Discrete Event Calculus Deduction using First-Order Automated Theorem Proving

Erik T. Mueller[1] and Geoff Sutcliffe[2]

[1] IBM Thomas J. Watson Research Center,
P.O. Box 704, Yorktown Heights, NY 10598 USA `etm@us.ibm.com`
[2] Department of Computer Science, University of Miami,
P.O. Box 248154, Coral Gables, FL 33124 USA `geoff@cs.miami.edu`

Abstract. The event calculus is a powerful and highly usable formalism for reasoning about action and change. The discrete event calculus limits time to integers. This paper shows how discrete event calculus problems can be encoded in first-order logic, and solved using a first-order logic automated theorem proving system. The following techniques are discussed: reification is used to convert event and fluent atoms into first-order terms, uniqueness-of-names axioms are generated to ensure uniqueness of event and fluent terms, predicate completion is used to convert second-order circumscriptions into first-order formulae, and a limited first-order axiomatization of integer arithmetic is developed. The performance of first-order automated theorem proving is compared to that of satisfiability solving.

1 Introduction

The event calculus (EC) [1] 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 [2] and robotics [3]. Kowalski and Sergot [4] 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 [5, 6].

The discrete event calculus (DEC) was developed by Mueller [7] in order to facilitate solution of event calculus reasoning problems using satisfiability (SAT) solvers. DEC facilitates this by

– limiting time to the integers to allow a SAT encoding (unlike EC, which allows continuous time), and
– eliminating triply quantified time from many of the axioms to reduce the size of the SAT encoding.

Mueller [7] proves that if time is restricted to the integers, then DEC is equivalent to an EC axiomatization of Miller and Shanahan [6]. The DEC axioms are given in the appendix of this paper. The predicates used in the axioms are:

– *happens*(E, T): Event E occurs at timepoint T.
– *holdsAt*(F, T): Fluent F is true at T.
– *releasedAt*(F, T): F is released from the commonsense law of inertia at T.

- *initiates*(E, F, T): If E occurs at T, then F is true and not released at $T + 1$.
- *terminates*(E, F, T): If E occurs at T, then F is false and not released at $T + 1$.
- *releases*(E, F, T): If E occurs at T, then F is released at $T + 1$.
- *trajectory*(F_1, T_1, F_2, T_2): If F_1 is initiated by an event that occurs at T_1, and T_2 is greater than zero, then F_2 is true at $T_1 + T_2$.
- *antiTrajectory*(F_1, T_1, F_2, T_2): If F_1 is terminated by an event that occurs at T_1, and T_2 is greater than zero, then F_2 is true at $T_1 + T_2$.

Since the introduction of the EC and DEC axiomatizations, several event calculus reasoning systems have been implemented, including:

- Shanahan's EC planner [8], which uses abductive logic programming,
- Shanahan and Witkowski's EC planner [9], which uses SAT solvers, and
- Mueller's DEC reasoner [10], which uses SAT solvers.

In this paper, we demonstrate the feasibility of using first-order logic automated theorem proving (ATP) systems [11] to solve event calculus reasoning problems. To our knowledge, this is the first time this has been done. We limit ourselves here to discrete time.

Our long-term goal is to develop a collection of systems for solving event calculus reasoning problems, both discrete and continuous, using both ATP and SAT. Depending on the user's problem and needs, one or more of these systems can be selected. The chief benefit of ATP is that it produces humanly-understandable proofs (refutations), while the chief benefits of SAT are its efficiency and ability to perform abduction and model finding as well as deduction.

In order to make DEC problems solvable using ATP systems, we

- use reification to convert event and fluent atoms into first-order terms,
- generate uniqueness-of-names axioms to ensure uniqueness of event and fluent terms,
- use predicate completion to convert second-order circumscriptions into first-order formulae, and
- use a limited first-order axiomatization of integer arithmetic.

Our method has been tested on two benchmark scenarios for the event calculus, which together cover many of the features of the event calculus: the supermarket trolley scenario and the kitchen sink scenario. For some theorems, human assistance in the form of lemma specifications is required to bring the problems within the reach of the current state of the art in ATP.

2 Encoding DEC Problems

Encoding DEC problems for ATP systems requires solving several technical and practical problems, which are discussed in this section. The techniques are described using examples from the kitchen sink scenario, in which a stopper is put into the drain of a kitchen sink and the water is turned on (the scenario is fully specified later in this paper).

2.1 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*$(2), 3)$. However, this is not a well-formed first-order logic formula if *waterLevel*(2) is an atom, since the first argument to the predicate symbol *holdsAt* is not a term. Similarly, *happens*$($*tapOn*$, 0)$ is not well-formed if *tapOn* is a proposition. The event calculus therefore uses the technique of *reification* [12], in which formulae of one first-order language become terms of another first-order language.

Reification for the event calculus uses techniques originally developed for the situation calculus by Lifschitz [13], which were adapted for the event calculus by Shanahan [5]. 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. Each DEC predicate and function has a sort signature that defines the sorts of its arguments. These signatures are specified for each of the DEC predicates in the introduction of this paper, e.g., the arguments of *holdsAt* are of sort fluent and timepoint, and the arguments of *happens* are of sort event and timepoint. 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, e.g., *waterLevel* terms are of sort fluent, and *tapOn* terms are of sort event.

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*$(3))$.

2.2 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 [13], 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 is an n-ary function symbol, f_j is an m-ary function symbol, and $x_1, \ldots, x_m, y_1, \ldots, y_{n/m}$ are distinct variables, for every $i, j \in \{1, \ldots, k\}$ such that $i < j$.

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. For example, if the fluent function symbols are *waterLevel* and *waterVolume*, we add *U*$[$*waterLevel*$,$ *waterVolume*$]$. From this, we can show, for example, that *waterLevel*$(2) \neq$ *waterVolume*(2) and *waterLevel*$(2) \neq$ *waterLevel*(3) (since $2 \neq 3$).

2.3 Circumscription

Consider the following scenario axioms:

$$\forall T\ \mathit{initiates}(\mathit{tapOn}, \mathit{filling}, T),$$
$$\forall T\ \mathit{terminates}(\mathit{tapOff}, \mathit{filling}, T),$$
$$\mathit{happens}(\mathit{tapOn}, 0).$$

These axioms specify events that initiate and terminate fluents, and a *tapOn* event. However, they do not specify what events do *not* initiate and terminate particular fluents, and they do not specify what events do *not* occur. Thus there are models of the DEC and these axioms in which, e.g., $\forall T\ \mathit{terminates}(\mathit{waterOutage}, \mathit{filling}, T)$ and $\mathit{happens}(\mathit{waterOutage}, 1)$ are true. The event calculus uses minimization of the extension of a predicate, or *circumscription* [14], to minimize unexpected effects of events and unexpected event occurrences, by minimizing the extensions of the predicates *initiates*, *terminates*, *releases*, and *happens*.

Computing circumscription is in general difficult [15]. 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 [16]. Fortunately, in many cases, including the benchmark scenarios considered in this paper, the circumscription can be computed using the following theorem [16, 17], which reduces circumscription to predicate completion:

Theorem 1. Let ρ be an n-ary predicate symbol and $\Gamma(x_1, \ldots, x_n)$ be a formula with only $x_1, \ldots, x_n$ free. If $\Gamma(x_1, \ldots, x_n)$ does not mention ρ, then the circumscription

$$\mathit{CIRC}[\forall x_1, \ldots, x_n\ (\Gamma(x_1, \ldots, x_n) \Rightarrow \rho(x_1, \ldots, x_n)); \rho]$$

is equivalent to

$$\forall x_1, \ldots, x_n\ (\Gamma(x_1, \ldots, x_n) \Leftrightarrow \rho(x_1, \ldots, x_n)).$$

2.4 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. General purpose first-order axiomatizations of integer arithmetic, such as Peano arithmetic, may produce very large search spaces and very large terms, which hinder the performance of ATP systems. An alternative approach is to make a computer algebra system available to an ATP system as a trusted external tool, and to have the ATP system recognize arithmetic expressions and relegate their solution to the computer algebra system [18].

For this work a first-order axiomatization of a small fragment of integer arithmetic has been encoded as first-order logic axioms. The axioms 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.

– Addition is dealt with by enumerating the results of adding all pairs of ordered integers, and providing the axiom of symmetry.

– Ordering is described by axioms that specify the adjacent pairs of integers in conjunction with a recursive definition of transitivity via the $\leq$ definition.
– The transitive sequence of ordered pairs is terminated by the axiom that specifies that there is nothing less than 0.
– 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).

$$\begin{array}{l}
0+0=0, \\
0+1=1, \\
\ldots \\
8+1=9, \\
\forall X, Y\; X+Y=Y+X, \\
\forall X, Y\; (X \leq Y \Leftrightarrow (X<Y \vee X=Y)), \\
\forall X\; (X<1 \Leftrightarrow X \leq 0), \\
\ldots \\
\forall X\; (X<9 \Leftrightarrow X \leq 8), \\
\neg \exists X\; X<0, \\
\forall X, Y\; (X<Y \Leftrightarrow (\neg(Y<X) \wedge Y \neq X)).
\end{array}$$

Note that only the “less” inequalities are used, with “greater” being expressed by reversal of arguments and negation.

3 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 formulae are written in the TSTP syntax [19], ready for submission to an ATP system. The ATP problems were submitted to the ATP system Vampire 7.0 [20]. Vampire is acknowledged to be a state-of-the-art ATP system—Vampire 7.0 won the FOF division of the 2004 CADE ATP system competition [21]. Initial testing was also performed using other ATP systems, including E [22] and SPASS [23], and the results showed that Vampire consistently outperforms those systems on these problems. Testing was done on a Dell P3 computer, with a 930 MHz CPU, 512 MB of memory, and the Linux 2.4.20-6 operating system. A CPU time limit of 300s was imposed on each run.

3.1 The Supermarket Trolley Scenario

The supermarket trolley scenario, introduced by Shanahan [5], 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 [5], reformulated using the technique of Miller and Shanahan [6], which involves adding *happens* preconditions to *initiates* and *terminates* axioms. The axiomatization is preprocessed to:

Circumscribed *initiates* axioms
$\forall E, F, T\ (\mathit{initiates}(E, F, T) \Leftrightarrow$
$((E = \mathit{push} \wedge F = \mathit{forwards} \wedge \neg \mathit{happens}(\mathit{pull}, T)) \vee$
$(E = \mathit{pull} \wedge F = \mathit{backwards} \wedge \neg \mathit{happens}(\mathit{push}, T)) \vee$
$(E = \mathit{pull} \wedge F = \mathit{spinning} \wedge \mathit{happens}(\mathit{push}, T)))).$

Circumscribed *terminates* axioms
$\forall E, F, T\ (\mathit{terminates}(E, F, T) \Leftrightarrow$
$((E = \mathit{push} \wedge F = \mathit{backwards} \wedge \neg \mathit{happens}(\mathit{pull}, T)) \vee$
$(E = \mathit{pull} \wedge F = \mathit{forwards} \wedge \neg \mathit{happens}(\mathit{push}, T)) \vee$
$(E = \mathit{pull} \wedge F = \mathit{forwards} \wedge \mathit{happens}(\mathit{push}, T)) \vee$
$(E = \mathit{pull} \wedge F = \mathit{backwards} \wedge \mathit{happens}(\mathit{push}, T)) \vee$
$(E = \mathit{push} \wedge F = \mathit{spinning} \wedge \neg \mathit{happens}(\mathit{pull}, T)) \vee$
$(E = \mathit{pull} \wedge F = \mathit{spinning} \wedge \neg \mathit{happens}(\mathit{push}, T)))).$

Circumscribed *releases* axioms
$\forall E, F, T\ \neg \mathit{releases}(E, F, T).$

Circumscribed event occurrences
$\forall E, T\ (\mathit{happens}(E, T) \Leftrightarrow$
$((E = \mathit{push} \wedge T = 0) \vee (E = \mathit{pull} \wedge T = 1) \vee$
$(E = \mathit{pull} \wedge T = 2) \vee (E = \mathit{push} \wedge T = 2))).$

Uniqueness-of-names axioms for events
$\mathit{push} \neq \mathit{pull}.$

Uniqueness-of-names axioms for fluents
$\mathit{forwards} \neq \mathit{backwards},$
$\mathit{forwards} \neq \mathit{spinning},$
$\mathit{spinning} \neq \mathit{backwards}.$

Initial conditions
$\neg \mathit{holdsAt}(\mathit{forwards}, 0),$
$\neg \mathit{holdsAt}(\mathit{backwards}, 0),$
$\neg \mathit{holdsAt}(\mathit{spinning}, 0),$
$\forall F, T\ \neg \mathit{releasedAt}(F, T).$

The axiomatization of the supermarket trolley scenario consists of 47 axioms (23 axioms for integer arithmetic, 12 axioms for DEC, 8 axioms for the domain theory, and 4 axioms for initial conditions).

Given the above axioms, the DEC axioms, and the integer arithmetic axioms, Vampire is quickly (less than 1s each) able to prove the theorems:

$\neg \mathit{holdsAt}(\mathit{spinning}, 1)$,
$\mathit{holdsAt}(\mathit{backwards}, 2)$,
$\neg \mathit{holdsAt}(\mathit{forwards}, 2)$,
$\neg \mathit{holdsAt}(\mathit{spinning}, 2)$,
$\neg \mathit{holdsAt}(\mathit{backwards}, 3)$,
$\neg \mathit{holdsAt}(\mathit{forwards}, 3)$,
$\mathit{holdsAt}(\mathit{spinning}, 3)$.

3.2 The Kitchen Sink Scenario

The kitchen sink scenario, introduced by Shanahan [24], is used to test *initiates* and *terminates* axioms representing the effects of events, *releases* axioms representing release from the commonsense law of inertia, *trajectory* axioms representing gradual change, trigger axioms representing triggered events, and state constraints. In this scenario, 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. The axiomatization of this problem is taken from Shanahan [5], and preprocessed to:

Circumscribed *initiates* axioms
$\forall E, F, T\ (\mathit{initiates}(E, F, T) \Leftrightarrow$
$((E = \mathit{tapOn} \wedge F = \mathit{filling}) \vee$
$(E = \mathit{overflow} \wedge F = \mathit{spilling}) \vee$
$\exists H\ (\mathit{holdsAt}(\mathit{waterLevel}(H), T) \wedge E = \mathit{tapOff} \wedge$
$F = \mathit{waterLevel}(H)) \vee$
$\exists H\ (\mathit{holdsAt}(\mathit{waterLevel}(H), T) \wedge E = \mathit{overflow} \wedge$
$F = \mathit{waterLevel}(H))))$.

Circumscribed *terminates* axioms
$\forall E, F, T\ (\mathit{terminates}(E, F, T) \Leftrightarrow$
$((E = \mathit{tapOff} \wedge F = \mathit{filling}) \vee$
$(E = \mathit{overflow} \wedge F = \mathit{filling})))$.

Circumscribed *releases* axioms
$\forall E, F, T\ (\mathit{releases}(E, F, T) \Leftrightarrow$
$\exists H\ (E = \mathit{tapOn} \wedge F = \mathit{waterLevel}(H)))$.

Circumscribed event occurrence and trigger axiom
$\forall E, T\ (\mathit{happens}(E, T) \Leftrightarrow ((E = \mathit{tapOn} \wedge T = 0) \vee$
$(\mathit{holdsAt}(\mathit{waterLevel}(3), T) \wedge \mathit{holdsAt}(\mathit{filling}, T) \wedge$
$E = \mathit{overflow})))$.

Trajectory axiom
$\forall H_1, T_1, H_2, O\ ((\mathit{holdsAt}(\mathit{waterLevel}(H_1), T_1) \wedge$
$H_2 = H_1 + O) \Rightarrow \mathit{trajectory}(\mathit{filling}, T_1, \mathit{waterLevel}(H_2), O))$.

State constraint
$\forall T, H_1, H_2\ ((\mathit{holdsAt}(\mathit{waterLevel}(H_1), T) \wedge$
$\mathit{holdsAt}(\mathit{waterLevel}(H_2), T)) \Rightarrow H_1 = H_2)$.

Uniqueness-of-names axioms for events
$\mathit{tapOff} \neq \mathit{tapOn}$,
$\mathit{tapOff} \neq \mathit{overflow}$,
$\mathit{overflow} \neq \mathit{tapOn}$.

Uniqueness-of-names axioms for fluents
$\forall X\ \mathit{filling} \neq \mathit{waterLevel}(X)$,
$\forall X\ \mathit{spilling} \neq \mathit{waterLevel}(X)$,
$\mathit{filling} \neq \mathit{spilling}$,
$\forall X, Y\ (\mathit{waterLevel}(X) = \mathit{waterLevel}(Y) \Leftrightarrow X = Y)$.

Initial conditions
$\mathit{holdsAt}(\mathit{waterLevel}(0), 0)$,
$\neg\mathit{holdsAt}(\mathit{filling}, 0)$,
$\neg\mathit{holdsAt}(\mathit{spilling}, 0)$,
$\forall H\ \neg\mathit{releasedAt}(\mathit{waterLevel}(H), 0)$,
$\neg\mathit{releasedAt}(\mathit{filling}, 0)$,
$\neg\mathit{releasedAt}(\mathit{spilling}, 0)$.

The axiomatization of the kitchen sink scenario consists of 54 axioms (23 axioms for integer arithmetic, 12 axioms for DEC, 13 axioms for the domain theory, and 6 axioms for initial conditions).

These axioms, in conjunction with the DEC axioms and the integer arithmetic axioms, form a specification of the problem for times and heights within the axiomatized integer range. Various theorems that describe the state of the system at specified times can be proved directly from the axioms, including (the CPU times taken are given in ()s):

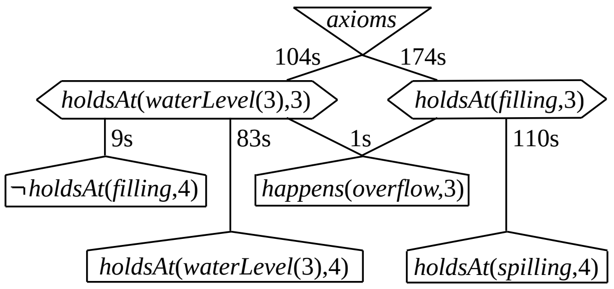
$\mathit{holdsAt}(\mathit{filling}, 1)$ (1s)
$\mathit{holdsAt}(\mathit{waterLevel}(1), 1)$ (2s)
$\mathit{holdsAt}(\mathit{filling}, 2)$ (3s)
$\mathit{holdsAt}(\mathit{waterLevel}(2), 2)$ (3s)
$\neg\exists E\ (\mathit{happens}(E, 2) \wedge \mathit{terminates}(E, \mathit{filling}, 2))$ (42s)
$\mathit{holdsAt}(\mathit{waterLevel}(3), 3)$ (104s)
$\neg\mathit{stoppedIn}(0, \mathit{filling}, 3)$ (110s)
$\mathit{holdsAt}(\mathit{filling}, 3)$ (174s)

For more complex theorems such as:

$$\begin{aligned}&\mathit{happens}(\mathit{overflow}, 3),\\ &\neg\mathit{holdsAt}(\mathit{filling}, 4),\\ &\mathit{holdsAt}(\mathit{waterLevel}(3), 4),\\ &\mathit{holdsAt}(\mathit{spilling}, 4),\end{aligned}$$

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. Such an incremental approach to proving hard theorems has been used in previous ATP applications, e.g., Art Quaife's development of Neumann-Bernays-Godel set theory [25]. 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.

Fig. 1. Lemmas for the Kitchen Sink Theorems



3.3 ATP vs. SAT

In this section we compare the performance of ATP and a SAT-based DEC reasoner [10] on a version of the supermarket trolley scenario with n agents and n trolleys: For each i in $\{1, \ldots, n\}$, agent i pushes and pulls trolley i at timepoint 0. The problem is to prove that for each i in $\{1, \ldots, n\}$, trolley i spins at timepoint 1. The results are shown in Table 1. The columns of this table are: (1) **n**: number of agents and trolleys, (2) **time**: wall time for the DEC reasoner, including running the Relsat 2.0 SAT solver, (3) **SAT time**: wall time for the SAT solver alone, (4) **vars**: number of variables in the SAT problem, (5) **clauses**: number of clauses in the SAT problem, (6) **time**: wall time for Vampire, (7) **gclauses**: number of generated clauses, and (8) **rclauses**: number of retained clauses. Times here are elapsed wall-clock time in seconds, averaged over 10 trials, on an IBM T30 computer, with a 1.8 GHz Intel Pentium 4 CPU, 512 MB of memory, and the Linux 2.4.9-31 operating system. Though SAT is more efficient than ATP for this scenario, ATP has the benefit that the derivation retains meaning and can be understood by humans.

	DEC reasoner				Vampire		
n	**time**	**SAT time**	**vars**	**clauses**	**time**	**gclauses**	**rclauses**
1	0.2	0.0	16	70	1.0	12,791	3,828
2	0.3	0.0	61	328	1.1	13,002	4,003
4	0.7	0.0	190	1,084	1.4	13,637	4,350
8	2.4	0.1	625	3,562	7.8	79,754	5,091
9	3.0	0.1	765	4,446	26.0	276,582	5,287
10	3.9	0.1	919	5,428	fail		

Table 1. DEC reasoner (SAT) vs. Vampire (ATP) on supermarket trolley problems (wall times in seconds)

4 Related Work

The general topic of commonsense reasoning is widely and deeply studied. It includes work that uses automated reasoning, whose roots are in John McCarthy's paper "Programs with Common Sense" [26]. The use of automated reasoning techniques in commonsense reasoning produced significant output, including, e.g., a special issue of the *Journal of Automated Reasoning* [27]. Despite the high level of activity, there appears to be little work on performing commonsense reasoning using classical first-order logic ATP systems. A similar state of affairs appears to exist in the subfield of reasoning about action and change. Existing systems for reasoning about action and change use task-specific logics and reasoning techniques, including: active logic reasoning [28], abductive logic programming [8], answer set computing algorithms [29], argumentation programming [30], model finding via constraint propagation [31], and SAT solving [32, 10, 9].

Planning is one type of reasoning about action and change in which first-order ATP systems have been employed. Green [33] implemented a system that used resolution theorem proving for planning. Citing performance problems with Green's system, Fikes and Nilsson [34] introduced STRIPS, which used means-ends analysis to search the space of plans, and resolution theorem proving only for proving subgoals and operator preconditions. Kautz and Selman [35] demonstrated the efficiency of SAT solving for planning. Modern planning systems use a variety of techniques including planning graph analysis, forward heuristic search, SAT solving, model checking, and planning by rewriting [36]. The TPTP problem library [37] contains a planning domain with 38 planning problems solved by ATP systems.

5 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.

The DEC problems are a reasonable challenge for ATP systems, from two perspectives. First, the use of arithmetic requires either an axiomatic solution (as described in this work), or the integration of arithmetic capabilities into the ATP system. Both of these alternatives are the focus of research in the ATP community, and this work in DEC further motivates that research. We are already in the process of replacing the small fragment of integer arithmetic, described in Section 2.4, with a more robust axiomatization of equality, addition, and order for byte arithmetic. The real arithmetic that is necessary for non-discrete time will be implemented using the built-in arithmetic capabilities of an ATP system, such as Otter [38]. To make ATP systems more applicable, it will be important for the ATP community to address the issue of standardizing mechanisms for built-in arithmetic.

Second, in the form described in this paper, the problems are suitable for testing ATP systems, because they lie at the frontier of the current state of the art. The problems have been incorporated into version 3.1.0 of the TPTP problem library [37] 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. We also plan to investigate the possibility of transforming problems into those with a finite Herbrand universe, so that specialized EPR (effectively propositional) solvers can be used.

6 Appendix: Discrete Event Calculus Axioms

The following DEC axioms were formed by Mueller [7], by introducing the new axioms DEC5 through DEC12, and adding them to the existing axioms DEC1 through DEC4 of Miller and Shanahan [6]:

Axiom DEC1
$\forall T_1, F, T_2\, (\mathit{stoppedIn}(T_1, F, T_2) \Leftrightarrow$
$\exists E, T\, (T_1 < T < T_2 \wedge \mathit{happens}(E, T) \wedge \mathit{terminates}(E, F, T)))$.

Axiom DEC2
$\forall T_1, F, T_2\, (\mathit{startedIn}(T_1, F, T_2) \Leftrightarrow$
$\exists E, T\, (T_1 < T < T_2 \wedge \mathit{happens}(E, T) \wedge \mathit{initiates}(E, F, T)))$.

Axiom DEC3
$\forall E, T_1, F_1, T_2, F_2\, ((\mathit{happens}(E, T_1) \wedge \mathit{initiates}(E, F_1, T_1) \wedge 0 < T_2 \wedge$
$\mathit{trajectory}(F_1, T_1, F_2, T_2) \wedge \neg \mathit{stoppedIn}(T_1, F_1, T_1 + T_2)) \Rightarrow$
$\mathit{holdsAt}(F_2, T_1 + T_2))$.

Axiom DEC4
$\forall E, T_1, F_1, T_2, F_2\, ((\mathit{happens}(E, T_1) \wedge \mathit{terminates}(E, F_1, T_1) \wedge 0 < T_2 \wedge$
$\mathit{antiTrajectory}(F_1, T_1, F_2, T_2) \wedge \neg \mathit{startedIn}(T_1, F_1, T_1 + T_2)) \Rightarrow$
$\mathit{holdsAt}(F_2, T_1 + T_2))$.

Axiom DEC5
$\forall F, T\ ((\mathit{holdsAt}(F, T) \wedge \neg \mathit{releasedAt}(F, T+1) \wedge$
$\neg \exists E\ (\mathit{happens}(E, T) \wedge \mathit{terminates}(E, F, T))) \Rightarrow$
$\mathit{holdsAt}(F, T+1)).$

Axiom DEC6
$\forall F, T\ ((\neg \mathit{holdsAt}(F, T) \wedge \neg \mathit{releasedAt}(F, T+1) \wedge$
$\neg \exists E\ (\mathit{happens}(E, T) \wedge \mathit{initiates}(E, F, T))) \Rightarrow$
$\neg \mathit{holdsAt}(F, T+1)).$

Axiom DEC7
$\forall F, T\ ((\mathit{releasedAt}(F, T) \wedge$
$\neg \exists E\ (\mathit{happens}(E, T) \wedge (\mathit{initiates}(E, F, T) \vee \mathit{terminates}(E, F, T)))) \Rightarrow$
$\mathit{releasedAt}(F, T+1)).$

Axiom DEC8
$\forall F, T\ ((\neg \mathit{releasedAt}(F, T) \wedge$
$\neg \exists E\ (\mathit{happens}(E, T) \wedge \mathit{releases}(E, F, T))) \Rightarrow$
$\neg \mathit{releasedAt}(F, T+1)).$

Axiom DEC9
$\forall E, T, F\ ((\mathit{happens}(E, T) \wedge \mathit{initiates}(E, F, T)) \Rightarrow \mathit{holdsAt}(F, T+1)).$

Axiom DEC10
$\forall E, T, F\ ((\mathit{happens}(E, T) \wedge \mathit{terminates}(E, F, T)) \Rightarrow \neg \mathit{holdsAt}(F, T+1)).$

Axiom DEC11
$\forall E, T, F\ ((\mathit{happens}(E, T) \wedge \mathit{releases}(E, F, T)) \Rightarrow \mathit{releasedAt}(F, T+1)).$

Axiom DEC12
$\forall E, T, F\ ((\mathit{happens}(E, T) \wedge (\mathit{initiates}(E, F, T) \vee \mathit{terminates}(E, F, T))) \Rightarrow$
$\neg \mathit{releasedAt}(F, T+1)).$

References

1. Shanahan, M.: The event calculus explained. In: Artificial Intelligence Today. Springer-Verlag, Heidelberg (1999) 409–430
2. Mueller, E.T.: Understanding script-based stories using commonsense reasoning. Cognitive Systems Research **5** (2004) 307–340
3. Shanahan, M.: Perception as abduction: Turning sensor data into meaningful representation. Cognitive Science **29** (2005) 103–134
4. Kowalski, R.A., Sergot, M.J.: A logic-based calculus of events. New Generation Computing **4** (1986) 67–95
5. Shanahan, M.: Solving the Frame Problem. MIT Press, Cambridge, MA (1997)
6. Miller, R., Shanahan, M.: Some alternative formulations of the event calculus. In: Computational Logic: Logic Programming and Beyond. Springer-Verlag, Heidelberg (2002) 452–490

7. Mueller, E.T.: Event calculus reasoning through satisfiability. Journal of Logic and Computation **14** (2004) 703–730
8. Shanahan, M.: An abductive event calculus planner. Journal of Logic Programming **44** (2000) 207–240
9. Shanahan, M., Witkowski, M.: Event calculus planning through satisfiability. Journal of Logic and Computation **14** (2004) 731–745
10. Mueller, E.T.: A tool for satisfiability-based commonsense reasoning in the event calculus. In: Proceedings of the 17th FLAIRS Conference, Menlo Park, CA, AAAI Press (2004) 147–152
11. Robinson, A., Voronkov, A.: Handbook of Automated Reasoning. Elsevier Science (2001)
12. McCarthy, J.: First order theories of individual concepts and propositions. In: Machine Intelligence 9. Ellis Horwood, Chichester, UK (1979) 129–148
13. Lifschitz, V.: Formal theories of action. In: The Frame Problem in Artificial Intelligence, Los Altos, CA, Morgan Kaufmann (1987) 35–57
14. McCarthy, J.: Circumscription—a form of non-monotonic reasoning. Artificial Intelligence **13** (1980) 27–39
15. Doherty, P., Łukaszewicz, W., Szałas, A.: Computing circumscription revisited: A reduction algorithm. Journal of Automated Reasoning **18** (1997) 297–336
16. Lifschitz, V.: Computing circumscription. In: Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Altos, CA, Morgan Kaufmann (1985) 121–127
17. Reiter, R.: Circumscription implies predicate completion (sometimes). In: Proceedings of the National Conference on Artificial Intelligence, Menlo Park, CA, AAAI Press (1982) 418–420
18. Kapur, D., Wang, D.: Special issue: Combining logical reasoning and algebraic computation. Journal of Automated Reasoning **21** (1998)
19. Sutcliffe, G., Zimmer, J., Schulz, S.: TSTP data-exchange formats for automated theorem proving tools. In: Distributed and Multi-Agent Reasoning. IOS Press (2004)
20. Riazanov, A., Voronkov, A.: The design and implementation of Vampire. AI Communications **15** (2002) 91–110
21. Nieuwenhuis, R.: Special issue: The CADE ATP System Competition. AI Communications **15** (2002)
22. Schulz, S.: E: A Brainiac Theorem Prover. AI Communications **15** (2002) 111–126
23. Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobald, C., Topic, D.: SPASS Version 2.0. In Voronkov, A., ed.: Proceedings of the 18th International Conference on Automated Deduction. Number 2392 in Lecture Notes in Artificial Intelligence, Springer-Verlag (2002) 275–279
24. Shanahan, M.: Representing continuous change in the event calculus. In: Proc. of ECAI 1990. (1990) 598–603
25. Quaife, A.: Automated Development of Fundamental Mathematical Theories. Kluwer Academic Publishers (1992)
26. McCarthy, J.: Programs with common sense. In: Proceedings of the Symposium on Mechanisation of Thought Processes, Her Majesty's Stationery Office (1959)
27. Lifschitz, V.: Special issue: Commonsense and nonmonotonic reasoning. Journal of Automated Reasoning **15** (1995)
28. Perlis, D.: Active logic, metacognitive computation, and mind. `http://www.activelogic.org` (2004)
29. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
30. Kakas, A., Miller, R., Toni, F.: E-RES - A system for reasoning about actions, events and observations. In: Proc. of the 8th International Workshop on Non-Monotonic Reasoning. (2000)

31. Kvarnström, J.: VITAL: Visualization and implementation of temporal action logic. Technical report, Linköping University (2001)
32. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artificial Intelligence **153** (2004) 49–104
33. Green, C.C.: The Application of Theorem Proving to Question-Answering Systems. PhD thesis, Department of Electrical Engineering, Stanford University (1969)
34. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2** (1971) 189–208
35. Kautz, H., Selman, B.: Pushing the envelope: Planning, propositional logic, and stochastic search. In: Proc. AAAI/IAAI 1996. (1996) 1194–1201
36. Long, D., Fox, M.: The 3rd International Planning Competition: Results and analysis. Journal of Artificial Intelligence Research **20** (2003) 1–59
37. Sutcliffe, G., Suttner, C.: The TPTP Problem Library: CNF release v1.2.1. Journal of Automated Reasoning **21** (1998) 177–203
38. McCune, W.: Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA (2003)