - $s_{t_1}A_{t_1}\ldots A_{t_2}$ contains a trace of $p$;
- $used\_in(p, q, t, t_1, t_2)$ - a trace of $p$ starting from $t$ is used in constructing a trace of $q$ from $t_1$ to $t_2$. Intuitively, this predicate records the actions belonging to the traces of $q$. The de£nition of this predicate will make sure that for a simple action $a$, only action $a$ is used to construct its trace, i.e., $used\_in(a, a, t_1, t_1, t_1 + d(a))$ is equivalent to $init(a, t_1)$ and $used\_in(b, a, t_1, t_1, t_1 + d(a))$ is false for every $b \neq a$.

Finally, we say that $trans(p, t_1, t_2)$ holds iff $tr(p, t_1, t_2)$ holds and every action $a \in A_j$ for $t_1 \leq j \leq t_2$, $used\_in(a, p, j, t_1, t_2)$ holds. The rules for $tr(p, t_1, t_2)$ are similar to the rules of the predicate $trans(p, t_1, t_2)$ from (Son, Baral, & McIlraith 2001) with changes that account for action duration and the new constructs such as $;^s_{[t_1,t_2]}$ and $;^e_{[t_1,t_2]}$ and checking for the condition of new constraint on a HTN-program. Below, we list some of the rules for $tr$.

$$tr(A, T_1, T_1 + D) \leftarrow init(A, T_1), duration(A, D). \quad (25)$$

$$tr(P, T_1, T_2) \leftarrow proc(P, P_1, P_2, start, M_1, M_2), T_1 + M_1 \leq T_3 \leq T_1 + M_2, T_1 \leq T_4 \leq T_2, tr(P_1, T_1, T_4), tr(P_2, T_3, T_2). \quad (26)$$

$$tr(P, T_1, T_2) \leftarrow proc(P, P_1, P_2, start, M_1, M_2), T_1 + M_1 \leq T_3 \leq T_1 + M_2, T_3 \leq T_4 \leq T_2, tr(P_1, T_1, T_2), tr(P_2, T_3, T_4). \quad (27)$$

$$tr(P, T_1, T_2) \leftarrow proc(P, P_1, P_2, end, M_1, M_2), T_3 \leq T_4, T_3 + M_1 \leq T_4 \leq T_3 + M_2, tr(P_1, T_1, T_3), tr(P_2, T_4, T_2). \quad (28)$$

$$tr(I, T_1, T_2) \leftarrow if(I, F, P_1, P_2), hf(F, T_1), tr(P_1, T_1, T_2). \quad (29)$$

$$tr(I, T_1, T_2) \leftarrow if(I, F, P_1, P_2), not\ hf(F, T_1), tr(P_2, T_1, T_2).^4 \quad (30)$$

[2] For an answer set $M$, $s_i(M) = \{f \mid h(f, i) \in M,\ f$ is a ¤uent literal$\}$ and $A_i(M) = \{a \mid init(a, i) \in M\}$.

[3] http://www.cs.nmsu.edu/˜tson/duration.

We next present the rules de£ning $tr$ for HTN-programs. First, we begin with the encoding of a HTN-program. A program $p = (S, C)$ is encoded by the set of atoms and rules encoding $S$ and $C$ where elements of $C$ will be represented by the predicates

- $order(*, +, +, start, m_1, m_2)$,
- $order(*, +, +, end, m_1, m_2)$,
- $postC(*, +, -, m_1, m_2)$,
- $preC(*, -, +, m_1, m_2)$, and
- $maintain(*, +, -, +, m_1, m_2)$

where $m_1$, $m_2$ are non-negative integers representing units of time and the place holder '*', '+', or '-' denotes the name of the constraint, a program, or a formula, respectively. To make sure that for every program $q$ belonging to $p$, the trajectory $s_{t_1}A_{t_1}\ldots A_{t_2}$ contains a trace of $q$, we generate the begin- and end-point of each $q$, denoted by $begin(p, q, t_{q_1})$ and $end(p, q, t_{q_2})$, respectively, and then check whether $tr(q, t_{q_1}, t_{q_2})$ holds or not. The key idea is to check whether this creates a trace for $p$. For this reason, we de£ne a predicate $on\_trace(p, t)$ and $nok(p, t_1, t_2)$ to say that the trace of $p$ contains the time moment $t$ and the current assignment of start- and end-point for the programs belonging to $S$ does not satisfy the constraints in $C$. Some of these rules are given below:

$$nok(N, T_1, T_2) \leftarrow htn(N, S, C), T_1 \leq T \leq T_2, not\ on\_trace(N, T). \quad (31)$$

$$nok(N, T_1, T_2) \leftarrow htn(N, S, C), I_1 \in S, I_2 \in S, begin(N, I_1, B_1), begin(N, I_2, B_2), O \in C, order(O, I_1, I_2, start, M_1, M_2), B_1 + M_1 > B_2. \text{ or } (B_1 + M_2 < B_2) \quad (32)$$

The £rst rule says that every time point between $T_1$ and $T_2$ must be on the trace of $p$. The second checks for the ordering constraints in $C$. Rules to check for truth constraints are to the second rule and are omitted here.

We now de£ne the rules for $used\_in(p, q, t, t_1, t_2)$. We have the following rules.

$$used\_in(A, A, T_1, T_1, T_2) \leftarrow action(A), tr(A, T_1, T_2). \quad (33)$$

$$used\_in(P, Q, T_3, T_1, T_2) \leftarrow T_1 \leq T_3 \leq T_2, T_4 \leq T_2, T_1 \leq T_5 \leq T_2, T_4 \leq T_3 \leq T_5, used\_in(P, Q_1, T_3, T_4, T_5), used\_in(Q_1, Q, T_4, T_1, T_2). \quad (34)$$

The £rst rule accounts for the fact that $used\_in(a, a, t, t, t + d(a))$ is always true if $init(a, t)$ is true. The second rule says that if $p$ is used in a trace of $q_1$ and $q_1$ is in a trace of $q$ then $p$ is also in a trace of $q$. These two rules are not enough, however, because so far we have not speci£ed when a program

[4] Rules for $tr$ to work with other programs such as the while-do or non-deterministic choice of arguments **pick** or actions | are de£ned similarly. We must note that in de£ning $tr$ using the rules for $trans$ given in (Son, Baral, & McIlraith 2001), we have improved them in several aspects. The style is similar though.

$p$ is used in a trace of $q$. This is done easily by using the weight rules. For instance, instead of the rule (26), we use the following rule:

$$\begin{array}{ll} 3\{tr(P,T_1,T_2), used\_in(P_1,P,T_1,T_1,T_2), & \\ \quad used\_in(P_2,P,T_3,T_1,T_2)\}3 \leftarrow & \\ \qquad proc(P,P_1,P_2,start,M_1,M_2), & (26') \\ \qquad T_1+M_1 \le T_3 \le T_1+M_2, T_1 \le T_4 \le T_2, & \\ \qquad tr(P_1,T_1,T_4), tr(P_2,T_3,T_2). & \end{array}$$

The intuition of the above rule is that whenever we used the rule to derive $tr(P,T_1,T_2)$ then we also record what sub-programs are used and starting from which moment (by requiring that $used\_in(P_1,P,T_1,T_1,T_2)$ and $used\_in(P_2,P,T_3,T_1,T_2)$ also true). Similarly modi£cation needs to be done for other rules de£ning $tr$. We omit them here for space reason.

Having de£ned $tr$ and $used\_in$, de£ning $trans$ is simple: for every program $P$, $trans(P,T_1,T_2)$ holds only if $tr(P,T_1,T_2)$ holds and every action $A$ initiated during $T_1$ and $T_2-1$ must be used to construct a trace of $P$ from $T_1$ to $T_2$. So, we have the following rules:

$$\begin{array}{rcll} not\_min(P,T_1,T_2) & \leftarrow & action(A), init(A,T), & (35) \\ & & T_1 \le T \le T_2, & \\ & & not\ used\_in(A,P,T,T_1,T_2). & \\ trans(P,T_1,T_2) & \leftarrow & T_1 \le T_2, tr(P,T_1,T_2), & (36) \\ & & not\ not\_min(P,T_1,T_2). & \end{array}$$

We illustrate this de£nition using a simple example.

**Example 5** Consider $p_1{=}((b;^s_{[0,0]}c);^e_{[0,0]}a)$ from Example 4 and the trajectory $\{b,c\},\emptyset,\{a\},\emptyset,\emptyset,\emptyset$.

For $p_1 = ((b;^s_{[0,0]}c);^e_{[0,0]}a)$ (wrt. the action theory in Example 1) from Example 4 and the trajectory (Example 3)

$$\mathbf{s_1^1}\ \{\mathbf{b},\mathbf{c}\}\ \mathbf{s_2^1}\ \emptyset\ \mathbf{s_3^1}\{\mathbf{a}\}\ \mathbf{s_4^1}\ \emptyset\ \mathbf{s_5^1}\ \emptyset\ \mathbf{s_6^1}\ \emptyset$$

with

- $s_1^1 = \{\neg f, \neg g, \neg h\}$,
- $s_2^1 = \{\neg f\}$,
- $s_3^1 = \{\neg f, g, h\}$,
- $s_4^1 = \{g, h\}$,
- $s_5^1 = \{g, h\}$,
- $s_6^1 = \{f, g, h\}$.

Clearly, $tr(b,1,3)$, $tr(c,1,3)$, $tr(a,3,6)$ hold (because of rule (25)). $used\_in(b,b,1,1,3)$, $used\_in(c,c,1,1,3)$, and $used\_in(a,a,3,3,6)$ hold (because of rule (33)). Furthermore, we have $tr(p_1^1,1,3)$, $used\_in(b,p_1^1,1,1,3)$ and $used\_in(c,p_1^1,1,1,3)$ hold (because of (26')). Similarly, $tr(p_1,1,6)$, $used\_in(b,p_1,1,1,6)$, $used\_in(c,p_1,1,1,6)$, and $used\_in(a,p_1,3,1,6)$ hold (because of (26') and (34)). This implies that $trans(p_1,1,6)$ holds. It is worth noting that $trans(b,1,3)$ does not hold since $c$ is initiated at 1 but $used\_in(c,b,1,1,3)$ does not hold.

We next present a theorem that extends a result from (Son, Baral, & McIlraith 2001) discussing a property of the above described program. Let $p$ be a program and $D$ be an action theory. Let $\Pi_n$ be the program consisting of

- the set of rules of $\Pi(D)$,
- the rules de£ning $tr$, $used\_in$, and $trans$, and
- the rule expressing our goal of £nding a trace of $p$

$$\leftarrow not\ trans(p,1,n)$$

where all the time variables in $\Pi_n$ are bounded by $n$. Extending a result from (Son, Baral, & McIlraith 2001), we have the following theorem.

**Theorem 2** For an action theory $D$ and a program $p$, (i) for every answer set $M$ of $\Pi_n$, $s_1(M)A_1\ldots s_n(M)A_n(M)$ is a trace of $p$; and (ii) if $s_1B_1s_2\ldots s_nB_n$ is a trace of $p$ then there exists an answer set $M$ of $\Pi_n$ such that $s_i = \{f \mid h(f,i) \in M\}$ and $B_i = \{a \mid init(a,i) \in M\}$.

## Experimental Evaluations

We have used Golog+HTN$^{TI}$ in planning for domains from the AIPS2002 competition. We concentrate on formulating and testing our language in complex domains in which actions have durations, which might depend on the concrete situation. For example, the duration of £lling a tank depends on various factors: the current level of fuel in the tank, the rate of which the fuel ¤ows, etc. Of the 13 planners competed in the AIPS2002 competition, only 5 have this capability (see (Long *et al.* )). As an example, our planner [5] can solve *all* 20 Zeno Flying problems in which the £rst seven problems were solved in less than 1 second, the next six problems were solved in less than 2 second and the last seven problems were solved within 10 to 60 seconds.

## Conclusion

In this paper we proposed a domain constraint language Golog+HTN$^{TI}$ that generalizes procedural (based on GOLOG) and HTN-based domain constraints to allow time intervals. In the process we generalize the connective ';' to two connectives '$;^s_{[t_1,t_2]}$' and '$;^e_{[t_1,t_2]}$' and similar generalizations of HTN constraints. We then show how the answer set planning paradigm can be used to plan where actions have duration, and we have domain constraints expressed in the language Golog+HTN$^{TI}$. In terms of related work (that we have not mentioned yet) Reiter in (Reiter 2001) discusses temporal GOLOG where time is a parameter of the actions. Two differences (as relevant to the focus of this paper) between his approach and ours is that we generalize the connective ';' rather than actions, and we consider HTN-based constructs not considered by Reiter. Other differences include his use of Situation calculus (and £rst-order logic) as opposed to our use of a propositional action theory. cc-Golog (Grosskreutz & Lakemeyer 2000) extended ConGolog to allow time to be added to the program but cc-Golog concentrates on accommodating even-driven behavior rather than for planning.

[5]We contacted authors of other planners to help us setting up their planners so that a fair comparison between our planner and theirs can be done. Unfortunately, we were not able to £nish this task in a timely fashion and did not yet have the comparison ready for this paper. We plan to have this issue resolved as soon as possible.

## References


Burgard, W.; Cremers, A. B.; Fox, D.; H́ahnel, D.; Lakemeyer, G.; D., S.; Steiner, W.; and Thrun, S. 1998. The interactive museum tour-guide robot. In *Proceedings of the 15th National Conference on Arti£cial Intelligence (AAAI-98)*, 11–18. AAAI Press.

Citrigno, S.; Eiter, T.; Faber, W.; Gottlob, G.; Koch, C.; Leone, N.; Mateis, C.; Pfeifer, G.; and Scarcello, F. 1997. The dlv system: Model generator and application frontends. In *Proceedings of the 12th Workshop on Logic Programming*, 128–137.

Erol, K.; Nau, D.; and Subrahmanian, V. 1995. Complexity, decidability and undecidability results for domain-independent planning. *Arti£cial Intelligence* 76(1-2):75–88.

Gelfond, M., and Lifschitz, V. 1990. Logic programs with classical negation. In Warren, D., and Szeredi, P., eds., *Logic Programming: Proceedings of the Seventh International Conf.*, 579–597.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *ETAI* 3(6).

Grosskreutz, H., and Lakemeyer, G. 2000. cc-golog: Towards more realistic logic-based robot controllers. In *Proceedings of the 17th National Conference on Arti£cial Intelligence*, 476–482. AAAI Press.

Levesque, H.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1-3):59–84.

Lifschitz, V. 1999. Answer set planning. In *International Conference on Logic Programming*, 23–37.

Long, D.; Fox, M.; Smith, D.; McDermott, D.; Bacchus, F.; and Geffner, H. International Planning Competition.

Niemelä, I., and Simons, P. 1997. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings ICLP & LPNMR*, 420–429.

Niemelä, I.; Simons, P.; and Soininen, T. 1999. Stable model semantics for weight constraint rules. In *Proceedings of the 5th International Conference on on Logic Programming and Nonmonotonic Reasoning*, 315–332.

Reiter, R. 2000. On knowledge-based programming with sensing in the situation calculus. In *Proc. of the Second International Cognitive Robotics Workshop, Berlin*.

Reiter, R. 2001. *KNOWLEDGE IN ACTION: Logical Foundations for Describing and Implementing Dynamical Systems*. MIT Press.

Son, T.; Baral, C.; and McIlraith, S. 2001. Domain dependent knowledge in planning - an answer set planning approach. In *Proceedings of the 6th International Conference on Logic Programming and NonMonotonic Reasoning*, 226–239.