Reasoning in the Event Calculus Using First-Order Automated Theorem Proving

Erik T. Mueller
IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598 USA
Email: etm@us.ibm.com

Geoff Sutcliffe
Department of Computer Science
University of Miami
P.O. Box 248154, Coral Gables, FL 33124 USA
Email: geoff@cs.miami.edu

Introduction

The event calculus (EC) (Shanahan 1999) is a powerful and highly usable formalism for reasoning about action and change, which is rapidly finding application in such areas as natural language processing and robotics. Kowalski and Sergot (1986) introduced the original event calculus, which was expressed as a logic program, and Shanahan and Miller introduced axiomatizations of the event calculus in first-order logic (Shanahan 1997; Miller & Shanahan 2002). The discrete event calculus (DEC) was developed by Mueller (2004) in order to facilitate solution of event calculus reasoning problems using satisfiability (SAT) solvers.

Since the introduction of the EC and DEC axiomatizations, several event calculus reasoning systems have been implemented. However, to our knowledge, nobody has ever attempted to solve event calculus reasoning problems using first-order logic automated theorem proving (ATP) systems.

Encoding DEC Problems

Encoding DEC problems for ATP systems requires solving several technical and practical problems.

Reification: In a first-order logic language, the proposition that the water level of a sink is 2, is represented using an atom such as *waterLevel*(2). In the event calculus, the truth of this proposition at timepoint 3 is represented using an atom such as *holdsAt*(*waterLevel*$(2), 3)$. This is not a well-formed first-order logic formula if *waterLevel*(2) is an atom. The event calculus therefore uses a technique pioneered by Lifschitz (1987) based on *reification* (McCarthy 1979), which allows formulae of one first-order language to become terms of another first-order language. Flat sorted first-order logic languages are used, with sorts for fluents, events, and timepoints, and additional sorts as required by the scenario under consideration. Atoms of the non-reified language become terms of the reified language, and their sort is determined by their function symbol as either event or fluent. Conformance to the sort signatures is ensured in an ATP system's reasoning through the conformance of the axioms and conjecture to the sort signatures, and the one-to-one unification of atoms' arguments. Note, however, that while this prevents the deduction of anomalies such as *holdsAt*(*tapOn*, *waterLevel*(3)), it does not allow deduction of the negations of such anomalies, e.g., it is not possible to deduce $\neg$*holdsAt*(*tapOn*, *waterLevel*(3)).

Unique Fluent and Event Objects: With the use of reification, it is necessary to add uniqueness-of-names axioms to ensure that fluent and event terms denote unique objects. We use the *U* notation of Lifschitz (1987), in which $U[f_1, \ldots, f_k]$ is a notational shorthand for the set of axioms

$f_i(x_1, \ldots, x_n) \neq f_j(y_1, \ldots, y_m)$

$f_i(x_1, \ldots, x_n) = f_i(y_1, \ldots, y_n) \Rightarrow (x_1 = y_1 \wedge \ldots \wedge x_n = y_n)$

where f_i and f_j are function symbols, and the x_k and y_k are distinct variables. The axioms given by $U[f_1, \ldots, f_m]$ and $U[e_1, \ldots, e_n]$ are added to each scenario's axiomatization, where $f_1, \ldots, f_m$ are the fluent function symbols and $e_1, \ldots, e_n$ are the event function symbols.

Circumscription: Axioms that specify what events initiate and terminate fluents, do not specify what events do *not* initiate and terminate particular fluents. The event calculus uses minimization of the extension of a predicate, or *circumscription* (McCarthy 1980), to minimize unexpected effects of events and unexpected event occurrences. The circumscription of a predicate in a first-order formula is defined by a second-order formula, and is not always equivalent to a first-order formula. Fortunately, in many cases, including the benchmark scenarios considered in this paper, the circumscription can be computed using the following theorem, which reduces circumscription to predicate completion:

Theorem 1 If the formula $\Gamma(x_1, \ldots, x_n)$, with free variables x_i, does not mention the predicate ρ, then the circumscription *CIRC*$[\forall x_1, \ldots, x_n\, (\Gamma(x_1, \ldots, x_n) \Rightarrow \rho(x_1, \ldots, x_n)); \rho]$ is equivalent to $\forall x_1, \ldots, x_n\, (\Gamma(x_1, \ldots, x_n) \Leftrightarrow \rho(x_1, \ldots, x_n))$.

Arithmetic: Problems in the event calculus include the use of integer arithmetic, e.g., to increment timepoints. Integer arithmetic is in general an undecidable theory, although fragments are decidable. For this work a first-order axiomatization of a small fragment of integer arithmetic has been encoded to capture the notions of equality, addition, and order, for the integers 0 to 9. Equality is dealt with through standard equality theory. The axioms for addition and order are listed below. The totality of the relationship between all pairs of integers is enforced by the last axiom. The last axiom also specifies that ordered integers are unequal (which is analogous to the uniqueness-of-names axioms generated for fluents and events).

$0 + 0 = 0, \quad 0 + 1 = 1, \quad \ldots, \quad 8 + 1 = 9$
$\forall X, Y \; X + Y = Y + X$
$\forall X, Y \; (X \leq Y \Leftrightarrow (X < Y \vee X = Y))$
$\forall X \; (X < 1 \Leftrightarrow X \leq 0), \quad \ldots, \quad \forall X \; (X < 9 \Leftrightarrow X \leq 8)$
$\neg \exists X \; X < 0$
$\forall X, Y \; (X < Y \Leftrightarrow (\neg(Y < X) \wedge Y \neq X))$

Testing

The above encoding techniques have been tested on two benchmark scenarios. In each case the axioms for the particular scenario are preprocessed by adding the necessary uniqueness-of-names axioms and performing the necessary predicate completions. The preprocessed axioms are then added to the DEC and integer arithmetic axioms. A conjecture formula is then added to produce a first-order ATP problem. The ATP problems were submitted to the state-of-the-art ATP system Vampire 7.0 (Riazanov & Voronkov 2002), with a 300s time limit.

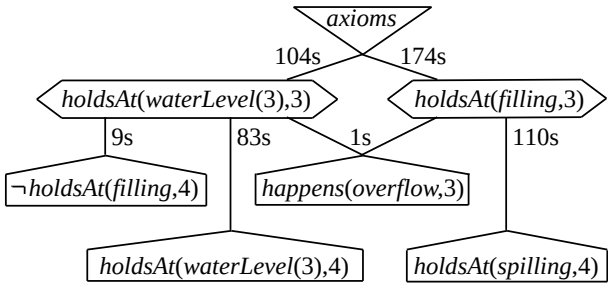
The Supermarket Trolley Scenario: The supermarket trolley scenario is used to test the handling of concurrent events with cumulative and canceling effects. The scenario is as follows: If a trolley is pushed, it moves forward. If it is pulled, it moves backward. If the trolley is simultaneously pulled and pushed, it spins around. The axiomatization of this problem is that of Shanahan (1997), reformulated using the technique of Miller and Shanahan (2002), which involves adding *happens* preconditions to *initiates* and *terminates* axioms. Various theorems, e.g., that when the trolley is both pushed and pulled at time 2 then it is spinning at time 3, were proved by Vampire in less than 1 second per proof.

The Kitchen Sink Scenario: In the kitchen sink scenario (Shanahan 1990) a stopper is put into the drain of a kitchen sink and the water is turned on. The task is to perform temporal projection, a form of deduction, in order to infer that the water level will rise, the water level will reach the rim of the sink, and then the water will overflow and start spilling. Various theorems that describe the state of the system at specified times were proved directly from the axioms within the 300s time limit, e.g., at time 3 the the water level is 3, and at time 3 the sink is filling. For more complex theorems, e.g., the water level at time 4 is still 3 (because the sink started overflowing at time 3), Vampire was unable to prove them directly from the axioms within the 300s time limit. For each of these it was necessary to specify which of the previously proved theorems should be used as lemmas, so that the harder theorem could be proved from the axioms and lemmas. Figure 1 shows the lemma structure used, leading to proofs of the most difficult theorems. Each link shows the CPU time taken to prove the lemma or theorem. When proving the theorems, the axioms as well as the indicated lemmas are used.

Conclusions

This paper shows, with techniques and examples, how DEC reasoning problems can be encoded in first-order logic. Solutions to the technical issues regarding the translation of DEC problems into pure first-order logic have been found, and the resulting ATP problems have been successfully tackled with a state-of-the-art ATP system. The result is a new and practical technology for solving DEC problems. Since the original submission of this paper, a complete axiomatization of equality, addition, and ordering for integers has been developed, and this will replace the fragment described. We plan to add the problems to the TPTP problem library (Sutcliffe & Suttner 1998) in a new domain focusing on commonsense reasoning. In the future it is planned to extend this work to the standard event calculus, in which time is not forced to be discrete.

Figure 1: Lemmas for the Kitchen Sink Theorems



References

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

Lifschitz, V. 1987. Formal theories of action. In *The Frame Problem in Artificial Intelligence*, 35–57.

McCarthy, J. 1979. First order theories of individual concepts and propositions. In *Machine Intelligence 9*. 129–148.

McCarthy, J. 1980. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence* 13:27–39.

Miller, R., and Shanahan, M. 2002. Some alternative formulations of the event calculus. In *Computational Logic: Logic Programming and Beyond*. 452–490.

Mueller, E. T. 2004. Event calculus reasoning through satisfiability. *Journal of Logic and Computation* 14(5):703–730.

Riazanov, A., and Voronkov, A. 2002. The design and implementation of Vampire. *AI Communications* 15(2-3):91–110.

Shanahan, M. 1990. Representing continuous change in the event calculus. In *Proc. of ECAI 1990*, 598–603.

Shanahan, M. 1997. *Solving the Frame Problem*. MIT Press.

Shanahan, M. 1999. The event calculus explained. In *Artificial Intelligence Today*. Springer-Verlag. 409–430.

Sutcliffe, G., and Suttner, C. 1998. The TPTP Problem Library: CNF release v1.2.1. *Journal of Automated Reasoning* 21(2):177–203.