491 Knowledge Representation

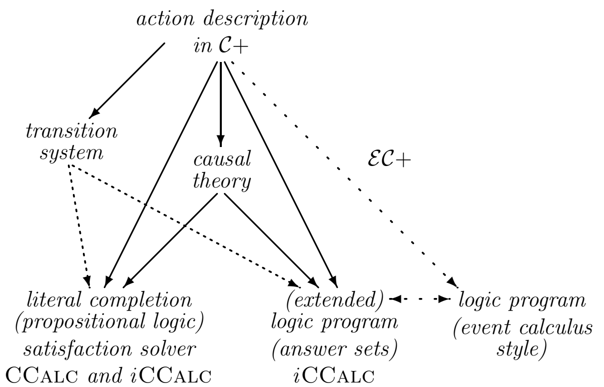
The Action Language $\mathcal{C}+$

Marek Sergot
Department of Computing
Imperial College, London

March 2009 v1.3f; November 2014 v2.0; November 2017 v2.1

Background The language $\mathcal{C}$ was introduced by Giunchiglia and Lifschitz [5]. It applies the ideas of 'causal theories' [7, 10] to reasoning about the effects of actions and the persistence ('inertia') of facts ('fluents'), building on earlier suggestions by McCain and Turner [9]. $\mathcal{C}+$ extends $\mathcal{C}$ by allowing multi-valued fluents as well as boolean fluents [4] and generalises the form of rules in the language in various ways. The definitive presentation of $\mathcal{C}+$, including various further extensions, is provided in [3]. A companion paper [1] shows how $\mathcal{C}+$ can be applied to some benchmark examples in the literature. An implementation supporting a wide range of querying and planning tasks is available in the form of the Causal Calculator (CCalc)[1]. We have our own implementation *i*CCalc which also supports a number of other extensions.

The language $\mathcal{C}+$ provides a means of constructing a *transition system* with certain properties. A separate language is used for making assertions about this transition system (what is true when) and querying it. One implementation route is via the translation of a $\mathcal{C}+$ action description into a causal theory, and thence into a set of formulas of (classical) propositional logic (its 'literal completion'). This is the method used by the Causal Calculator (CCalc). An alternative implementation route is provided by translations into extended logic programs [8], which works better and is much faster. That is the method that will be emphasised here. (Rob Craven developed another translation into logic programs with a different computational behaviour ($\mathcal{EC}+$ in the diagram. Not covered in these notes.)



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

Transition systems

A *labelled transition system* is a structure $\langle S, \mathbf{A}, R\rangle$ in which

- S is a (non-empty) set of 'states';
- $\mathbf{A}$ is a (non-empty) set of transition labels (also called 'events');
- R is a set of transitions, $R \subseteq S \times \mathbf{A} \times S$.

It does not matter whether we think of the labelled transitions as a single three-place relation R, as here, or as a family of binary relations $\{R_\varepsilon\}_{\varepsilon\in\mathbf{A}}$. The former is chosen here for consistency with published accounts of the language $\mathcal{C}+$.

A transition system can be depicted as a labelled directed graph. Every state s is a node of the graph. Labelled directed edges of the graph are the tuples (s, ε, s') of R.

We are free to interpret the labels on the transitions in various ways. The usual way is to see each label as corresponding to execution of an action or perhaps several actions concurrently. It is then usual to call the transition label an 'event'.

The triple (s, ε, s') represents execution of event ε in state s leading (possibly non-deterministically) to the state s'.

An event ε is *executable* in s when there is at least one tuple (s, ε, s') in R.

An event ε is *deterministic* in s if there is at most one such s'.

Paths, 'runs', or 'histories'

A *run* or *trace* of a transition system is a finite or infinite (ω length) path through the system. (One or other of the terms *run* or *trace* is often reserved to refer to infinite length paths. We will use 'run' and 'path' interchangeably and avoid the use of the term 'trace'. The account of $\mathcal{C}+$ in [3] uses the term 'history'.)

Let $\langle S, \mathbf{A}, R\rangle$ be a transition system. A *run* (or *path* or *history*) of length m is a sequence

$$s_0\, \varepsilon_0\, s_1 \,\cdots\, s_{m-1}\, \varepsilon_{m-1}\, s_m \qquad (m \geq 0)$$

such that $s_0, s_1, \ldots, s_m \in S$, $\varepsilon_0, \ldots, \varepsilon_{m-1} \in \mathbf{A}$, and $(s_i, \varepsilon_i, s_{i+1}) \in R$ for $0 \leq i < m$.

Sometimes there is a distinguished set $S_0 \subseteq S$ of *initial states*. All runs (or histories) are then defined so that their first state $s_0 \in S_0$. If there is a single initial state $S_0 = \{s_0\}$ then the set of all runs of the transition system can be seen as a *tree* rooted in s_0.

Query languages

A wide variety of languages—we will call them query languages—can be interpreted on labelled transition systems. These include simple propositional languages, as well as temporal logics such as CTL and LTL widely used for expressing and verifying properties of transition systems in software engineering.

Multi-valued signatures

A *multi-valued propositional signature* σ consists of:

- a set of symbols called *constants*,
- for each constant c, a non-empty set $dom(c)$ of values, called the *domain* of c. For simplicity we will assume that there are at least two distinct values in every $dom(c)$ (otherwise c is *trivial* — meaningful but causes some minor technical complications in definitions and so on, which I want to avoid).

An *atom* of a signature σ is an expression of the form $c=v$ where c is a constant in σ and $v \in dom(c)$.

A *formula* φ of signature σ is any truth-functional compound of atoms of σ.

$$\varphi ::= \bot \mid \top \mid \text{any atom } c=v \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi.$$

($\bot$ and $\top$, representing 'false' and 'true', and all atoms are formulas. If φ is a formula so is $\neg\varphi$. If φ and ψ are formulas, then so are $\varphi \wedge \psi$ and $\varphi \vee \psi$ and $\varphi \to \psi$.)

A *Boolean constant* is one whose domain is the set of truth values $\{\mathsf{t}, \mathsf{f}\}$. If p is a Boolean constant, p is shorthand for the atom $p=\mathsf{t}$ and $\neg p$ for the atom $p=\mathsf{f}$.

An *interpretation* of σ is a function that maps every constant in σ to an element of its domain. An interpretation I *satisfies* an atom $c=v$, written $I \models c=v$, if $I(c) = v$. The satisfaction relation $\models$ is extended from atoms to formulas in accordance with the standard truth tables for the propositional connectives. $\mathrm{I}(\sigma)$ stands for the set of all interpretations of σ. As usual, when X is a set of formulas, $I \models X$ signifies that I is a *model* of X, i.e., that $I \models \varphi$ for every $\varphi \in X$.

It is often convenient to represent an interpretation I by the set of atoms satisfied by I.

Reduction to Boolean signatures

Multi-valued signatures are for convenience. As long as the set of constants is finite, and the domain of every constant is finite, a multi-valued signature can be translated to an equivalent Boolean signature.

An atom $c=v$ can be viewed as a classical, propositional atom. Then add the following set of additional formulas:

$$\bigvee_v (c=v) \ \wedge \ \bigwedge_{v \neq w} \neg(c=v \wedge c=w) \qquad \text{for all } c \in \sigma$$

There are various optimisations — Details omitted.

Transition systems in $\mathcal{C}+$

States Let σ^{f} be a multi-valued signature of constants called 'state variables', or more usually in AI terminology, *fluent constants*. Given a labelled transition system $\langle S, \mathbf{A}, R\rangle$ we add a valuation function which specifies, for every fluent constant $f \in \sigma^{\mathrm{f}}$ and every state $s \in S$, a value in $dom(f)$. We shall be dealing with the *special case* of transition systems in which

- each state $s \in S$ is an interpretation of σ^{f}, $S \subseteq \mathrm{I}(\sigma^{\mathrm{f}})$.

Not all interpretations of σ^{f} are states, in general. (There are usually constraints that have to be satisfied.)

It is convenient to adopt the convention that an interpretation I of σ^{f} is represented by the set of atoms of σ^{f} that are satisfied by I. A state is then a (complete, and consistent) set of fluent atoms, and a separate valuation function is unnecessary. We say a formula φ 'holds in' state s or 'is true in' state s as alternative ways of saying that s satisfies φ.

Transition labels Although it is much less common, an idea employed in $\mathcal{C}+$ is that another category of constants and formulas — *action formulas* — can be interpreted on the transition labels/events of a transition system. So, let σ^{a} be a multi-valued signature of constants called *action constants*, disjoint from σ^{f}. Given a labelled transition system $\langle S, \mathbf{A}, R\rangle$ we add a valuation function for action constants which specifies, for every action constant $a \in \sigma^{\mathrm{a}}$ and every label/event $\varepsilon \in \mathbf{A}$, a value in $dom(a)$. Again, we deal with a special case, the case of labelled transition systems in which the set $\mathbf{A}$ of labels/events is the set of interpretations of σ^{a}. In other words the transition systems of interest will be those of the form $\langle \sigma^{\mathrm{f}}, S, \mathrm{I}(\sigma^{\mathrm{a}}), R\rangle$, on which we will interpret various query languages of signature $\sigma^{\mathrm{f}} \cup \sigma^{\mathrm{a}}$, or variations thereof. $(\sigma^{\mathrm{f}}, \sigma^{\mathrm{a}})$ is the 'action signature' of the transition system.

Note that since a transition label/event ε is an interpretation of σ^{a}, it is meaningful to say that ε satisfies an action formula α ($\varepsilon \models \alpha$). When $\varepsilon \models \alpha$ we say that the event ε is of type α. When $\varepsilon \models \alpha$ we also say that the transition (s, ε, s') is a transition of type α.

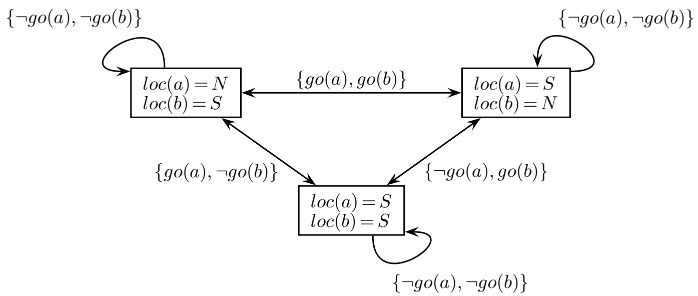
Since a transition label is an interpretation of the action constants σ^{a}, it can also be represented by the set of atoms that it satisfies. The suggested reading of a transition label $\{a_1 = v_1, a_2 = v_2, \ldots, a_n = v_n\}$ for an action signature with action constants $a_1, a_2, \ldots, a_n$ is that it represents a composite action in which the elementary actions $a_1 = v_1, a_2 = v_2 \ldots, a_n = v_n$ are performed (or occur) concurrently. Where a is a Boolean action constant, $\neg a$, i.e. $a = \mathsf{f}$, can be read as indicating that action a is not performed; and where all action constants are Boolean, the action $\{a_1 = \mathsf{f}, \ldots, a_n = \mathsf{f}\}$ can be read as representing the 'null' event.

For example: suppose there are three agents, a, b, and c which can move in direction E, W, N, or S, or remain idle. Suppose (for the sake of an example) that they can also whistle as they move (they are trains, let us say). Let the action signature consist of action constants $move(a)$, $move(b)$, $move(c)$ with domains $\{E, W, N, S, idle\}$, and Boolean action constants $whistle(a)$, $whistle(b)$, $whistle(c)$. Then one possible interpretation of the action signature, and therefore one possible transition label, is

$$\{move(a) = E, move(b) = N, move(c) = idle, whistle(a), \neg whistle(b), whistle(c)\}$$

Because of the way that action formulas are evaluated on a transition (s, ε, s'), an action formula can also be regarded as expressing a property of the transition (s, ε, s') as a whole. The term 'transition constants' might have been better for σ^{a} therefore; I will stick to the $\mathcal{C}+$ terminology and call them 'action constants'.

Example Let σ^{f} be the set of fluent constants $\{loc(a), loc(b)\}$ with possible values $\{N, S\}$, and let σ^{a} be the set of Boolean action constants $\{go(a), go(b)\}$. Consider the transition system $\mathcal{T}$ depicted in the following diagram:



There is no state $\{loc(a) = N, loc(b) = N\}$ in $\mathcal{T}$ (for the sake of the example).

Query language: example (time-stamped query language)

Query languages can be interpreted on the paths ('runs') of a transition system. There are many, many possibilities. One candidate, and the only one we will consider, is the query language used in CCALC. This uses propositional formulas of time-stamped fluent and action constants: the time-stamped fluent atom $f[i] = v$ represents that fluent atom $f = v$ holds at integer time i, or more precisely, that $f = v$ is satisfied by the state s_i of a path $s_0\,\varepsilon_0\,\cdots\,\varepsilon_{i-1}\,s_i\cdots$ of the transition system; the time-stamped atom $a[i] = v$ represents that action atom $a = v$ is satisfied by the transition label ε_i of a path $s_0\,\varepsilon_0\,\cdots\,s_i\,\varepsilon_i\,s_{i+1}\cdots$.

Time-stamped formulas are therefore evaluated on *paths* of the transition system. Note that the paths of length 0 are the *states* and the paths of length 1 are the *transitions*.

You can stop reading here and just skip to the example that follows. If you are interested in a more careful exposition, here are the details.

Time-stamping: details (can be skipped)

In general, given a multi-valued signature σ and a non-negative integer i, we write $\sigma[i]$ for the signature consisting of all constants of the form $c[i]$ where c is a constant of σ, with $dom(c[i]) = dom(c)$. For any non-negative integer m, we write σ_m for the signature $\sigma[0] \cup \cdots \cup \sigma[m]$.

The time-stamped query language used in CCALC to express properties of paths of length m of a transition system with action signature $(\sigma^{\mathrm{f}}, \sigma^{\mathrm{a}})$ is the propositional language of signature $\sigma^{\mathrm{f}}_m \cup \sigma^{\mathrm{a}}_{m-1}$. In other words, the formulas of this query language are:

- atoms $f[i] = v$ where $i \in 0\,..\,m$ and $f = v$ is a fluent atom of σ^{f};
- atoms $a[i] = v$ where $i \in 0\,..\,m{-}1$ and $a = v$ is an action atom of σ^{a};
- all truth-functional compounds of the above.

Let $\pi = s_0\,\varepsilon_0\,s_1\,\cdots\,s_{m-1}\,\varepsilon_{m-1}\,s_m$ be a path of length m of a transition system $\mathcal{T}$ of action signature $(\sigma^{\mathrm{f}}, \sigma^{\mathrm{a}})$. An atom $f[i] = v$ for any fluent constant f of σ^{f} and $0 \le i \le m$ is *true on* path π (or 'holds on' path π, or 'is satisfied by' path π), written $\mathcal{T}, \pi \models_m f[i] = v$, when $s_i \models f = v$; for action constants a of σ^{a} and $0 \le i < m$, $\mathcal{T}, \pi \models_m a[i] = v$ when $\varepsilon_i \models a = v$; and $\models_m$ is extended to formulas φ of signature $\sigma_m = \sigma^{\mathrm{f}}_m \cup \sigma^{\mathrm{a}}_{m-1}$ by the usual truth tables for the propositional connectives.

We will say φ is true on paths of length m of $\mathcal{T}$, written $\mathcal{T} \models_m \varphi$ when $\mathcal{T}, \pi \models_m \varphi$ for all paths π of length m of $\mathcal{T}$.

Equivalently, ...

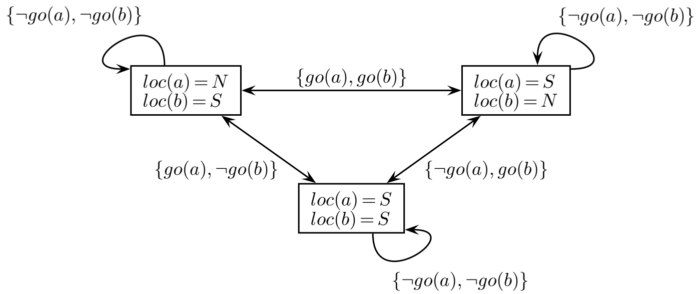
Let $\pi[i]$ denote the ith component of a path π: that is, when $\pi = s_0\,\varepsilon_0\,s_1\,\cdots\,s_i\,\varepsilon_i\,s_{i+1}\,\ldots$, let $\pi[i] = s_i \cup \varepsilon_i$. Clearly, $\pi[i]$ is an interpretation of $\sigma^{\mathrm{f}} \cup \sigma^{\mathrm{a}}$ when π is a path of a transition system with action signature $(\sigma^{\mathrm{f}}, \sigma^{\mathrm{a}})$.

For any formula ψ of signature $\sigma^{\mathrm{f}} \cup \sigma^{\mathrm{a}}$, let $\psi[i]$ stand for the formula of signature $\sigma^{\mathrm{f}}[i] \cup \sigma^{\mathrm{a}}[i]$ obtained by time-stamping every constant in ψ with i, that is, replacing every constant c in ψ by the constant $c[i]$. Clearly, every formula φ of signature $\sigma_m = \sigma^{\mathrm{f}}_m \cup \sigma^{\mathrm{a}}_{m-1}$ is a truth-functional compound of formulas of the form $\psi[i]$ where $0 \le i \le m$ and ψ is a formula of signature $\sigma = \sigma^{\mathrm{f}} \cup \sigma^{\mathrm{a}}$.

Now, for any path π of length m of a transition system $\mathcal{T}$ of action signature $(\sigma^{\mathrm{f}}, \sigma^{\mathrm{a}})$, we have

$$\mathcal{T}, \pi \models_m \psi[i] \quad \text{iff} \quad \pi[i] \models_\sigma \psi$$

Example (contd) Consider again the transition system $\mathcal{T}$:



Time-stamped formulas are evaluated on *paths* of the transition system.

$\mathcal{T}, \pi \models_m \varphi$ means time-stamped formula φ is true on the path π of length m of $\mathcal{T}$.

$\mathcal{T} \models_m \varphi$ when $\mathcal{T}, \pi \models_m \varphi$ for all paths π of length m of $\mathcal{T}$.

We have, amongst other things:

$$\begin{array}{ll}
\mathcal{T} \models_1 & (loc(a)[0] = N \ \wedge \ go(a)[0]) \rightarrow loc(a)[1] = S \\
\mathcal{T} \models_2 & (loc(a)[0] = N \ \wedge \ go(a)[0] \ \wedge \ go(a)[1]) \rightarrow loc(a)[2] = N \\
\mathcal{T} \models_1 & (loc(a)[0] = N \ \wedge \ \neg go(a)[0]) \rightarrow loc(a)[1] = N \\
\mathcal{T} \models_2 & (loc(a)[0] = N \ \wedge \ go(b)[0] \ \wedge \ go(a)[1]) \rightarrow loc(a)[2] = N \\
\mathcal{T} \models_m & (loc(a)[i] = N \ \wedge \ loc(a)[i{+}2] = N) \rightarrow (go(a)[i] \leftrightarrow go(a)[i{+}1]) \quad \text{for all } 0 \leq i \leq m{-}2 \\
\mathcal{T} \models_m & (loc(a)[i] = S \ \wedge \ loc(b)[i] = S) \rightarrow \neg(go(a)[i] \ \wedge \ go(b)[i]) \quad \text{for all } 0 \leq i \leq m{-}1
\end{array}$$

(Thanks to Robin Gallimard for pointing out an error in an earlier version of these notes.)

Examples of computational tasks

Prediction Given a transition system $\mathcal{T}$ and a time-stamped query language of signature $(\sigma^{\mathrm{f}}, \sigma^{\mathrm{a}})$:

- Initially F holds.
- Partially specified events of type $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ happen.
- Does it follow that G holds in state m?

F and G are formulas of σ^{f} and α_i are formulas of σ^{a}.

We want to know whether, for every path/run $\pi = s_0\,\varepsilon_0\,s_1 \cdots s_{m-1}\,\varepsilon_{m-1}\,s_m, \cdots$ of $\mathcal{T}$ such that $s_0 \models F$ and $\varepsilon_i \models \alpha_i$ for each $i \in 0..m{-}1$, we have $s_m \models G$.

Or in other words is it the case that

$$\mathcal{T} \models_m (F[0] \wedge \alpha_0[0] \wedge \alpha_1[1] \wedge \cdots \wedge \alpha_{m-1}[m{-}1] \rightarrow G[m])$$

A variant of the problem:

- Initially F holds.
- Partially specified events of type $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ happen.
- Is it *possible* that G holds in state m?

Is there a possible run/path π through the transition system $\mathcal{T}$ such that

$$\mathcal{T}, \pi \models_m F[0] \wedge \alpha_0[0] \wedge \alpha_1[1] \wedge \cdots \wedge \alpha_{m-1}[m{-}1] \wedge G[m]$$

What is this path π?

'Postdiction' (stupid term)

- Partially specified events of type $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ happen.
- G holds now (at time m).
- Does it follow that initially F?

We want to know whether

$$\mathcal{T} \models_m (\alpha_0[0] \wedge \alpha_1[1] \wedge \cdots \wedge \alpha_{m-1}[m{-}1] \wedge G[m] \rightarrow F[0])$$

And as before, checking whether there is a possible path/run π of $\mathcal{T}$ such that

$$\mathcal{T}, \pi \models_m (\alpha_0[0] \wedge \alpha_1[1] \wedge \cdots \wedge \alpha_{m-1}[m{-}1] \wedge G[m] \wedge F[0])$$

asks whether it is *possible* that initially F. π, if it exists, shows how it is possible.

Temporal interpolation Prediction and 'postdiction' are both special cases of the general problem in which:

- Partially specified events of type $\alpha_0, \alpha_1, \ldots, \alpha_k$ happen.
- Certain combinations of fluents (partially specified states) hold at given times.

We want to determine what holds in each state, or what *possibly* holds in each state.

'Planning'

- Initially F.
- Goal: G.

Find the shortest sequence of fully specified actions (i.e., events, or transition labels) $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{k-1}$ such that there is a path/run $s_0\, \varepsilon_0\, s_1 \cdots s_{k-1}\, \varepsilon_{k-1}\, s_k$ of $\mathcal{T}$ in which $s_0 \models F$ and $s_k \models G$.

We try *consecutively* for $k = 0, 1, \ldots$ up to some specified maximum value m to find a path π of $\mathcal{T}$ such that:

$$\mathcal{T}, \pi \models_k F[0] \wedge G[k]$$

If there is such a path $\pi = s_0\, \varepsilon_0\, s_1 \cdots s_{k-1}\, \varepsilon_{k-1}\, s_k$ then it contains a representation of the plan: $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{k-1}$.

But note This is often called *planning* in the AI literature but it's not *really* planning. There is more to planning that just finding a suitable sequence of events $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{k-1}$ that gets us from the initial state to the goal state. For instance, some of these ε_i might be *non-deterministic*. Calling this a 'plan' is then wishful thinking. It would be like saying that my plan for getting rich is to bet £1000 on a particular horse, because there is one possible path from where we are now to where I am rich in which I bet on this horse and it wins. Similarly (it comes to the same thing) some of these events ε_i may represent actions by other agents over whom I have no control. I might as well say my plan to get rich is that some rich person gives me £1000, because there is a possible path which gets me from where I am to where I am rich in which that happens. There's obviously more to planning. No time for further discussion of real planning methods in this course.

Other possible problems

- Given a sequence of (partially specified) events $\alpha_0\, \alpha_1 \ldots \alpha_k$ (no gaps), is this consistent with a given transition system? This can be combined with partial information about these actions, and about some or all of the states. This is an instance of the temporal interpolation problem above.
- Given a sequence of (partially specified) events $\alpha_0\, \alpha_1 \ldots \alpha_k$, but with possible gaps, is *this* consistent with a given transition system ? What are the complete (no gap) sequences of events? This is obviously much harder.

The Action Description Language $\mathcal{C}+$

The language $\mathcal{C}+$ has evolved through several versions. Here we follow the (definitive) presentation in [3] though we will deal with a slightly simplified version of the language to avoid unnecessary detail.

An *action description* in $\mathcal{C}+$ is a set of $\mathcal{C}+$ laws that define a *transition system* of a certain kind.

Syntax

An *action signature* is a (non-empty) set σ^{f} of *fluent constants* and a (non-empty) set σ^{a} of *action constants*.

A *fluent formula* is any truth-functional compound of fluent atoms (i.e., a formula of signature σ^{f}). An *action formula* is any formula of signature σ^{a}. The language also allows formulas of signature $\sigma^{\mathrm{f}} \cup \sigma^{\mathrm{a}}$.

So we have:

- fluent atoms $f = v$, p, $\neg p$
- action atoms $a = v$, a, $\neg a$

The full language also has *rigid fluents* (which do not change value from state to state), and a sub-category of fluents called *statically determined fluents*. I will not bother with rigid fluents. I will ignore statically determined fluents for now so as not to distract attention from the main ideas.

There are three kinds of expressions in $\mathcal{C}+$:

(1) **Static laws**

$$\textsf{caused } F \textsf{ if } G$$

where F and G are fluent formulas (i.e., formulas of signature σ^{f}).

Static laws are used to express constraints that hold in all states.

(2) **Fluent dynamic laws**

$$\textsf{caused } F \textsf{ if } G \textsf{ after } \psi$$

where F and G are fluent formulas, and ψ is a formula of signature $\sigma^{\mathrm{f}} \cup \sigma^{\mathrm{a}}$.

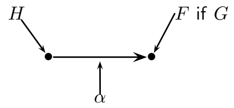
Informally, in a transition (s, ε, s'), formulas F and G are evaluated at s' (the resulting state), fluent atoms in ψ are evaluated at s (i.e., in the state immediately before the transition), and action atoms in ψ are evaluated on the transition ε itself, as explained below.

Fluent dynamic laws are primarily used to express how the values of fluents are affected by different kinds of actions, and to specify which fluents are 'inertial'.

It might be helpful to note that a fluent dynamic law can be written equivalently as a set of laws of the form

$$\textsf{caused } F \textsf{ if } G \textsf{ after } H \wedge \alpha$$

where H is a fluent formula (no action constants) and α is an action formula (no fluent constants).



(3) **Action dynamic laws**

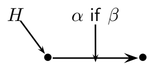
$$\textsf{caused } \alpha \textsf{ if } \psi$$

where α is an action formula (i.e., a formula of signature σ^{a}) and ψ is any formula of signature $\sigma^{\mathrm{f}} \cup \sigma^{\mathrm{a}}$.

An action dynamic law can be written equivalently as a set of laws of the form

$$\textsf{caused } \alpha \textsf{ if } \beta \wedge H$$

where H is a fluent formula (no action constants) and β is an action formula.



Two special cases:

$$\textsf{caused } \alpha \textsf{ if } \beta$$

(Every transition/event of type β is also a transition/event of type α.)

$$\textsf{caused } \alpha \textsf{ if } H$$

(Whenever a state satisfies fluent formula H there is a transition/event of type α from that state.)

Note In the rest of the notes I usually omit the keyword caused . This is to save space.

There are also various (optional) **abbreviations** for commonly occuring patterns of laws. See below.

Definite action descriptions

An action description D is *definite* when, for all static laws, fluent dynamic laws and action dynamic laws in D:

- the head of every law is either a fluent atom or the symbol $\perp$, and
- no atom is the head of infinitely many laws of D.

(Remember that a Boolean fluent constant p and its negation $\neg p$ are treated as atoms, and hence are included in the definition.)

Definite action descriptions are the ones of practical interest.

Static laws:	caused $f = v$ if G or caused $\perp$ if G
Fluent dynamic laws:	caused $f = v$ if G after ψ or caused $\perp$ if G after ψ
Action dynamic laws:	caused $a = v$ if ψ

Example The effects of toggling a switch between on and off can be represented by a Boolean fluent *on* and a Boolean action constant *toggle* and the following pair of laws:

$toggle$ causes on if $\neg on$
$toggle$ causes $\neg on$ if on

These are shortand for the following fluent dynamic laws:

on if $\top$ after $toggle \wedge \neg on$
$\neg on$ if $\top$ after $toggle \wedge on$

Example: 'inertia' Default persistence ('inertia') of fluents is not a built-in feature of the $\mathcal{C}+$ language. One specifies explicitly which fluents are 'inertial' by means of a $\mathcal{C}+$ law of the form

inertial f

This is shorthand for the set of fluent dynamic laws of the form

$f = v$ if $f = v$ after $f = v$, for every $v \in dom(f)$.

How this form of rule works to express default persistence of $f = v$ will become clearer when we look at the semantics of $\mathcal{C}+$ laws.

Example Not all fluents are inertial. Here is a traffic light:

$light = yellow$ if $\top$ after $light = green$
$light = red$ if $\top$ after $light = yellow$
$light = red\_yellow$ if $\top$ after $light = red$
$light = green$ if $\top$ after $light = red\_yellow$

Example Shooting someone with a loaded gun makes them not alive.

$\neg alive$ if $\top$ after $shoot \wedge loaded$

Example Suppose the shooter is not always accurate. Suppose shooting someone with a loaded gun is *non-deterministic*

% *alive* is inertial
$alive$ if $alive$ after $alive$
$\neg alive$ if $\neg alive$ after $\neg alive$

% shooting is non-deterministic (even with a loaded gun)
$\neg alive$ if $\neg alive$ after $shoot \wedge loaded$

Abbreviations

The language $\mathcal{C}+$ provides various (optional) abbreviations. Here are the most common. (We won't bother with the full list.)

default F	F if F
default F if G	F if $F \wedge G$
inertial f	$f = v$ if $f = v$ after $f = v$ for all $v \in dom(f)$
α causes G	G if $\top$ after α
α causes G if ψ	G if $\top$ after $\alpha \wedge \psi$
nonexecutable α	$\bot$ if $\top$ after α (or: α causes $\bot$)
nonexecutable α if ψ	$\bot$ if $\top$ after $\alpha \wedge \psi$ (or: α causes $\bot$ if ψ)
α may cause G	G if G after α
α may cause G if ψ	G if G after $\alpha \wedge \psi$

How these rules work to express *defaults* will become clearer when we look at the semantics of $\mathcal{C}+$ laws.

(You don't have to learn these abbreviations off by heart!!)

Semantics

(The rationale behind these definitions is *far from obvious.* They come from the formalism of 'nonmonotonic causal theories' in which $\mathcal{C}+$ has its roots. It is *NOT NECESSARY* to memorize the definitions in this section. It is not even necessary to read them.)

An action description D of $\mathcal{C}+$ defines a labelled transition system

$$\langle \sigma^{\mathrm{f}}, \sigma^{\mathrm{a}}, S, \mathbf{A}, R \rangle$$

- a *state*
  - is an interpretation of σ^{f} (the fluent constants)
  - that satisfies $G \rightarrow F$ for every static law caused F if G in D
    (and some extra conditions for 'statically determined' fluents)
- a *transition label* (or *event*)
  - is an interpretation of σ^{a} (the action constants)
- a *transition* is a triple (s, ε, s') where s and s' are states and ε is a transition label/event. s is the *initial* state of the transition and s' is the *resulting state.* A transition *defined by* a *definite* action description D must satisfy the following additional constraints.

$$\begin{aligned} T_{static}(s) &=_{\mathrm{def}} \{F \mid F \text{ if } G \text{ is in } D,\ s \models G\} \\ E(s, \varepsilon, s') &=_{\mathrm{def}} \{F \mid F \text{ if } G \text{ after } \psi \text{ is in } D,\ s' \models G,\ s \cup \varepsilon \models \psi\} \\ A(\varepsilon, s) &=_{\mathrm{def}} \{A \mid A \text{ if } \psi \text{ is in } D,\ s \cup \varepsilon \models \psi\} \end{aligned}$$

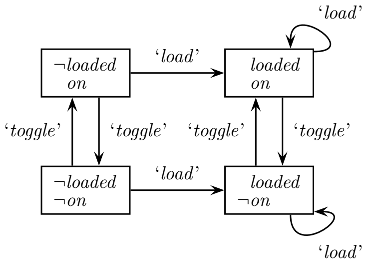
  $\langle s, \varepsilon, s' \rangle$ is a transition iff:
  - $s \models T_{static}(s)$
  - $s' = T_{static}(s') \cup E(s, \varepsilon, s')$
  - $\varepsilon \models A(\varepsilon, s)$

  (and some extra details for 'statically determined' fluents)

You can ignore these formal definitions. Their purpose is to justify the translation of $\mathcal{C}+$ action descriptions to logic programs which comes presently.

Example (first, one without any static laws)

Signature: Boolean fluent constants *loaded*, *on*; Boolean action constants *load*, *toggle*.



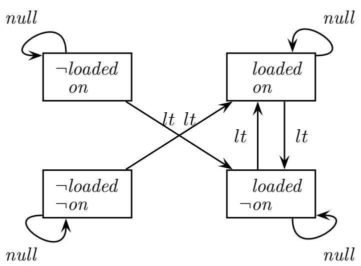
inertial *loaded*
inertial *on*

load causes *loaded*
toggle causes *on* if $\neg on$
toggle causes $\neg on$ if *on*

(Action *load* is supposed to mean something like ‘ensure that loaded’. Otherwise we would change the action description to *load* causes *loaded* if $\neg loaded$.)

In the diagram, transition labels ‘*load*’ and ‘*toggle*’ are shorthand for $\{load, \neg toggle\}$ and $\{\neg load, toggle\}$, respectively.

There are two other events/labels in this transition system, not shown in the diagram above. They are the events $\{load, toggle\}$ and $\{\neg load, \neg toggle\}$ (‘null’ event).



Here, the label *lt* is shorthand for $\{load, toggle\}$ and *null* is shorthand for the ‘null’ event $\{\neg load, \neg toggle\}$.

If we wanted to eliminate the ‘null’ event, we could add the following law to the action description:

$\perp$ if $\top$ after $\neg load \wedge \neg toggle$

for which there is a standard abbreviation in $\mathcal{C}+$:

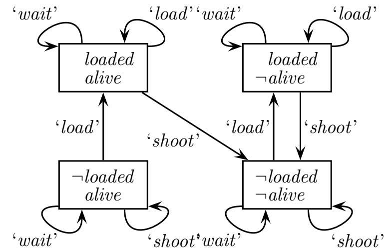
nonexecutable $\neg load \wedge \neg toggle$

If we wanted to eliminate the possibility of concurrent execution of *load* and *toggle* we would add

nonexecutable $load \wedge toggle$

Example (‘Yale Shooting Problem’)

Signature: Boolean fluent constants *loaded*, *alive*; Boolean action constants *load*, *shoot*, *wait*.



inertial *loaded*
inertial *alive*

load causes *loaded*
shoot causes $\neg alive$ if *loaded*
shoot causes $\neg loaded$ (**)

nonexecutable $shoot \wedge load$
nonexecutable $wait \wedge shoot$
nonexecutable $wait \wedge load$
$\perp$ after $\neg wait \wedge \neg shoot \wedge \neg load$

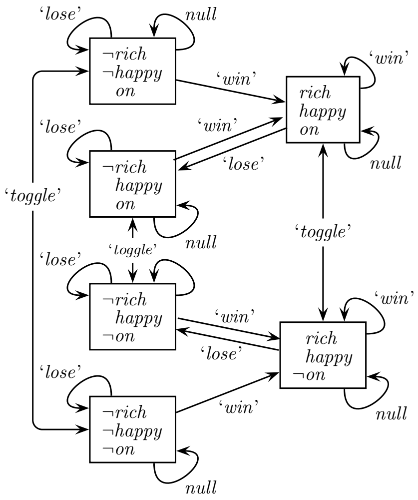
(**) says that shooting the gun unloads it. That isn't part of the original statement of the ‘Yale Shooting Problem’. I just thought I would include it.

It is not possible to load and shoot a gun at the same time: $shoot \wedge load$ events are eliminated by the first of the nonexecutable laws.

Alternatively: we could eliminate the action constant *wait* and represent it instead by the ‘null’ event $\{\neg shoot, \neg load\}$. The last three lines of the action description could then be deleted.

Example (completely artificial; just for the sake of an example)

Signature: Boolean fluent constants *rich*, *happy*, *on*; Boolean action constants *win*, *lose*, *toggle*.



inertial *rich*
inertial *on*
inertial *happy*

win causes *rich*
lose causes $\neg$*rich*
toggle causes *on* if $\neg$*on*
toggle causes $\neg$*on* if *on*

happy if *rich* (a static law)

nonexecutable *lose* $\wedge$ *win*

Because of the static law, there are only 6 states not $2^3 = 8$. The diagram does not show the transitions with labels $\{toggle, win, \neg lose\}$ and $\{toggle, \neg win, lose\}$.

Example (Winning the lottery)

Winning the lottery causes one to become (or remain) rich. Losing one’s wallet causes one to become (or remain) not rich. A person who is rich is happy.

Signature: Boolean fluent constants *alive*, *rich*, *happy*; Boolean action constants *birth*, *death*, *win*, *lose*.

inertial *alive*
inertial *rich*
inertial *happy*

birth causes *alive*
nonexecutable *birth* if *alive*

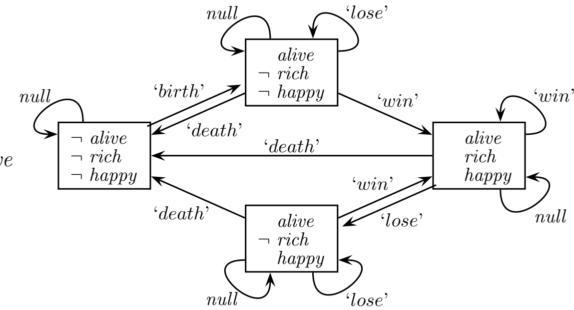
death causes $\neg$*alive*
nonexecutable *death* if $\neg$*alive*

win causes *rich*
nonexecutable *win* if $\neg$*alive*

lose causes $\neg$*rich*
nonexecutable *lose* if $\neg$*alive*

happy if *rich*
$\bot$ if $rich \wedge \neg alive$
$\bot$ if $happy \wedge \neg alive$

nonexecutable $birth \wedge death$
nonexecutable $birth \wedge win$
nonexecutable $birth \wedge lose$
nonexecutable $win \wedge lose$



Because of the static laws, there are only four states in the transition system and not $2^3 = 8$. Transition labels ‘*birth*’, ‘*death*’, ‘*win*’, ‘*lose*’ in the diagram are shorthand for the events $\{birth, \neg death, \neg win, \neg lose\}$, $\{\neg birth, death, \neg win, \neg lose\}$, $\{\neg birth, \neg death, win, \neg lose\}$, $\{\neg birth, \neg death, \neg win, lose\}$, respectively. The label *null* is shorthand for the ‘null’ event $\{\neg birth, \neg death, \neg win, \neg lose\}$.

Notice that as formulated here, the example allows for reincarnation: a person can be born, die, and be born again. The possibility of reincarnation can be eliminated easily enough, for example by adding another fluent constant *dead* together with a static law $\bot$ if $alive \wedge dead$; the simpler version with reincarnation is perfectly adequate for present purposes.

The diagram does not show transitions of type $death \wedge lose$ (i.e., transitions with label $\{\neg birth, death, \neg win, lose\}$). Their effects are exactly the same as ‘*death*’ transitions.

There are no transitions of type $win \wedge death$ because they would lead to states with $rich \wedge \neg alive$. There are no such states because of the static law $\bot$ if $rich \wedge \neg alive$.

The laws nonexecutable $birth \wedge death$ and nonexecutable $win \wedge lose$ could be deleted without affecting the transition system. There are no transitions of type $birth \wedge death$ because they would lead to states with $alive \land \neg alive$, and there are no such states. There are no transitions of type $win \land lose$ because they would lead to states with $rich \land \neg rich$.

In fact, all the nonexecutable statements in this example could be omitted (they are all implied). (This isn't obvious, but turns out to be the case on closer examination.)

Statically determined fluent constants

A state of the transition system is uniquely determined by the values of the fluent constants.

In the previous example there are three Boolean fluent constants:

$$alive,\ rich,\ happy$$

All three are declared inertial. A static law

$$\textsf{caused}\ happy\ \textsf{if}\ rich$$

eliminates certain combinations.

The language $\mathcal{C}+$ actually has two different kinds of fluent constant: *simple* fluent constants and *statically determined fluent constants*.

A state of the transition system is uniquely determined by the values of the *simple* fluent constants.

The values of the *statically determined fluent constants* are defined in terms of the simple fluent constants (or other statically determined fluent constants). One does not write dynamic laws saying how the values of statically determined fluent constants change from state to state. Their values are defined in terms of other fluents.

(Statically determined fluent constants are an optional extra. They don't change expressive power but are sometimes useful.)

Example (An alternative, different version of the previous one)

Signature: (simple) Boolean fluent constants $alive, rich$;
(Boolean action constants $birth, death, win, lose$ as before);
Boolean statically determined fluent constant $happy$.

Suppose a person is happy if and only if he is rich.

$$\textsf{default}\ \neg happy$$
$$\textsf{caused}\ happy\ \textsf{if}\ rich$$

Other features of the action description as before, except that ...

We do not specify now that $happy$ is inertial. It is statically determined. Its value is determined by the value of $rich$ (a simple fluent constant). In this version, if a person ceases to be rich, s/he ceases to be happy. ($happy$ is not inertial.)

Example (Going to work 1)

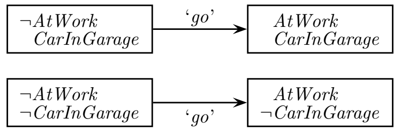
This illustrates non-deterministic actions.

Let the Boolean action constant go represent 'Jack goes to work'. Jack can go to work by walking or, if his car is in his garage, he can drive. For simplicity, to simplify the diagrams, we ignore the possibility that Jack goes in the opposite direction.

The following action description

> $\textsf{inertial}\ AtWork$
> $\textsf{inertial}\ CarInGarage$
> $go\ \textsf{causes}\ AtWork$
> $\textsf{nonexecutable}\ go\ \textsf{if}\ AtWork$

makes 'go' deterministic in all states, as shown in the following diagram



(Reflexive edges corresponding to the event $\{\neg go\}$ are not shown.)

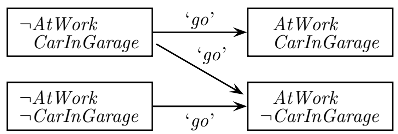
But what we expect (or want) is that go is non-deterministic in those states where $CarInGarage$ is true, because here Jack can either walk to work or drive and thereby move his car. To obtain this effect one adds another statement to the action description:

> $go\ \textsf{may cause}\ \neg CarInGarage\ \textsf{if}\ CarInGarage$

This is an abbreviation for the dynamic law

> $\neg CarInGarage\ \textsf{if}\ \neg CarInGarage\ \textsf{after}\ CarInGarage \land go$

With this additional statement we obtain the following transition diagram ('null' events $\{\neg go\}$ omitted):



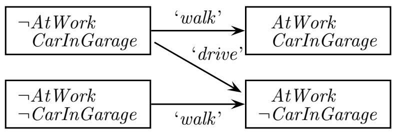
Example (Going to work 2)

Alternatively, we could distinguish between walking to work and driving to work. Let us have two Boolean action constants *walk* and *drive* to represent walking and driving to work respectively. The action description

inertial $AtWork$
$walk$ causes $AtWork$
$drive$ causes $AtWork$
$drive$ causes $\neg CarInGarage$ if $CarInGarage$

nonexecutable $walk$ if $AtWork$
nonexecutable $drive$ if $AtWork$
nonexecutable $drive$ if $\neg CarInGarage$
nonexecutable $walk \wedge drive$

defines the following transition system ('null' events $\{\neg walk, \neg drive\}$ omitted):



The first two causes laws could be replaced by the (equivalent) law: $(walk \vee drive)$ causes $AtWork$

We could also represent that *walk* and *drive* are both kinds of *go* by means of action dynamic laws:

caused go if $walk$
caused go if $drive$

Or (equivalently as it turns out) by the pair of fluent dynamic laws:

nonexecutable $walk \wedge \neg go$ $\quad$ ($walk \wedge \neg go$ causes $\bot$)
nonexecutable $drive \wedge \neg go$ $\quad$ ($drive \wedge \neg go$ causes $\bot$)

We might also wish to add (in the absence of another kind of *go*, such as cycling):

nonexecutable $go \wedge \neg walk \wedge \neg drive$

This would not change the form of the transition system shown above except to replace transition labels '*walk*' and '*drive*' by $\{go, walk\}$ and $\{go, drive\}$ respectively.

Notice that the transition label $\{go, walk\}$ cannot distinguish between two concurrent but unrelated actions *go* and *walk* and one action 'go by walking'. We have an extended version of $\mathcal{C}+$ which is intended to address such issues, amongst other things.

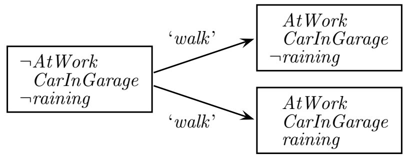
Example (Going to work 3)

Here is an example of another source of non-determinism. Some fluents vary from state to state but are not 'caused' by any kind of action. Such fluents are called 'exogenous'.

Take the previous example and add a Boolean constant *raining*. We express that *raining* is exogenous by adding the following pair of static laws:

$raining$ if $raining$
$\neg raining$ if $\neg raining$

Here is a fragment of the transition system obtained:



The pair of static laws for *raining* above may also be written more concisely in $\mathcal{C}+$ as:

exogenous $raining$

In general, for a fluent constant f, the abbreviation

exogenous f

stands for the set of static laws $f = v$ if $f = v$, for every $v \in dom(f)$..

Example (Going to work 4)

This is just to illustrate the use of multi-valued fluents and action constants. (Action constants can also be multi-valued.)

Suppose there are three agents a, b, c. Each has a car.

There are three locations: *home*, *work*, *pub*.

Fluent symbols:
$loc(x) = p$: agent x is at location p
$car(x) = p$: agent x's car is at location p

Action symbols:
$walk(x) = dest$: x walks to *dest*
$drive(x) = dest$: x drives to *dest*
Note that the domain of $walk(x)$ and $drive(x)$ are 'destinations' not locations:
$dom(walk(x)) = dom(drive(x)) = \{home, work, pub, none\}$.
This is because every action constant must have a value in every model and obviously we want transitions in which an agent does not walk and/or does not drive. (Very easy to forget. I forgot in an earlier draft of these notes and only noticed when I executed the example in *i*CCalc.

Here x ranges over the agents and p, p' over the locations:

inertial $loc(x)$
inertial $car(x)$
$walk(x) = p$ causes $loc(x) = p$
$drive(x) = p$ causes $loc(x) = p$
$drive(x) = p$ causes $car(x) = p$
nonexecutable $drive(x) = p \wedge walk(x) = p'$
nonexecutable $drive(x) = p$ if $loc(x) \neq car(x)$

Note that

(1) p and p' in these laws range over *locations* not 'destinations'.
(2) $drive(x) = p$ when $loc(x) = p$ is possible (in this example), and means that x drives around and ends up back where he/she started. And similarly for $walk(x)$.
(3) The condition $loc(x) \neq car(x)$ in the last line is valid syntax in CCalc and *i*CCalc. The last line is shorthand for the following $\mathcal{C}+$ laws:

nonexecutable $drive(x) = p$ if $loc(x) = p' \wedge \neg(car(x) = p')$ (for all locations p, p')

Example ('Yale Shooting Problem', again)

The 'Yale Shooting Problem' (YSP) is one of the classics in temporal reasoning in AI. The significance of the 'problem' (if it is a problem; not everyone agrees that it is) is that attempts to formalise it using a variety of general purpose non-monotonic reasoning formalisms failed to give an adequate representation. One loads a gun; waits; then shoots. Intuitively, the target should be dead (not alive) after this sequence. But various formalisations of the persistence (frame axiom/law of inertia) gave a surprising result: there was one model (extension, answer set, ...) in which the target was indeed not alive, but another unintended anomalous model (extension, answer set, ...) in which the gun was mysteriously no longer loaded after the wait, and so after the shooting, the target was still alive.

I don't want to get into details of whether this really is a problem or not, or what the diagnosis of the problem is (if it is a problem). What happens in $\mathcal{C}+$?

Here is the earlier $\mathcal{C}+$ action description.

Signature: Boolean fluent constants *loaded*, *alive*; Boolean action constants *load*, *shoot*, *wait*.

inertial *loaded*
inertial *alive*
load causes *loaded*
shoot causes $\neg$*alive* if *loaded*
shoot causes $\neg$*loaded*
nonexecutable *shoot* $\wedge$ *load*
nonexecutable *wait* $\wedge$ *shoot*
nonexecutable *wait* $\wedge$ *load*
$\bot$ after $\neg wait \wedge \neg shoot \wedge \neg load$

With this action description, any path of the transition system which has *load* at time 0, *wait* at time 1, and *shoot* at time 2, has *alive* false at time 3, just as expected. In this action description, the *wait* at time 1 does not mysteriously result in the gun becoming unloaded.

But suppose we *did* want to allow for this possibility? Suppose, for example, that *wait* could be an extremely long wait during which the gun could lose its ability to fire (and thus become 'unloaded'). How could we get this effect in $\mathcal{C}+$?

Answer: *wait* would then be an action with *non-deterministic* effects. It may but need not, result in $\neg$*loaded* after *wait*.

How to express this? Add another causal law:

wait may cause $\neg$*loaded*

Causal theories

The details here are *NOT EXAMINABLE*. They are provided for background for the action language $\mathcal{C}+$, and for general interest.

Nonmonotonic causal theories (or just 'causal theories' for short) is a general purpose non-monotonic representation formalism. A causal theory is a set of causal rules of the form

$$F \Leftarrow G$$

where F and G are formulas (defined on next page). Informally, this is to be read as saying that F is 'caused' if G is true (which is not the same as saying that G is the cause of F). We don't need to rely on this informal reading to use the formalism.

The main interest for us is that causal theories are intimately connected to the action language $\mathcal{C}+$ which we will be looking at later. You can think of them as a kind of stepping stone between $\mathcal{C}+$ and logic programs, which are our main interest.

Definite clausal theories A causal theory Γ is *definite* if

- the head of every rule of Γ is an atom or $\bot$, and
- no atom is the head of infinitely many rules of Γ.

Note that, as defined earlier, when p is a Boolean constant, $\neg p$ is shorthand for the atom $p = \mathsf{f}$ and so covered by the definition.

Translation to (classical) propositional logic

Definite causal theories (defined above) can be translated via the process of 'literal completion' into expressions of (classical) propositional logic. The process is analogous to the Clark completion for logic programs.

Literal completion

For a *definite* causal theory Γ, translate to set of (classical) formulas $comp(\Gamma)$:

$$\left.\begin{array}{c} F \Leftarrow G_1 \\ \vdots \\ F \Leftarrow G_n \end{array}\right\} \quad \text{becomes} \quad F \leftrightarrow G_1 \vee \cdots \vee G_n$$

If F is an atom and there are no causal rules with F as the head then $F \leftrightarrow \bot$ (which is logically equivalent to $\neg F$).

A causal rule $\bot \Leftarrow G$ becomes $\neg G$.

Models of Γ are the (classical) models of the formulas $comp(\Gamma)$.

Relationship to other formalisms

A causal theory of a Boolean signature can be viewed as a **Reiter default theory**.

Translate causal rule $F \Leftarrow G$ to the default rule $\dfrac{:G}{F}$.

More precisely: Let Γ be a causal theory of Boolean signature. Let $D(\Gamma)$ be the set of default rules obtained by translating every rule in Γ as described above.

An interpretation I is a model of Γ iff $th(I)$ is an extension of the default theory $(D(\Gamma), \emptyset)$. ($th(I)$ stands for the set of all formulas true in I.)

Extended logic programs (IMPORTANT)

Suppose that a causal theory Γ has a Boolean signature and is definite.

Every such causal theory can be written equivalently as a set of causal rules of the form

$$L \Leftarrow L_1 \wedge \cdots \wedge L_n \qquad (n \geq 0)$$

or

$$\bot \Leftarrow L_1 \wedge \cdots \wedge L_n \qquad (n \geq 0)$$

where L and every L_i is a Boolean atom, i.e., of the form c or $\neg c$ ($c = \mathsf{t}$ and $c = \mathsf{f}$). Translate each such rule to an extended logic programming clause:

$$L \leftarrow not\ \overline{L_1}, \ldots, not\ \overline{L_n}$$

where as usual $\overline{L_i}$ stands for the literal complementary to L_i. (Translate $L \Leftarrow \top$ to $L \leftarrow$.)

Translate

$$\bot \Leftarrow L_1 \wedge \cdots \wedge L_n \qquad (n \geq 0)$$

to the constraint

$$\leftarrow not\ \overline{L_1}, \ldots, not\ \overline{L_n}$$

Call this the program $lp(\Gamma)$.

A set of literals I that is an interpretation of Γ is a model of Γ iff I is an answer set of $lp(\Gamma)$.

Notice: the above does not say that every answer set of an extended logic program $lp(\Gamma)$ is a model of Γ. It says that every interpretation I – every consistent and complete set of atoms of the signature of Γ – is a model of Γ iff it is an answer set of $lp(\Gamma)$.

Defaults

In causal theories, a causal rule of the form

$$F \Leftarrow F$$

effectively says 'F holds by default'.

It should now be clear why. The causal rule $F \Leftarrow G$ is (nearly exactly) equivalent to the Reiter default rule

$$\frac{: F}{F}$$

(If it is consistent that F, then F, or 'F holds by default'.)

Consider the corresponding translation to (extended) logic programs. For an atom p,

$$p \Leftarrow p \quad \text{translates to} \quad p \leftarrow \mathit{not}\ \neg p$$

One last little note

The very observant might have noticed that the causal rule

$$L \Leftarrow L_1 \wedge \cdots \wedge L_n$$

translates to the Reiter default $\dfrac{: L_1 \wedge \cdots \wedge L_n}{L}$

Whereas the logic program clause

$$L \leftarrow \mathit{not}\ \overline{L}_1, \ldots, \mathit{not}\ \overline{L}_n$$

translates to the Reiter default $\dfrac{: L_1, \ldots, L_n}{L}$

These are not equivalent, in general. Compare, for example

$$\frac{: p \wedge q}{r} \quad \text{and} \quad \frac{: p, q}{r} \quad \text{when } (\neg p \vee \neg q)$$

But they *are* equivalent when causal rules have only *atoms* in the head, i.e., when they can be translated to a logic program.

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

For any action description D in $\mathcal{C}+$, and any non-negative integer m, it is possible to construct a causal theory Γ^D_m such that the models of Γ^D_m correspond to the paths of length m of the transition system defined by D. The language $\mathcal{C}+$ can thus be regarded as a higher-level notation for defining causal theories of a particular kind, and indeed this is exactly as it is presented in [3].

The translation is obtained by time-stamping every fluent and action atom with a non-negative integer, just as we did for the time-stamped query language earlier:

$f[i] = v$ represents that fluent $f = v$ holds at integer time i, or more precisely, that $f = v$ holds in the ith state of a history (path) of the transition system.

$a[i] = v$ represents that action atom $a = v$ is satisfied by the transition from the ith state of the history (path) to the $i{+}1$th state.

For any formula ψ, $\psi[i]$ stands for the result of time-stamping all fluent and action constants in ψ with i. For example: $(p \vee \neg q)[i]$ is shorthand for $p[i] \vee \neg q[i]$, that is, $p[i] = \mathsf{t} \vee q[i] = \mathsf{f}$.

Given an action description D, the causal theory Γ^D_m is constructed as follows.

- Static law
  $$\textsf{caused } F \textsf{ if } G \quad \mapsto \quad F[i] \Leftarrow G[i] \quad (i \in 0 \mathinner{..} m)$$
- Fluent dynamic law
  $$\textsf{caused } F \textsf{ if } G \textsf{ after } \psi \quad \mapsto \quad F[i{+}1] \Leftarrow G[i{+}1] \wedge \psi[i] \quad (\ i \in 0 \mathinner{..} m{-}1)$$
- Action dynamic law
  $$\textsf{caused } \alpha \textsf{ if } \psi \quad \mapsto \quad \alpha[i] \Leftarrow \psi[i] \quad (\ i \in 0 \mathinner{..} m{-}1)$$

We also require the following **exogeneity laws**:

- For every *simple* fluent constant f and every $v \in \mathit{dom}(f)$:
  $$f[0] = v \Leftarrow f[0] = v$$
- For every action constant a, every $v \in \mathit{dom}(a)$:
  $$a[i] = v \Leftarrow a[i] = v \qquad (i \in 0 \mathinner{..} m{-}1)$$

Look *very carefully* at the range of the time index i in all of the above causal laws.

The exogeneity laws are necessary. Why? Because to get a model of the causal theory Γ^D_m (and a representation of the transition system defined by D) we must have a consistent and *complete* valuation for every fluent constant in every state, and for every action constant at every transition. The exogeneity laws ensure this — assuming that if we give a valuation for the (simple) fluent constants in the initial state the fluent dynamic laws will ensure they get a valuation in every subsequent state. The fluent dynamic laws must be written accordingly.

Note that there are no exogeneity laws for *statically determined fluent constants* — their values are determined by the values of the simple fluents. Statically determined fluents must be defined accordingly: by means of 'default' statements if necessary.

For illustration, here are the translated forms of the most commonly used abbreviations of $\mathcal{C}+$:

default F	$F[i] \Leftarrow F[i]$
default F if G	$F[i] \Leftarrow F[i] \wedge G[i]$
inertial f	$f[i{+}1]{=}v \Leftarrow f[i{+}1]{=}v \wedge f[i]{=}v$ for all $v \in dom(f)$
α causes G	$G[i{+}1] \Leftarrow \alpha[i]$
α causes G if ψ	$G[i{+}1] \Leftarrow \alpha[i] \wedge \psi[i]$
nonexecutable α	$\bot \Leftarrow \alpha[i]$
nonexecutable α if ψ	$\bot \Leftarrow \alpha[i] \wedge \psi[i]$
α may cause G	$G[i{+}1] \Leftarrow G[i{+}1] \wedge \alpha[i]$
α may cause G if ψ	$G[i{+}1] \Leftarrow G[i{+}1] \wedge \alpha[i] \wedge \psi[i]$

Clearly any interpretation X of the signature of Γ^D_m can be written in the form

$$s_0[0] \cup \varepsilon_0[0] \cup \cdots \cup \varepsilon_{m-1}[m{-}1] \cup s_m[m]$$

where $s_0, \ldots, s_m$ are interpretations of σ^{f} and $\varepsilon_0, \ldots, \varepsilon_{m-1}$ are interpretations of σ^{a}.

So, here is the *key* point …

$$\text{Models of } \Gamma^D_m \quad \overset{1-1}{\Longleftrightarrow} \quad \text{paths/histories of length } m \text{ in } D$$

In particular, Γ^D_1 represents paths of length 1 of D, i.e., the *transitions* of the transition system described by D.

Γ^D_0 represents paths of length 0, i.e., the *states* of the transition system described by D.

Example

toggle causes *on* if $\neg$*on*
toggle causes $\neg$*on* if *on*

load causes *loaded*

inertial *on*
inertial *loaded*

$on[i{+}1] \leftrightarrow (toggle[i] \wedge \neg on[i]) \vee (on[i{+}1] \wedge on[i])$
$\neg on[i{+}1] \leftrightarrow (toggle[i] \wedge on[i]) \vee (\neg on[i{+}1] \wedge \neg on[i])$
$loaded[i{+}1] \leftrightarrow load[i] \vee (loaded[i{+}1] \wedge loaded[i])$
$\neg loaded[i{+}1] \leftrightarrow \neg loaded[i{+}1] \wedge \neg loaded[i]$

$$\left.\begin{array}{l} on[0] \leftrightarrow on[0] \\ \neg on[0] \leftrightarrow \neg on[0] \\ loaded[0] \leftrightarrow loaded[0] \\ \neg loaded[0] \leftrightarrow \neg loaded[0] \\ toggle[i] \leftrightarrow toggle[i] \\ \neg toggle[i] \leftrightarrow \neg toggle[i] \\ load[i] \leftrightarrow load[i] \\ \neg load[i] \leftrightarrow \neg load[i] \end{array}\right\} \quad (**)$$

on if $\top$ after *toggle* $\wedge \neg$*on*
$\neg$*on* if $\top$ after *toggle* $\wedge$ *on*

loaded if $\top$ after *load*

on if *on* after *on*
$\neg$*on* if $\neg$*on* after $\neg$*on*
loaded if *loaded* after *loaded*
$\neg$*loaded* if $\neg$*loaded* after $\neg$*loaded*

$on[i{+}1] \Leftarrow toggle[i] \wedge \neg on[i]$
$\neg on[i{+}1] \Leftarrow toggle[i] \wedge on[i]$

$loaded[i{+}1] \Leftarrow load[i]$

$on[i{+}1] \Leftarrow on[i{+}1] \wedge on[i]$
$\neg on[i{+}1] \Leftarrow \neg on[i{+}1] \wedge \neg on[i]$
$loaded[i{+}1] \Leftarrow loaded[i{+}1] \wedge loaded[i]$
$\neg loaded[i{+}1] \Leftarrow \neg loaded[i{+}1] \wedge \neg loaded[i]$

$$\left.\begin{array}{l} on[0] \Leftarrow on[0] \\ \neg on[0] \Leftarrow \neg on[0] \\ loaded[0] \Leftarrow loaded[0] \\ \neg loaded[0] \Leftarrow \neg loaded[0] \\ toggle[i] \Leftarrow toggle[i] \\ \neg toggle[i] \Leftarrow \neg toggle[i] \\ load[i] \Leftarrow load[i] \\ \neg load[i] \Leftarrow \neg load[i] \end{array}\right\} \quad (*)$$

Note the introduction of the 'exogeneity laws' (*) in the second step.

Some versions of $\mathcal{C}+$ require an *explicit* declaration

exogenous a

for every (exogenous) action constant a. Perhaps this is better because then one does not forget.

Notice that all the formulas (**) in the completion resulting from the exogeneity laws (*) are tautologies (always true) and so, since they are trivially satisfied, they can be deleted from the completion before it is passed to the sat-solver as an obvious optimisation step. And that is exactly what happens in practice.

But that does *not* mean that the exogeneity laws (*) themselves are unnecessary. See what happens if they are omitted. In that case the completion would contain (by definition)

$$on[0] \leftrightarrow \bot, \quad \neg on[0] \leftrightarrow \bot$$

i.e., both $\neg on[0]$ and $on[0]$. And similarly for all the other exogeneity laws. The completion would obviously be unsatisfiable.

The 'Causal Calculator'

$\mathcal{C}+$ is a language for defining transition systems. That's all. Other languages can be interpreted on these structures:

- temporal
- epistemic (cf. 'interpreted systems')

  ⋮
- narratives and planning
  - e.g. as supported by the 'causal calculator' CCALC

Given an action description D of signature $(\sigma^{\mathrm{f}}, \sigma^{\mathrm{a}})$, non-negative integer m, and query ψ of the time-stamped signature σ_m, CCALC

- performs the translation of D to Γ^D_m,
- constructs $comp(\Gamma^D_m)$,
- invokes a standard propositional sat-solver to find (classical) models of $comp(\Gamma^D_m) \cup \psi$, and then
- post-processes the sat-solver output to show the models obtained.

In practice, the first two steps may be combined into one, possibly with some additional optimisations to simplify the set of formulas passed to the sat-solver.

In addition, CCALC provides

- a language for specifying the action signature (sorts, variables, various shorthand notations)
- a language for asserting narratives and for expressing common forms of queries.

I won't show the details. The query language in particular is very ugly.

Translation to logic programs (ASP)

This is our main interest. Think of causal theories as an (optional) stepping stone in the translation to logic programs.

The translation for boolean signatures is very straightforward. The details for multi-valued signatures are a bit fiddly.

The details are in a separate set of notes ('Addendum').

References

[1] V. Akman, S. T. Erdoğan, J. Lee, V. Lifschitz, and H. Turner. Representing the Zoo World and the Traffic World in the language of the Causal Calculator. *Artificial Intelligence*, 153:105–140, 2004.

[2] Michael Gelfond and Vladimir Lifschitz. Action Languages. *Electronic Transactions on AI*, 3(16), 1998. `http://www.ep.lin.se/ea/cis/1998/016`.

[3] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153:49–104, 2004.

[4] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, and Hudson Turner. Causal laws and multi-valued fluents. In *Proc. of the Fourth Workshop on Nonmonotonic Reasoning, Action, and Change, Seattle*, August 2001.

[5] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630. AAAI Press, 1998.

[6] Enrico Giunchiglia and Vladimir Lifschitz. Action languages, temporal action logics, and the situation calculus. In *Working Notes of the IJCAI-99 Workshop on Nonmonotonic Reasoning, Action, and Change*, 1999.

[7] Vladimir Lifschitz. On the logic of causal explanation. *Artificial Intelligence*, 96:451–465, 1997.

[8] Vladimir Lifschitz and Hudson Turner. Representing transition systems by logic programs. In *Proc. Fifth International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 92–106, 1999.

[9] Norman McCain and Hudson Turner. Causal theories of action and change. In *Proc. AAAI-97*, pages 460–465. AAAI Press, 1997.

[10] Hudson Turner. A logic of universal causation. *Artificial Intelligence*, 113:87–123, 1999.