Modeling and planning through answer set programming

Fernando Zacarias Flores, Rosalba Cuapa Canto, Marquez Rodriguez Beatriz, Angeles López María José

***Abstract*— The line of research on the frame problem start in 1980, when formal non-monotonic reasoning arises. Recently, planning applications are being modeled with answer set programming. This allows us to represent a given computational problem through a logical program. Finally, we can use Smodels or DLV, to find an answer set for this program. Thus, a stack-type parking planning system is modeled using this methodology.**

***Index Terms*—Planning, Answer Set Programming, Parking, Action Language.**

I. INTRODUCTION

Answer set programming is a new paradigm used to represent a given computational problem through a logic program whose answer sets correspond to its solution. Answer set programming has been applied in several areas: reasoning about actions and changes, planning, configuration, wire routing, phylogenetic inference, semantic web, information integration, etc. Research on planning requires the resolution of two key problems: First, declarative and elaboration tolerant languages to describe planning domains. Second, efficient and scalable reasoning algorithms. Action Description Languages are formal models to represent knowledge on actions and change [1]. These specifications are given through declarative assertions that permit to describe actions and their effects on states. Furthermore, to express queries on the underlying transition system.

In this paper, we use a novel paradigm based on the plans generation by reducing this problem to find a satisfactory interpretation for a set of propositional formulas. Thus, reducing a planning problem to the problem of finding a stable model ("answer set") for a logic problem. One of the outstanding characteristics of this paradigm is that the representation of properties of actions is easier when logic programs are used instead of axiomatizations in classical logic, in view of the nonmonotonic character of negation as failure.

DLV and Smodels are two answer set solvers available today [1], [2]. Both proposals are based on anwer set programming – ASP. In this paper we show through examples how action languages like K allow to formalize complex planning problems involving non-determinism and incomplete knowledge in a very flexible manner. Language K adopts a logic programming view where fluents representing the epistemic state of an agent might be true, false or undefined in each state.

Fernando Zacarias Flores, Computer Science, Benemérita Universidad Autónoma de Puebla, Puebla, Méxio, 2222295500, 2224530177

Rosalba Cuapa Canto, Faculty of Architecture, Benemérita Universidad Autónoma de Puebla, Puebla, Mexico, 2222295500

Marquez Rodriguez Beatriz and Angeles López María José, Computer Science, Benemérita Universidad Autónoma de Puebla, Puebla

II. BACKGROUND

First, we select the K language for modeling our proposal due to its flexibility and its ability to model transitions between world states and reason about them as a particular case. Besides, K is closer in spirit to answer set semantics [3] than to classical logics. Also, we show through examples how action languages like K allow to formalize complex planning problems involving non-determinism and incomplete knowledge in a very flexible manner.

Our case of use in this paper is a stack-type parking planning system. This type of problem arises in the historic centers of cities considered World Heritage (as Puebla, México). This systems type allows to model and automate a real parking. Planning involves the representation of actions and world models, reasoning about the effects of actions, and techniques for efficiently searching the space of possible plans.

A. Basic definitions

In this paradigm, the planning consists of a description of a world, in which the initial situation is defined, and a desired situation. The objective is to find a sequence of actions. (which can change the situations), such that the desired situation is reached. In addition, not all actions are applicable in every situation.

Thus, a planning problem is modeled in the following form:

- First, is necessary to define a set of fluents, which characterize the situations such as initial configuration or final situation (goal).
- Second, we must define the set of actions, with a definition of their respective preconditions and effects.
- Third, it is necessary to define the configuration of the initial situation from which it will start (state 0). It is a set of fluents describing the initial situation.
- Fourth, we have to define a set of fluents describing the desired situation or goal.
- Finally, it is necessary to determine the objects involved in planning system as well as the auxiliary rules necessary for the proper system functioning.

Right away, using the definition given in [4], [5], fixed-length solutions to a planning problem should be calculated as follows:

A planning problem is defined as a pair of a planning environment PE and a query q, which specifies the goal. A planning problem is represented as a combination of a background knowledge, which is a stratified Datalog program, and a program as described above. For a given

planning problem P and an integer n that defines the plan length that we want to find, a plan is a sequence $a_1, \ldots, a_n$ such that there are n + 1 situations $S_0, \ldots, S_n$, such that for each a_i, S_{i-1} is consistent with a_i's preconditions, and Si is modified from Si-1 by exactly the effects of a_i.

III. Classical Planning Problems

If In organizations, industrials zones and corporations that have extensive facilities, also known as campuses, is very common that people have to find some particular site, for instance, how go from electronic department to general library? (see figure 1).

Figure 1. Campus of Autonomous University of Puebla

Thus, it's important that we understand the general progression of technology and try to plan for innovation at each stage of its life cycle. In figure 2 we show the interface of the mobile application developed (in apple's iPAD 3 with retina display) for our campus "Autonomous University of Puebla". Next, we can define the domain our problem as follow:

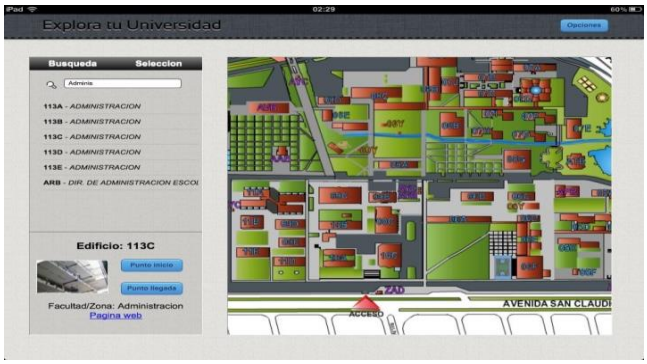

Figure 2. Main system interface

fluents:
% Fluents represent basic properties of the world
% which can change over time.
% They are comparable to first-order predicates or
% propositional assertions.
start(X2, Y2) requires campus(X2,Y2).
user(X3, Y3) requires campus(X3,Y3).
position_building(X1,Y1) requires campus(X1,Y1).
position_lake(X4, Y4) requires campus(X4,Y4).
actions:
% Actions represent dynamic momenta of the world, and
% their execution can % change the state of the world
% (of knowledge).
reachTarget.
moveRight costs 1.
moveLeft costs 1.
moveUp costs 1.
moveDown costs 1.

In this case, the actions have a cost (measured in meters). in this application each movement represents the distance between the starting point and the point reached.

Each move has costs 1, resulting in plans, where a minimum number of moves are executed to achieve the plan, furthermore, user gets a route and this can be recalculated at any time. The rules following the declarations of actions and fluents (always) describe the transitions and constraints on the initial states of the domain (for space reasons we do not present all code). Finally, the goal section defines the goal to be reached and the plan length.

always:
% always represents the transitions, these are atomic
% changes, represented by a previous state, a set of
% actions, and a resulting state.

executable reachtarget if position_target(X,Y), user(X,Y).
caused -targetdown after reachtarget.

caused user(X,Y) after reachtarget, user(X,Y).
caused userlive after reachtarget, userlive.

goal: -targetdown, user(E,E), userlive ? (12)

The rule goal, defines the goal (desired target) to be reached and the plan length.

A. *Execution in the campus domain in Dlv^K*

In general, assume that the above background knowledge and planning program are giving in files cu.bk and cu.plan, respectively. The execution of the command is:

C:\ dlv cu.bk cu.plan –FP –n=1
% Computes the result on server and this is sent to the
% mobile device in no more than 30 segs.

PLAN: moveUp; moveUp; moveUp; moveUp; moveRight; moveRight; moveDown; moveDown; moveLeft; reachtarget; COST 9.

IV. Parking Robotization for Historic Cities

Advances in mobility are clearly illustrated by the rapid development of urbanization in developing countries. The parking problem has been becoming much more seriously important in many metropolises, particularly in cities declared world heritage. With the aim of seeking solutions as to how the parking system could operate more efficiently by using new paradigms and new methodologies such as answer set programming – ASP.

The parking problems urge that the traffic professionals should seek more efficient solutions as to how the parking system could be used more efficiently and how parking planning and management could be improved by using new paradigms and new methodologies. Recently, action languages have received considerable attention in solving planning problems, such as those required in a parking

system. Some relevant action languages such as DlvK and Smodels have grown lately, due to their expressive power and efficiency in solving planning problems

A. *The advantage of answer set programming*

The answer set programming approach to planning is that the representation of properties of actions is easier when logic programs are used instead of axiomatizations in classical logic, coupled with the non-monotonic character of negation as failure. Some results using Smodels for planning are reported in [6], [7]. Furthermore, answer set programming is a novel approach to knowledge representation and reasoning. ASP enables default reasoning, which is necessary in commonsense reasoning. It supports event calculus reasoning and handles some types of event calculus formulas whose circumscription cannot be computed using predicate completion. Also, asp include effect constraints, disjunctive event axioms, and definitions of compound events.

B. *Answer Set Solvers*

The System DLV compute answer sets for finite programs without negation as failure in the heads of rules. On the other hand, Smodels requires additionally that its input program contain no disjunctions. This limitation can be overcome by two circumstances:

First, the input language of SMODELS allows us to express any "exclusive disjunctive rule", that is, a disjunctive rule as follow:

L1 ; . . . ; Ln ← Body. accompanied by the constraints

← Li, Lj , Body ($1 \leq i < j \leq n$).

Second, SMODELS allows us to represent the important disjunctive combination in the head of a rule by enclosing L in braces: {L}

Answer set programming has found applications to several practically important computational problems. One of these problems is planning.

V. MANAGEMENT PLANNING SOLUTIONS FOR TRAFFIC OPERATION EFFECTIVENESS

The parking problems urge that the traffic professionals should seek more efficient solutions as to how the parking system could be used more efficiently, and how parking planning and management could be improved by using new technologies and new methodologies. The main reasons for parking problems in Puebla can be concluded as the disparity between the supply of parking facilities and parking demand. The supply of new parking facilities, including sidewalk parking, has completely congested the historic centers of many cities and Puebla is not the exception.

The high-density parking configuration - where inter-vehicle distance is kept to a minimum - improves considerably land use. In order to make parking planning strategy be efficient in most situations, instead of processing it immediately we hold parking queries in a queue for a while and the number of queries we hold is a controllable parameter.

VI. CASE STUDY FOR PARKING PLANNING

The high-density parking system demands solving the following tasks: (a) selection of vehicle destination i.e., final parking position, (b) conflict-free motion planning for vehicle input and vehicle output and (c) variable vehicle size compacting. In conflict-free path planning it is ensured that the trajectories of vehicles are not overlapping in time and space.

As we mentioned earlier, our methodology is based on the logic programming paradigm and the front-end know as action language called "*K*". This paradigm allows us to model transitions between the knowledge states. Thus, DLVK is a knowledge-based planning system. It is based on the declarative language K, which is similar in spirit to the logic-based language C, but includes some logic-programming features (e.g., default negation and strong negation). K offers the following distinguishing features:

1) nondeterministic effects: actions may have multiple possible outcomes.
2) handling of incomplete knowledge: for a fluent f, in a state neither f nor its opposite ¬f may be known.
3) optimistic and secure (conformant) planning: construction of a "credulous" plan or a "sceptical" plan, which works in all cases.
4) parallel actions: More than one action may be executed simultaneously.

The general system functioning is shown in figure 3 and figure 4. Figure 3 shows the initial state of the parking lot, where you can see that the requested car is marked by the red circle.

Figure 3. stack parking in downtown Puebla

Next, figure 4 shows how the objective of delivering the car indicated outside the parking lot is achieved. Finally, the three cars (on the left side of figure 4) that are outside the parking lot must be returned to the parking lot.

Figure 4. attending car request

A. Parking System modeling

As we describe in section II, a planning problem is modeled by defining the following 4 sections: a set of fluents, a set of actions, define the initial situation and goal and finally, the objects involved in planning system.

First, we must define *the objects* involved in the planning problem as follows:

car(a). car(b). car(c). car(d). car(e). car(f). car(g). car(h). car(i). car(j). car(k). car(l). car(m). car(n). car(o). car(p). car(q). car(r). car(s).

true.

location(calle) :- true. % the street is defined as infinite

location(B) :- car(B). % cars occupy a place

Second, we define set of fluents, these allow us to characterize the world, i.e., predicates describing relevant properties of the domain of discourse. In this context, fluents necessarily is either true or false.

***fluents*:** on(B, L) requires car(B), location(L).

occupied(B) requires location(B).

The fluent "on" allows us to describe where each of the cars in the parking lot are located. Complementary to the fluent "on" we define the fluent "occupied".

Third, define set of actions, the actions allow us to modify the world through the execution of them. For this reason, it is very important to consider the causes that the execution of each action causes on the context where they are executed.

***actions*:** in(B, L) requires car(B),location(L),on(B,L1),

L1==street.

out(B, L) requires car(B), location(L), L==street.

As we can see, the action "in" defines the action of putting a car into the parking lot. To be able to execute the action "in" it is required to have a car, a place available inside the parking lot and the car must be on the street. On the other hand, the action "in" is complementary to the action "out" and this requires that the following requirements be met: a car, a place available in the street (this is always true since the street was declared infinite).

Among the effects caused and considered in our modeling are the following:

executable in(B,L) if not occupied(B), not occupied(L),B<>L.

The execution of action "in" requires that both car and place in the parking lot be free to put it in. In addition, verifying that both "B" and "L" are not equal guarantees the application of the action in the correct way.

caused occupied(B) if on(B1,B), car(B).

This action modifies our world by indicating that the place inside the parking lot was occupied "occupied(B)" and is characterized by the fluent "on (B1, B)".

caused on(B,L) after in(B,L).

This other cause modifies the configuration of our world indicating that the car "B" has been parked in the place "L" inside the parking lot.

executable out(B1,L1) if not occupied(B1).

On the other hand, the execution of the "out" action unlike the "in" cause does not need to require that the street "L" is not occupied, because it is considered infinite, so there is no problem of space, i.e, there is always a place. Here, it is important to note that the causes generated by the execution of these actions are similar to those of the previous one, except that the street is not occupied because it is infinite.

Finally, we define the initial configuration and goal in this problem as follow:

***initially*:** on(a,fondo). on(b,a). on(c,b). on(d,c). on(e,d). on(f,fondo). on(g,f). on(h,g). on(i,h). on(j,fondo). on(j,i). on(k,j). on(l,k). on(m,l). on(n,m). on(o,fondo). on(p,o). on(q,p). on(r,q). on(s,r).

goal: on(a,fondo), on(c,a), on(d,c), on(e,d),

on(f,fondo), on(g,f), on(h,g), on(b,calle) ? (7)

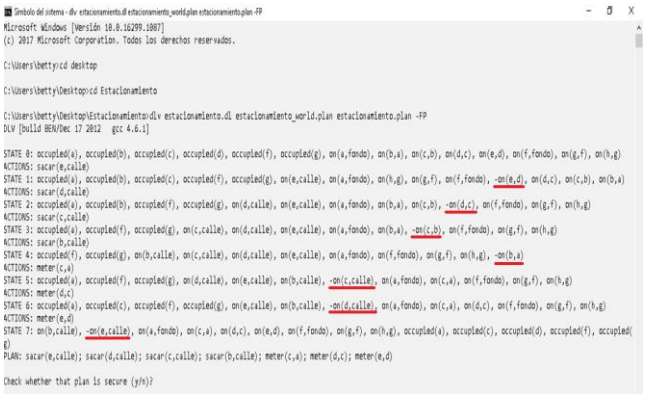

Figure 5. Attending a car request in the parking lot

As we can see in Figure 5, the plan obtained by our proposal is as follows:

PLAN: out(e,calle); out(d,calle); out(c,calle); out(b,calle); in(c,a); in(d,c); in(e,d);

As mentioned before, the cars are put on the street (which is infinite) and then put those that will not be delivered and only leave the one requested by the customer.

It is important to note that our language not only presents a plan, i.e., you can request more plans and also verify if it is safe. This is a relevant feature that allows you to have several alternatives for solving the problem.

VII. CONCLUSION

As you can see, the modeling is simple and clear due to the expressive power of the K language, i.e., K is very expressive in terms of planning and reasoning about actions, allowing to encode even hard planning problems with alternative preconditions of actions, and nondeterministic actions effects. In the proposal presented on the pile-type parking, it is important to note that it can easily be modified to model a vertical parking, either up or as a basement.

With the application developed for our campus and particularly parking planning application, we could create a mobile tool that allows those responsible for parking administration to control customer requests in an automated way.

On the other hand, first application that determines the route to get from one point to another is an example of classical planning. With this, visitors to our campus can go to any point in a simple way. However, also some limitations and possibilities for improvements and further research have developed throughout our work.

Another advantage of this paradigm is the ease that language gives us to be able to communicate it with other languages (such as Java) that allow development as a mobile application.

ACKNOWLEDGMENT

Thank you very much to the Autonomous University of Puebla for their financial support. This work was supported by thematic research network PROMEP called "Combinatorial Algorithms and Pattern Recognition".

REFERENCES

[1] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (Jul 2006).

[2] I. Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning, volume 1265 of Lecture Notes in Artificial Intelligence, pages 420-429, Dagstuhl, Germany, July 1997.

[3] M. Gelfond and |V. Lifschitz. Classical negation in logic programs and disjuctive databases. New generation computing 9, pp. 365-385, 1991.

[4] T. Eiter, W. Faber, N. Leone, G. Pfeifer and A. Polleres. "A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity". INFSYS Research report 1843-01-11, Technische Universitat Wien, 2002.

[5] T. Eiter, W. Faber, N. Leone, G. Pfeifer and A. Polleres. "A Logic Programming Approach to Knowledge-State Planning, II: The DLVK System". INFSYS Research report 1843-01-12, Technische Universitat Wien, 2003.

[6] I. Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, Ann. Math. Artificial Intelligence 25 (1999) 241–273.

[7] Zeynep Gozen Saribatur, Thomas Eiter: Reactive Policies with Planning for Action Languages. JELIA 2016: 463-480.