Progression of Situation Calculus Action Theories with Incomplete Information

Stavros Vassos and Hector Levesque

Department of Computer Science
University of Toronto
Toronto, Ontario M5S 3G4
Canada

Department of Computer Science
University of Toronto
Toronto, Ontario M5S 3G4
Canada

Problem description

Get a robot or agent to reason efficiently about action and change under the following three conditions:

1. the agent has *incomplete knowledge* about certain changing elements in the world (fluents);
2. the agent has both *physical* (world-changing) and *sensing* (knowledge-producing) actions at its disposal;
3. the agent is *long-lived* and may have performed thousands or even millions of actions to date.

The challenge is to reason efficiently about the overall state of the world, taking into account what has been done and what has been sensed, as a large number of such actions are performed.

Our approach

- Represent the world as a *situation calculus* [1] basic action theory [2] of a special form.
- Use a *regular database* to represent the *possible values* for each dynamic element (fluent) of the world.
- *Update* the database after each action keeping track of the constraints that arise between the fluents.

Formally, solve the *projection task* for situation calculus basic action theories of a special form using a *progression mechanism*.

(See the paper for the details!)

Action theories

- Use *functional fluents* to represent the dynamic elements of the world. E.g. for the temperature of a room:

  use $temp = 22$ instead of $temp(22)$.

  In general, focus on atoms of the form $\tau = d$, where τ is a *ground fluent term* and d is a constant.
- There can be *incomplete knowledge* about the *value* of a ground fluent term, but not about its arguments. E.g.,

  $$temp = 22 \vee temp = 23 \vee temp = 24.$$

  In general, we only allow for incomplete knowledge of the form

  $$\vec{\tau} = \vec{d}_1 \vee \cdots \vee \vec{\tau} = \vec{d}_n$$

  where $\vec{\tau}$ is a *tuple of ground fluent terms* and each $\vec{d}_i$ is a constant tuple of the same length.
- Actions have *local effects* [3].

Running example: a fantasy world

The agent is trapped inside a dungeon. The main door can only be unlocked by casting a magical spell. The agent has to cast the spell while being in the same room with a magical orb that is hidden somewhere in the dungeon, otherwise the spell has no effect. Finally, even though the agent cannot find the magical orb, she knows that it is hidden somewhere in rooms *Room1* or *Room2*.

We use three (functional) fluents to represent this scenario.

- $hero$ holds the location of the agent. E.g. $hero =$ *Room1*.
- orb holds the location of the orb. E.g. $orb =$ *Room1*
- $door$ holds the state of the door. E.g. $door =$ *Closed*.

There are two actions available.

- $go(y)$: the agent moves to room y;
- $cast$: the agent casts the magical spell.

The initial state as a database

$hero$	orb	$door$
Room1	*Room1* *Room2*	*Closed*

- $hero =$ *Room1*
- $orb =$ *Room1* $\vee$ $orb =$ *Room2*
- $door =$ *Closed*

- Each *table* in the database has *one column* and stores the *possible values* for a ground fluent term.
- Whenever there is *complete knowledge* about the term, the corresponding table has *exactly one* row.

  E.g., the ground fluent term $door$ has only one possible value (*Closed*) and so the table for $door$ has exactly one row: *Closed*.
- Whenever there is *incomplete knowledge*, the table has more than one rows that list all the possible values for the term.

  E.g., the table for orb has two rows: *Room1* and *Room2*.
- In the general case, a table in the database may store the possible values for a *tuple* of ground fluent terms (read on!).

Actions update the database: go

The agent executes action $go($*Room2*$)$ and moves to *Room2*. This affects the fluent $hero$ and therefore the corresponding table.

$hero$	orb	$door$
Room1	*Room1* *Room2*	*Closed*

$\Rightarrow$

$hero$	orb	$door$
Room2	*Room1* *Room2*	*Closed*

Actions update the database: $cast$

The agent executes action $cast$. This affects the fluent $door$ but now the outcome depends on the fluent orb which has *two possible values*:

- If the orb is located at *Room2* (i.e. at the same room where the agent is) then the door gets opened.
- If the orb is located at *Room1* then the door remains closed.

In the updated database, we need to store the *possible values* for the *pair* $\langle orb, door \rangle$ instead of each of the fluents orb and $door$ separately.

- The table for $\langle orb, door \rangle$ has *two columns*, one for each of the fluents orb and $door$.
- The table has *two rows*: each row is a possible value for the pair $\langle orb, door \rangle$.

$hero$	orb	$door$
Room2	*Room1* *Room2*	*Closed*

$\Rightarrow$

$hero$	orb	$door$
Room2	*Room1* *Room2*	*Closed* *Open*

The general case

At any given time:

- All ground fluent terms in the language are *grouped* into *disjoint tuples*.
- The possible values for each such tuple are stored in a *separate database table*.
- The result of doing a database join operation on all the tables is a relation that lists *all the models* of the *underlying theory*.

When an action is performed:

- The grouping of the ground fluent terms changes so that *interconstrained* fluents appear in the same tuple.
- The possible values for fluent tuples are updated according to the axioms describing the effects of actions.

Logical correctness

The proposed mechanism is a *logically sound* and *logically complete* way of implementing the *projection task* for a restricted form of situation caclulus basic action theories.

(See the paper for details!)

Efficiency

Worst-case scenario:

- *All* ground fluents are *interconstrained* and there is *incomplete knowledge* about their value. All ground fluents need to be grouped into *one huge table*!

Nevertheless, we argue that our method *can be kept practical*:

- The worst-case scenario may arise often when representing logical puzzles, but it appears *rarely* when reasoning about the effects of actions in *agent worlds*.
- In practice, *only small bundles* of ground fluents are both interconstrained and unknown at any time.

Conclusion

Under reasonable assumptions our approach is an appealing solution for *long-lived* agents that constantly need to reason about action and change in *incompletely known* worlds.

References

[1] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Mach. Int.*, 1969.

[2] R. Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.

[3] Y. Liu and H. J. Levesque. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *Proc. IJCAI-05*, Edinburgh, Scotland, August 2005.

[4] E. Amir and S. Russell. Logical filtering. In *Proc. IJCAI-03*, pages 75–82, 2003.

[5] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

[6] L. Antova, C. Koch, and D. Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *Proc. ICDE-07*, Istanbul, Turkey, April 2007. (To appear).

[7] G. De Giacomo, H. J. Levesque, and S. Sardina. Incremental execution of guarded theories. *Comp. Logic*, 2(4):495–525, 2001.

[8] G. De Giacomo and H. J. Levesque. An incremental interpreter for high-level programs with sensing. *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 86–102, 1999.

[9] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66(4):775–808, June 2003.

[10] T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, 37(1-3):95–138, October 1998.

[11] J. Funge. Cognitive modeling for computer games. In *AAAI Fall Symp. on Cognitive Robotics*, pages 44–51, Orlando, FL., 1998.

[12] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[13] G. Lakemeyer and H. J. Levesque. Chapter: Cognitive robotics. In V. Lifschitz, F. van Harmelen, and F. Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007. (Forthcoming).

[14] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[15] H. J. Levesque. Planning with loops. In *Proc. IJCAI-05*, Edinburgh, Scotland, August 2005.

[16] F. Lin and R. Reiter. How to progress a database. *Artificial Intelligence*, 92(1-2):131–167, 1997.

[17] F. Lin. Chapter: Situation calculus. In V. Lifschitz, F. van Harmelen, and F. Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007. (Forthcoming).

[18] Y. Liu and H. J. Levesque. A tractability result for reasoning with incomplete first-order knowledge bases. In *Proc. IJCAI-03*, pages 83–88, Acapulco, Mexico, August 2003.

[19] R. Petrick and H. J. Levesque. Knowledge equivalence in combined action theories. In *Proc. KR-02*, Toulouse, France, April 2002.

[20] R. Petrick. *A knowledge-based representation for effective acting, sensing, and planning.* PhD thesis, University of Toronto, 2006.

[21] S. Sardina and S. Vassos. The Wumpus World in IndiGolog: A Preliminary Report. In *Proc. NRAC-05*, pages 90–95, 2005.

[22] R. Scherl and H. J. Levesque. Knowledge, action, and the frame problem. *Artificial Intelligence*, 144(1–2):1–39, 2003.

[23] M. Thielscher. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1-2):277–299, July 1999.

[24] M. Thielscher. FLUX: A logic programming method for reasoning agents. *Theory and Practice of Log. Prog.*, 5(4-5):533–565, 2004.

[25] T. C. Son and C. Baral. Formalizing sensing actions a transition function based approach. *Art. Int.*, 125(1-2):19–91, 2001.