Visualizing Transition Diagrams of Action Language Programs

Özcan Koç, Ferda N. Alpaslan, Nihan K. Çiçekli
Department of Computer Engineering
Middle East Technical University,
06531 Ankara, TURKEY
Phone: +90–312–210–2080
Fax: +90–312–210–1259
okoc@udel.edu, alpaslan@ceng.metu.edu.tr, nihan@cs.ucf.edu

Abstract

The subject of action languages is one of the prominent research topics in current Artificial Intelligence (AI) research. One of the problems in teaching and learning action languages as well as writing causal theory expressions is the difficulty of visualizing transition diagrams in mind. A tool, called Tdv, which extends Ccalc [GL98b] and uses `GraphViz`[KN91] software, is developed to visualize transition diagrams of $\mathcal{C}$ programs.

Keywords : Action Languages, causal theories, transition diagrams, visualization.

1 Introduction

The subject of action languages is one of the prominent research topics in current Artificial Intelligence (AI) research. An action language allows its users to study the change and properties of actions by means of fluents and causal relations among them. A *fluent* is a judgement about the status of objects in the world—*i.e.* a logical world model. Causal relationships among fluents are expressed using logical propositions. For example,

Shoot(gun) `causes` ¬*Alive* `if` *loaded(gun)*

describes the effect of *Shoot* action on *Alive* fluent. Action languages and related research are discussed in many recent papers [GL98a], [BG97]. $\mathcal{C}$ is an action language developed by Enrico Giunchiglia and Vladimir Lifschitz[GL98b] and uses the idea of causal theories. $\mathcal{C}$ is a language having two kinds of *laws*. Static laws are of the form

`caused` F `if` G

and dynamic laws are of the form

`caused` F `if` G `after` H

where F and G are propositional combinations of fluent names and H is a propositional combination of fluent and action names. The dynamic laws are used to show the direct effects of the actions. The $\mathcal{C}$ language also has some additional expressions that can be written in the form of static and dynamic laws. These are itemized below:

U `causes` F `if` G
`inertial` F
`always` F
`nonexecutable` U `if` F
`default` F `if` G
U `may cause` F `if` G

where F and G are propositional combinations of fluent names and U is a propositional combination of fluent and action names. Examples of simple planning problems that are solved using $\mathcal{C}$ can be found in [McC99]. A $\mathcal{C}$ program defines a *transition system* which consists of sets of fluents and relationships among these fluent sets. Although $\mathcal{C}$ offers a complete system in formalizing action language domains, it lacks a tool to obtain a visual appearance of the resulting transition systems. Drawing a transition diagram is the best way of representing the whole state space of the problem domain. Doing it manually is time consuming and open to errors. Moreover, an automated tool would be helpful for the learners and teachers of action languages.

The Causal Calculator (Ccalc) [McC99] is a system, written in Prolog, for query answering and satisfiability in the context of planning. It is de-

veloped for the language of causal theories [MT97]. The input to CCALC is given in $\mathcal{C}$ which is translated to causal theory by using rewrite rules. The causal theory is grounded to obtain a ground causal theory. The ground causal theory is translated to propositional logic by means of literal completion. Then, the propositional logic formulas are put into the conjunctive normal form (CNF). To find a model of the system, CCALC uses a so–called *satisfiability solver* (SAT). These are propositional provers based on Davis–Putnam method and expect their input in CNF. The models found by SATs are then used for planning. The idea behind this approach can be found in [KS92].

We have developed a software, called TDV(Transition Diagram Visualizer), to draw transition diagrams defined by programs written in action language $\mathcal{C}$. TDV extends the CCALC package and uses `GraphViz`[KN91] package for drawing.

The rest of this paper is organized as follows: Section 2 explains transition diagrams, Section 3 discusses implementation details, Section 4 compares the performance of algorithms, Section 5 presents the visual customizations and Section 6 contains the conclusion.

2 Transition Diagrams

Programs written in action languages consist of causal clauses which constitue a causal theory. Every causal theory defines a transition system. A *transition system* is a set of transitions of the form $<s, A, s'>$, where s is initial state A is a set of states and s', is resulting state

At each state, every fluent has a value of *true* or *false*. Hence, a state s is a set of all literals, *i.e.* positive or negative fluents. A is a, possibly empty, set of concurrent actions that a causal theory allows to execute concurrently. If we assume that there are n fluents and m actions, there are $2^n \times m \times 2^n$ possible transitions for non–concurrent case and $2^n \times 2^m \times 2^n$ possible transitions for the concurrent case. Among these, some are *causally explained*[1]. A *transition diagram* refers to a set of causally explained transitions.

```
:- include 'C.t'.
:- sorts latch.
:- variables L  :: latch.
:- constants
 l1, l2          :: latch;
 up(latch)       :: defaultFalseFluent;
 open            :: inertialFluent;
 toggle(latch)   :: action.
caused open if up(l1) && up(l2).
caused -open if -up(l1) ++ -up(l2).
toggle(L) causes up(L) if -up(L).
toggle(L) causes -up(L) if up(L).
```

Figure 1: $\mathcal{C}$ code for suitcase domain

We can represent transition diagrams by means of labeled directed graphs. In such a representation, nodes correspond to states and edges represent transitions. Figure 2 presents the transition diagram of Lin's suitcase domain [Fan95] whose code is presented in Figure 1 as an example. In the suitcase domain, there is a spring–loaded suitcase with two latches. The suitcase is open whenever both latches are up. According to Figure 2, the domain has 3 fluents—*open, up(l1)* and *up(l2)*—and 2 actions—*toggle(l1)* and *toggle(l2)*. Since this is a concurrent example[2], there are $2^3 \times 2^2 \times 2^3 = 256$ possible transitions. Among these 14 of them are causally explained. In $\mathcal{C}$, states may change (or remain same) without executing any action. Transitions labeled with 0 represent such null actions. Note that *up(l1)* and *up(l2)* fluents are defined as `defaultFalseFluent`. If we modify this definition as `up(latch) :: inertialFluent;` then, we would obtain a transition diagram as illustrated in Figure 3. In this case, transition diagram has again 14 causally explained transitions with certain differences, *i.e.* some of the transitions are different. These two figures emphasize the importance of formalization.

Although the Figures 2 and 3 are strongly connected directed graph, transition diagrams may be unconnected. A directed graph is said to be strongly connected if and only if there is a path—in the sense of graph theory—between every two vertices in the graph. Figure 4 illusrates a transi-

[1] A transition $< s, A, s' >$ is *causally explained* if its resulting state s' is the only interpretation of σ^{fl}, *i.e* fluent symbols, that satisfies all formulas caused in this transition[GL98b].

[2] $\mathcal{C}$ assumes that any program is concurrent unless it contains a `noconcurrency`—or equivalently `nonexecutable _A && _A1 if _A@<_A1`—statement.

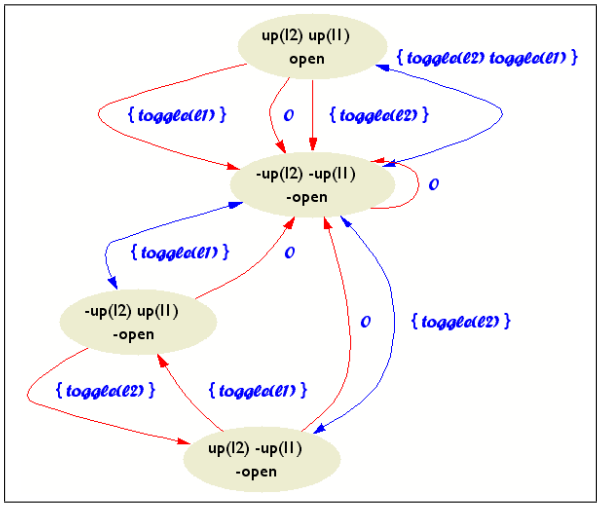

Figure 2: Transition diagram of suitcase example

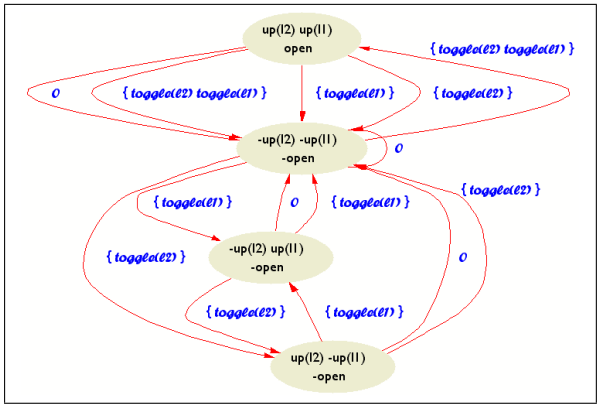

Figure 3: Transition diagram of suitcase example (with inertial fluents)

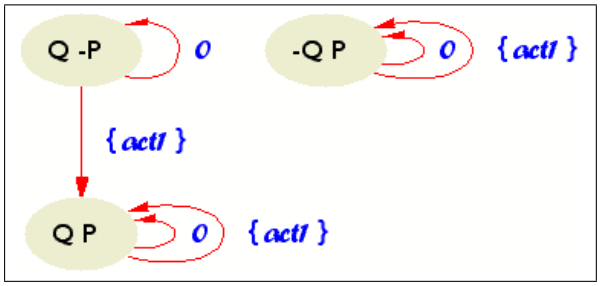

Figure 4: A transition diagram which is not strongly connected.

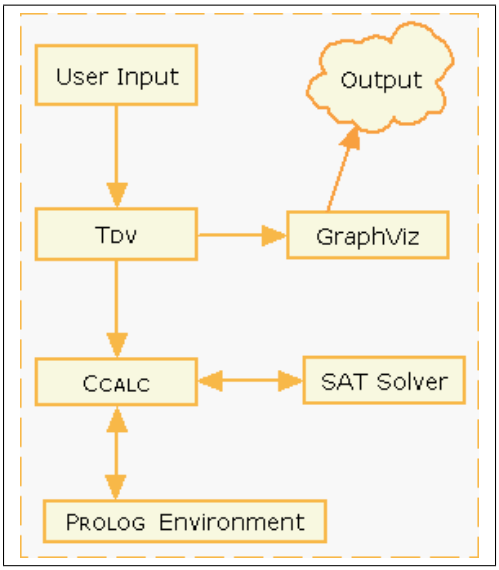

Figure 5: Tdv architecture

tion diagram which is not strongly connected. This graph is not strongly connected, because it is not possible to reach $\{\neg P\ \neg Q\}$, or any other node, from $\{P\ Q\}$.

3 Visualization of Transition Diagrams

In order to visualize the transition diagram of a $\mathcal{C}$ program, Td software extends the Ccalc software as illustrated in Figure 5. Ccalc passes the information about the nodes and transitions among the nodes to Tdv. After extracting the node and transition information, the result is written to a file in a format readable by AT&T's `GraphViz` software and the file is sent to *dot*[3] as input. *dot* produces the transition diagram. Ccalc neither produces nodes or transitions, nor uses this information. Hence, this information is obtained by posing planning problems to Ccalc as outlined in algorithm, named `algorithm 1`, below:

Algorithm

1. Ask Ccalc to find a plan of length 1 with initial condition `true` and goal `true`. Ccalc

[3] *dot* is a part of `GraphViz` package.

produces a plan with an initial state s_0, a final state s_1 and a set of actions A_0.

2. While there are more transitions, construct a new planning problem with

   - current state is s,
   - goal state s' is different from previously found goals,
     i.e. $s' \neq s_1 \wedge s' \neq s_2 \wedge \ldots s' \neq s_n$ where s_i are goals found so far,
   - set of actions A is different than previously found action sets, *i.e.* $A \neq A_1 \wedge A \neq A_2 \wedge \ldots A \neq A_n$ where A_i are actions found so far.

   After finding the new transition $< s, A, s' >$, add s' to the set of non–processed nodes if we see this node for the first time.

3. When there are no more transitions, mark current state s as processed, take another non–processed node as new s, and go to step 2.

4. Repeat step 3 until all nodes are processed.

5. Output the resulting nodes and transitions to a file.

As we discussed before, a state may change, or remain same, without performing any action. In such a case, how to specify that at least one action should be used in order to get a different transition is a problem needs special handling. This is formulated by the following statement:

```
0: \/V_A1: o(V_A1,0)
```

This statement specifies that at least one action should occur, and it solves the problem. Although the algorithm outlined above is correct, transition diagrams produced by the algorithm may be incomplete. As we discussed in the previous section, transition diagrams may be unconnected. The above algorithm, however, finds only a connected subgraph of the transition diagram. To overcome this difficulty, we can generate all states by taking permutations of 2^n literals beforehand, and use them as non–processed nodes. This algorithm is named `algorithm 2`. However, this would decrease the performance considerably for graphs which are not fully connected. A hybrid of these two algorithms, named `algorithm 3`, is also possible. In this case, a set of desired nodes is given to the system as non–processed nodes. System discovers the remaining nodes by applying the first algorithm. All of these 3 algorithms were implemented in Tdv.

4 Performance Issues

In order to discuss the complexities of the algorithms, let us consider a domain with n fluents, m actions, T causally explained transitions and N nodes. We assume N and T are actual number of nodes and edges, respectively, in the resulting transition diagrams, hence they satisfy

- $0 \leq N \leq 2^n$ (concurrent and non–concurrent case)
- $0 \leq T \leq N \times 2^m \times N$ (concurrent) or $0 \leq T \leq N \times (m+1) \times N$ (non–concurrent case)

Therefore, `algorithm 1` runs in $O(T)$[4] in general and in $O(N \times 2^m \times N)$ for the worst case. `Algorithm 2` first discovers *all* possible nodes ($O(2^n)$), then it finds all causally explained transitions ($O(T)$). So, `algorithm 2` runs in $O(2^n) + O(T)$, which impiles a worst case of $O(2^n \times 2^m \times 2^n)$ and a best case of $O(2^n)$. `Algorithm 3`, like `algorithm 1`, runs faster than `algorithm 2`. This algorithm runs in $O(T_1 + T_2 + T_3 + \cdots + T_n)$, where each T_i denotes the number of causally explained transitions in discovered subgraph i. As we can easily see, the summation satisfies the $0 \leq T_1 + T_2 + T_3 + \cdots + T_n \leq T$ inequality, since the summation of number of causally explained transitions in subgraphs cannot exceed the total number of causally explained transitions.

Our experiments, *see* appendix, showed that our program runs reasonably fast for domains with 2^{11} or less nodes. Each of the examples took less than six minutes on an average Pentium II PC with 64 MB RAM. We observed that, most of this time is spent for file I/O operations of Ccalc. We believe that running times may be decreased to 30–75% if Ccalc is compiled with the SAT solvers. On the other hand, we should note that the problem is, *by definition*, exponential; *i.e.* there can be up to $2^n \times 2^m \times 2^n$ transitions. So, for larger domains, Tdv cannot produce results fastly, or even indefinitely. For example, Tdv is not able to produce the

[4] If the graph is not connected, T is the number of causally explained transitions in the subgraph discovered by the algorithm.

transition diagram of airport problem[LMRT00], since there are 42 fluents—which impiles $2^{42} = 4.398.046.511.104$ possible nodes.

5 Visual Customizations

`GraphViz` package offers some customizations about the visual appearance of graphs, and we have an interface for these options. TDV supports the customizations of shape of nodes, node and edge fonts, size/color of node and edge fonts, multi-line labels for nodes and edges, thickness of edges, scale of graphs and it is able to produce a smaller version of transition diagrams by merging[5] bi–directional edges, *etc.*

In addition to these customization, *dotty* program of `GraphViz` package allows its users to modify the layout of the graph.

6 Conclusion and Suggested Work

Action laguages is one of the important topics in current AI research. In order to help the study of action languages research a tool which is capable of displaying the transition diagrams of $\mathcal{C}$ programs is developed. We will improve our tool by implementing algorithm 3. Three different algorithms were developed to achieve speed and completeness. Also, several visual customizations are offered for convenience. System is tested in SWI–Prolog under Windows NT/2000. Minor revisions may be done to run the program under different operating systems. Furthermore, we believe that it would be useful to develop a web based interface to achive a cross–platform system.

Acknowledgements

We are grateful to Vladimir Lifschitz and Esra Erdem for their suggestions and useful comments.

[5] If there are two nodes X and Y such that there is a transition from X to Y with action set A and another transition from Y to X with the same set of actions , *i.e.* A, TDV can display a single edge with two arrowheads which reduces the number edges displayed. Such kind of edges are called bi–directional edges.

Appendix

Yale Shooting Domain

```
:- sorts gun.
:- variables G :: gun.
:- constants
 g1, g2                :: gun;
 load(gun), shoot(gun) :: action;
 alive, loaded(gun)    :: inertialFluent.
load(G) causes loaded(G).
shoot(G) causes -alive if loaded(G).
shoot(G) causes -loaded(G).
```

$\mathcal{C}$ code for yale shooting example

References

[BG97] Chitta Baral and Michael Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming 31*, 1997.

[Fan95] Lin Fangzhen. Embracing causality in specifying the indirect effects of actions. *Proc. of IJCAI-95*, pages 1985–1991, 1995.

[GL98a] Michael Gelfond and Vladimir Lifschitz. Actions languages. *Electronic Transactions on AI*, page 3, 1998.

[GL98b] Enrico Giunchiglia and Vladimir Lifschitz. An actions language based on causal explanation: Preliminary report. *Proc. AAAI–98*, pages 623–630, 1998.

[KN91] Eleftherios Koutsofios and Stephen North. Drawing graphs with dot. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, September 1991.

[KS92] H. Kautz and B. Selman. Planning as satisfaibility. *Proc. of ECAI–92*, pages 359–379, 1992.

[LMRT00] Vladimir Lifschitz, Norman McCain, Emilio Remolina, and Armando Tacchella. Getting to the airport: the oldest planning problem in AI. *Logic-Based Artificial Intelligence*, pages 147–165, 2000.

[McC99] Norman McCain. Using the causal calculator with the $\mathcal{C}$ input language. Technical report, 1999.

[MT97] Norman McCain and Hudson Turner. Causal theories of action and change. *Proc. AAAI–97*, pages 460–465, 1997.

Transition diagram of yale shooting example