A Tool for Satisfiability-Based Commonsense Reasoning in the Event Calculus

Erik T. Mueller
IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598 USA

Abstract

We present a tool for commonsense reasoning in the classical logic event calculus using satisfiability. We describe the tool, which takes event calculus reasoning problems as input, encodes the problems as propositional satisfiability problems, runs a satisfiability solver on the encoded problems, and produces solutions to the reasoning problems as output. We describe the encoding method and how various commonsense phenomena are expressed using the tool. We evaluate the tool on 14 benchmark commonsense reasoning problems for the event calculus, compare its performance with the causal calculator on eight zoo world test problems, and discuss two natural understanding applications built using the tool.

Introduction

Commonsense reasoning is an important capability for intelligent systems. Many mechanisms have been used to perform commonsense reasoning including analogical reasoning, Bayesian networks, case-based reasoning, logic, marker passing, multiagent systems, and neural networks. Within logic-based artificial intelligence, after Kautz and Selman (1996) showed that high performance planning could be achieved by encoding planning problems as satisfiability problems, several commonsense reasoning systems have appeared that make use of satisfiability and constraint solving techniques. The causal calculator (Giunchiglia *et al.* 2004) uses satisfiability to solve reasoning problems expressed in the language of causal theories (McCain & Turner 1997). The VITAL system (Kvarnström 2001) uses constraint propagation to solve reasoning problems expressed in temporal action logics (Doherty *et al.* 1998). The system of Shanahan and Witkowski (2002) uses satisfiability to solve planning problems for a restricted subset of the classical logic event calculus (Shanahan 1997; Miller & Shanahan 2002).

In this paper, we present a satisfiability-based commonsense reasoning tool that supports a larger subset of the classical logic event calculus. The tool takes a reasoning problem as input and produces solutions to the problem as output. It supports such reasoning types as deduction, temporal projection, abduction, planning, model checking, model finding, and counterexample finding.

Like Shanahan and Witkowski's system, our tool supports:

- The *commonsense law of inertia*, which states that things do not change unless directly affected by events. For example, moving a glass in one room does not mysteriously cause the location of a glass in another room to change.
- *Conditional effects of events*. For example, the results of turning on a television set depend on whether it is plugged in.

In addition, our tool supports the following features of the event calculus not supported by Shanahan and Witkowski's system:

- *Release from the commonsense law of inertia*. For example, if a person is holding a glass, then the location of the glass is released from the commonsense law of inertia so that the location of the glass is permitted to vary.
- *Event ramifications* or indirect effects of events. The tool supports *state constraints*. For example, a glass moves along with the person holding it. The tool supports *causal constraints*, which deal with the instantaneous propagation of interacting indirect effects, as in idealized electrical circuits.
- *Events with nondeterministic effects*. For example, flipping a coin results in the coin landing either heads or tails.
- *Gradual change* such as the changing height of a falling object or volume of a balloon in the process of inflation.
- *Triggered events* or events that are triggered under certain conditions. For example, if water is flowing from a faucet into a sink, then once the water reaches a certain level the water will overflow.
- *Concurrent events with cumulative or canceling effects*. For example, if a shopping cart is simultaneously pulled and pushed, then it will spin around.

The performance of our tool is comparable to that of the causal calculator. An advantage of our tool is the ease of use of the event calculus, which is a straightforward extension of first-order logic.

!	logical negation (NOT, ¬)
&	conjunction (AND, ∧)
|	disjunction (OR, ∨)
->	implication (→)
<->	bi-implication (↔)
[ ]	universal quantification (∀)
{ }	existential quantification (∃)

Table 1: Symbols used in formulas

A Simple Example

Suppose we wish to represent the knowledge that waking up causes a person to be awake, and then use this to deduce that if James is asleep and wakes up, then James will be awake. We create the following problem description:

```
sort actor
fluent Awake(actor)
event WakeUp(actor)
[actor,time] Initiates(WakeUp(actor),Awake(actor),time).
actor James
!HoldsAt(Awake(James),0).
Happens(WakeUp(James),0).
range time 0 1
```

We represent the knowledge about waking up: We define a sort *actor* to represent a person or animal. We define a fluent *Awake* and an event *WakeUp*. We add an *Initiates* axiom that states that if an actor wakes up, then the actor will be awake. The tool uses a formula syntax similar to that of the Bliksem resolution theorem prover (de Nivelle 1999). Table 1 shows the symbols used in formulas. Then we represent a narrative about James: We define a constant *James* whose sort is *actor*. We state that James is not awake at time step 0 and that he wakes up at time step 0. We state that reasoning should be limited to time points 0 through 1.

When we run the tool on this problem description, it produces:

```
5 variables and 9 clauses
1 model
---
model 1:
0
Happens(WakeUp(James), 0).
1
+Awake(James).
encoding 0.0s, solution 0.0s
```

The tool finds one solution or model in which James is awake at time step 1. For each model the tool shows the fluents that are true at time point 0 (none in this case). Thereafter, the tool indicates fluents that become true with a plus sign and fluents that become false with a minus sign. The tool shows event occurrences at the end of each time point. The tool shows the number of variables and clauses of the satisfiability encoding of the problem, and the time taken to encode and solve the problem. When the tool does not find any models, it provides a near miss solution along with unsatisfied clauses to help with debugging.

The Classical Logic Event Calculus

The classical logic event calculus (Shanahan 1999a) is a reformulation in many-sorted first-order logic of the original event calculus (Kowalski & Sergot 1986). Following Shanahan and Witkowski (2002), we restrict the event calculus to finite sets of variables, constants, function symbols, predicate symbols, and sorts. A finite set of time points is associated with the *time* sort.

The predicate symbols of the event calculus are as follows:

1. *Happens(event, time)*: *event* occurs at *time*.
2. *HoldsAt(fluent, time)*: *fluent* is true at *time*.
3. *ReleasedAt(fluent, time)*: *fluent* is released from the commonsense law of inertia at *time*.
4. *Initiates(event, fluent, time)*: if *event* occurs at *time*, then *fluent* becomes true at *time+1* and is no longer released from the commonsense law of inertia at *time+1*.
5. *Terminates(event, fluent, time)*: if *event* occurs at *time*, then *fluent* becomes false at *time+1* and is no longer released from the commonsense law of inertia at *time+1*.
6. *Releases(event, fluent, time)*: if *event* occurs at *time*, then *fluent* becomes released from the commonsense law of inertia at *time+1*.
7. *Trajectory(fluent1, time, fluent2, offset)*: if *fluent1* is initiated by an event that occurs at *time*, and *offset* is greater than zero, then *fluent2* is true at *time+offset*.

A paper by Miller and Shanahan (2002) presents an axiomatization of the event calculus as well as a number of alternative axiomatizations that subtract or add various features. Our tool supports a formulation that combines axioms from the paper's Section 3.2 (for a version of the event calculus in which initiating and terminating a fluent at the same time produces inconsistency), Section 3.5 (gradual change), and Section 3.7 (release from the commonsense law of inertia).

Satisfiability Encoding Method

We now explain the method used by our tool to encode reasoning problems as satisfiability problems. We will be forming a conjunction of problem formulas.

First, we apply syntactic transformations to any input formulas containing the predicate symbols *Initiates*, *Terminates*, *Releases*, or *Trajectory*. These transformations serve to eliminate atoms involving these predicates which if retained would result in extremely large satisfiability problems. For example, the atom *Initiates(event,fluent,time)* would give rise to *number-of-event-ground-terms* × *number-of-fluent-ground-terms* × *number-of-time-points* ground atoms. We add the transformed formulas to the conjunction of problem formulas. In the above simple example, we transform:

```
[actor,time] Initiates(WakeUp(actor),Awake(actor),time).
```

into:

```
[actor,time]
Happens(WakeUp(actor),time) ->
HoldsAt(Awake(actor),time+1) &
!ReleasedAt(Awake(actor),time+1).
```

Second, we add to the conjunction of problem formulas any input formulas not transformed above. In our simple example, we add:

```
!HoldsAt(Awake(James),0).
Happens(WakeUp(James),0).
```

Third, we complete the *Happens* predicate in the conjunction of problem formulas, using standard techniques (Clark 1978, pp. 303–305). This rules out extraneous events (Davis 1990, p. 208) or event occurrences neither explicitly asserted nor triggered by conditions.

Fourth, in order to enforce the commonsense law of inertia, we generate explanation closure frame axioms (Haas 1987) from *Initiates*, *Terminates*, and *Releases* axioms and add them to the conjunction of problem formulas. Explanation closure frame axioms specify for each fluent all the events that can cause the fluent's truth value to change. Frame axioms are not generated for fluents declared as *non-inertial* or always released from the commonsense law of inertia. We generate frame axioms using a technique similar to that described by Reiter (2001, pp. 28–32). He described the technique for the situation calculus; we have adapted it to the event calculus. We have also added a disjunct that enables fluents to be released from the commonsense law of inertia. In our simple example, we add the following frame axioms:

```
[actor,time]
HoldsAt(Awake(actor),time+1) ->
HoldsAt(Awake(actor),time)|
ReleasedAt(Awake(actor),time+1)|
Happens(WakeUp(actor),time).

[actor,time]
!HoldsAt(Awake(actor),time+1) ->
!HoldsAt(Awake(actor),time)|
ReleasedAt(Awake(actor),time+1).
```

The first axiom states that if an actor is awake at *time+1*, then either (1) the actor was awake at *time*, (2) the fluent *Awake* is released from the commonsense law of inertia at *time+1*, or (3) the actor woke up at *time*. The second axiom states that if an actor is not awake at *time+1*, then either the actor was not awake at *time* or *Awake* is released from the commonsense law of inertia at *time+1*. The second axiom does not contain a *Happens* atom because no *Terminates* axioms were specified for the *Awake* fluent in the problem description. In addition, we add frame axioms for *ReleasedAt*:

```
[actor,time]
ReleasedAt(Awake(actor),time+1) ->
ReleasedAt(Awake(actor),time).

[actor,time]
!ReleasedAt(Awake(actor),time+1) ->
!ReleasedAt(Awake(actor),time)|
Happens(WakeUp(actor),time).
```

We also add the initial conditions for *ReleasedAt*:

```
[actor] !ReleasedAt(Awake(actor),0).
```

Fifth, we transform the conjunction of problem formulas into a propositional calculus formula by instantiating quantifiers, simplifying, and converting to short conjunctive normal form by renaming subformulas (Plaisted & Greenbaum 1986). In our simple example, we end up with the following nine clauses:

```
HoldsAt(Awake(James),1)|!Happens(WakeUp(James),0).
!Happens(WakeUp(James),0)|!ReleasedAt(Awake(James),1).
!HoldsAt(Awake(James),0).
Happens(WakeUp(James),0).
Happens(WakeUp(James),0)|ReleasedAt(Awake(James),1)|
  HoldsAt(Awake(James),0)|!HoldsAt(Awake(James),1).
HoldsAt(Awake(James),1)|ReleasedAt(Awake(James),1)|
  !HoldsAt(Awake(James),0).
ReleasedAt(Awake(James),0)|!ReleasedAt(Awake(James),1).
Happens(WakeUp(James),0)|ReleasedAt(Awake(James),1)|
  !ReleasedAt(Awake(James),0).
!ReleasedAt(Awake(James),0).
```

The above clauses are, in order, the transformed *Initiates* axiom, the input formulas that were not transformed, the frame axioms, and the *ReleasedAt* initial condition.

Finally, we map ground atoms to new variables of the satisfiability problem:

```
Happens(WakeUp(James),0)   → 1
HoldsAt(Awake(James),1)    → 2
ReleasedAt(Awake(James),1) → 3
HoldsAt(Awake(James),0)    → 4
ReleasedAt(Awake(James),0) → 5
```

The encoding method is implemented in C and consists of 4,144 lines of code. The method is described in more detail elsewhere (Mueller 2004b).

The encoding method is used by the tool as follows: We format the encoded problem in the standard DIMACS format for satisfiability problems:

```
p cnf 5 9
2 -1 0
-1 -3 0
-4 0
1 0
1 3 4 -2 0
2 3 -4 0
5 -3 0
1 3 -5 0
-5 0
```

We pass the problem to an off-the-shelf satisfiability solver. Our tool uses the Relsat (Bayardo Jr. & Schrag 1997) complete satisfiability solver. In order to find near miss solutions when there are no models, it uses Walksat (Selman, Kautz, & Cohen 1993). For each solution produced by the solver:

```
Solution 1:  1 2
```

we convert the solution back into a list of ground atoms by mapping true variables back to true ground atoms and false variables back to false ground atoms. We map the above solution back to:

```
Happens(WakeUp(James),0)
HoldsAt(Awake(James),1)
!ReleasedAt(Awake(James),1)
!HoldsAt(Awake(James),0)
!ReleasedAt(Awake(James),0)
```

Expressing Commonsense Phenomena

We now explain how various commonsense phenomena are expressed in the event calculus using the tool. We express conditional effects of events using axioms of the form:

$$condition \rightarrow Initiates(event, fluent, time)$$
$$condition \rightarrow Terminates(event, fluent, time)$$

where a condition is a conjunction of inequalities and atoms of the form $\mathit{HoldsAt}(\beta, \mathit{time})$ or $\neg \mathit{HoldsAt}(\beta, \mathit{time})$. For example, we may specify that if a television set is turned on, then it will go on only if it is plugged in:

```
[actor,switch,tv,time]
TVOf(switch)=tv & HoldsAt(PluggedIn(tv),time) ->
Initiates(TurnOn(actor,switch),TVOn(tv),time).
```

One way of expressing event ramifications is with *state constraints*, which are of the form:

$\mathit{condition1} \leftrightarrow \mathit{condition2}$
$\mathit{condition1} \rightarrow \mathit{condition2}$

For example, we may specify that a switch is *Off* whenever it is not *On*:

```
noninertial Off
[switch,time] HoldsAt(Off(switch),time) <->
              !HoldsAt(On(switch),time).
```

In this case, *Off* is always released from the commonsense law of inertia and always depends on *On*.

We may specify that fluents are only sometimes released from the commonsense law of inertia using axioms of the form:

$\mathit{condition} \rightarrow \mathit{Releases}(\mathit{event}, \mathit{fluent}, \mathit{time})$

For example, we may specify that picking up an object releases its location:

```
[actor,physobj,location,time]
Releases(PickUp(actor,physobj),At(physobj,location),time).
```

Then we may use a state constraint to express that when an actor is holding an object, the location of the object depends on the location of the actor:

```
[actor,location,physobj,time]
HoldsAt(Holding(actor,physobj),time) &
HoldsAt(At(actor,location),time) ->
HoldsAt(At(physobj,location),time).
```

Finally, when an actor sets down an object, the location of the object will no longer be released from the commonsense law of inertia:

```
[actor,physobj,location,time]
HoldsAt(At(actor,location),time) ->
Initiates(SetDown(actor,physobj),At(physobj,location),time).
```

We may also express event ramifications with *causal constraints* by adding four new predicates and four new axioms to the formulation of the event calculus, as described by Shanahan (1999b).

One method for expressing events with nondeterministic effects uses a *determining fluent*, whose value determines the result of an event. We may specify that when a wheel is spun, it takes on one of several values at random:

```
noninertial WheelValueDeterminingFluent
[wheel,value,time]
HoldsAt(WheelValueDeterminingFluent(wheel,value),time) ->
Initiates(Spin(wheel),WheelValue(wheel,value),time).
range value 7 10
```

Another method for expressing events with nondeterministic effects uses *disjunctive event axioms*, which are of the form:

$\mathit{Happens}(\mathit{event}, \mathit{time}) \rightarrow$
$\mathit{Happens}(\mathit{event1}, \mathit{time}) \vee \ldots \vee \mathit{Happens}(\mathit{eventn}, \mathit{time})$

For example, we might know that moving involves either driving or running. If an actor runs, the actor gets tired. But if we are told simply that an actor moved, whether the actor will be tired is nondeterministic. We express this as follows:

```
[actor,time]
Happens(Move(actor),time) ->
Happens(Run(actor),time)|Happens(Drive(actor),time).
xor Run, Drive
[actor,time] Initiates(Run(actor),Tired(actor),time).
```

We express gradual change with axioms of the form:

$\mathit{condition} \rightarrow \mathit{Trajectory}(\mathit{fluent1}, \mathit{time}, \mathit{fluent2}, \mathit{offset})$

For example, we might specify that n time points after an object starts falling, its height is n^2 less than it was when it started falling:

```
[object,time]
Initiates(StartFalling(object),Falling(object),time).

[object,height,time]
Releases(StartFalling(object),Height(object,height),
         time).

[object,height1,height2,offset,time]
HoldsAt(Height(object,height1),time) &
height2=height1-offset*offset ->
Trajectory(Falling(object),time,
           Height(object,height2),offset).

[object,height1,height2,time]
HoldsAt(Height(object,height1),time) &
HoldsAt(Height(object,height2),time) ->
height1=height2.
```

We express events that are triggered under certain conditions using axioms of the form:

$\mathit{condition} \rightarrow \mathit{Happens}(\mathit{event}, \mathit{time})$

For example, we may extend the above example with a collision event that is triggered when the falling object hits the ground. The collision event stops the object from falling and the height of the object will no longer be released from the commonsense law of inertia:

```
[object,time]
HoldsAt(Falling(object),time) &
HoldsAt(Height(object,0),time) ->
Happens(HitsGround(object),time).

[object,time]
Terminates(HitsGround(object),Falling(object),time).

[object,height,time]
HoldsAt(Height(object,height),time) ->
Initiates(HitsGround(object),Height(object,height),time).
```

Given the above example and the narrative:

```
!HoldsAt(Falling(Leaf),0).
HoldsAt(Height(Leaf,4),0).
Happens(StartFalling(Leaf),0).
```

the tool produces:

```
0
Height(Leaf, 4).
Happens(StartFalling(Leaf), 0).
1
```

Problem	Vars	Clauses
BusRide	30	74
ChessBoard	27	52
CoinToss	24	65
Commuter	—	—
DeadOrAlive	36	76
Happy	24	46
KitchenSink	102	593
RussianTurkey	35	87
StolenCar	10	20
StuffyRoom	27	57
Supermarket	104	752
ThielscherCircuit	68	142
WalkingTurkey	—	—
Yale	28	64

Table 2: Results on event calculus benchmark problems

```
-Height(Leaf, 4).
+Falling(Leaf).
+Height(Leaf, 3).
2
-Height(Leaf, 3).
+Height(Leaf, 0).
Happens(HitsGround(Leaf), 2).
3
-Falling(Leaf).
```

We express concurrent events with cumulative or canceling effects using axioms of the form:

$$incident \rightarrow Initiates(event, fluent, time)$$
$$incident \rightarrow Terminates(event, fluent, time)$$

where an incident is a conjunction of atoms of the form $Happens(\alpha, time)$ or $\neg Happens(\alpha, time)$. For example, we may specify that the result of approval by itself is happiness, while the result of simultaneous approval and disapproval is confusion and not happiness:

```
[actor1,actor2,time]
!Happens(DisapproveOf(actor1,actor2),time) ->
Initiates(ApproveOf(actor1,actor2),Happy(actor2),time).

[actor1,actor2,time]
Happens(DisapproveOf(actor1,actor2),time) ->
Initiates(ApproveOf(actor1,actor2),Confused(actor2),time).
```

Evaluation on Event Calculus Problems

As shown in Table 2, our tool solves 12 of the 14 benchmark commonsense reasoning problems that have been described for the event calculus by Shanahan (1997; 1999a). All of the problems are solved in less than one second. Commuter involves compound events and WalkingTurkey involves effect constraints; these features are not supported by our tool.

Causal Calculator Performance Comparison

We now compare the performance of our tool and the causal calculator (CCALC) in an important benchmark domain, the zoo world proposed by Erik Sandewall and formalized in the language of CCALC (Akman *et al.* 2004). We translated the CCALC formalization of the zoo world into the event calculus, and used our tool to solve the same set of zoo world test problems solved by CCALC. The CCALC formalization consists of 62 causal laws and our event calculus translation consists of 78 axioms. As shown in Table 3, the performance of our tool on the test problems is comparable to that of CCALC. The CCALC runs were performed with CCALC 2.0 beta 8.3, SWI-Prolog 5.0.10, and Relsat 2.0. The CCALC encoding time is the sum of the grounding and completion times. Encoding and solution times reported in this paper are elapsed wall-clock time in seconds on a machine with a 1.8 GHz Intel Pentium 4 processor and 512 megabytes of RAM.

Problem		Vars	Clauses	Encode	Solve
ZooTest1		3,239	23,819	23.35	1.18
	c	2,693	31,881	14.83	12.24
ZooTest2		982	5,671	1.86	0.14
	c	1,116	8,870	3.67	0.17
ZooTest3		1,805	12,604	8.74	0.53
	c	1,726	17,895	15.08	0.61
ZooTest4.1		3,239	23,816	23.22	1.19
	c	2,770	32,193	14.98	4.14
ZooTest4.2		3,980	29,572	34.24	1.39
	c	3,292	39,354	14.90	11.54
ZooTest5.1		1,670	12,906	21.92	1.50
	c	1,483	18,120	44.69	5.04
ZooTest5.2		1,670	12,908	22.11	0.66
	c	1,483	18,122	44.69	5.06
ZooTest6		1,088	6,996	4.02	0.30
	c	1,127	10,428	14.85	1.10

Table 3: Comparison with CCALC (c) on zoo problems (wall times in seconds)

Natural Language Applications

In addition to solving benchmark problems, our tool is useful for real-world applications. It provides an effective mechanism for carrying out inferences and filling in missing information in natural language understanding applications (Mueller 2003). The tool has been used to build two applications that expand a semantic parse of a narrative text into a more detailed model of the narrative. Questions about the narrative are then answered by consulting the model.

A commonsense knowledge base has been developed for use with our tool, which is used by the two applications. The knowledge base consists of 445 axioms.

The first application (Mueller 2004a) used the tool to build models of terrorist incidents given arbitrary templates of the sort produced by MUC3 and MUC4 information extraction systems. The application produced models for 652 terrorism templates. It generated questions and answers such as the following:

> Q: Were the villagers angry at the unidentified persons after the unidentified persons threatened the villagers?
> A: Yes.
> Q: Was the grocery store intact after the fire damaged the grocery store?
> A: No.

Problem	Vars	Clauses	Encode	Solve
ARSON	1,010	4,765	1.03	0.09
KIDNAPPING	5,553	27,070	19.86	0.91
SHOOTINGATTACK	1,933	10,138	3.28	0.28
BOMBING	8,258	40,277	44.79	1.90

Table 4: Terrorist incident understanding statistics (wall times in seconds)

Q: Were the unidentified persons present when the fire damaged the grocery store?
A: Yes.

Table 4 provides statistics on processing typical reasoning problems for each type of terrorist incident.

The second application (Mueller 2004c) used information extraction techniques to extract episodes involving dining in a restaurant from narrative texts, used the tool to build models of the dining episodes, and generated questions and answers about the episodes. The application produced models for 61 web texts and 52 American literature excerpts. A typical reasoning problem had 5,160 variables and 43,205 clauses, was encoded in 25.40 seconds, and was solved in 1.32 seconds.

Conclusions

We have presented a tool for automated commonsense reasoning in the classical logic event calculus. The tool works by converting a first-order logic problem into propositional logic, invoking an off-the-shelf satisfiability solver on the propositional logic problem, and then decoding the solutions produced by the solver. The tool successfully solves 12 of 14 benchmark commonsense reasoning problems described for the event calculus, has performance comparable to the causal calculator in the zoo world domain, and has been used to perform inferencing in two natural language understanding applications.

References

Akman, V.; Erdogan, S. T.; Lee, J.; Lifschitz, V.; and Turner, H. 2004. Representing the zoo world and the traffic world in the language of the causal calculator. *Artificial Intelligence* 153:105–140.

Bayardo Jr., R. J., and Schrag, R. C. 1997. Using CSP look-back techniques to solve real world SAT instances. In *Proceedings of AAAI-1997*, 203–208. Menlo Park, CA: AAAI Press.

Clark, K. L. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. New York: Plenum. 293–322.

Davis, E. 1990. *Representations of Commonsense Knowledge*. San Mateo, CA: Morgan Kaufmann.

de Nivelle, H. 1999. Bliksem 1.10 user manual.

Doherty, P.; Gustafsson, J.; Karlsson, L.; and Kvarnström, J. 1998. TAL: Temporal Action Logics language specification and tutorial. *Linköping Electronic Articles in Computer and Information Science* 3(015).

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153:49–104.

Haas, A. R. 1987. The case for domain-specific frame axioms. In Brown, F. M., ed., *The Frame Problem in Artificial Intelligence*, 343–348. Los Altos, CA: Morgan Kaufmann.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-1996*, 1194–1201. Menlo Park, CA: AAAI Press.

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

Kvarnström, J. 2001. VITAL: Visualization and implementation of temporal action logic.

McCain, N., and Turner, H. 1997. Causal theories of action and change. In *Proceedings of AAAI-1997*, 460–465. Menlo Park, CA: AAAI Press.

Miller, R., and Shanahan, M. 2002. Some alternative formulations of the event calculus. In *Lecture Notes in Computer Science*, volume 2408. Heidelberg: Springer-Verlag. 452–490.

Mueller, E. T. 2003. Story understanding through multi-representation model construction. In Hirst, G., and Nirenburg, S., eds., *Text Meaning: Proceedings of the HLT-NAACL 2003 Workshop*, 46–53. East Stroudsburg, PA: Association for Computational Linguistics.

Mueller, E. T. 2004a. Combining information extraction and commonsense reasoning for text understanding. Manuscript submitted for publication.

Mueller, E. T. 2004b. Event calculus reasoning through satisfiability. Manuscript submitted for publication.

Mueller, E. T. 2004c. Modelling space and time in narratives about restaurants. Manuscript submitted for publication.

Plaisted, D. A., and Greenbaum, S. 1986. A structure-preserving clause form translation. *Journal of Symbolic Computation* 2:293–304.

Reiter, R. 2001. *Knowledge in Action*. Cambridge, MA: MIT Press.

Selman, B.; Kautz, H.; and Cohen, B. 1993. Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26. Boston, MA: American Mathematical Society.

Shanahan, M., and Witkowski, M. 2002. Event calculus planning through satisfiability. Manuscript submitted for publication.

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

Shanahan, M. 1999a. The event calculus explained. In *Lecture Notes in Computer Science*, volume 1600. Heidelberg: Springer-Verlag. 409–430.

Shanahan, M. 1999b. The ramification problem in the event calculus. In *Proceedings of IJCAI-1999*, 140–146. San Mateo, CA: Morgan Kaufmann.