CCALC and Satisfiability Planning

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

Abstract
This paper is described in the spirit of what is known as the logical approach to AI. McCarthy proposed that to address a planning problem in AI by first giving a "formal statement of the premises" & then to develop a program capable of common sense reasoning about action and change. A program that is able to perform such tasks as prediction, explanation, and planning. The goal here is to take a careful look at the early history of AI and to identify some of the logical and algorithmic ideas related to problems that have emerged over the years. An experiment is presented that summarizes the duration of the computation (in CPU seconds) using Sato when its input is generated by Ccalc.

Keywords: *Propositional logic system, Transition System, Causal logic, Sat-Planning, Solvers, Ccalc, C.t (Causal Theory), Grounding*

1. Introduction

The cause "*is the sum total of the conditions positive and negative taken together .... which being realized, the consequent invariably follows*."[1]
John Stuart Mill

The research on artificial intelligence (AI) has mostly been aimed at developing a computer program capable of commonsense reasoning about action and change specifically, and is able to perform such tasks as prediction, explanation, and planning [John McCarthy 1959]. A program of this kind might "reason" by manipulating explicit, declarative representations of the relevant knowledge.

Perhaps computing answer sets for a planning problem we do not need only answer set solvers, a propositional logic system *CCALC*[2] could be used that provide Satisfiability solver to compute answer sets for a planning problem. *CCALC* was developed in SICStus Prologg, it also runs in Gnu Prolog. It employs to two satisfiability checkers: (i) rel_sat [Bayardo and Schrag, 1997] and (ii) sato [Zhang, 1997] these are written in *C++* and *C*. The Causal Calculator is an implementation of the propositional causal logic from [McCain and Turner, 1997]. Causal Calculator is a system for representing commonsense knowledge about action and change and is intended for query answering and planning with respect to action domains, formalized directly or indirectly in the language of causal theories [McCain and Turner, 1997] a simple nonmonotonic formalism which extends standard propositional logic. *CCALC* is useful as a tool for investigating, debugging, and validating action domain descriptions in the language of causal theories and in *\C*. Presumably *CCALC* could also be applied to constraint satisfaction problems that do not involve actions or effects, but so far this has not been seriously attempted. The input language of the new *CCALC* provides a convenient and concise syntax for writing action descriptions in the "definite" fragment of *C+*. A description is translated by *CCALC* first into a causal theory and then into a set of propositional formulas. The models of this set formulas correspond to paths in the given transition system and *CCALC* finds the models by invoking satisfiability solvers, such as *mCHAFF*[3], *RELSAT*[4], and *SATO*[5], in the spirit of satisfiability planning. *CCALC* can be used for solving problems in planning (finding a sequence of

[1] The quotation is from A System of Logic, page 332, Volume VII, *The Collected Works of John Stuart Mill*, [Robson, 1973].
[2] http://www.cs.utexas.edu/users/tag/cc/
[3] http://www.ee.princeton.edu/~chaff/software.php
[4] http://www.almaden.ibm.com/cs/people/bayardo/resources.html
[5] http://www.cs.uiowa.edu/~hzhang/sato.html

actions that leads to a goal), temporal projection (computing the effect of a sequence of actions), and postdiction (computing possible initial states given a sequence of actions and a resulting state). The idea of planning to satisfiability is due to Kautz and Selman [1992, 1996] provides tantalizing evidence that large, classical planning problems may be efficiently solved by translating them into propositional satisfiability problems, using stochastic search techniques, and translating the resulting truth assignments back into plans for the original problems. SAT-PLAN showed that a general propositional theorem prover could indeed be competitive with some of the best specialize planning systems. The success of SAT-PLAN can be attributed to two factors:

- The use of a logical representation that has good computational properties. Both the fact that SAT-PLAN uses propositional logic instead of first-order logic, and the particular conventions suggested for representing time and actions, are significant. Differently declarative representations that are semantically equivalent can still have quite distinct computational profiles.
- Many researchers in different areas of computer science are creating faster SAT engines every year. Furthermore, these researchers have settled on common representations that allow algorithms and code to be freely shared and fine-tuned. As a result, at any point in time the best general SAT engines tend to be faster (in terms of raw inferences per second) than the best specialized planning engines. In principle, of course, these same improvements could be applied to the specialized engines but by the time that is done, there will be a new crop of SAT solvers.

CCALC has been successfully applied to several challenge problems in the theory of commonsense knowledge [Lifschitz, 2000], [Lifschitz *et al*., 2000], [Campbell and Lifschitz, 2003], [Lee and Lifschitz, 2003a] and to wire-routing problems in VLSI domains [Erdem *et al*., 2000]. Several researchers have used *CCALC* to formalize the specification of multi-agent computational systems [Artikis *et al*., 2003a, Artikis *et al*., 2003b, Chopra and Singh, 2003]. There is preliminary work done on formalizing workflows in the language of *CCAL*C. A workflow is a series of tasks performed by various cooperative and coordinated agents in series or in parallel to achieve a desired goal. *CCALC* has been applied to formalizing workflow requirements and to testing their correctness. Regarding *CCALC* can also be applied to verifying some network security protocols these wide applications of *CCALC* are not surprising because ccalc can be regarded as a general-purpose reasoning tool. Just as a pocket calculator automates the computation of numbers, *CCALC* automates reasoning about actions.

The remainder of this paper is organized as follows. In Section 1.1 a rough idea about formalization of commonsense knowledge is discussed and interpreted. Section 2 describes the input transition system of *CCALC*.

In Section 3, we will review briefly the Satisfiability planning. Section 4 concerns with a simple planning experiment and we discuss about the CPU run time for grounding in the first encoding and plan generation in the second encoding that shows how to present planning problem to the *CCALC* system. We concludes in Section 5. Section 6 has a small glimpse about future plans. Section 7 gives acknowledgement about the paper and finishing paper with references in Section 8.

1.1. Formalizing Medium-size Action Domains

Research on formalizing commonsense knowledge has been mostly focused on “toy problems”, which can be formalized by several lines of axioms. Since the addition of system *CCALC* that generates large domain descriptions by grounding relatively small “schematic descriptions” whose schematic variables range over finite domains. We are, therefore, now able to formalize and test medium-size action domains which normally requires several pages of axioms.

The expressive possibilities of language *C+* can be tested by formalizing two well known action domains which are as under, the Zoo World and the Traffic World, proposed by *Erik Sandewall* as part of his workshop on logic modeling [Akman *et al*., 2004]. The Zoo World consists of several cages and the exterior, gates between them, and animals of several species, that also including humans. Actions in Zoo World domain includes moving within and between cages, opening and closing the gates, and mounting and riding on animals. The second well known action domain is the Traffic World which includes vehicles, moving those vehicles continuously between road crossings

subject to a number of restrictions, such as speed limits and keeping a fixed safety distance away from other vehicles that are moving on the same road. So at this moment, the system *CCALC* is the only system that has been successfully applied to formalizing both these domains.

2. Input Transition System

CCALC, as stated earlier, is a program for solving computational problems related to action and change. Problems of this kind described in terms of "transition systems" in a form of directed graphs having vertices that corresponding to the states of the graph and the edges corresponding to events. To understand transition system, consider a system having 2 states defined by the value of a Boolean-valued parameter say ("fluent") *p* also having an action *a*. Executing an action *a* on *p* causes *p* to be true. That transition system can be graphically represented in Fig. **1** as follows,

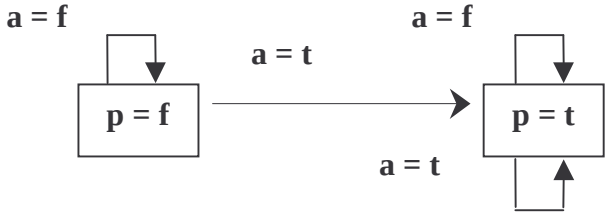

FIG. 1: Input Transition System

as we know that the following transition system having two edges labeled a=t that actually represent the events in which action a is executed, these edges lead to the state p=t. The other two edges are labeled as a=f (action a is not executed), so these edges are the loops. If a system involves several fluents like p_1, p_2, p_3, ... simultanously several actions like a1, a2, a3, ... then every vertex and every edge is labeled by a set of equations specifying the values of all fluents in the corresponding state, and specifying which actions are executed in the corresponding event, for instance:

$$p_1 = t,\ p_2 = t,\ p_3 = f,.....$$
$$a_1 = t,\ a_2 = t,\ a_3 = f,....$$

value t to several action symbols indicates that the corresponding actions are executed concurrently. The above transition system can be described in the input language of system *CCALC* as follows:

```
:- constants
p :: inertialFluent;
a :: exogenousAction.
a causes p.
```

This description consists of the declarations of the constants *p* and *a* and a *proposition*.

3. Satisfiability Planning

Planning is a sub area of AI. In a planning problem, we want to find a plan—a sequence of actions that leads to a given goal. The system *CCALC* [McCain, 1998], [McCain and Turner, 1998] reason about actions and find plans using a satisfiability checker. Currently, it can use two checkers: *RELSAT* [Bayardo and Schrag, 1997] or *SATO* [Zhang, 1997]. It's input can be represented in the action language *C* [Giunchiglia and Lifschitz, 1998]. *CCALC* can transform it's input into a propositional theory in clausal form in three ways: using the basic mode, using history mode or using the transition mode. The transition mode differs in that *CCALC* first constructs a propositional theory describing the effects of actions at time 0. It converts this classical propositional theory into clausal form and then "shifts" the clauses to generate similar descriptions for other time instants. The computational procedure used by *CCALC*, called "literal completion", is a modification of Clark's completion [Clark, 1978] applicable to programs with classical negation which is due to McCain and Turner [1997]. After transforming the causal theory to a propositional theory in clausal form, *CCALC* can sort the clauses in this theory. Then *CCALC* uses a propositional solver to find a model of this classical propositional theory. This is an approach which in particular used, for finding plans in the spirit of satisfiability planning (*a powerful approach to domain-independent planning).* Recently there has been increased use of SAT solvers for solving real-world problems in such fields as planning, model-checking adn so on, by encoding as a SAT instance. SAT problems

is fundamental in solving in solving many application problems in automated reasoning, computer-aided design, computer-aided manufacturing, machine vision, database, robotics, scheduling, integrated circuit design, computer architecture design, and computer networking. Methods to solve problem play a crucial role in the development of efficient computer systems. There has been a strong relationship between the theory, the algorithms, and the applications of SAT problem. The SAT instances arising from these problems are often quite large and require hours, even days, to solve and take up significant system resources.

4. Experiment and Discussion

A *C* input file for the Causal Calculator consists of declarations, propositions in language *C*, problems and comments. Among it's declarations, a *C* input file usually contains a directive to include the "standard" file *C.t*[6] (Causal Theory) which contains rewrite rules (*sort names inertialFluent, defaultFalseFluent, and the action that appear in the constants declaration of the encoding*) for translation from *C* into the language of causal theories, as well as various sorts, variables, constants, and domain independent causal laws which are useful in formalizing action domains.

We perform an experiment on *CCALC* version 2.0 and *SATO* version 3.2 with a well known planning problem "Getting to the Airport", in our approach first we compile the description and get the running time result for the grounding process and then consider the description of the planning problem.

Encoding

```
% File 'airport-domain'
:- macros
  disjoint(#1,#2) -> (-((#1)=(#2)) & -at(#1,#2) & -at(#2,#1)).
:- sorts
  object >> region;
  mode.
:- variables
  X,Y,Z                 :: object;
  U,V                   :: region;
  M                     :: mode.
:- objects
  i, car, desk          :: object;
  home, airport, county    :: region;
  walking, driving        :: mode.
:- constants
  at(object,object)       :: inertialFluent;
  walkable(region),
  drivable(region)        :: boolean;
  go(object,mode)         :: exogenousAction.
exogenous walkable(U), drivable(U).
% Properties of at
caused  at(X,Z) if at(X,Y) & at(Y,Z).
caused -at(X,Z) if at(X,Y) & disjoint(Y,Z).
caused -at(X,X).
% Effects of actions
go(X,M)       causes at(i,X).
go(X,driving) causes at(car,X).
% Action preconditions
nonexecutable go(X,M) if at(i,X).
```

[6] The sort names inertialFluent, defaultFalseFluent, and action appear in the constant declaration are declared in C.t.

```
nonexecutable go(X,driving) if -at(i,car).
nonexecutable go(X,walking)
   if -[\/U | walkable(U) & at(i,U) & at(X,U)].
nonexecutable go(X,driving)
   if -[\/U | drivable(U) & at(i,U) & at(X,U)].
% Restrictions on the concurrent execution of actions
nonexecutable go(X,walking), go(Y,driving).
nonexecutable go(X,M), -go(Y,M) if at(X,Y) & -at(i,Y).
% Sizes of objects
constraint -at(car,desk).
constraint walkable(home).
constraint -walkable(county).
constraint -drivable(home).
constraint drivable(county).
```

The file is loaded by calling loadf/1 passing the file name as the argument. The language of the theory contains 90 ground atoms, and the generated ground theory — say *D* — consists of 1282 rules. The theory *D* is definite. Accordingly, the translation into propositional logic is carried out by the method of literal completion — let say the resulting theory *lcomp(D).* In clausal form, *lcomp(D)* reduces to 3036 clauses after eliminating tautologies. The over all complition process took 2.54 seconds.

- **Behavior of Causal Calculator in Planning**

The behavior of Causal Calculator in planning can be summarized as follows. Given a *C* input file describing an action domain, the propositions and schemas in the file are translated into laws and schemas of the language of causal theories. The Causal Calculator instantiates the schemas with respect to the language signature, replacing schematic variables by ground terms of the language. The resulting ground theory is translated into propositional logic and transformed into a set of clauses. When a planning problem is posed, a range of times is specified (either implicitly or explicitly) by a set of natural numbers {0, 1, ..., n} (n >=1). From the previously generated set of clauses another set of clauses is obtained whose models correspond to the causally possible histories of the domain over the specified range of times. The second set of clauses is combined with clauses describing a possible initial state and a set of time specific goals. Then a model of the resulting set of clauses is sought. And if a model is found a plan is extracted from it and displayed.

Encoding for Planning Problem

```
% File 'airport-problem'
:- include ’airport-domain’.
:- query
maxstep :: 2;
0: at(i,desk),
   at(desk,home),
   at(car,home),
   at(home,county),
   at(airport,county),
   -at(desk,car),
   disjoint(home,airport);
maxstep: at(i,airport).
```

Line begins with the time stamp 0 is an initial condition and expresses the initial state s_0 of the planning problem. Similarly, line with time stamp 2 tells that the goal is to make *at(i, airport)* becomes true after 2 successful actions execution. *CCALC* produces the following result with *SATO* run time,

```
calling sato….
run time (seconds)                    0.03
```

0. drivable(airport) drivable(county) drivable(home) walkable(home) at(airport,county) at(car,county) at(car,home) at(desk,county) at(desk,home) at(home,county) at(i,county) at(i,desk) at(i,home)
Actions: go(car,walking)

1. drivable(airport) drivable(county) drivable(home) walkable(home) at(airport,county) at(car,county) at(car,home) at(desk,county) at(desk,home) at(home,county) at(i,car) at(i,county) at(i,home)
   Actions: go(airport,driving)
2. drivable(airport) drivable(county) drivable(home) walkable(home) at(airport,county) at(car,airport) at(car,county) at(desk,county) at(desk,home) at(home,county) at(i,airport) at(i,car) at(i,county)

5. Conclusions

The solution to the problems presented in this paper is based on a number of ideas some of them are old, some new coming from several areas of logic-based AI.

The idea of an action language as a language for describing transition systems was articulated by Pednault [1994]. After action language *A* was defined and related to logic programming in [Gelfond and Lifschitz, 1993], it was extended and modified by several authors. Language *C* is one of the most expressive action description language introduced so far. Satisfiability planning, the success of this idea (due to Kautz and Selman [1992, 2000]) is determined by remarkable progress in the development of fast propositional solvers. Between 1991 and 1996 the size of hard satisfiability problems that could be feasibly solved grew from ones involving less than 100 variables to one involving over 10,000 variables [Selman *et al*., 1997]. On the other hand, the Causal Calculator demonstrates that the method is applicable to planning problems described in very expressive languages.

6. Future Directions

The logical approach to Artificial Intelligence is beginning towards the next step after spending it's time on investigating appropriate systems of logic. Implementations that will benefit from expressive formalisms will be applied to more complicated and challenging problems. *C+* and *CCALC* will be investigated further to resolve some theoretical issues for applying them to commonsense reasoning. For instance, sensing actions which actions that affects the agent's knowledge, but have no effect on the world or probabilistic reasoning have not been considered in *C+*. Used as a planner, *CCALC* does not rely on domain-specific control knowledge. It has been noted that declarative control knowledge sometimes drastically improves performance. Although *CCALC* can solve prediction and postdiction problems with incomplete information, it does not handle "conformant planning"--- generating plans that are guaranteed to succeed with an incomplete initial state. Extensive research on logic programming shows a way to extend SAT-based answer set solvers to deal with disjunctive programs. Currently, *DLV*[7] is the only system that is capable of handling disjunctive programs, and it would be interesting to compare it with an extension of a SAT-based answer set solver.

I believe that a system such as *CCALC* can serve as a general-purpose reasoning tool which is far more expressive than most of the other logical systems.

7. Acknowledgements

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

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

8. References

[1] V. Lifschitz, N. McCain, E. Remolina and A. Tacchella, "Getting to the airport: the oldest planning problem in AI", *Logic Based Artificial Intelligence*, Kluwer, 2000, pp. 147 – 165.

[2] V. Lifschitz, "On logic of causal explanation", *Artificial Intelligence*, 1997, pp. 451 – 465.

[3] V. Akman, S. T. Erdogan, J. Lee, V. Lifschitz and H. Turner, "Representing the zoo world and the traffic world in the language of the causal calculator", *Artificial Intelligence*, 2004, pp. 105 – 140.

[4] A. Artikis, M. Sergot and J. Pitt, "Specifying electronic societies with the causal calculator", *Proc. of Workshop on Agent Oriented Software Engineering III (AOSE)*, LNCS 2585, 2000, pp. 1 – 15, Springer.

[5] E. Erdem, V. Lifschitz and M. Wong, "Wire routing and satisfiability planning", *Proc. of CL*, 2000, pp. 822 – 836.

[6] J. Lee and V. Lifschitz, "Loop formulas for disjunctive logic programs", *Proc. of ICLP-03*, 2003, pp. 451 – 461.

[7] N. McCain, H. Turner and A. Cohn, "Satisfiability planning with causal theories", *Proc. of Sixth Int. Conf. on Principles of Knowledge Representation and Reasoning*, pp. 212 – 223.

[8] H. Zhang, "An efficient propositional prover", *Proc. of the 14th Int. Conf. on Automated Deduction*, 1997, pp. 272 – 275.

[9] N. McCain and H. Turner, "Causal theory of action and change", *Proc. of AAAI-97*,1997, pp. 460 – 465.

[10] C. Baral and M. Gelfond, "Logic programming and knowledge representation", *Journal of Logic Programming*, 1994, pp. 74 – 148.

[11] J. Lee, "Nondefinite vs. Definite causal theories", *LPNMR*, 2004, pp. 141 – 153.

Amount of Figures, Diagrams, and Tables

This paper contains only one Figure.

Received: 2005-04-18