Reasoning about actions in presence of resources: applications to planning and scheduling

Chitta Baral[†] Tran Cao Son[‡] Le-Chi Tuan [†]

[†] Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
{chitta,lctuan}@asu.edu

[‡] Department of Computer Science
New Mexico State University
PO Box 30001, MSC CS
Las Cruces, NM 88003, USA
tson@cs.nmsu.edu

Abstract. In this paper we show how the paradigm of model based planning using logic programming with the answer set semantics (LPASS) can be used to do planning and scheduling in presence of actions that have duration, resources, machines, overlapping plans, and alternative resource and machine requirements for actions. We also briefly discuss what the notion of a state and state transition in such a case is and relate it to our logic programming formulation.

1 Introduction and motivation

Recent research on model based planning [8, 20, 5] has led to the bridging of the gap between reasoning about actions and simple classical planning. In the model based planning approach the effect of actions on states of the world is described using a logical language – propositional logic is used in [8], while logic programming with stable model semantics is used in [20, 5] – together with the description of the initial and goal states. A particular plan length n is assumed, and the various possible action occurrences in the different time points are succinctly expressed in the same logical language. Finally, constraints to the effect that the state reached at the time point $n+1$ must be a goal state is also added. Each of the 'models' (stable models or answer sets in case logic programming is used) of the resulting theory, if any, now encode a plan that achieves a goal state from the initial state. If no 'model' exists, it means that there is no plan of length n that can achieve a goal state from the initial state. Thus, once the encoding is done, planning is reduced to 'model finding'.

Our goal in this paper is to use the model based planning paradigm in more realistic planning domains. In particular, we would like to account for: (a) actions that have duration, (b) plans with overlapping actions, (c) resources that can be consumed or produced by actions, (d) machine units that are unsharable when in use, and (e) the availability of alternate machines and resources for an action. There have been some recent planners [22, 9, 1, 18, 11, 10, 12, 16] that allow for some of the above features. *But none of them are pure model based planners.* The closest are LPSAT [22]– where a satisfiability solver is integrated with an incremental simplex algorithm, and ILP-PLAN [9] – where integer linear programming (ILP) is used. In a latter section we give a more detailed comparison between our approach in this paper and the above mentioned planners.

Besides being able to plan in realistic planning domains, we would like to be able to: (i) express various kinds of constraints (such as temporal [2], procedural [13] and hierarchical) which drastically cut down the planning time; (ii) express qualification, ramification and causal constraints which make the representation of state transitions due to actions simpler, shorter and elaboration tolerant; (iii) have the planning program do scheduling without much change; and (iv) specify optimization criteria among plans. For (i)-(iii) it is necessary that we use model based planning using an expressive language.

We choose **L**ogic **P**rogramming with **A**nswer **S**et **S**emantics (or LPASS)[1] [6] as our starting point, over classical logic, because the former is more expressive – can express transitive closure and aggregates, can represent causality more easily; and has been shown to be able to encode various constraints [19]. (A more detailed comparison of LPASS with other logics is detailed in the draft of the book by the first author available at http://www.cs.public.asu.edu/∼cbaral.) To be able to specify optimization criteria, and to be able to do more efficient aggregation using weight constraints we use the Smodels system [15] that implements an extension of LPASS. Although, propositional satisfiability can be directly encoded in ILP, it is not yet known how to directly encode LPASS in ILP. (Note that the method in [4] is not a direct encoding.) Hence, to the best of our knowledge, LPASS$_{smodels}$ (the extension used in the Smodels system) is the only known language that can do model based planning in realistic domains together with (i)-(iv) above.

In this paper we first show – in Section 3 – how planning in realistic domains together with (iii) and (iv) can be done using LPASS$_{smodels}$. (The expression of (i) and (ii) with respect to simple planning, as described in [19, 21], can be directly imported here.) The rest of the paper is structured as follows. In Section 4 we show how RCPS (resource constraint project scheduling) can be encoded using the framework in Section 3. In Section 5 we compare our approach with other planning and scheduling systems. In Section 6 we briefly discuss how to formulate the notion of states and state transitions in a action domain where actions have duration, and we have resources and machines. We then show how part of our encoding in Section 3 can be used to define the state transition. Finally we conclude in Section 7.

2 Background: LPASS$_{smodels}$

An LPASS program is a collection of rules of the form

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

where a_i's are atoms. For an atom a, "$\mathit{not}\ a$" 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. LPASS differs from Prolog in that the ordering of

[1] LPASS is also referred to as A-Prolog and AnsProlog by some. We would appreciate any additional suggestions.

atoms in the body of a rule and the ordering of rules in a file do not matter in LPASS while they matter in Prolog. Moreover, Prolog may get into an infinite loop when the dependency between atoms as dictated by the rules have loops with "*not*" edges.

The semantics of an LPASS program is defined using answer sets. LPASS programs whose rules do not have *not* in the body – referred to as definite LPASS 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 is defined 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 definite LPASS program P^S.*

The LPASS program consisting of the following two rules has a 1-1 correspondence between its answer sets and the possible action occurrences at the various time points.

$o\_occurs(A,T) \leftarrow occurs(B,T), A \neq B.$
$occurs(A,T) \leftarrow action(A), time(T), not\ o\_occurs(A,T).$

LPASS$_{smodels}$, although with the same formal expressiveness of Co-NP as LPASS, allows additional rules such as weight constraints and cardinality constraints. For example, the above two rules can be expressed by the following single rule of LPASS$_{smodels}$.

$1\ \{occurs(A,T) : action(A)\}\ 1 \leftarrow time(T).$

Intuitively, the above rule means that for any particular value t of T, if $action(a_1), \ldots, action(a_n)$ are the only facts about actions then in each answer set exactly one $occurs(a_i,t)$ $(1 \leq i \leq n)$ would be true.

Similarly, the rule $2\ \{occurs(A,T) : action(A)\}\ 5 \leftarrow time(T).$ means that for any particular value t of T, if $action(a_1), \ldots, action(a_n)$ are the only facts about actions then in each answer set between 2 to 5 (inclusively) $occurs(a_i,t)$ $(1 \leq i \leq n)$ would be true. We can also assign weights to each $occur(a_i,t)$. In that case the last rule would mean that for any particular value t of T, if $action(a_1), \ldots, action(a_n)$ are the only facts about actions then in each answer set S the sum of the weights of $occurs(a_i,t)$ $(1 \leq i \leq n)$ that are true in S must be between 2 to 5 (inclusively).

LPASS$_{smodels}$ allows optimization statements of the form

$minimize\ \{occurs(A,T) : action(A) : time(T)\}.$

and $maximize\ \{occurs(A,T) : action(A) : time(T)\}.$

The former (latter) returns the answer set which has the minimum (maximum) sum of the weights for its $occurs$ atoms. Multiple optimization statements are allowed in Smodels and in that case lexicographic ordering is used. More formal characterization of LPASS$_{smodels}$ can be found in [17].

3 Planning and reasoning about actions

In this section we *first* present the LPASS$_{smodels}$ encoding of our planner that can reason about consumable and producible resources, that can account for non-sharable machines, that allows actions with durations, and that produces plans with overlapping actions. We then discuss how optimization criteria can be expressed and how to express alternative resource consumption and machine usage for the same action.

3.1 Describing an example domain

We demonstrate our encoding using an example domain. Our example domain has four actions: a1, a2, a3 and a4; four fluents: p1, p2, p3 and p4; two resources: r1 and r2; and one machine: m.

1. The executability conditions of the actions are described by the following facts.

   $exec(a1, true).\ exec(a2, true).\ exec(a3, s1).\ exec(a4, s2).$

   $in(p2, s1).\ in(p1, s2).\ in(p3, s2).$

   Intuitively, the above means that $a1$ and $a2$ are always executable and $a3$ is executable at a time point if $p2$ is true at that time point and $a4$ is executable at a time point if $p1$ and $p3$ are true in that time point.
2. The effect of the actions on the fluents is described by the following facts.

   $causes(a1, p1).\ causes(a2, p2).\ causes(a3, p3).\ causes(a4, p4).$

   In a simple domain with duration-less (instantaneous) actions $causes(a1, p1)$ means that after the execution of $a1$, $p1$ will be true. When $a1$ is not instantaneous the meaning is a little different. In that case $p1$ becomes true as soon as $a1$ is finished, and *moreover if* $p1$ *was false before* $a1$ *was started then during the execution of* $a1$ *the value of* $p1$ *is 'unknown' or 'in-flux'*. This is to model the fact that we do not know when exactly during the execution of $a1$, $p1$ becomes true. To stay within the the NP-complete class while planning we do not do case analysis with respect to the truth value 'in-flux' (or 'unknown').

   Here, we assume actions do not have conditional effects. This can be modeled by having $causes(a, p, s)$ and $in(p1, s), \ldots, in(pk, s)$ to express the conditional effect a **causes** p **if** $p1, \ldots, pk$ and making appropriate changes in the frame axiom and effect axioms.
3. The effect of actions on the resources is described by the following facts.

   $consumes(a1, r1, 1).\ consumes(a2, r1, 1).$
   $consumes(a3, r1, 1).\ consumes(a4, r1, 1).$

   $produces(a1, r2, 1).\ produces(a2, r2, 1).$
   $produces(a3, r2, 1).\ produces(a4, r2, 1).$

   Intuitively, $consumes(a1, r1, 1)$ means that as soon as the execution of $a1$ starts there is a reduction of the resource $r1$ by 1 unit. It does not mean that one unit of r1 is immediately consumed by $a1$, rather it encodes the conservative reasoning that one less unit of $r1$ is available for other actions. Similarly, $produces(a1, r2, 1)$ means that at the end of execution of $a1$, there is an increase of $r2$ by one unit.

4. The needs of the actions in terms of the machines is described by the following facts.

   $needs(a1, m, 3).\ needs(a2, m, 2).$

   Machines are different from consumable and producible resources in that they only become temporarily unavailable by a particular amount during the execution of particular actions.
5. The duration of the actions is described by the following facts.

   $duration(a1, 3).\ duration(a2, 2).\ duration(a3, 3).\ duration(a4, 4).$
6. The capacity of the machines, the initial situation and the conditions on the final state are described by the following facts.

   $capacity(m, 4).$

   $initially(neg(p1)).\ initially(neg(p2)).\ initially(neg(p3)).\ initially(neg(p4)).$

   $initially(r1, 5).\ initially(r2, 1).$

   $finally(p4).\ finally(r2, 5).$

3.2 The domain independent encoding

In this section we present the LPASS$_{smodels}$ rules[2] that can be added to *any* example domain – that conforms with the syntax of Section3.1, to construct plans for it by finding an answer set of the resulting program. In other words our formulation is not just for planning with respect to the example in Section 3.1. It is a general one. It can of course be further generalized by additional features such as: conditional effects, and constraints that causes ramification and qualification.

1. The following rules encode when the goal – as described by the $finally$ facts – is satisfied at a time point T.

   $not\_goal(T) \leftarrow finally(X), not\ holds(X, T).$

   $not\_goal(T) \leftarrow finally(R, N), not\ holds(R, N, T).$

   $goal(T) \leftarrow time(T), not\ not\_goal(T).$

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

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

3. The following rules define when an action is executable at a time point, based only on the truth of fluents.

   $executable(A, T) \leftarrow not\ not\_executable(A, T).$

   $not\_executable(A, T) \leftarrow exec(A, S), in(F, S), not\ holds(F, T).$

[2] The rules we present are simplified – for presentation and brevity purposes – from the actual Smodels code. Some of the Smodels code is available at http://www.public.asu.edu/∼cbaral/resources/.

4. The following rules define when an action ends and when it is under execution.

   $ends(A, T + D) \leftarrow initiated(A, T), duration(A, D).$

   $in\_exec(A, T) \leftarrow initiated(A, T'), duration(A, D), T' < T \leq T' + D.$

5. The following rules reason about truth value of fluents at different time points. The second rule encodes the delayed effect of an action taking into account the duration of the action. The third and fourth rules encode the frame axiom. They are slightly different from the normal LPASS encoding of the frame axiom so as to take into account action durations and its impact as described in Section 3.1 (part 2).

   $holds(F, 1) \leftarrow initially(F).$

   $holds(F, TD) \leftarrow TD < plan\_size, ends(A, TD), causes(A, F).$

   $ab(F, T + 1) \leftarrow T < plan\_size, initiated(A, T), causes(A, F).$

   $holds(F, T + 1) \leftarrow contrary(F, G), T < plan\_size, holds(F, T),$
   $not\ ab(G, T + 1).$

6. The following rules encode the change in the quantity of consumable and producible resources due to action executions.

   $holds(R, N, 1) \leftarrow initially(R, N).$

   $decrease(A, R, N, T) \leftarrow initiated(A, T), consumes(A, R, N).$

   Intuitively, $decrease(A, R, N, T)$ means that there is a decrease of N units of resource R at time point T due to action A. Note that if A consumes N units of R then the decrease is immediate.

   $weight\ \ decrease(A, R, N, T) = N.$ (*1)

   $increase(A, R, N, T) \leftarrow ends(A, T), produces(A, R, N).$

   Intuitively, $increase(A, R, N, T)$ means that there is an increase of N units of resource R at time point T due to action A. Note that if A produces N units of R then the increase in R does not happen immediately after A is initiated. It happens when the execution of A ends.

   $weight\ \ increase(A, R, N, T) = N.$ (*2)

   $tot\_decrease(R, C, T) \leftarrow resource(R), time(T),$
   $C[decrease(A, R, N, T) : action(A) : int(N)]C.$

   $tot\_increase(R, C, T) \leftarrow resource(R), time(T),$
   $C[increase(A, R, N, T) : action(A) : int(N)]C.$

   The last two rules aggregate the total decrease and increase of each resource R at each time point T. The aggregation is done using the weight constraint expressions $C[decrease(A, R, N, T) : action(A) : int(N)]C$ and $C[increase(A, R, N, T) : action(A) : int(N)]C$. To be able to evaluate these weight expressions the weight declarations (*1) and (*2) are used to assign weights to each atom $decrease(A, R, N, T)$ and $increase(A, R, N, T)$.

$$holds(R,N,T+1) \leftarrow holds(R,NN,T), tot\_decrease(R,C,T),$$
$$tot\_increase(R,CC,T+1), N = NN + CC - C.$$

The last rule takes into account the decrease and increase of resources to define how many units of each resource is available at each time point. The asymmetry in the time parameter in $tot\_decrease(R,C,T)$ and $tot\_increase(R,C,T+1)$ is because, when an action starts in time T, the decrease in resource due to that action takes effect from time $T+1$, while when an action ends at time $T+1$, the increase in resource due to that action takes effect from time $T+1$.

$$\leftarrow holds(R,NN,T), tot\_decrease(R,C,T),$$
$$tot\_increase(R,CC,T+1), NN + CC < C.$$

The last rule is a constraint that prevents particular action executions, in the absence of adequate resources.

7. 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. This need not be the case in general. Our point here is that LPASS allows us to express such restrictions very easily.

$$has\_occurred\_before(A,T) \leftarrow initiated(A,TT),$$
$$duration(A,D), TT < T < TT + D.$$

$$initiated(A,T) \leftarrow executable(A,T),$$
$$not\ has\_occurred\_before(A,T),$$
$$not\ not\_initiated(A,T).$$

$$not\_initiated(A,T) \leftarrow not\ initiated(A,T).$$

8. The following rules are used to make sure that the various actions occurring at each of the time points do not overuse the machine resources. In case of overuse the corresponding answer sets are eliminated.

$$in\_use(A,M,N,T) \leftarrow in\_exec(A,T), needs(A,M,N).$$

Intuitively, $in\_use(A,M,N,T)$ means that N units of machine-resource M are being used at time point T because of action A.

$$weight\ \ in\_use(A,M,N,T) = N.$$

$$resource\_constraint\_violated \leftarrow time(T), capacity(M,C), CC = C + 1,$$
$$CC[in\_use(A,M,N,T) : action(A) : int(N)].$$

Intuitively, the above rule says that for any resource M, and at any time T, if the total number of the M that are being used is more than the capacity C (i.e., greater than or equal to C +1) of that resource, then $resource\_constraint\_violated$ is true.

$$\leftarrow resource\_constraint\_violated.$$

When we run this program we obtain an answer set containing the following action initiations: $\{initiated(a2,1), initiated(a1,3), initiated(a3,3),\ initiated(a4,6)\}$, and the goal is satisfied at the time point 10. The truth value of the fluents at the different time points are as follows: $p1$ is $false$ from 1-3, *in-flux* at 4 and 5, and true at 6 and beyond; $p2$ is $false$ at 1, *in-flux* at 2, and $true$ at 3 and beyond; $p3$ is $false$ from 1-3, *in-flux* at 4 and 5, and $true$ at 6 and beyond; and $p4$ is $false$ from 1-6, *in-flux* from 7-9, and true at 10 and beyond.

3.3 Optimizations

LPASS$_{smodels}$ allows us to easily express various kinds of optimizations. For example, minimization of the total number of action initiations can be achieved by the following two rules:

$$action\_pair(A,T) \leftarrow action(A), time(T).$$

$$minimize\ \{initiated(A,T) : action\_pair(A,T)\}.$$

We can also assign cost to each action by the expression $weight\ initiated(A,T) = w$ and then the above minimization statement will give us the plan with minimal total cost. We can also easily express additional optimization conditions such as maximizing a particular resource at the end, minimizing inventory, etc. The main advantage of LPASS$_{smodels}$ over ILP is that it can easily mix logical (including temporal) conditions with the optimizations. For example, we can express minimization of inventory of those resources that are not touched for a given time period.

3.4 Alternative machines

Another feature that trips ILP formulations is the presence of alternative ways to do the same action. For example, suppose a particular action $a1$ can be done 3 ways: using 3 units of m1, 2 units of m2, and 2 units of m3; or using 2 units of m4 and 6 units of m5; or using 3 units of m2, 1 unit of m3, and 2 units of m5. This can be represented by replacing $needs$ facts by the following $needs\_bag$ facts.

$$needs\_bag(a1,s1).\ in(m1,3,s1).\ in(m2,2,s1).\ in(m3,2,s1).$$

$$needs\_bag(a1,s2).\ in(m4,2,s2).\ in(m5,6,s1).$$

$$needs\_bag(a1,s3).\ in(m2,3,s3).\ in(m3,1,s3).\ in(m5,2,s3).$$

In addition we need the following change in the formulation in Section 3.2.

1. When actions are initiated, we need the following rules which enumerate the different $needs\_bag$ that are selected.

   $$\begin{aligned} &other\_initiated(A,S,T) \leftarrow initiated(A,S',T), S \neq S' \\ &initiated(A,S,T) \leftarrow initiated(A,T), needs\_bag(A,S), \\ &\qquad\qquad\qquad\qquad not\ other\_initiated(A,S,T). \end{aligned}$$

   Intuitively, $initiated(A,S,T)$ means that action A is initiated at time point T and S is the needs_bag that is selected.
2. We now have to redefine in_use so that it takes into account the needs_bag that is selected.

   $$\begin{aligned} &in\_use(A,M,N,T) \leftarrow initiated(A,S,T'), in(M,N,S), duration(A,D), \\ &\qquad\qquad\qquad\qquad T' < T < T' + D + 1. \end{aligned}$$

4 RCPS

In this section we show how RCPS (Resource Constraint Project Scheduling) can be easily done using the framework in the previous section. In RCPS, we are given a set of jobs, a set of resources and a fixed deadline. Each job has a set of tasks (operations) and each task operates uninterruptedly on a resource in a given period of time. Each task may consume some units of an assigned resource. Each resource has a capacity and may accommodate several tasks simultaneously. There are different constraints namely precedence constraint, resource constraint and deadline constraint. The precedence constraint is to enforce tasks occurrence in some pre-defined order. The resource constraint is to prevent all tasks operating on a single resource from over-using that resource. The deadline constraint is to enforce that all the tasks end their operation by the deadline. *A schedule is a mapping from tasks to integral start times*. We need to find a schedule that satisfies these three constraints.

4.1 An example domain

Consider an RCPS problem where the set of resources $\mathcal{R}$ is equal to $\{r_1, r_2\}$, where r_1 has a capacity of 2 units and r_2 has a capacity of 7 units. The set of tasks has six instances $\{j_{11}, j_{12}, j_{13}, j_{21}, j_{22}, j_{23}\}$. All tasks have the same duration of 3 time units. Task resource consumption is described as follows: j_{11} and j_{21} each consumes 2 units of r_1, j_{11} also requires 3 units of r_2. All other tasks, except j_{21}, each consume 2 units of r_2. The set of precedence constraints are: $j_{11} \prec j_{12}$, $j_{11} \prec j_{13}$, $j_{21} \prec j_{23}$ and $j_{22} \prec j_{23}$. We are given a deadline of 15 time units. We need to find schedules that satisfy all three types of constraints for this particular instance of the RCPS problem. This domain is described by the following facts.

1. The capacity of the various resources are expressed by the following:

   $capacity(r1, 2).\ capacity(r2, 7).$
2. The various tasks, their needs and their durations are expressed by the following:

   $task(j11).\ needs(j11, r1, 2).\ duration(j11, 3).$
   $task(j12).\ needs(j12, r2, 2).\ duration(j12, 3).$
   $task(j13).\ needs(j13, r2, 2).\ duration(j13, 3).$
   $task(j21).\ needs(j21, r1, 2).\ duration(j21, 3).$
   $task(j22).\ needs(j22, r2, 2).\ duration(j22, 3).$
   $task(j23).\ needs(j23, r2, 2).\ duration(j23, 3).$
3. The precedence constraints are expressed by the following:
   $\leftarrow$ *not* $precedes(j11, j12).$
   $\leftarrow$ *not* $precedes(j11, j13).$
   $\leftarrow$ *not* $precedes(j21, j23).$
   $\leftarrow$ *not* $precedes(j22, j23).$

4.2 The domain independent encoding

We now describe the domain independent rules which when added to the facts and constraints corresponding to particular domains will give us answer sets, each corresponding to a valid schedule.

1. The following LPASS$_{smodels}$ rule ensures that in an answer set, each task gets initiated at a unique time point, and enumerates the various possible initiations.
   $1\ \{initiated(O, T) : time(T)\}\ 1 \leftarrow task(O).$

2. Enforcing constraints:
   (a) The following rule specifies that task $O1$ precedes task $O2$ if $O2$ starts after the finishing time of $O1$:
   $precedes(O1, O2) \leftarrow task(O1), task(O2), neq(O1, O2), initiated(O1, T1),$
   $duration(O1, D1), initiated(O2, T2), duration(O2, D2),$
   $T1 + D1 \leq T2, T2 + D2 \leq deadline.$
   (b) To express the constraint that tasks can not overuse resources at any time moment we need the rules from part 7, of Section 3.2.
   (c) To constrain that all tasks must finish before the deadline can be expressed by the following rule:
   $\leftarrow task(O), time(T), initiated(O, T), duration(O, D), T+D > deadline.$

We were also able to encode other kinds of scheduling problems, such as the job shop scheduling problem and tournament scheduling problem in LPASS$_{smodels}$.

5 Comparison with other work on planning and scheduling

The following table compares our approach with some of the other related works in terms of the following features: R (resources), MC (machines), AD (actions with durations), OA (plans with overlapping actions), O (optimizations), A (alternatives), C (constraints), S (can do scheduling); 'y' means 'yes' and 'n' means that it has not been shown how to account for the corresponding features. We would like to stress that 'n' only means that the corresponding articles do not mention how to account for the corresponding feature; in many of the cases the feature in question could perhaps be accommodated with small changes, and in other cases perhaps large changes are necessary. Additional analysis is necessary to differentiate between the two cases.

In case of IxTeT actions with durations are allowed when they consider machine usage, plans with overlapping actions are considered for scheduling, and a limited optimization is allowed.

Features Lang./Sys.	R	MC	AD	OA	O	A	C	S
This paper	y	y	y	y	y	y	y	y
LPSAT [22]	y	n	n	n	n	n	n	n
ILP-PLAN [9]	y	n	n	n	y	n	n	n
TL$_{PLAN}$+ [1]	y	n	y	y	n	n	y	n
TGP [18]	n	n	y	y	n	n	n	n
TALplanner+ [11]	y	y	y	y	n	n	y	n
IPP [10]	y	n	n	n	n	n	n	n
IxTeT [12]	y	y	m/c	sch.	limited	y	n	y
[16]	y	n	n	n	n	n	n	n

A quick comparison of the above systems on their other features is as follows: While our approach is model based and uses Smodels, an expressive KR language, LPSAT uses a satisfiability solver integrated with an incremental simplex algorithm; ILP-PLAN uses ILP; TL$_{PLAN}$+ builds on TL$_{PLAN}$ and exploits temporal constraints with forward planning; TGP extends graph plan; TALplanner+ uses procedural search algorithm on top of TAL+ reasoning system; IPP is based on Graphplan; IxTeT is based on least commitment planning and temporal constraint reasoning; and [16] uses techniques from propositional satisfiability testing and is not restricted to forward or backward chaining.

6 Reasoning about actions

In reasoning about actions often one of the main goals is to have an elaboration tolerant representation of state transitions due to action executions. This representation can either be in a high level language with its own semantics (as in the language $\mathcal{A}$ [7]) with correct translations to logical languages such as LPASS, or the representation can be directly in a language like LPASS as in [3]. In either case the logical encoding may be used as part of a planner.

In this paper we gave the encoding of a planner in Section 3.1 and 3.2. We now briefly discuss what our view of a state is and how state transitions are defined when we are dealing with actions that have durations, and in presence of resources and machines; and how this relates to the encoding in Section 3.1 and 3.2. (Due to lack of space we do not give detailed motivation or explore the properties of our transition function here. We plan to do that in the extended version of this paper.)

A state σ is a snapshot of a factory floor. It has information about fluent values (true, false, in-flux), available resources, and what actions are under execution and for how long. Transitions are due to a (possibly empty) set of action initiations after particular time durations, and passage of time since σ. Thus $\Phi(\sigma, \{(a_1, t_1), \ldots, (a_n, t_n)\}, t)$, where $t_i \leq t$, represents the new state reached after t units of time has passed since σ and the actions a_i's are initiated after t_i unit of time since σ. Note that $\Phi(\sigma, \emptyset, t)$ is not necessarily equal to σ, as some of the actions that were under execution in σ have either ended or have been under execution for a longer time than in σ. Although, $\Phi(\sigma, \{(a_1, t_1), \ldots, (a_n, t_n)\}, t)$ can be easily defined when $n = 1$, in the general case, there are too many ordering possibilities to have an independent simple definition. On the other hand, we can define it using some of the LPASS rules in Section 3.1 and 3.2 in the following way.

1. Representing the domain: We represent the domain using rules similar to the ones in part 1-5, and the $capacity$ atoms of part 6 of Section 3.1.
2. Representing the start state σ: The fluents that are true and that are false, and the resource values are represented using the $initially$ predicate as in part 6 of Section 3.1. In addition if an action a is in execution for time duration d at state σ, we write $in\_exec\_for(a, 1, d)$.
3. We define the termination time of actions that are in execution in σ by the following rule:

   $ends(A, T) \leftarrow in\_exec\_for(A, 1, D'), duration(A, D), T = D - D' + 1.$

4. We have the rules defining executability of actions (part 3 of section 3.2), rules about which actions are under execution and action termination (part 4 of section 3.2), rules for reasoning about truth values of fluents (part 5 of Section 3.2), rules for reasoning about resources (part 6 of Section 3.2), and rules for staying within the machine resource capacities (part 8 of section 3.2).
5. We have the following facts to encode the action occurrences dictated by $\Phi(\sigma, \{(a_1, t_1), \ldots, (a_n, t_n)\}, t)$.

   $initiated(a_1, t_1{+}1).$ $\ldots$ $initiated(a_n, t_n{+}1).$

6. Finally we need rules to reason about which actions are under execution and for how long in the state $\Phi(\sigma, \{(a_1, t_1), \ldots, (a_n, t_n)\}, t)$.

$$in\_exec\_for(A, T, T - T') \leftarrow initiated(A, T'), duration(A, D'), T' + D' > T.$$

$$in\_exec\_for(A, T, D + T - 1) \leftarrow in\_exec\_for(A, 1, D), duration(A, D'),\\ D + T - 1 < D'.$$

The holds atoms about the fluent and resources at time point $t+1$, and the $in\_exec\_for(A, t+1, D)$ atoms in the answer set of the above program define the state $\Phi(\sigma, \{(a_1, t_1), \ldots, (a_n, t_n)\}, t)$.

7 Conclusion and future work

In this paper we have shown how several aspects of realistic planning and scheduling together with optimizations can be expressed in LPASS$_{smodels}$. Our focus has been more on knowledge representation than on efficiency. We used a contrived example instead of the standard logistics transport example as the latter does not express all the different features that we were considering. We are of course able to model the logistics example, though the run time were not as good as in [1, 9].

From the knowledge representation point of view we are able to show that LPASS$_{smodels}$, with its prowess to express logical aspects is perhaps a better alternative to ILP and ILP extensions that translate logical specifications to ILP [14]. We hope this will encourage more attempts at – currently there are only 2, Smodels and dlv – faster implementation of LPASS, making LPASS a suitable alternative even from the efficiency point of view. An alternative future direction of research would be to develop direct translations from LPASS$_{smodels}$ to ILP.

Another contribution of our LPASS formulation in this paper is that it presents a solution of the frame problem in presence of resources and actions with duration. We also showed how our formulation can be used to define a state transition in the domain where actions have duration, and we have resources and machines. In the future we will expand on this with independent definitions of Φ for special cases and prove the correctness of our formulation with respect to this alternative definition. Another interesting facet – which we are not able to elaborate for lack of space – is that when we express a production environment domain with the optimization requirement to maximize the production of a certain widget, our formulation gives an answer set that corresponds to pipelining. Thus we are able to derive that pipelining leads to maximizing production.

References

1. F. Bacchus and M. Ady. Planning with resources and concurrency – a forward chaining approach. In *AAAI-2000 workshop on the Integration of AI and OR Techniques for Combinatorial Optimization*, 2000.
2. F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 16:123–191, 2000.
3. C. Baral and M. Gelfond. Reasoning agents in dynamic domains. In J Minker, editor, *Logic Based AI*. Kluwer, 2000.
4. C. Bell, A. Nerode, R. Ng, and V.S. Subrahmanian. Mixed integer programming methods for computing non-monotonic deductive databases. *Journal of ACM*, 41(6):1178–1215, 1994.

5. Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in non-monotonic logic programs. In *Proc. of European conference on Planning*, pages 169–181, 1997.
6. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080. MIT Press, 1988.
7. M. Gelfond and V. Lifschitz. Representing actions in extended logic programs. In K. Apt, editor, *Joint International Conference and Symposium on Logic Programming.*, pages 559–573. MIT Press, 1992.
8. H. Kautz and B. Selman. Planning as satisfiability. In *Proc. of ECAI-92*, pages 359–363, 1992.
9. H. Kautz and J. Walser. State-space planning by integer optimization. In *Proc. of AAAI 99*, pages 526–533, 1999.
10. J. Koehler. Planning under resource constraints. In *Proc. of ECAI'98*, 1998.
11. J. Kvarnstrom, P. Doherty, and P. Haslum. Extending talplanner with concurrency and resources. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-00)*, 2000.
12. P. Laborie and M. Ghallab. IxTeT: an integrated approach for plan generation and scheduling, 2000.
13. Y. Lesperance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. A logical approach to high level robot programming – a progress report. In *Working notes of the 1994 AAAI fall symposium on Control of the Physical World by Intelligent Systems, New Orleans, LA*, November 1994.
14. Q. Li, Y. Guo, and T. Ida. Modeling integer programming with logic: Language and implementation. *IEICE Transactions*, E83(8):1673–1680, 2000.
15. I. Niemela and P. Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. 4th international conference on Logic programming and non-monotonic reasoning*, pages 420–429, 1997.
16. J. Rintanen and H. Jungholt. Numeric state variables in constraint-based planning. In *Proc. European Conference in Planning (ECP 99)*, 1999.
17. P. Simons. Extending the stable model semantics with more expressive rules. In *Proc. of International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'99*, 1999.
18. D. Smith and D. Weld. Temporal planning with mutual exclusion reasoning. In *Proc, of IJCAI'99*, pages 326–337, 1999.
19. T. Son, C. Baral, and S. McIlraith. Extending answer set planning with sequence, conditionals, loops, non-deterministic choice and procedural constructs. In *Proc. of AAAI Spring symposium on Answer Set Programming*, 2001.
20. V. Subrahmanian and C. Zaniolo. Relating stable models and ai planning domains. In L. Sterling, editor, *Proc. ICLP-95*, pages 233–247. MIT Press, 1995.
21. L. Tuan and C. Baral. Effect of knowledge representation on model based planning: experiments using logic programming encodings. In *Proc. of AAAI Spring symposium on Answer Set Programming*, 2001.
22. S. Wolfman and D. Weld. The LPSAT engine and its application to resource planning. In *Proc, of IJCAI'99*, pages 310–317, 1999.