Planning with Fully and Partially Specified Initial State Facts Using Dlv^k

Asim Ali Shah
Department of Intelligent Systems Computer Science Institute,
University of Leipzig Augustusplatz 10/11, 04109 Leipzig -Germany
asimali@informatik.uni-leipzig.de

Abstract
Planning allows one to sequence a series of actions to achieve a certain goal. The paper presents a short overview of Disjunctive Logic Programming (DLP) under the answer set semantics an advanced formalism for knowledge representation and reasoning. The DLV^K system, which implements the declarative planning language K on top of the DLV logic programming system is used to implement a new approach to knowledge representation and reasoning, in order to compute solutions to the planning problem considering the blocks world planning domain under fully and partially specified initial state facts. We compute the stable models (answer sets) and compare the performance of the system in run time CPU seconds while increasing the plan length. We also use the security check feature provided by the DLV^K system to verify whether the plan generated is secure or not. The work presented in this paper contributed mainly as a technique that is compatible with extensions and improvements of the existing system rather than as a concrete planning system.

Keywords: *Dlp, World State, Knowledge State, Dlv^k, Background Knowledge, Blocks World Domain, Answer Set Semantics, Declarative Planning Language*

INTRODUCTION

"*An automatic programming the design of a course of action that when executed will result in the achievement of some desired goal.*" Recently several declarative planning languages and formalisms have been introduced, which allow for an intuitive encoding of complex planning problems involving ramifications, incomplete information, non-deterministic action effects, or parallel actions [Giunchiglia & Lifschitz, 1998; Lifschitz, 1999b; Lifschitz & Turner, 1999; McCain & Turner, 1998; Giunchiglia, 2000; Cimatti & Roveri, 2000; Eiter *et al*., 2000b, 2003b]. Planning was one of the main areas that motivated the development of nonmonotonic reasoning systems and consists of a set of operators or action types. Each operator may be executed only in some particular set of world states (its preconditions), and has some particular set of effects on its world state (its effects). A planning problem consists of a planning domain together with an initial state of the world, and a desired goal state of the world. A planner solves a planning problem by producing a sequence of actions, each of which is legal in it's starting world state, which takes the initial state to a goal state. An often-cited example of a planning domain is the infamous blocks world, a model of stacks of blocks on an infinite table. An action in planning domain is applicable only if some preconditions hold in the current state executing this action changes the current state by modifying the truth-values of some fluents. In planning domain possibly one do not have a complete view of the world, so possibly he has to reason with incomplete knowledge in an environment and should interact with others by means of communication. For instance, robots usually do not have a complete view of the world and in case if their knowledge is complete a number of fluents may be unknown, the sensors provide the information required by the robot to perform its task and tells the robot when to execute which action despite uncertainty in sensing and control. In [11] a declarative planning language *K* is proposed which is closer in spirit to answer set semantics than to classical logics. A very flexible language and is capable of modelling transitions between states of the world (i.e., states of complete knowledge). While its way of representing planning problem in a declarative way it is more capable of representing transitions between incomplete states of the knowledge. By implementing the planning language *K* on top of the DLV[1] system a prototype

[1] http://www.dbai.tuwien.ac.at/proj/dlv/

system DLV^{K2} is developed which works as a front-end to the DLV logic programming system. Generally it comes with two parsers, one accepts the files with a filename extension ***.plan*** and the other accepts files with ***.bk*** extension it is a command line oriented system and is invoked by an option *–FP*. The basic aim of this paper is to reduce planning in language *K* to answer set programming and test the system DLV^{K} against the blocks world planning domain from the literature under fully and partially specified initial state facts that leads to the goal state, and check the performance of the system execution on the problem in run time CPU seconds while increasing the plan length. Also we use the security check feature provided in [8] to know whether the plan generated is secure or not. The road map for rest of the paper is organized as follows; Section 2 presents short overview about disjunctive logic programming syntax and semantics. In order to have a look at the key components of the DLV^{K} a short outlook is given in Section 3. Section 4 explains knowledge representation in DLV^{K}. Section 5 presents results of the experimental work and performance of the DLV^{K} system is discussed in Section 6. Section 7 presents conclusions about the work and Section 8 describes acknowledgements of the paper and finally we close with references in Section 9ss.

DISJUNCTIVE LOGIC PROGRAMMING

In [Koch *et al*., 2003] Disjunctive logic programming together with stable model semantics is a one of the powerful nonmonotonic formalism using for knowledge representation and reasoning. Providing a high-level overview of DLV implementation under the consistent answer set semantics, a view of logic programs as sets of inference rules (default inference rules), where a stable model is a set of literals closed under the program itself [Gelfond & Lifschitz, 1988, 1991].

2.1 Syntax

A variable or constant is a term. An atom is of the form p(t1 ,..., tn), where p is a predicate of arity n ≥ 0 and t1 ,..., tn are terms, for nullary predicates (n = 0) we usually omit the parentheses. A classical literal is an atom a or a classically negated atom $\neg$a. A negation as failure literal is either a positive literal *c* or a negative literal not *c*, where *c* is a classical literal. A disjunctive rule *r* is of the form,

$$a1\ v....v\ an \leftarrow b1 \wedge .... \wedge bk \wedge not\ bk+1 \wedge .... \wedge not\ bm \quad n \geq 1, m \geq 0$$

where *a1 .... an , b1.... bm* are classical literals and rule *r* needs to be safe, i.e., each variable occuring in rule *r* must also appear in one of the positive body literals *b1.... bk* as well. The disjunction *a1 v....v an* is the head of rule r, while the conjunction *b1* $\wedge$.... $\wedge$ *bk* $\wedge$ *not bk+1* $\wedge$.... $\wedge$ *not bm* is the body of rule *r*.

H(r) denotes the set of head literals {a1,....,an} and *B(r)* the set of body literals {*b1.... bk , not bk+1.... not bm*}. *B+(r)* (resp., *B-(r)*) denotes the set of classical literals occurring positively (resp., negatively) in *B(r)*: *B+(r)* = {b1,.....,bk} and *B-(r)* = {bk+1,....,bm}. Constraints are rules with an empty head (n = 0). A program is a finite set of rules (including constraints). A not-free (resp., v-free) program is called positive (resp., normal). A literal, a program or a rule is called ground if not containing any variables.

2.2 Semantics

Usually, the Herbrand Universe *UP* is the set of all constants appearing in a program P and the Herbrand Base *BP* is the set of all ground (classical) literals constructible from the predicate symbols appearing in P and the constants of *UP*. A partial (or three valued) interpretation w.r.t. a program *P* is a pair (*T*,*F*) of subsets of *BP*. For x $\in$ *T*, x $\in$ *F*, and x $\in$ *BP* – (*T* $\cup$ *F*) it is to say x is true, false and undefined respectively, *I*′ = (*T* ′ , *F* ′) is an extension of *I*, if *T* ′ $\supseteq$ *T* and *F* ′ $\supseteq$ *F*. A total interpretation *I* is a consistent partial interpretation where *T* $\cup$ *F* = *BP* it is often simply represented as *I* = *T*. For Total and partial models see [Prz90, JNS+03]. For any rule *r*, the Ground Instantiation Ground(*r*) denotes the set of rules obtained by applying all possible substitutions

[2] http://www.dbai.tuwien.ac.at/proj/dlv/K/

δ from the variables in r to elements of *UP*. According to Lifschitz [96] the answer sets of a program *P* can be defined in two steps using the ground instantiation Ground (*P*), first the answer sets of positive programs and then a reduction of programs containing negation as failure to positive ones by deleting all rules $r \in P$ for which $B\text{-}(r) \cap I \neq 0$ holds and deleting the negative body from the remaining rules. Then by using that to define answer sets of arbitrary programs.

3 DLV^K IN SHORT

In [11] the key components for the DLV*K* system are described, here we are giving a short view about some of these:

(1) parallel action execution
Allows an execution of more than one action at a time. And these actions should qualify with the help of an executability condition.

(2) weak and strong negation
Allows feature of using weak and strong negation in the rules. Intuitively, the use of weak negation allows for a simple and statement of inertia rules for fluents. The negation as failure construct allows for expressing defeasible rules and default conclusions, by which a more natural modeling of rational planning agents, which have to deal with incomplete information, becomes possible at qualitative level.

(3) complete and incomplete information
States are consistent sets of ground literals by default, not necessarily every atom must appear so representing states of knowledge. But instead, DLV^K also allows representation of transitions between possible states of the world.

(4) background knowledge
Referred to as a static knowledge, which is represented by a logic program. Includes rules and facts that containing program defines predicates, which are normally not subject to change.

4 KNOWLEDGE REPRESENTATION WITH DLV^K

Representing knowledge in DLV*K* we turned to a puzzle called blocks world, the system would typically find a procedure for solving a problem by breaking up the lengthy search for a solution into a sequence of achievable interim goals. Blocks world requires sorting initial state of the blocks using world state and knowledge state technique in achieving the goal state.

4.1 World State (Fully Specified Initial State Facts)

Let's consider the well-known blocks world-planning domain in [8] with complete initial knowledge that involves blocks and a table. The blocks world domain consists of the planning problem with an extension ***.plan*** and the optional background knowledge ***.bk***. The blocks world configuration is represented in Fig **1**.
The problem involves sorting the initial state of the blocks in a way to achieve a stack of blocks that represents the goal state.

initial: **goal:**

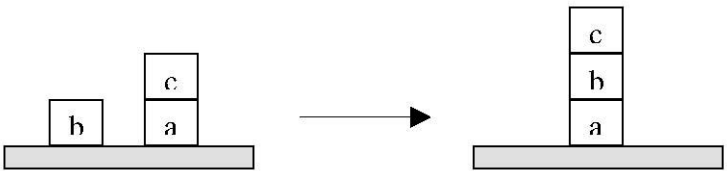

Fig 1: Block's world planning problem with complete initial state

The encoding of blocks world domain for *world state* is as follows,
fluents: on(B,L) requires block(B), location(L).

occupied(B) requires location(B).
actions: move(B,L) requires block(B), location(L).
initially: on(a,table). on(b,table). on(c,a).
always: caused occupied(B) if on(B1,B), block(B).
executable move(B,L) if not occupied(B), not occupied(L),
B <> L.
nonexecutable move(B,L) if occupied(B).
nonexecutable move(B,L) if occupied(L).
noConcurrency.
caused on(B,L) after move(B,L).
caused -on(B,L1) after move(B,L), on(B,L1), L <> L1.
inertial on(B,L).
goal: on(c,b), on(b,a), on(a,table) ? (3)

The static background knowledge □ consists of the following rules and actions:

block(a). block(b). block(c).
location(table).
location(B) :-block(B).

The execution of above encoding with DLV^K computes the following plan that describes the goal state:

PLAN: move (c, table), move (b, a), move (c, b)

4.2 Knowledge State (Partially Specified Initial State Facts)

In the case where information is incomplete but the missing quantities are required for the problem solution, the system must be able to hypothesise a constrained range of values for the unknown quantity. Similarly the configuration for blocks world domain with incomplete initial knowledge represented in Fig **2**. A short description of the problem is as follows; this time a further block *d* is added to the planning domain and the background knowledge while the intuition that whether the block is located on the table or on the stack of blocks is not known.

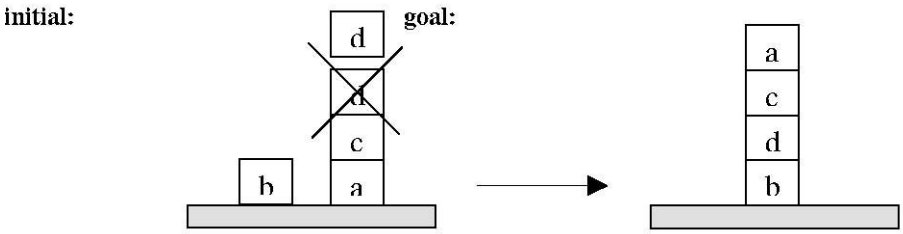

Fig 2: Block's world planning problem with incomplete initial state

The encoding of blocks world domain for knowledge state is as follows,

fluents: on(B,L) requires block(B), location(L).
occupied(B) requires location(B).
supported(B) requires block(B).
actions: move(B,L) requires block(B), location(L).
initially: on(a,table). on(b,table). on(c,a). -on(d,c).
total on(d,Y).
forbidden on(B,L), on(B,L1), L <> L1.
forbidden on(B1,B), on(B2,B), block(B), B1 <> B2.
caused supported(B) if on(B,table).
caused supported(B) if on(B,B1), supported(B1).
forbidden not supported(B).
always: caused occupied(B) if on(B1,B), block(B).s
executable move(B,L) if B <> L.
nonexecutable move(B,L) if occupied(B).
nonexecutable move(B,L) if occupied(L).

noConcurrency.
caused on(B,L) after move(B,L).
caused -on(B,L1) after move(B,L), on(B,L1), L <> L1.
inertial on(B,L).

goal: on(a,c), on(c,d), on(d,b), on(b,table) ? (4)

The encoding for the static background knowledge of the blocks world domain with incomplete initial state is as follows,

block(a). block(b). block(c). block(d).
location(table).
location(B) :-block(B).

The execution of above encoding with DLV^K computes the following resulting plan:

PLAN: move (d, table), move (d, b), move (c, d), move (a, c)

This plan is valid on all possible initial legal states and hence the effects of all actions are determined.

EXPERIMENTAL RESULTS

Our experimental results for blocks-world encoding in Section 4 under complete and incomplete initial facts are given below,

(1) State with complete initial knowledge

In order to specify the goal situation of just one big tower we need to fix the length of the plan in order to disallow all answer sets (possible plans) where the blocks are not in the desired position in the goal situation and then invoke DLV^K that generate the plan for the desired plan length. The results for initial run for planlength (3) are summarized in Table **1**. The execution successfully terminates with Cpu run time 1.468000s and we got only one plan that is secure. A plan *p* is called secure plan iff for all possible legal initial states s0 executability of *p* is guaranteed and leads to the goal see [12] for further reading.

Table 1: Block's world with complete initial state

ws	plan length	no. of plans	secure plans	in-secure plans	running time (sec)
1	3	1	1	0	1.468000s
2	4	11	11	0	11.708000s

Similarly, for next plan length (4) invoking DLV^K generates 11 plans with Cpu run time 11.708000s seconds and the answer to the question whether to search for other plans results positively and each time the check was secure.

(2) State with incomplete initial knowledge

This time we consider the blocks world domain with incomplete initial information as discussed in Section 4.2. For the initial run we specify the plan length (4) by invoking DLV^K the first attempts we arrive at is not secure and we have been asked once again to answer the query whether to search for other plans (y/n)? by pressing "y" at each query the system executes 34 plans among which are 30 in-secure plans while only 4 secure as shown in Table **2** and the process finishes in 30.525000s Cpu seconds.

Table 2: Block's world with incomplete initial state

ks	plan length	no. of plans	secure plans	in-secure plans	running time (sec)
1	4	34	4	30	30.525000s
2	5	203	46	157	151.819000s

Simultaneously, for plan length (5) the execution process successfully terminates with CPU run time 151.819000s including 203 plans. Responding positively to the query for security check

identify that only 46 among 203 are secure and the rest 157 are not secure plans.

6 PERFORMANCE

The results summarized in Table **1** showed that DLV^K under the world-state encodings on Blocks World domain is significantly faster and more efficient than that of the results showed in Table **2** for blocks world domain under the knowledge-state encoding. In these experiments the sensitivity to increasing plan length observed, where execution times seems to grow drastically as the plan length increases.

7 CONCLUSIONS

Language *k* is very expressive and can be checked against complete (fully) and incomplete (partial) initial states offers proven concepts from logic programming to represent knowledge about the action domain. Our contribution in experimental Section **5** from viewpoint of increasing planlength give a promising intuition about the DLV^K system performance in run time CPU seconds that it can handle efficiently planning problems of short plan length while having decrease in efficiency due to increase in time taken by the system to execute the plans on increasing plan length.

8 ACKNOWLEDGEMENTS

The current research is funded by a grant from the German Research Council DFG (Deutsche Forschungsgemeinschaft), which is gratefully acknowledged.

9 REFERENCES

[1] E. Erdem, "Applications of Logic Programming to Planning"; *Computational Experiments*, 1999, un-published draft. http://www.cs.utexas.edu/users/esra/papers.html.

[2] V. Lifschitz, "Answer Set Planning"; In Schreye, D. D. (Ed.), *ICLP'99*, The MIT Press, Las Cruces, New Mexico, 1999b, pp.23-37.

[3] N. Kushmerick, S. Hanks, and D. S. Weld, "An Algorithm for Probabilistic Planning"; *Artificial Intelligence 76*(1-2), 1995, pp.239-286.

[4] G. Pfeifer, W. Faber, N. Leone, and G. Ielpa, "Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV"; *INFSYS Research Report* 1843-03-07, 2003.

[5] A. Cimatti, and M. Roveri, "Conformant Planning via model checking"; In Proceedings of the *Fifth European Conference on Planning* ECP, 1999, pp.21-34.

[6] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres, "Planning under Incomplete Knowledge"; *First International Conference, Proceedings, J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, Eds. Number 1861 in Lecture Notes in AI (LNAI)*. Springer Verlag, 2000, pp.807-821.

[7] W. Faber, N. Leone, and G. Pfeifer, "Pushing goal derivation in dlp computations"; In Proceedings of the *5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99), M. Gelfond, N. Leone, and G. Pfeifer, Eds. Number 1730 in Lecture Notes in AI (LNAI)*. Springer Verlag, El Paso, Texas, 1999, pp. 177-191.

[8] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A Polleres, "A logic programming approach to knowledge-state planning"; *Artificial Intelligence*, Vol. 144(1-2), 2003, pp.157-211.

[9] I. Niemelä, "Logic programming with stable model semantics as constraint programming paradigm" *Annals of Mathematics and Artificial Intelligence*, Vol. 25(3-4), 1999, pp. 241-273.

[10] M. A. Peot, "Decision-Theoretic Planning"; Ph.D. thesis, Stanford University, Stanford, CA, 1998.

[11] T. Eiter, W. Faber, N. Leone. G. Pfeifer, A. Polleres. "A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity"; Report-on: *INFSYS RR-1843-01-11*, *Artificial Intelligence.*, Vol. 48, 2004.

[12] A. Polleres, "The DLV^K System for Planning with Incomplete Knowledge"; M.S. thesis, *Institut für Information systeme*, Technische Universität Wien, Wien, Österreich, 2001.

[13] V. Subrahmanian, and C. Zaniolo, "Relating stable models and AI planning domains"; In Proceedings of the *12^{th} International Conference on Logic Programming*, L. Sterling, Ed. MIT Press, Tokyo, 1995, pp.233-247.

[14] J. D. Ullman, "Principles of Database and Knowledge Base Systems", Computer Science Press., Vol. 1, 1989.

[15] Y. Dimopoulos, B. Nebel, and J. Koehler, "Encoding planning problems in nonmonotonic logic programs"; *In Proceedings of the European Conference on Planning* (ECP-97). Springer Verlag, 1997, pp.169-181.

[16] P. M. Dung, "On the relations between stable and well-founded semantics of logic programs". *Theoretical Computer Science.*, Vol. 105,1992, pp. 7-25.

[17] C. Baral, V. Kreinovich, and R. Trejo, "Computational complexity of planning and approximate planning in the presence of incompleteness"; *Artificial Intelligence.*, Vol. 122, 1-2, 2000, pp.241-267.

Received: 2004-12-21