CHAPTER

Logics for Commonsense Reasoning 16

Why the event calculus? How does the event calculus stack up against other logical formalisms for commonsense reasoning? There are many such formalisms. In this chapter, we review four important families—the situation calculus, the features and fluents framework, action languages, and the fluent calculus—discussing their relationship to the event calculus.

16.1 THE SITUATION CALCULUS

The main difference between the situation calculus and the event calculus is that the situation calculus uses branching time, whereas the traditional event calculus uses linear time. In the event calculus, there is a single time line on which events occur, and all events are considered to be actual events. In the situation calculus, time is represented by a tree, and each event may give rise to a different possible future; therefore, events are considered to be hypothetical events. A point in time is represented by a situation. In this section, we use a standard version of the situation calculus described by Raymond Reiter. Notational variants of the situation calculus are shown in Table 16.1.

16.1.1 RELATIONAL AND FUNCTIONAL FLUENTS

The truth value of a relational fluent is represented by

$$(\neg)F(x_1,\ldots,x_n,\sigma)$$

which represents that fluent $F(x_1,\ldots,x_n)$ is true (false) in situation σ. The value of a functional fluent is represented by

$$f(x_1,\ldots,x_n,\sigma)=v$$

which represents that fluent $f(x_1,\ldots,x_n)$ has value v in situation σ.

16.1.2 ACTIONS

The performance of an action is represented by

$$\sigma' = do(\alpha,\sigma)$$

Table 16.1 Situation calculus notational variants[a]

Source	Fluent	Successor
McCarthy and Hayes (1969)	$F(x_1, \ldots, x_n, \sigma)$	$result(\pi, \alpha, \sigma)$
McCarthy (1980)	$F(x_1, \ldots, x_n, \sigma)$	$result(\alpha, \sigma)$
Russell and Norvig (2009)	$F(x_1, \ldots, x_n, \sigma)$	$Result(\alpha, \sigma)$
Hayes (1973)	$F(x_1, \ldots, x_n, \sigma)$	$R(\alpha, \sigma)$
Hayes (1971), Reiter (2001), and Brachman and Levesque (2004)	$F(x_1, \ldots, x_n, \sigma)$	$do(\alpha, \sigma)$
Green (1969) and Winston (1984)	$F(x_1, \ldots, x_n, \sigma)$	$A(x_1, \ldots, x_n, \sigma)$
Kowalski (1979)	$Holds(\phi, \sigma)$	$result(\alpha, \sigma)$
Shanahan (1997b)	$Holds(\phi, \sigma)$	$Result(\alpha, \sigma)$
E. Davis (1990)	$true\_in(\sigma, \phi)$ $value\_in(\sigma, \phi)$	$result(\sigma, \alpha)$
Charniak and McDermott (1985)	$(T\ \sigma\ \phi)$	$result\ \sigma\ (do\ \pi\ \alpha)$
Genesereth and Nilsson (1987)	$T(\phi, \sigma)$	$Do(\alpha, \sigma)$

[a] α, *action;* π, *person;* σ, *situation;* ϕ, *fluent;* A, *action predicate;* F, *fluent predicate;* $x_1, \ldots, x_n$, *action and fluent arguments.*

which represents that situation σ' is the successor situation that results from performing action α in situation σ. There is an initial situation, called S_0.

16.1.3 ACTION EFFECTS

An *effect axiom* of the form

$$R(x_1, \ldots, x_n, \sigma) \supset (\neg)F(x_1, \ldots, x_n, do(\alpha, \sigma))$$

represents that under the conditions given by $R(x_1, \ldots, x_n, \sigma)$, fluent $F(x_1, \ldots, x_n)$ is true (false) in the situation that results from performing action α in situation σ.

The effect axioms for each fluent $F(x_1, \ldots, x_n)$ are compiled into a single successor state axiom as follows. The conditions under which an action α initiates the fluent are collected into a single formula $\gamma_F^+(x_1, \ldots, x_n, \alpha, \sigma)$, and the conditions under which α terminates the fluent are collected into a single formula $\gamma_F^-(x_1, \ldots, x_n, \alpha, \sigma)$. The *successor state axiom* is then as follows:

$$F(x_1, \ldots, x_n, do(\alpha, \sigma)) \equiv$$
$$\gamma_F^+(x_1, \ldots, x_n, \alpha, \sigma) \vee (F(x_1, \ldots, x_n, \sigma) \wedge \neg\gamma_F^-(x_1, \ldots, x_n, \alpha, \sigma))$$

This states that a fluent is true in the situation that results from performing an action α in situation σ if and only if (1) α initiates the fluent or (2) the fluent is true in σ and α does not terminate the fluent. (Notice the resemblance of $\gamma_F^+(x_1, \ldots, x_n, \alpha, \sigma)$ to $Initiates(\alpha, F(x_1, \ldots, x_n), \sigma)$ in the event calculus and of $\gamma_F^-(x_1, \ldots, x_n, \alpha, \sigma)$ to $Terminates(\alpha, F(x_1, \ldots, x_n), \sigma)$.)

16.1.4 ACTION PRECONDITIONS

An *action precondition axiom* of the form

$$Poss(A(x_1, \ldots, x_n), \sigma) \equiv P(x_1, \ldots, x_n, \sigma)$$

represents that it is possible to perform the action $A(x_1, \ldots, x_n)$ in situation σ if and only if $P(x_1, \ldots, x_n, \sigma)$ is true. A situation is executable if and only if all the actions leading to that situation are possible to perform. Nonexecutable situations are those resulting from one or more impossible actions.

16.1.5 EQUIVALENCE OF THE SITUATION CALCULUS AND THE EVENT CALCULUS

Robert Kowalski and Fariba Sadri have proved the equivalence of a version of the situation calculus similar to that of Reiter and a version of the event calculus. Their version of the situation calculus assumes that no fluents are true in the initial situation and includes terms representing transitions from one situation to another. Their version of the event calculus differs from the one described in this book as follows:

- It supports branching time.
- It does not support concurrent events.
- It does not support release from the commonsense law of inertia.
- It does not support continuous change.
- It has an initial situation, and no fluents are true in the initial situation.
- It includes terms representing transitions from one situation to another.

Their version supports a distinction between actual and hypothetical situations, but this is not required for the equivalence proof. Their version of the event calculus does not support continuous time, which is also the case for the discrete event calculus. An equivalence result between the situation calculus and a branching time version of the discrete event calculus is presented in Appendix D.

16.1.6 DISCUSSION

Like the event calculus, the situation calculus supports context-sensitive effects of events, indirect effects, preconditions of actions, and the commonsense law of inertia.

The situation calculus can be used to represent and reason about hypothetical events and situations, which cannot be done in the event calculus unless it is extended to support branching time, as proposed by Kowalski and Sadri.

The situation calculus supports functional fluents, which are not supported by the event calculus. Nonetheless, a functional fluent may be represented in the event calculus by (1) using a relational fluent, (2) adding an extra argument to the relational fluent, and (3) adding state constraints ensuring that the relational fluent is functional. For example, if we wish to represent a functional fluent $F(x) = y$, we instead use a relational fluent $F(x, y)$ and the following state constraints:

$$HoldsAt(F(x, y_1), t) \wedge HoldsAt(F(x, y_2), t) \Rightarrow y_1 = y_2$$
$$\exists y\, HoldsAt(F(x, y), t)$$

The basic situation calculus does not support continuous change because of the discrete nature of the tree of situations. The basic situation calculus also does not support triggered events because many hypothetical events are possible in any given situation. Nondeterminism is also not supported in the basic situation calculus. A number of extensions to the situation calculus have been proposed to address these phenomena.

The basic situation calculus does not support the representation of actual situations and actual events. Two methods for extending the situation calculus to support these have been proposed:

- Situations are designated as actual using a predicate *actual*(s), and actions are designated as actually occurring in a situation using a predicate *occurs*(a, s). Time may also be added, and a function *start*(s) may be used to represent the start time of a situation.
- A time line is added to the situation calculus, times are associated with actual situations using a function *State*(t), and actions are designated as actually occurring at a time using a predicate *Happens*(a, t).

Concurrent events cannot normally be represented in the situation calculus, because each transition from situation to situation represents a single event. Extensions have been proposed in which sets of concurrent events can be associated with transitions.

16.2 THE FEATURES AND FLUENTS FRAMEWORK

The features and fluents framework was developed by Erik Sandewall. The framework includes a specialized logic called discrete fluent logic and a methodology for assessing the range of applicability of logics. Discrete fluent logic (DFL) was recast in many-sorted first-order logic with equality by Patrick Doherty and Witold Łukaszewicz. This work gave rise to a class of logics called temporal action logics (TAL) developed by Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnström. TAL uses the pointwise minimization of occlusion with nochange premises (PMON) entailment method, which is one of the entailment methods defined in the features and fluents framework.

16.2.1 TEMPORAL ACTION LOGICS

TAL has many similarities with the event calculus. Like the event calculus, and unlike the situation calculus, TAL uses linear time. (In general, the features and fluents framework supports both linear and branching time.) Like the event calculus, TAL uses circumscription to enforce the commonsense law of inertia. TAL has a predicate, *Occlude*(t, f), similar to the event calculus predicate *ReleasedAt*(f, t), which represents that a fluent f is released from the commonsense law of inertia at timepoint t. The event calculus inherited this construct from the features and fluents framework. TAL uses discrete time, like the discrete event calculus. Whereas the discrete event calculus uses the integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ for representing time, TAL uses the non-negative integers $\{0, 1, 2, \ldots\}$. The relation between the event calculus and TAL is explored in Appendix E.

TAL consists of two languages: a narrative description language, $\mathcal{L}(ND)$, and a fluent logic language, $\mathcal{L}(FL)$. Commonsense reasoning problems are described using $\mathcal{L}(ND)$, which is a specialized language. Descriptions in $\mathcal{L}(ND)$ are then translated into $\mathcal{L}(FL)$, which is a many-sorted first-order language. We now describe how things are represented in $\mathcal{L}(ND)$ and translated into $\mathcal{L}(FL)$.

Observation statements

The $\mathcal{L}(ND)$ expression

$$[\tau]\, f(\omega_1, \ldots, \omega_n) \mathrel{\hat{=}} v$$

represents that fluent $f(\omega_1, \ldots, \omega_n)$ has the value v at timepoint τ. f is called a feature hence the name features and fluents. (We call f a fluent symbol.)

If f has a Boolean domain, then the $\mathcal{L}(FL)$ translation of $[\tau]\, f(\omega_1, \ldots, \omega_n) \mathrel{\hat{=}} T$ is the observation statement

$$Holds(\tau, f(\omega_1, \ldots, \omega_n))$$

and the translation of $[\tau]\, f(\omega_1, \ldots, \omega_n) \mathrel{\hat{=}} F$ is the observation statement

$$\neg Holds(\tau, f(\omega_1, \ldots, \omega_n))$$

If f has a non-Boolean domain, then the $\mathcal{L}(FL)$ translation of $[\tau]\, f(\omega_1, \ldots, \omega_n) \mathrel{\hat{=}} v$ is the observation statement

$$Holds(\tau, f(\omega_1, \ldots, \omega_n, v))$$

Axioms in $\mathcal{L}(FL)$ ensure that f is functional:

$$\begin{gathered} Holds(\tau, f(\omega_1, \ldots, \omega_n, v_1)) \land Holds(\tau, f(\omega_1, \ldots, \omega_n, v_2)) \rightarrow \\ v_1 = v_2 \end{gathered} \tag{16.1}$$

$$\exists v\, Holds(\tau, f(\omega_1, \ldots, \omega_n, v)) \tag{16.2}$$

Action occurrences and effects

Action occurrences and the effects of actions are represented as follows. The $\mathcal{L}(ND)$ action occurrence statement

$$[\tau_1, \tau_2]\; a(\omega_1, \ldots, \omega_n)$$

represents that action $a(\omega_1, \ldots, \omega_n)$ occurs between timepoints τ_1 and τ_2. The $\mathcal{L}(ND)$ action law schema

$$[\tau_1, \tau_2]\; \alpha \leadsto ([\tau_1]\; \gamma \rightarrow [\tau_1, \tau_2]\; \phi := v)$$

represents that if action α occurs between τ_1 and τ_2 and condition γ is true at τ_1, then the value of fluent ϕ will be v starting at τ_2. The value of ϕ is unchanged at τ_1 and is permitted to vary at timepoints $\tau_1 + 1$ to $\tau_2 - 1$.

If $\tau_2 = \tau_1 + 1$, then the action is called a single-step action. In this case, the meaning is the same as in the discrete event calculus:

- TAL $[\tau, \tau+1]\,\alpha$ is the same as DEC $Happens(\alpha, \tau)$
- TAL $[\tau, \tau+1]\,\alpha \rightsquigarrow ([\tau]\,\gamma \rightarrow [\tau, \tau+1]\,\phi := T)$ is the same as DEC $Initiates(\alpha, \phi, \tau)$
- TAL $[\tau, \tau+1]\,\alpha \rightsquigarrow ([\tau]\,\gamma \rightarrow [\tau, \tau+1]\,\phi := F)$ is the same as DEC $Terminates(\alpha, \phi, \tau)$

Action occurrences statements and action law schemata are translated into $\mathcal{L}(FL)$ schedule statements as follows. For each action occurrence statement $[\tau_1, \tau_2]\ a(\omega_1, \ldots, \omega_n)$ and corresponding action law schema $[t_1, t_2]\ a(x_1, \ldots, x_n) \rightsquigarrow\ \Delta$, we form a $\mathcal{L}(ND)$ reassignment formula $[\tau_1, \tau_2]\ \Delta'$, where Δ' is Δ with τ_1 substituted for t_1, τ_2 substituted for t_2, ω_1 substituted for x_1, ω_2 substituted for x_2, and so on. For example, from

$$[1,2]\ WakeUp(Nathan)$$

and

$$[t_1, t_2]\ WakeUp(a) \rightsquigarrow awake(a) := T$$

we form the reassignment formula

$$[1,2]\ awake(Nathan) := T \tag{16.3}$$

We then translate the reassignment formula into $\mathcal{L}(FL)$. The translation of a reassignment formula $[\tau_1, \tau_2]\, f(\omega_1, \ldots, \omega_n) := v$ is the schedule statement

$$Holds(\tau_2, f(\omega_1, \ldots, \omega_n, v)) \wedge$$
$$\forall t, d\ (t_1 < t \leq t_2 \rightarrow Occlude(t, f(\omega_1, \ldots, \omega_n, d)))$$

For example, the translation of (16.3) is

$$Holds(2, awake(Nathan)) \wedge Occlude(2, awake(Nathan))$$

The predicate $Occlude(t, f)$ represents that the truth value of fluent f may change between timepoints $t-1$ and t, which is the same as $ReleasedAt(f, t)$ in the event calculus. This constraint is enforced by the following nochange axiom of $\mathcal{L}(FL)$, which states that if a fluent f is not occluded at timepoint $t+1$, then the truth value of f does not change from timepoint t to $t+1$:

$$\neg Occlude(t+1, f) \rightarrow Holds(t, f) \equiv Holds(t+1, f) \tag{16.4}$$

This is equivalent to the conjunction of the axioms

$$Holds(t, f) \wedge \neg Occlude(t+1, f) \rightarrow Holds(t+1, f) \tag{16.5}$$

and

$$\neg Holds(t, f) \wedge \neg Occlude(t+1, f) \rightarrow \neg Holds(t+1, f) \tag{16.6}$$

These axioms resemble axioms of the discrete event calculus. Axiom (16.5) resembles DEC5:

$$\begin{gathered}(HoldsAt(f,t) \wedge \neg ReleasedAt(f,t+1) \wedge \\ \neg\exists e\,(Happens(e,t) \wedge Terminates(e,f,t))) \Rightarrow \\ HoldsAt(f,t+1)\end{gathered}$$

Axiom (16.6) resembles DEC6:

$$\begin{gathered}(\neg HoldsAt(f,t) \wedge \neg ReleasedAt(f,t+1) \wedge \\ \neg\exists e\,(Happens(e,t) \wedge Initiates(e,f,t))) \Rightarrow \\ \neg HoldsAt(f,t+1)\end{gathered}$$

Scenario descriptions

A TAL scenario description is given by

$$CIRC[\Sigma; Occlude] \wedge \Psi \wedge \Gamma \wedge \mathrm{TAL}$$

where Σ is a conjunction of schedule statements derived from action occurrence statements and action law schemata; Ψ is a conjunction of domain constraints, or state constraints; Γ is a conjunction of observation statements; and TAL is a conjunction of foundational axioms and the nochange axiom (16.4). We use a notation that highlights the similarity of TAL scenario descriptions and event calculus domain descriptions, which are defined in Section 2.7. The TAL notation for Σ is Γ_{SCD}, Ψ is Γ_{DOMC}, Γ is Γ_{OBS}, and $\mathrm{TAL} = \Gamma_{FA} \wedge \Gamma_{NCG}$, where $\Gamma_{NCG} = (16.4)$. The scenario description may entail various formulas, such as *Holds* formulas, as in the event calculus.

Discussion

Like the event calculus, TAL supports actions with duration, the commonsense law of inertia, dynamic release from the commonsense law of inertia, concurrent events, context-sensitive effects, indirect effects, and nondeterministic effects.

Unlike the event calculus, TAL supports functional fluents. As described in Section 16.1, functional fluents can be represented in the event calculus using relational fluents and state constraints. These state constraints are similar to the $\mathcal{L}(FL)$ axioms (16.1) and (16.2). TAL supports a type of fluent called a *durational fluent*, which is a fluent whose value is always a default value except when the fluent is occluded.

Several features of the event calculus are not supported by TAL: continuous change, continuous time, time less than zero, and triggered events (because action occurrence statements have no provision for conditions). Event occurrences are represented explicitly in the event calculus using *Happens*; in TAL, event occurrences are represented explicitly in $\mathcal{L}(ND)$ but not in $\mathcal{L}(FL)$. (In some versions of TAL, a predicate *Occurs* is introduced.)

16.3 ACTION LANGUAGES

Instead of using the general-purpose language of predicate logic for reasoning about action, specialized action languages can be developed. The original action languages were STRIPS and ADL; a number of action languages, such as $\mathcal{A}$, $\mathcal{AR}_0$, $\mathcal{A}_C$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{C}+$, $\mathcal{BC}$, and $\mathcal{E}$, have since been developed.

The simple action language $\mathcal{A}$ was introduced by Michael Gelfond and Vladimir Lifschitz. In $\mathcal{A}$, action effects are represented using expressions of the form

$$a \textbf{ causes } f \textbf{ if } f_1, \ldots, f_n$$

where a is an action, and $f, f_1, \ldots, f_n$ are (possibly negated) fluents. $\mathcal{AR}_0$ extends $\mathcal{A}$ for indirect effects and nondeterminism. $\mathcal{A}_C$ extends $\mathcal{A}$ for concurrent actions. $\mathcal{B}$ extends $\mathcal{A}$ for indirect effects. $\mathcal{C}$ is based on a causal theory of actions. $\mathcal{C}+$ is an extension of $\mathcal{C}$. $\mathcal{BC}$ extends $\mathcal{B}$ with features from $\mathcal{C}$ such as static causal laws. $\mathcal{E}$ is similar to the event calculus, and a mapping between the event calculus and $\mathcal{E}$ has been described. The $\mathcal{E}$ language is discussed in Section 15.5. In this section, we discuss action language $\mathcal{C}+$.

16.3.1 $\mathcal{C}+$

The $\mathcal{C}+$ action language was developed by Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Like the event calculus and TAL, $\mathcal{C}+$ uses linear time. Like the discrete event calculus, $\mathcal{C}+$ uses discrete time. Unlike the event calculus, the situation calculus, and TAL, which are based on the first-order and sometimes second-order logic, $\mathcal{C}+$ is based on propositional logic. $\mathcal{C}+$ uses concepts like reduct that are similar to those used in answer set programming.

Reasoning problems in $\mathcal{C}+$ can be transformed into SAT problems, as can reasoning problems of the discrete event calculus. The SAT-based approach used in the discrete event calculus derives from the SAT-based approach used in $\mathcal{C}+$.

Syntax of $\mathcal{C}+$

We start with a *signature*, which consists of a set of constants, and a domain $dom(c)$ associated with each constant c. The domain of a Boolean constant is $\{\mathbf{t}, \mathbf{f}\}$. If c is a constant and $v \in dom(c)$, then $c = v$ is a $\mathcal{C}+$ *atom*. A $\mathcal{C}+$ *formula* is defined as follows:

- An atom is a formula.
- The symbols $\bot$ (*false*) and $\top$ (*true*) are formulas (zero-place connectives).
- If f_1 is a formula, and f_2 is a formula, then $\neg f_1, f_1 \wedge f_2$, and $f_1 \vee f_2$ are formulas.
- If c is a Boolean constant, then $c = \mathbf{t}$ is abbreviated c and $c = \mathbf{f}$ is abbreviated $\neg c$.

The set of fluents and the set of actions are disjoint subsets of the set of constants. Fluents are further divided into simple fluents and statically determined fluents. A *fluent formula* is a formula whose constants are fluents, and an *action formula* is a formula whose constants are actions.

Although $\mathcal{C}+$ is based on propositional logic, a first-order notation is often used. We may define a set of constants $p(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are taken from finite sets. For example, we may represent the location of an object using the set of constants $InRoom(x)$, where $x \in \{Nathan, Book\}$.

A $\mathcal{C}+$ *causal law* is an expression of the form

caused f **if** g **after** h

or

caused f **if** g

where f, g, and h are formulas. A number of abbreviations for causal laws are defined.

- The effects of actions are represented using expressions of the form

  a **causes** f **if** c

  where a is an action formula, and f and c are fluent formulas. This is an abbreviation for **caused** f **if** $\top$ **after** $a \wedge c$. The formula f cannot contain statically determined fluents. The condition c enables context-sensitive effects of actions to be represented.
- If f is a simple fluent, then the expression

  inertial f

  specifies that f is subject to the commonsense law of inertia at all times. This is an abbreviation for **caused** $f = v$ **if** $f = v$ **after** $f = v$, for each $v \in dom(f)$.
- Action preconditions are represented using expressions of the form

  nonexecutable a **if** c

  where a is an action formula, and c is a fluent formula. This stands for **caused** $\bot$ **if** $\top$ **after** $a \wedge c$.
- Triggered actions are represented using causal laws of the form

  caused a **if** c

  where a is an action formula, and c is a fluent formula.
- If a is an action, then the expression

  exogenous a

  specifies that a may or may not occur at each timepoint. This stands for **caused** $a = v$ **if** $a = v$, for each $v \in dom(a)$. If **exogenous** a is not specified, then in order for a to occur it must be caused to occur.
- State constraints are represented using expressions of the form

  constraint c

  where c is a fluent formula. This stands for **caused** $\bot$ **if** $\neg c$.

$\mathcal{C}+$ treats indirect effects in an elegant way. They are represented using causal laws of the form

caused f **if** g

where f and g are fluent formulas. Such a causal law represents that f is an indirect effect of any actions that bring about g. For example, we may represent that if an agent is holding an object, and the agent is in a room, then the object will also be in that room:

$$\textbf{caused } InRoom(o) = r \textbf{ if } Holding(a) = o \wedge InRoom(a) = r$$

The closest analog to this in the event calculus is the pair of effect constraints:

$$\begin{gathered} HoldsAt(Holding(a, o), t) \wedge \\ Initiates(e, InRoom(a, r), t) \Rightarrow \\ Initiates(e, InRoom(o, r), t) \\ HoldsAt(InRoom(a, r), t) \wedge \\ Initiates(e, Holding(a, o), t) \Rightarrow \\ Initiates(e, InRoom(o, r), t) \end{gathered}$$

But these formulas do not allow for the possibility that $Holding(a) = o$ or $InRoom(a) = r$ may be the result of interacting concurrent events.

We have so far described the syntax of $\mathcal{C}+$. The semantics of a $\mathcal{C}+$ theory is given by a translation into a causal theory. We first define the syntax and semantics of causal theories, and then specify the translation of a $\mathcal{C}+$ theory into a causal theory.

Syntax of causal theories

We have a signature, as in $\mathcal{C}+$. Causal theory atoms and formulas are as in $\mathcal{C}+$. A *causal rule* is an expression of the form

$$f_1 \Leftarrow f_2$$

where f_1 and f_2 are causal theory formulas. This represents that f_1 has a cause if f_2 is true. f_1 is called the head of the causal rule, and f_2 is called the body of the causal rule. A *causal theory* consists of a set of causal rules.

Semantics of causal theories

An *interpretation* I for a signature is a mapping from each constant c of the signature to an element of $dom(c)$. We define what it means for a formula ψ to be true in an interpretation I, written $I \models \psi$, as follows:

- If c is a constant and $v \in dom(c)$, then $I \models c = v$ if and only if $I(c) = v$.
- It is not the case that $I \models \bot$.
- It is the case that $I \models \top$.
- If α is a formula, then $I \models \neg\alpha$ if and only if it is not the case that $I \models \alpha$.
- If α and β are formulas, then $I \models \alpha \wedge \beta$ if and only if $I \models \alpha$ and $I \models \beta$.
- If α and β are formulas, then $I \models \alpha \vee \beta$ if and only if $I \models \alpha$ or $I \models \beta$.

An interpretation I is a *model* of a set of formulas Φ, written $I \models \Phi$, if and only if $I \models \psi$ for every $\psi \in \Phi$.

The *reduct* of a causal theory T relative to an interpretation I, written $reduct(T, I)$, is the set of every head of a causal rule $r \in T$ such that the body of r is true in I. An interpretation I is a *model* of a causal theory T, written $I \models T$, if and only if I is the unique model of the set of formulas $reduct(T, I)$. That is, there must be exactly one model of $reduct(T, I)$, and that model must equal I. A causal theory T *entails* a formula ψ, written $T \models \psi$, if and only if, for every I such that $I \models T$, we have $I \models \psi$.

Reduction to propositional logic

A causal theory T is *definite* if and only if (1) the head of every causal rule in T is an atom or $\bot$ and (2) no atom is the head of an infinite number of causal rules of T. If a causal theory is definite, its semantics may be restated in terms of a reduction to propositional logic.

From a causal theory, we form a conjunction of propositional formulas, called the completion of the causal theory, as follows: First, for each atom a, we add to the conjunction

$$a \equiv g_1 \vee \cdots \vee g_m$$

where $m \geq 0$ and $g_1, \ldots, g_m$ are the bodies of the causal rules in which a appears as the head. Second, for each causal rule of the form $\bot \Leftarrow f$, we add to the conjunction

$$\neg f$$

Third, we add to the conjunction

$$\left(\bigvee_{v} c = v \right) \wedge \left(\bigwedge_{v_1, v_2} v_1 = v_2 \vee \neg (c = v_1 \wedge c = v_2) \right)$$

for all constants c, where v, v_1, and v_2 range over $dom(c)$, so that each constant equals exactly one value.

Translation from $\mathcal{C}+$ to causal theories

We define the length-N translation of a $\mathcal{C}+$ theory into a causal theory, where N is a positive integer. A $\mathcal{C}+$ theory can be viewed as representing a *transition system*, or a directed graph whose nodes represent states, and whose links represent actions or sets of concurrently executed actions. There is a one-to-one correspondence between the models of the length-N translation of a $\mathcal{C}+$ theory and the paths of length N through the transition system represented by the theory.

We define a set of *timepoints* $\{0, 1, 2, \ldots, N\}$. The set of constants in the causal theory is the set of all pairs $t : c$, where t is a timepoint, and c is a constant in the $\mathcal{C}+$ theory. (For simplicity, our discussion ignores the fact that actions do not occur at timepoint N.) The domain associated with each $t : c$ in the causal theory is the domain associated with c in the $\mathcal{C}+$ theory. The *translation* of a $\mathcal{C}+$ formula f at timepoint t, written $t : f$, is a causal theory formula obtained by substituting $t : c$ for every constant c in f.

> If f is a fluent, $v \in dom(f)$, and t is a timepoint, then $t : f = v$ represents that f has the value v at t.
> If a is a Boolean action and t is a timepoint, then $t : a$ represents that a occurs at t.

In order to expose the meanings of the $\mathcal{C}+$ expressions already mentioned, we specify here the translation of each expression separately. The translation may be defined more simply in terms of just the two causal laws **caused** f **if** g **after** h and **caused** f **if** g.

- A $\mathcal{C}+$ expression representing the effects of actions

  $$a \textbf{ causes } f \textbf{ if } c$$

  is translated into a set of causal rules

  $$t+1 : f \Leftarrow (t : a) \wedge (t : c)$$

  for each $t \in \{0, 1, 2, \ldots, N-1\}$.
- A $\mathcal{C}+$ expression of the form

  $$\textbf{inertial } f$$

  is translated into a set of causal rules

  $$t+1 : f = v \Leftarrow (t+1 : f = v) \wedge (t : f = v)$$

  for each $v \in \mathit{dom}(f)$ and $t \in \{0, 1, 2, \ldots, N-1\}$.
- We also add causal rules stating that the initial values of simple fluents are arbitrary. For each simple fluent f and $v \in \mathit{dom}(f)$ in the $\mathcal{C}+$ theory, we add

  $$0 : f = v \Leftarrow 0 : f = v \tag{16.7}$$

  For a statically determined fluent, (16.7) is not added. The value of a statically determined fluent at a timepoint t is strictly determined by the values of other fluents and actions at t. Statically determined fluent constants can be used for default reasoning.
- A $\mathcal{C}+$ precondition

  $$\textbf{nonexecutable } a \textbf{ if } c$$

  is translated into a set of causal rules

  $$\bot \Leftarrow (t : a) \wedge (t : c)$$

  for each $t \in \{0, 1, 2, \ldots, N\}$
- A $\mathcal{C}+$ expression representing a triggered action

  $$\textbf{caused } a \textbf{ if } c$$

  is translated into a set of causal rules

  $$t : a \Leftarrow t : c$$

  for each $t \in \{0, 1, 2, \ldots, N\}$.
- A $\mathcal{C}+$ expression of the form

  $$\textbf{exogenous } a$$

  is translated into a set of causal rules

  $$t : a = v \Leftarrow t : a = v$$

  for each $v \in \mathit{dom}(a)$ and $t \in \{0, 1, 2, \ldots, N\}$.
- A $\mathcal{C}+$ state constraint

  $$\textbf{constraint } c$$

  is translated into a set of causal rules

  $$\bot \Leftarrow \neg t : c$$

  for each $t \in \{0, 1, 2, \ldots, N\}$.

- A $\mathcal{C}+$ expression representing an indirect effect

  $$\textbf{caused } f \textbf{ if } g$$

  is translated into a set of causal rules

  $$t : f \Leftarrow t : g$$

  for each $t \in \{0, 1, 2, \ldots, N\}$.

Discussion

$\mathcal{C}+$ supports many of the same features as the event calculus. Like the event calculus, it supports the commonsense law of inertia, concurrent events with cumulative and canceling effects, context-sensitive effects, indirect effects, nondeterministic effects (via **may cause**), preconditions, state constraints, and triggered events. Unlike the event calculus, $\mathcal{C}+$ does not support continuous change or continuous time.

16.4 THE FLUENT CALCULUS

The fluent calculus was developed by Michael Thielscher, Steffen Hölldobler, and Josef Schneeberger. The fluent calculus adds the notion of states to the notions of actions, fluents, and situations of the situation calculus.

16.4.1 STATES

A *state* represents a set of fluents and is defined inductively as follows:

- $\emptyset$ is a state.
- If f is a fluent, then f is a state.
- If z_1 and z_2 are states, then $z_1 \circ z_2$ is a state.
- Nothing else is a state.

The function $\circ$ is associative and commutative. The expression $Holds(f, z)$ represents that fluent f is true in state z and is defined as follows:

$$Holds(f, z) \stackrel{\text{def}}{\equiv} \exists z' \, (z = f \circ z')$$

Situations and states are related as follows. The function $State(s)$ represents the state associated with situation s. The expression $Holds(f, s)$ represents that fluent f is true in situation s and is defined as:

$$Holds(f, s) \stackrel{\text{def}}{\equiv} Holds(f, State(s))$$

As in the situation calculus, $Do(a, s)$ represents the situation that results from performing action a in situation s and the initial situation is called S_0.

16.4.2 PLUS AND MINUS MACROS

The macros $+$ (plus sign) and $-$ (minus sign) are used to specify the addition and removal of fluents from states:

$$z_1 - \varnothing = z_2 \overset{\text{def}}{\equiv} z_2 = z_1$$

$$z_1 - f = z_2 \overset{\text{def}}{\equiv} (z_2 = z_1 \lor z_2 \circ f = z_1) \land \neg Holds(f, z_2)$$

$$z_1 - (f_1 \circ f_2 \circ \ldots \circ f_n) = z_2 \overset{\text{def}}{\equiv} \exists z\,(z_1 - f_1 = z \land z - (f_2 \circ \ldots \circ f_n) = z_2)$$

$$z_2 = (z_1 - \vartheta^-) + \vartheta^+ \overset{\text{def}}{\equiv} \exists z\,(z_1 - \vartheta^- = z \land z_2 = z \circ \vartheta^+)$$

The states ϑ^- and ϑ^+ represent sets of fluents.

16.4.3 STATE UPDATE AXIOMS

The effects of each action a are represented using *state update axioms* for a, which are of the form:

$$Poss(a, s) \land \Delta(s) \supset State(Do(a, s)) \;=\; (State(s) - \vartheta^-) + \vartheta^+$$

where $Poss(a, s)$ represents that a is possible in situation s, $\Delta(s)$ is a formula containing s, the state ϑ^- represents the set of fluents terminated by a under the conditions $\Delta(s)$, and the state ϑ^+ represents the set of fluents initiated by a under the conditions $\Delta(s)$. Notice that for a given action a, if we wish to be able to determine the effects of a in all cases for which $Poss(a, s)$ is true, then the conditions $\Delta(s)$ of the state update axioms for a must exhaust all those cases. For example, the following state update axiom for the action $WakeUp(p)$ states that, if a person wakes up, then the person will no longer be asleep and the person will be awake:

$$State(Do(WakeUp(p), s)) \;=\; (State(s) - Asleep(p)) + Awake(p)$$

16.4.4 NONDETERMINISTIC EFFECTS

The effect of flipping a coin is represented using the following state update axiom:

$$\begin{gathered} State(Do(Flip, s)) = State(s) + Heads \lor \\ State(Do(Flip, s)) = State(s) - Heads \end{gathered}$$

16.4.5 CONCURRENT ACTIONS

The fluent calculus deals with concurrent actions with cumulative or canceling effects by introducing concurrent actions. A *concurrent action* represents a set of actions and is defined inductively as follows:

- The empty action ϵ is a concurrent action.
- If a is an action, then a is a concurrent action.
- If c_1 and c_2 are concurrent actions, then $c_1 \circ c_2$ is a concurrent action.
- Nothing else is a concurrent action.

The function $\circ$ on concurrent actions is associative and commutative. The expression $HoldsIn(c_1, c_2)$ represents that c_1 is contained in c_2, and is defined as follows:

$$HoldsIn(c_1, c_2) \overset{\text{def}}{\equiv} \exists c' \, (c_2 = c_1 \circ c')$$

The effects of concurrent actions are then specified using state update axioms. For example, the following state update axioms represent that in order for a door to open, it must be pushed at the same time that its latch is turned.

$$Poss(Turn \circ c, s) \wedge \neg HoldsIn(Push, c) \supset \\ State(Do(Turn \circ c, s)) = State(Do(c, s))$$

$$Poss(Push \circ c, s) \wedge \neg HoldsIn(Turn, c) \supset \\ State(Do(Push \circ c, s)) = State(Do(c, s))$$

$$Poss(Turn \circ Push \circ c, s) \supset \\ State(Do(Turn \circ Push \circ c, s)) = State(Do(c, s)) + Open$$

This is represented in the event calculus as follows:

$$Happens(Turn, t) \Rightarrow Initiates(Push, Open, t)$$

(The representation of cumulative and canceling effects in the event calculus is discussed in Section 8.2.)

16.4.6 DISCUSSION

Like the event calculus, the fluent calculus supports the commonsense law of inertia, concurrent events with cumulative and canceling effects, context-sensitive effects, indirect effects, nondeterministic effects, and preconditions. Methods have also been proposed for handling continuous change and triggered events in the fluent calculus. The treatment of indirect effects in the fluent calculus is described in the Bibliographic notes of Chapter 6.

16.5 DISCUSSION AND SUMMARY

Many of the best features of logics for commonsense reasoning have been incorporated into the classical logic event calculus. Yet the event calculus retains a certain simplicity compared to other logics. Figure 16.1 outlines the relationships between the logics and indicates some of the origins of features of the event calculus.

The situation calculus came first and has had a major influence on all the logics. The notion of fluents, use of frame axioms, use of reification, and distinction between primitive and derived fluents are all from the situation calculus. The treatment of concurrent events with cumulative or canceling effects was first perfected in the situation calculus and then integrated into the event calculus.

The classical logic event calculus derives from the original logic programming version of the event calculus, which introduced events, event occurrences, and fluents

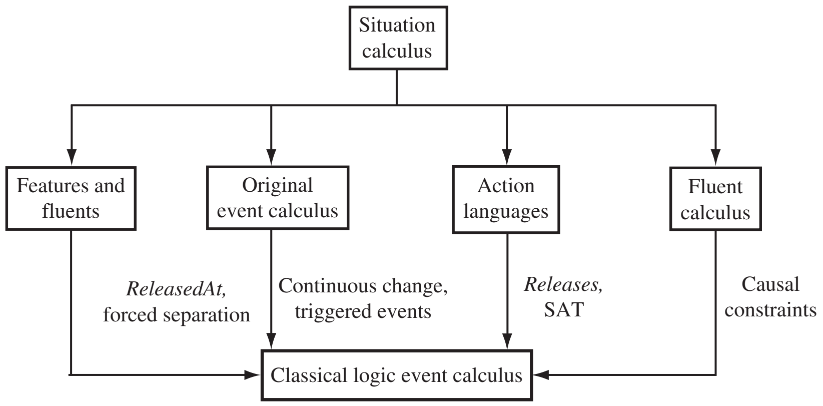

FIGURE 16.1

Logics for commonsense reasoning.

that are initiated and terminated. The treatment of continuous change and triggered events in the classical logic event calculus derives from work performed in the original event calculus.

The features and fluents framework has had several influences on the classical logic event calculus. The *ReleasedAt* predicate of the classical logic event calculus derives from the occlusion construct of features and fluents. The technique of forced separation, in which event calculus axioms are outside the scope of any circumscription, derives from the filtered preferential entailment or filtering of features and fluents.

Several features of the classical logic event calculus derive from action languages. The *Releases* predicate of the event calculus is taken from action languages. The use of SAT solvers to solve event calculus reasoning problems was inspired by the use of SAT solvers in action languages.

Work on the instantaneous propagation of interacting indirect effects was first performed in the fluent calculus. This work led to the introduction of causal constraints into the classical logic event calculus.

Logics for commonsense reasoning can model time as branching or linear and as discrete or continuous. Table 16.2 shows which logics use which models of time.

Table 16.2 Models of Time Used by Logics

	Branching Time	**Linear Time**
Discrete time	Situation calculus Fluent calculus Branching discrete event calculus	TAL $\mathcal{C}+$ Discrete event calculus
Continuous time		Event calculus

BIBLIOGRAPHIC NOTES

The situation calculus

The version of the situation calculus we describe is that of Reiter (2001). Based on previous proposals (F. Lin & Reiter, 1994; Pinto, 1998b; Reiter, 1993), Pirri and Reiter (1999, p. 328) and Reiter (2001, p. 50) provide the following four foundational axioms for the situation calculus:

$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2 \tag{16.8}$$

$$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s) \tag{16.9}$$

$$\neg(s \sqsubset S_0) \tag{16.10}$$

$$s \sqsubset do(a, s') \equiv s = s' \vee s \sqsubset s' \tag{16.11}$$

Axiom (D.36) is a formula of second-order logic, an induction axiom, which allows us to prove general properties of situations and limits the set of situations to the smallest set S such that (1) $S_0 \in S$ and (2) $s \in S$ and a is an action implies $do(a, s) \in S$ (Reiter, 1993). Axiom (D.38) defines the ordering relation $\sqsubset$ on situations.

Axioms (D.35) and (D.37) restrict the graph of situations to a tree; that is, these axioms identify a situation with the sequence of actions that led to it. The original situation calculus of McCarthy and Hayes (1969) does not contain this restriction. They define a situation as “the complete state of the universe at an instant of time” (p. 477). Shanahan (1997b, p. 43) calls (D.35) and (D.37) the axioms of arboreality. He provides the following alternative, called the situation-state axiom, which identifies a situation with the fluents that are true in it:

$$s1 = s2 \leftrightarrow \forall f\ [Holds(f, s1) \leftrightarrow Holds(f, s2)]$$

Thielscher (1999b) takes a third approach: He identifies a situation with a sequence of actions and adds a separate notion of state. He introduces a function *State* that maps a situation to a state, as discussed in Section 16.4.

The equivalence of versions of the situation calculus and the event calculus was proved by Kowalski and Sadri (1997), based on their previous work (Kowalski & Sadri, 1994). Gelfond, Lifschitz, and Rabinov (1991) discuss some of the apparent limitations of the situation calculus and outline ways of overcoming them. In order to deal with indirect effects in the situation calculus, they may be represented explicitly as effect axioms (F. Lin & Reiter, 1994, sec. 5; Reiter, 2001, pp. 401-402). Other methods are discussed in the Bibliographic notes of Chapter 6.

Continuous change

Gelfond, Lifschitz, and Rabinov (1991, pp. 170-171) sketch a method for handling continuous change in the situation calculus. They introduce a function *Time* that maps situations to real numbers. Situations have the cardinality of the real numbers. Changing quantities are represented using fluents. Building on previous work (Pinto & Reiter, 1993), Pinto (1994, chap. 6) presents a way of representing continuous change and triggered events in the situation calculus. He argues (pp. 3, 67) against the approach of Gelfond et al. involving a continuum of situations. Instead, he proposes to represent episodes of continuous change within single situations. He introduces the function $start(s)$, which represents the start time of a situation s, and $end(s,a)$, which represents the end time of a situation s if action a is performed in s (p. 29), where a time is a non-negative real number.

Pinto and Reiter (1993) use a predicate $start(s,t)$, which represents that t is the start time of s. Instead of representing continuous change using fluents, Pinto (1994) introduces the notion of a continuous parameter (p. 68). He then uses $holds(p \doteq f, s)$ to represent that in situation s, the value of parameter p is given by function of time f. Thus, starting at time $start(s)$, the value of p at time t equals $f(t)$. Actions initiate and terminate episodes of continuous change. An effect axiom of the form

$$Poss(a,s) \supset holds(p \doteq f, do(a,s))$$

represents that, if a is performed in s, it initiates an episode of continuous change within the situation $do(a,s)$ in which p is described by f.

Actual situations and events

Pinto and Reiter (1993) and Pinto (1994, sec. 4.2) add actual situations and actual events to the situation calculus. The predicate $actual(s)$ represents that s is an actual situation. The initial situation is actual. Every actual situation has at most one successor situation that is actual. The predecessor situation of an actual situation is actual. The predicate $occurs(a,s)$ represents that action a actually occurs in situation s and is defined by (p. 29):

$$occurs(a,s) \equiv actual(do(a,s))$$

Thus, actions occur along a single time line, as in the event calculus. If $occurs(a,s)$, then a occurs at time $start(do(a,s))$. That is, an action a performed in situation s is considered to occur at the start time of the successor situation $do(a,s)$.

Natural actions

Pinto (1994, p. 68) introduces the notion of a *natural action*, or action that is not performed by an agent but rather occurs as a result of physical laws, such as a falling ball hitting the floor. The predicate $\Pi_t(a,s,t)$ represents that natural action a is possible at time t in situation s. A natural action a is possible in a situation s—that is, $Poss(a,s)$—if and only if a is possible at $end(s,a)$ in s and no other action is possible at a time greater than $pstart(s)$ and less than or equal to $pend(s,a)$ in s. Only one natural action is possible in each situation (p. 75). Thus, a sequence of possible natural actions uses linear time rather than branching time (p. 85). If a natural action a is possible in a situation s and a non-natural action does not occur before $end(s,a)$, then a must occur in s (p. 75). For example, the natural action *Fall* occurs in situation S_1, which gives rise to successor situation S_2. This action initiates an episode of continuous change within S_2 in which the height of the falling object is given by an equation of free-fall motion. A natural action *HitGround* occurs in

S_2, which gives rise to successor situation S_3. This is reminiscent of the operation of *Trajectory* in the event calculus.

Reiter (1996) presents some variations on Pinto's proposal. The last argument of every action predicate symbol is a time, which is a real number. The function $time(a)$ represents the time of the action a. The predicate $\Pi_t(a, s, t)$ is dispensed with because the time of a is embedded in a and whether a natural action is possible at a time is represented directly using *Poss*.

Narrative time line

R. Miller and Shanahan (1994) introduce a narrative time line consisting of the non-negative real numbers into the situation calculus. The predicate $Happens(a, t)$ represents that action a actually occurs at time t. The function $State(t)$ represents the actual situation at time t. If an action a occurs at time t_1, $State(t_1) = s_1$, and no other actions occur between t_1 and $t_2 > t1$, then $State(t_2) = do(a, s_1)$. Thus, times greater than the time of occurrence of an action correspond to the successor situation given that action. All times before the first action occurs correspond to the initial situation S_0.

Continuous change and triggered events

Using the approach of R. Miller and Shanahan (1994), and building on work by Baker (1991), R. Miller (1996) proposes a way of representing continuous change and triggered events in the situation calculus. He introduces valuations and continuously varying parameters. A *valuation* is a function from a parameter to a real number. The predicate $Triggers(v, s, a)$ represents that action a is triggered given valuation v and situation s. For example, we may represent that, if a sink is filling, then the water will overflow when its height reaches a certain level:

$$\begin{gathered} Triggers(v, s, a) \leftrightarrow \\ v(Height) = H \wedge Holds(Filling, s) \wedge a = Overflow \end{gathered}$$

An action occurs if and only if it is performed or it is triggered.

Release from inertia

A proposal for permanently releasing a fluent from the commonsense law of inertia in the situation calculus is provided by Lifschitz (1990b, p. 370). He introduces the predicate $Frame(f)$, which represents that fluent f is always subject to the law; $\neg Frame(f)$ represents that f is always released from the law.

Concurrent events

Ways of dealing with concurrent events in the situation calculus are discussed in the Bibliographic notes of Chapter 8.

Nondeterminism

F. Lin (1996) treats nondeterminism in the situation calculus. Mateus, Pacheco, and Pinto (2002) present a probabilistic version of the situation calculus. See also the Bibliographic notes of Chapter 9.

Knowledge machine

The Knowledge Machine (KM) system (P. Clark & Porter, 2004) contains a situation mechanism (P. Clark & Porter, 2000) that implements a version of the situation calculus. KM allows us to specify whether each fluent should or should not be subject to the commonsense law of inertia. A fluent subject to this law persists from a situation to a successor situation, provided that it is consistent for it to persist. The effects of actions are represented in KM using preconditions, add lists, and delete lists, along the lines of STRIPS (Fikes & Nilsson, 1971). The KM situation mechanism performs temporal projection. It does not perform abduction or postdiction, and it does not support nondeterminism, continuous change, or actions with duration.

The features and fluents framework

The features and fluents framework was introduced by Sandewall (1989a, 1989b, 1991, 1994). The framework has many facets. Sandewall (1994) formalizes a semantics for worlds inhabited by agents, called inhabited dynamical systems (IDS). He defines the syntax and semantics of several varieties of a logic called DFL. This logic is used to define *chronicles*, which specify a set of objects, the effects of events, event occurrences, and observed states. (Chronicles are similar to event calculus domain descriptions, discussed in Section 2.7.)

Assessment of logics

Sandewall presents the following systematic methodology for assessing logics for reasoning about action and change (p. 65). First, we formally define a set of chronicle classes. For example, Sandewall defines a chronicle class $\mathcal{K}\mathsf{sp} - \mathbf{IAd}$. Chronicles of this class involve accurate knowledge ($\mathcal{K}$), complete knowledge of the initial state (s), pure prediction (p), context-free inertia and integer time ($\mathbf{I}$), alternative effects of events ($\mathbf{A}$), and deterministic change ($\mathbf{d}$). Benchmark commonsense reasoning problems in $\mathcal{K}\mathsf{sp} - \mathbf{IAd}$ include the Yale shooting scenario (Hanks & McDermott, 1987, pp. 387-390), the Stockholm delivery scenario (Sandewall, 1994, pp. 31-33), and the ferryboat connection scenario (Sandewall, 1994, pp. 154-155).

Second, for each chronicle class, we formally define the set of intended models for any chronicle of the class. Specifically, we define a model set function *Mod* from chronicles to sets of intended IDS models (p. 182), and a coercion function Ci from sets of IDS models to sets of DFL models (p. 185).

Third, we define *entailment methods*, or functions from chronicles to sets of DFL models (p. 195). Sandewall defines a number of such methods, including prototypical global minimization, original chronological minimization (OCM), prototypical chronological minimization (PCM), chronological minimization of occlusion and change (CMOC), chronological assignment and minimization of occlusion and change (CAMOC), global minimization of occlusion with nochange premises (GMON), and PMON premises (PMON, the basis of TAL).

Finally, we prove that one or more entailment methods are correctly applicable (p. 201) to one or more chronicle classes. An entailment method is *correctly applicable* to a chronicle class if and only if, for every chronicle *c* of the class, the set of intended DFL models $\mathsf{Ci}(Mod(c))$ equals the set of DFL models produced by applying the entailment method to c.

Sandewall focuses on the $\mathcal{K}-\mathbf{IA}$ set of chronicle classes (pp. 159-168). He assesses the applicability of 12 entailment methods relative to the classes (pp. 201-211, 228-230, 244-252). The CAMOC and PMON entailment methods are found to be correctly applicable to $\mathcal{K}$-**IA**. Sandewall (1996) assesses methods for dealing with indirect effects of events. Brandano (2001) makes some progress on assessing the event calculus using Sandewall's methodology.

Temporal action logics

Doherty and Łukaszewicz (1994) and Doherty (1994) recast features and fluents in first-order logic. TAL was previously called PMON; the most complete description of TAL (PMON) is provided by Doherty (1996) and a later treatment of TAL is provided by Doherty, Gustafsson, Karlsson, and Kvarnström (1998). Our description is a composite of Doherty (1996) and Doherty, Gustafsson, Karlsson, and Kvarnström (1998). Indirect effects may be represented in TAL using domain constraints or using the causal constraints introduced by Gustafsson and Doherty (1996). Karlsson, Gustafsson, and Doherty (1998) treat delayed effects of events in TAL. Karlsson and Gustafsson (1997) present TAL-C, an extension of TAL that supports concurrent events with cumulative and canceling effects. Kvarnström (2001) presents the VITAL program, a tool for performing commonsense reasoning using TAL. VITAL works by using "constraint propagation to find all possible models for a given narrative" (p. 41). TAL is used by the TALplanner planning system (Kvarnström, 2005; Kvarnström & Doherty, 2000b).

Action languages

Pednault (1989) introduced ADL, an action description language with a syntax similar to that of STRIPS (Fikes & Nilsson, 1971). Gelfond and Lifschitz (1993) introduced action language $\mathcal{A}$, Kartha and Lifschitz (1994) introduced $\mathcal{AR}_0$, Baral and Gelfond (1997) introduced $\mathcal{A}_C$, Gelfond and Lifschitz (1998) introduced $\mathcal{B}$, and Kakas and Miller (1997a, 1997b) introduced action language $\mathcal{E}$. R. Miller and Shanahan (2002) describe a mapping between $\mathcal{E}$ and the classical logic event calculus. Vladimir Lifschitz and Erik Sandewall have debated the merits of action languages versus classical logic (Lifschitz, 1997a, 1997b; Sandewall, 1997). E. Giunchiglia and Lifschitz (1998) introduced action language $\mathcal{C}$, based on the language of causal theories of McCain and Turner (1997). E. Giunchiglia, Lee, Lifschitz, McCain, and Turner (2004) introduced action language $\mathcal{C}+$. Geffner (1990) introduced the distinction between what is true and what has a cause. E. Giunchiglia, Lee, Lifschitz, McCain, and Turner (2004, pp. 88, 89, 98) show that a causal theory in which all constants are Boolean can be translated into a theory of Reiter's (1980b) default logic. The causal calculator (CCALC) (E. Giunchiglia, Lee, Lifschitz, McCain, & Turner, 2004; McCain, 1997) uses satisfiability to solve reasoning problems expressed in the CCALC language, which is similar to $\mathcal{C}+$. Lee, Lifschitz, and Yang (2013) introduced $\mathcal{BC}$.

The fluent calculus

The fluent calculus was introduced and developed by Hölldobler and Schneeberger (1990), Hölldobler and Thielscher (1995), Bornscheuer and Thielscher (1996), and Thielscher (1995, 1996, 1997, 1999b). Thielscher (1999b, 2000) treats nondeterministic effects in the fluent calculus. The state update axiom for flipping a coin is based on Thielscher (1999b, p. 290). Thielscher (2000) discusses concurrent events in the fluent calculus. The state update axioms for opening a door are from Thielscher (1999c). The door latch scenario is due to Allen (1991). Thielscher (1995, 1996, 1999b, 2000) discusses indirect effects in the fluent calculus. Thielscher (1999a, 2000) treats continuous change in the fluent calculus.

Inferential frame problem

Thielscher (1999b) discusses how fluent calculus state update axioms solve the inferential frame problem of the situation calculus. The problem is to reduce the computational cost of determining the truth values of fluents in the successor situation $Do(a, s)$ that results from performing action a in situation s. In the situation calculus, we must apply the successor state axiom for each fluent of interest. In the fluent calculus, we may reason about the truth values of fluents in the successor situation simply by applying a single-state update axiom for the action a. This reasoning is possible even in the presence of partial information about the truth values of fluents. Thielscher (2005b) presents the FLUX program for performing commonsense reasoning using the fluent calculus.

Cyc

A system that uses logic for commonsense reasoning is Cyc (Lenat, 1995; Lenat & Guha, 1990). Under development since 1983, Cyc is a large commonsense knowledge base, inference engine, and natural language processing system. Knowledge is represented in Cyc using CycL, whose syntax is based on the syntax of first-order logic and Lisp (Cycorp, 2014c, chap. 2). The inference engine (Cycorp, 2014a, 2014b, 2014d) uses the resolution principle (Robinson, 1965) and heuristic search (Pearl, 1984). It uses negation as failure to implement the unique names assumption (McCarthy, 1986; Reiter, 1980a) and minimization of abnormalities for default reasoning (McCarthy, 1984a). Cyc also has specialized inference procedures to deal with reflexivity, symmetry, transitivity, dates, lists, numbers, sets, strings, and inheritance (Cycorp, 2014c, chap. 11).

Cyc has two basic ways of representing time-varying properties. In the first representation, an element *e* of the collection `SomethingExisting` is divided into elements of `SomethingExisting` that represent *e* at different slices of time (Cycorp, 2002). These objects are called *temporal subabstractions* of *e* (Lenat & Guha, 1990, pp. 211-213). For example, the following formula represents that `Nathan2007` is a temporal subabstraction of the person `Nathan`:

```
(subAbstrac Nathan Nathan2007)
```

Temporal subabstractions can be divided into further temporal subabstractions:

```
(subAbstrac Nathan2007 Nathan20070304)
(subAbstrac Nathan2007 Nathan20070305)
(subAbstrac Nathan2007 Nathan20070306)
⋮
```

Assertions are made about temporal subabstractions. The following formula states that Nathan was a scientist on March 4, 2007:

```
(occupationOf Nathan20070304 scientist)
```

The second representation uses the `TemporalThing` collection, which has `TimeInterval`, `TimePoint`, and `SomethingExisting` as specializations (Cycorp, 2002). A formula of the form `(holdsIn` t f`)` represents that formula f is true at all moments of temporal thing t. For example, the previous assertion can be represented as follows:

```
(holdsIn
(DayFn 4 (MonthFn March (YearFn 2007)))
(occupationOf Nathan scientist))
```

Assertions involving temporal subabstractions and those involving `holdsIn` are related by several axioms (Guha & Lenat, 1990b).

Cyc represents event occurrences using instances of the `Event` collection, which is a specialization of the `Situation-Temporal` collection (Cycorp, 2014c, sec. 8.1). Attributes of event instances are represented using various predicates. The time of occurrence of an event may be represented in a variety of ways:

`(dateOfEvent` e d`)`
`(startingDate` e d`)`
`(endingDate` e d`)`
`(contiguousAfter` e_1 e_2`)`
`(endsAfterEndingOf` e_1 e_2`)`
`(overlapsStart` e_1 e_2`)`
`(startsAfterStartingOf` e_1 e_2`)`
`(temporallyFinishedBy` e_1 e_2`)`
`(temporallyIntersects` e_1 e_2`)`
⋮

where e, e_1, and e_2 are event instances, and d is a date. The expression `(subEvents` e_1 e_2`)` represents that event e_2 is a part of event e_1. The expression `(eventOccursAt` e l`)` represents that l is the location of event e. The `Action` collection is a specialization of the `Event` collection. The expression `(performedBy` e a`)` represents that a is the agent of action e.

The representation of event occurrences in Cyc is based on the proposal of Davidson (1967/2001) to represent event occurrences as objects and to represent the meanings of action sentences using predicates that take those objects as arguments. Under this proposal, the sentence “I flew my spaceship to the Morning Star” (p. 114) is represented as (p. 119)

$$(\exists x)(\text{Flew}(\text{I, my spaceship}, x)\,\&\ \text{To}(\text{the Morning Star}, x))$$

Cyc does not have a standard representation for the preconditions and effects of events but, rather, uses a variety of predicates (Parmar, 2001). Cyc cannot simulate the occurrence of events (Parmar, 2001), although previous versions of Cyc had the following rudimentary mechanism for temporal projection (Guha & Lenat, 1990a, pp. 46 and 47; Guha & Lenat, 1990b). Associated with each formula is an *interval of persistence* that specifies how long the formula is expected to remain true. A frame axiom states that if (1) a formula is proved true at timepoint t_0 without using this frame axiom, (2) the formula's interval of persistence is i, (3) $t_0 < t_1 < t_0 + i$, and (4) the formula is not abnormal with respect to this frame axiom at timepoint t_1, then the formula is true at t_1. Guha (1990) describes a way of handling the Yale shooting scenario in Cyc.

OpenCyc (Cycorp, 2002b) includes an implementation of a hierarchical task network planner (Ghallab, Nau, & Traverso, 2004, chap. 11). The planner supports preconditions, context-sensitive effects, and compound events. It does not support concurrent events with cumulative and canceling effects, continuous change, indirect effects, nondeterministic effects, release from the commonsense law of inertia, or triggered events. Cyc is discussed further in Section 19.1.1.

Other formalisms

Other logical formalisms for commonsense reasoning include McDermott's (1982) temporal logic, motivated action theory (Stein & Morgenstern, 1994), and $\mathcal{LAP}_{\leadsto}$ (Castilho, Gasquet, & Herzig, 1999), which is based on dynamic logic (Harel, 1984; Pratt, 1976).

EXERCISES

16.1 (Research Problem) Write a program that translates event calculus domain descriptions into the input language of CCALC (E. Giunchiglia, Lee, Lifschitz, McCain, & Turner, 2004; McCain, 1997).

16.2 (Research Problem) Write a program that translates event calculus domain descriptions into the input language of VITAL (Kvarnström, 2001).