Automating Commonsense Reasoning Using the Event Calculus

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

Preprint of December 23, 2006

Citation: Mueller, Erik T. (in press). Automating commonsense reasoning using the event calculus. *Communications of the ACM.*

Commonsense reasoning is the human ability to make inferences about properties and events in the everyday world. The automation of commonsense reasoning, long a goal of the field of artificial intelligence [3] and an area of active research in the last decade [8], is attaining a level of maturity. Automating commonsense reasoning allows us to build applications that are more user-friendly and more understanding of the world [2].

Several major computational approaches to commonsense reasoning have been explored. Analogical processing implements the notion that people reason about novel situations by analogy to familiar ones. Probability theory allows us to reason given uncertain knowledge of the state of the world and how the world works. Qualitative reasoning focuses on reasoning about physical systems. Methods based on natural language make use of large textual corpora of commonsense knowledge. Society of mind approaches stress the use of multiple interacting methods and representations.

One approach that has achieved a high degree of success because of its steadfast focus on hard benchmark problems of commonsense reasoning, is logic. One logic-based formalism that stands out as both comprehensive and easy to use is the event calculus [4, 9].

Problem Statement

A method for commonsense reasoning must be able to perform various types of reasoning. Projection consists of determining what will happen next, or what the results of a sequence of actions will be. Explanation consists of determining what happened, or what the initial situation was. A method for commonsense reasoning must be able to reason about various everyday domains, especially time (action and change), space, and mental states.

Event Calculus

The event calculus fits the bill; it handles the important aspects of commonsense reasoning, as shown in Table 1.

Action	Context-sensitive effects
	Triggered events
	Concurrency
Change	Commonsense law of inertia
	Indirect effects
	Continuous change
	Nondeterminism
Object identity	
Space	Metric space
	Relational space
Mental states	Beliefs
	Goals and plans
	Emotions
Reasoning types	Projection
	Explanation

Table 1: Aspects of commonsense reasoning handled by the event calculus

The entities of the event calculus are *events*, *properties*, and *timepoints*. Two types of facts are based on these entities—the fact that an event *occurs* at a timepoint, and the fact that a property *holds* at a timepoint. Each fact has a *truth value*—it is either true or false.

To perform commonsense reasoning using the event calculus, we first identify an area of interest and then provide commonsense knowledge related to that area. We specify the effects that result from an event occurring at a timepoint. We may specify that, in a given context, a given event *initiates* a given property. This means that, if the event occurs at some timepoint in the context, then the property will be true after that timepoint. We may similarly specify that an event *terminates* a given property, which means that the property will be false after the event occurs.

The *frame problem* in artificial intelligence is the problem of representing and reasoning about what properties do not change when an event occurs. In the event calculus, properties normally obey the *commonsense law of inertia* [9]—the truth value of a property stays the same unless the property is directly initiated or terminated by an event. But *indirect effects* are also possible. For example, if a person picks up an object and walks into another room, not only will the person be in the other room; the object will also be in that room. The event calculus allows us to specify that, in a given context, an event *releases* a property from the commonsense law of inertia, so that the property can vary according to some law. For example, we may specify that the location of an object varies with the person holding it.

Events enable the description of discrete change. To describe continuous change, we may specify that a released property follows a certain *trajectory* or function of time.

We may specify that, in a given context, the occurrence of an event is *triggered*. For example, when a ball flying toward a wall reaches the wall, it bounces off the wall. The *preconditions* for occurrence of an event may be specified. For example, to walk through a door, the door must be open.

Default reasoning is reasoning in which we reach a conclusion based on limited information, and possibly later retract that conclusion when new information comes in. The event calculus supports default reasoning using *circumscription*, which can be used to minimize unexpected events, minimize unexpected effects of events, and minimize exceptional conditions. For example, circumscription can be used to represent that, unless we know otherwise, the refrigerator is plugged in.

Example

Here is an example of how we use the event calculus to represent knowledge about falling objects, and then to reason about a particular falling object. We represent the knowledge as follows:

Starting falling initiates falling.
Starting falling releases height.
The height of a falling object is $h_0 - \frac{1}{2}gt^2$.[1]
Falling and a height of zero trigger hitting the ground.
Hitting the ground terminates falling.
Hitting the ground initiates a height of zero.

We can then use this knowledge to reason. Suppose we are asked to determine what happens given the following information:

The initial height of an apple is four meters.
The apple starts falling at timepoint zero.

This is an instance of projection. We reason as follows:

Because starting falling initiates falling, after the apple starts falling, it will be falling.
Because the height of a falling object is $h_0 - \frac{1}{2}gt^2$, the apple starts falling at timepoint zero, and $h_0 = 4$, the height of the apple will be zero at timepoint $\sqrt{8/g}$.
Because falling is subject to the commonsense law of inertia, the apple is falling at timepoint $\sqrt{8/g}$.
Because the apple is falling and the height of the apple is zero at timepoint $\sqrt{8/g}$, the apple will hit the ground at that timepoint.

Representing Domains

The constructs of the event calculus enable representation of a surprising number of areas.

The event calculus can be used to represent and reason about various notions of space, such as object-level space:

An object is at one location at a time.
Moving from L to M initiates a location of M.
Moving from L to M terminates a location of L.
An agent picking up an object initiates the agent holding the object.
An agent letting go of an object terminates the agent holding the object.
An agent picking up an object releases its location.
If A is holding O, then O is where A is.
If A is holding O and O is at L, then A letting go of O initiates O being at L.

The event calculus can be used to represent and reason about the goal-based behavior of a rational agent using the following knowledge:

Adding a goal initiates an active goal.
Dropping a goal terminates an active goal.
Adding a plan initiates an active plan.

[1] h_0 is the object's initial height in meters, g is the acceleration due to gravity (9.8 meters/second^2), and t is the elapsed time in seconds.

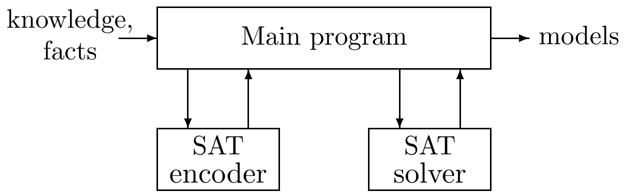

Figure 1: Discrete Event Calculus Reasoner

Fact	**M1**	**M2**	**M3**	**M4**
H awake at 0	F	F	F	F
H wakes up at 0	T	T	T	T
H awake at 1	T	T	T	T
K awake at 0	F	F	T	T
K wakes up at 0	F	T	F	T
K awake at 1	F	T	T	T

Table 2: Four models of a problem

Dropping a plan terminates an active plan.
An active goal with no active plan triggers adding an appropriate plan.
An active plan triggers executing the next step.
An active goal that is achieved triggers dropping the active goal and its active plan.

Discrete Event Calculus Reasoner

The Discrete Event Calculus Reasoner[2] solves event calculus projection and explanation problems efficiently using satisfiability (SAT) [6]. At its core, the Discrete Event Calculus Reasoner is a model finding program. As shown in Figure 1, the program takes commonsense knowledge and the truth values of some facts as input, encodes a SAT problem, invokes a complete SAT solver, and decodes the results of the SAT solver to produce models as output. Each model assigns a truth value to every fact.

Each model produced by the Discrete Event Calculus Reasoner fills in facts not provided as input. For example, suppose we know that waking up initiates being awake, Holly is not awake at timepoint 0, and she is awake at timepoint 1. Further suppose that we limit the timepoints to 0 and 1, and the agents to Holly and Ken. The program produces the four models shown in Table 2. This is an instance of explanation: The fact that Holly woke up at timepoint 0 explains how it was that she was not awake at timepoint 0 and awake at timepoint 1. Not much is known about Ken. He might have been awake or not awake at timepoint 0. But if he did wake up at timepoint 0, then he was awake at timepoint 1.

Applications

Commonsense reasoning using the event calculus is finding application in many areas, including business systems, natural language understanding, and robotics.

Business systems. The event calculus can be used to make business systems more flexible and

[2]`http://decreasoner.sourceforge.net/`

more aware of the human world. It is being used to increase the flexibility of customer and merchant applications implementing the NetBill payment protocol [12]. Instead of representing the protocol in a traditional fashion as a finite state machine (FSM), the event calculus is used to represent and reason about the commitments underlying the protocol. An example of a commitment is that sending a price quote for a product commits the sender to delivering the product if the customer sends a purchase request.

Commonsense reasoning allows the NetBill applications to adapt to novel and exceptional situations. For example, the NetBill protocol specifies that a transaction begins with a price quote request from the customer. Yet by reasoning about commitments, it is possible for the merchant application to determine that it is in fact possible to send an unsolicited price quote to the customer. Of course, the merchant is then committed to deliver the product if the customer orders it at the quoted price. Another example is that the customer application can determine that it is possible to bargain hunt by sending a purchase request at a low price without having previously received a price quote. The customer is then committed to pay if the merchant delivers the product at that price, which the merchant isn't committed to do.

The event calculus is being used to model business workflows [1]. The model of a workflow can be used for various purposes, such as (1) simulating the workflow, (2) managing the execution of the workflow by participants, and (3) querying the past, present, and possible future states of the workflow.

Natural language understanding. The event calculus is well-suited to the representation of the meaning of natural language, and making inferences in natural language understanding systems. It is being used to represent tense and aspect [11], and for inferencing in narrative comprehension systems [7].

Given a commonsense knowledge base about a domain and a narrative consisting of known properties and events in that domain, the event calculus can be used to fill in missing properties and events. A system for understanding terrorism news stories [7] uses commonsense reasoning to fill in such things as the following:

After being threatened, the victim is angry at the perpetrator.
After the target is set on fire, it is eventually destroyed.

Although this work is in its infancy, it points the way to the development of question answering systems with an ability to understand text more deeply than is currently possible.

Robotics. The event calculus is suitable for representation and reasoning within robotics systems. It is being used within a robot's high-level vision system [10] to improve object recognition by taking advantage of reasoning about the changes of the appearance of an object over time.

The high-level vision system is divided into three layers. The first layer takes low-level edge information and produces hypotheses about regions. The second layer produces hypotheses about appearance. The third layer produces hypotheses about what objects are in view based on the changes in appearance over time.

A Unix Example

How does one actually go about using the Discrete Event Calculus Reasoner? Suppose we wish to reason about the Unix system. We enter the following text into a file called `Unix.e`:

```
sort dir
sort filename

fluent Cwd(dir)
fluent In(filename, dir)
event Mv(filename, filename)

[filename1, filename2, dir, time]
HoldsAt(Cwd(dir), time) & filename1!=filename2 ->
Initiates(Mv(filename1, filename2), In(filename2, dir), time).

[filename1, filename2, dir, time]
HoldsAt(Cwd(dir), time) & filename1!=filename2 ->
Terminates(Mv(filename1, filename2), In(filename1, dir), time).
```

We define two sorts (or types): one for directories and one for filenames. We define two fluents (or properties): one that represents the current working directory and one that represents that a filename is in a directory. We define an event that represents that a `mv` command is issued. We then write two logical axioms. These axioms specify that, if one is in a directory and one issues the command

`mv` *filename1 filename2*

where *filename1* is different from *filename2*, then *filename2* will be in the directory and *filename1* will no longer be in the directory. The notation `[filename1, filename2, dir, time]` means "for all `filename1`, `filename2`, `dir`, and `time`." The notation "`->`" means "implies."

Suppose we wish to solve the following projection problem. Initially, the current working directory is D and the only filename in directory D is N1. We issue the command `mv N1 N2`. What is the result? To determine the answer, we enter the following problem into a file:

```
load Unix.e

dir D
filename N1, N2

HoldsAt(Cwd(D), 0).
[filename] HoldsAt(In(filename, D), 0) <-> filename=N1.
Happens(Mv(N1, N2), 0).

completion Happens

range time 0 1
```

The notation "`<->`" means "if and only if." When we provide this file as input to the Discrete Event Calculus Reasoner, it produces the following output:

```
1 model
---
model 1:
0
In(N1, D).
Cwd(D).
Happens(Mv(N1, N2), 0).
1
```

```
-In(N1, D).
+In(N2, D).
```

The result is that filename N1 is no longer in D and N2 is now in D. By default, the Discrete Event Calculus Reasoner shows only the differences from one timepoint to the next. A property that changes from true to false is indicated with a minus sign ("-") and a property that changes from false to true is indicated with a plus sign ("+").

Suppose we wish to solve the following explanation problem. At timepoint 0, N1 is the only filename in directory D, whereas at timepoint 1, N2 is the only filename in directory D. What happened? We enter the following into a file:

```
load Unix.e

dir D
filename N1, N2

[filename] HoldsAt(In(filename, D), 0) <-> filename=N1.
[filename] HoldsAt(In(filename, D), 1) <-> filename=N2.

range time 0 1
```

When we run this through the Discrete Event Calculus Reasoner, it produces the following:

```
1 model
---
model 1:
0
In(N1, D).
Cwd(D).
Happens(Mv(N1, N2), 0).
1
-In(N1, D).
+In(N2, D).
```

The program determines that the command `mv N1 N2` was issued when the current working directory was D.

Related Work

Other logic-based formalisms for commonsense reasoning include the situation calculus, the fluent calculus, temporal action logics (TAL), and action languages. The situation calculus is similar to the event calculus, but uses branching time instead of linear time. The fluent calculus is an extension of the situation calculus. TAL is similar to the event calculus. Action languages use specialized syntaxes rather than first-order or second-order logic.

Software tools are available for reasoning using these formalisms:

- KM[3] implements the situation calculus.
- FLUX[4] implements the fluent calculus.

[3]`http://www.cs.utexas.edu/users/mfkb/km.html`
[4]`http://www.fluxagent.org/`

- VITAL[5] implements TAL.
- The causal calculator (CCALC)[6] implements the CCALC action language, which is based on the distinction between what is true and what has a cause.
- $\mathcal{E}$-RES[7] implements the $\mathcal{E}$ action language, which is based on the event calculus.

Formalisms and tools for automated commonsense reasoning are closely related to those for planning. The task in planning is to take as input an initial state and a goal state, and produce as output a sequence of actions that bring about the goal state starting from the initial state. Most planners support the PDDL planning domain definition language[8] that is used in international planning competitions. PDDL treats many of the same phenomena of action and change as the event calculus such as context-sensitive effects and the commonsense law of inertia.[9] The main difference between planners and logical commonsense reasoning tools is that planners are highly optimized for planning and typically support only planning, whereas commonsense reasoning tools support planning as well as other forms of reasoning, including projection,[10] postdiction (determining the initial state given a sequence of events and a final state), and model finding (filling in missing properties and events given observed properties and events).

Nonlogical Methods

A number of nonlogical methods for commonsense reasoning have been developed by artificial intelligence researchers [8]. How do they compare with the event calculus?

When using the event calculus to solve a problem in a domain, we typically assume that knowledge about the domain has been previously entered. Analogical processing methods can be used to generate candidate inferences in a novel domain by finding analogies to familiar domains.

The event calculus does not deal with probabilities, although it does allow representation of nondeterministic effects of events. Methods based on probability theory can be used to quantify uncertainty about commonsense knowledge, uncertainty about a given scenario, and uncertainty about commonsense inferences.

Qualitative reasoning methods are particularly strong at modeling and simulating the behavior of physical mechanisms such as clocks, electrical circuits, pressure regulators, and water tanks.

Natural-language-based methods [2] have the advantage that commonsense knowledge can be entered without having to learn any special notation. To use the event calculus one must have a basic familiarity with first-order logic.

The society of mind [5] allows multiple commonsense reasoning methods to be combined so that the best techniques can be used to make progress at each point in reasoning. The event calculus can be one of these techniques.

Conclusions

The automation of commonsense reasoning is coming to fruition. Future challenges include the development of useful commonsense knowledge for application areas and the efficient solution of large commonsense reasoning problems.

---

[5] `http://www.ida.liu.se/~jonkv/vital/`

[6] `http://www.cs.utexas.edu/users/tag/cc/`

[7] `http://www.ucl.ac.uk/~uczcrsm/LanguageE/`

[8] `http://www.cs.yale.edu/homes/dvm/`

[9] Few PDDL tools implement continuous change and triggered events. Exceptions are TM-LPSAT (`http://cs1.cs.nyu.edu/~jiae/`) and VAL (`http://planning.cis.strath.ac.uk/VAL/`).

[10] Projection is performed by Drew McDermott's PDDL solution checker (`ftp://ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz`).

The event calculus handles the important problems of commonsense reasoning, and is highly usable. It is being applied to business systems, natural language understanding, and vision. It will be interesting in the coming years to explore its advantages for creating more flexible and user-friendly systems.

References

[1] Nihan Kesim Cicekli and Yakup Yildirim. Formalizing workflows using the event calculus. In Mohamed T. Ibrahim, Josef Küng, and Norman Revell, editors, *Database and Expert Systems Applications*, volume 1873 of *Lecture Notes in Computer Science*, pages 222–231. Springer, Berlin, 2000.

[2] Henry Lieberman, Hugo Liu, Push Singh, and Barbara A. Barry. Beating common sense into interactive applications. *AI Magazine*, 25(4):63–76, 2004.

[3] John McCarthy. Programs with common sense. In *Mechanisation of Thought Processes: Proceedings of a Symposium held at the National Physical Laboratory on 24th, 25th, 26th and 27th November 1958*, volume 1, pages 75–91, London, 1959. Her Majesty's Stationery Office.

[4] Rob Miller and Murray Shanahan. Some alternative formulations of the event calculus. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski, Part II*, volume 2408 of *Lecture Notes in Computer Science*, pages 452–490. Springer, Berlin, 2002.

[5] Marvin Minsky. *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. Simon & Schuster, New York, 2006.

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

[7] Erik T. Mueller. Understanding script-based stories using commonsense reasoning. *Cognitive Systems Research*, 5(4):307–340, 2004.

[8] Erik T. Mueller. *Commonsense Reasoning*. Morgan Kaufmann, San Francisco, 2006.

[9] Murray Shanahan. *Solving the Frame Problem*. MIT Press, Cambridge, MA, 1997.

[10] Murray Shanahan. Perception as abduction: Turning sensor data into meaningful representation. *Cognitive Science*, 29:103–134, 2005.

[11] Michiel van Lambalgen and Fritz Hamm. *The Proper Treatment of Events*. Blackwell, Malden, MA, 2005.

[12] Pınar Yolum and Munindar P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1–3):227–253, 2004.