Background  The language $C$ 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]. A summary of $C$ is included in the survey article by Gelfond and Lifschitz [2]. [6] discusses relationships to situation calculus. $C^+$ extends $C$ by allowing multi-valued fluents as well as boolean fluents [4] and generalises the form of the rules in language $C$ in various ways. The definitive presentation of $C^+$, including various further extensions, is provided in [3]. A companion paper [1] shows how $C^+$ 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)\(^1\).

The language $C^+$ 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 a translation of a $C^+$ 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). It performs the required translations and then invokes a standard propositional satisfaction solver. An alternative implementation route is provided by translations into extended logic programs [8]. We have our own implementation of CCalc which supports both implementation routes. The method via logic programs and answer sets works better and is much faster. That is the one that will be emphasised here. (Rob Craven developed another translation into logic programs with a different computational behaviour ($EC^+$ 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 $\langle S, A, R \rangle$ in which

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

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 $\{ R_c \}_{c \in A}$. The former is chosen here for consistency with published accounts of the language $C^+$.

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 $(s, \varepsilon, s')$ 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 $(s, \varepsilon, s')$ represents execution of event $\varepsilon$ in state $s$ leading (possibly non-deterministically) to the state $s'$.

An event $\varepsilon$ is executable in $s$ when there is at least one tuple $(s, \varepsilon, s')$ in $R$.

An event $\varepsilon$ 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 ($\omega$ 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 $C^+$ in [3] uses the term ‘history’.)

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

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

such that $s_0, s_1, \ldots, s_m \in S$, $\varepsilon_0, \ldots, \varepsilon_{m-1} \in A$, 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.
We begin with states. Let $\sigma'$ be a multi-valued signature of constants called 'state variables', or more usually in AI terminology, fluent constants. Given a labelled transition system $(S, A, R)$ we add a valuation function which specifies, for every fluent constant $f \in \sigma'$ and every state $s \in S$, a value in $\text{dom}(f)$. We shall be dealing with the special case of transition systems in which

- each state $s \in S$ is an interpretation of $\sigma'$, $S \subseteq I(\sigma')$.

The valuation function for fluent constants is then redundant: the state already specifies the value of each fluent constant in that state. It is often convenient to adopt the convention that an interpretation $I$ of $\sigma'$ is represented by the set of atoms of $\sigma'$ that are satisfied by $I$. A state is then a (complete, and consistent) set of fluent atoms. We sometimes say a formula $\varphi$ ‘holds in’ state $s$ or ‘is true in’ state $s$ as alternative ways of saying that $s$ satisfies $\varphi$.

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

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

Moreover, since a transition label is an interpretation of the action constants $\sigma^a$, 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. $a = f$, can be read as indicating that action $a$ is not performed; and where all action constants are Boolean, the action $\{a_1 = f, \ldots, a_n = f\}$ 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 $\text{move}(a)$, $\text{move}(b)$, $\text{move}(c)$ with domains $\{E, W, N, S, \text{idle}\}$, and Boolean action constants $\text{whistle}(a)$, $\text{whistle}(b)$, $\text{whistle}(c)$. Then one possible interpretation of the action signature, and therefore one possible transition label, is

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

Because of the way that action formulas are evaluated on a transition $(s, \varepsilon, s')$, an action formula can also be regarded as expressing a property of the transition $(s, \varepsilon, s')$ as a whole. We won’t bother with that reading in these notes.

**Example** Let $\sigma'$ be the set of fluent constants $\{\text{loc}(a), \text{loc}(b)\}$ with possible values $\{N, S\}$, and let $\sigma^a$ be the set of Boolean action constants $\{\text{go}(a), \text{go}(b)\}$. Consider the transition system $T$ depicted in the following diagram:

There is no state $\{\text{loc}(a) = N, \text{loc}(b) = N\}$ in $T$ (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. One candidate 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 $s_0, s_1, s_2, \ldots, s_i, s_{i+1}, \ldots$ of the transition system; the time-stamped atom $a[i] = v$ represents that action atom $a = v$ is satisfied by the transition label $\varepsilon_i$ of a path $s_0, s_1, s_2, \ldots, s_i, \varepsilon_i, s_{i+1}, \ldots$.

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

In general, given a multi-valued signature $\sigma$ and a non-negative integer $i$, we write $\sigma[i]$ for the signature consisting of all constants of the form $e[i]$ where $c$ is a constant of $\sigma$, with $dom(e[i]) = dom(c)$. For any non-negative integer $m$, we write $\sigma_m$ for the signature $\sigma[0] \cup \cdots \cup \sigma[m]$. The time-stamped query language used in CICAL to express properties of paths of length $m$ of a transition system $T$ of action signature $(\sigma^t, \sigma^s)$ is the propositional language of signature $\sigma^t_m \cup \sigma^s_m$. In other words, the formulas of this query language are:

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

Let $\pi = s_0 \cdot e_0 \cdot s_1 \ldots \cdot e_i \cdot s_{i+1} \ldots$ be a path of length $m$ of a transition system $T$ of action signature $(\sigma^t, \sigma^s)$. An atom $f[i] = v$ for any fluent constant $f$ of $\sigma^t$ and $0 \leq i \leq m$ is true on path $\pi$ (or ‘holds on’ path $\pi$, or ‘is satisfied by’ path $\pi$), written $T, \pi \models_m f[i] = v$, when $s_i \models f = v$; for action constants $a$ of $\sigma^s$ and $0 \leq i < m$, $T, \pi \models_m a[i] = v$ when $e_i \models a = v$; and $T, \pi \models_m$ is extended to formulas $\varphi$ of signature $\sigma_m = \sigma^t_m \cup \sigma^s_{m-1}$ by the usual truth tables for the propositional connectives.

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

Equivalently, 

Let $\pi[i]$ denote the $i$th component of a path $\pi$: that is, when $\pi = s_0 \cdot e_0 \cdot s_1 \ldots \cdot e_i \cdot s_{i+1} \ldots$, let $\pi[i] = s_i \cup e_i$. Clearly, $\pi[i]$ is an interpretation of $\sigma^t_m \cup \sigma^s_{m-1}$ when $\pi$ is a path of a transition system with action signature $(\sigma^t, \sigma^s)$.

For any formula $\psi$ of signature $\sigma^t_m \cup \sigma^s_{m-1}$, let $\psi[i]$ stand for the formula of signature $\sigma^t[i] \cup \sigma^s[i]$ obtained by time-stamping every constant in $\psi$ with $i$, that is, replacing every constant $c$ in $\psi$ by the constant $c[i]$. Clearly, every formula $\varphi$ of signature $\sigma_m = \sigma^t_m \cup \sigma^s_{m-1}$ is a truth-functional compound of formulas of the form $\psi[i]$ where $0 \leq i \leq m$ and $\psi$ is a formula of signature $\sigma = \sigma^t \cup \sigma^s$.

Now, for any path $\pi$ of length $m$ of a transition system $T$ of action signature $(\sigma^t, \sigma^s)$, we have

$$T, \pi \models_m \psi[i] \iff \pi[i] \models \psi$$

Example (contd) Consider again the transition system $T$:

- $\{\neg go(a), \neg go(b)\}$
- $\{go(a), go(b)\}$
- $\{\neg go(a), \neg go(b)\}$
- $\{go(a), go(b)\}$
- $\{\neg go(a), \neg go(b)\}$
- $\{go(a), go(b)\}$
- $\{\neg go(a), \neg go(b)\}$
- $\{go(a), go(b)\}$
- $\{\neg go(a), \neg go(b)\}$

We have, amongst other things:

- $T \models_1 (\text{loc}(a)[0] = N \land \text{go}(a)[0]) \rightarrow \text{loc}(a)[1] = S$
- $T \models_2 (\text{loc}(a)[0] = N \land \text{go}(a)[0] \land \text{go}(a)[1]) \rightarrow \text{loc}(a)[2] = N$
- $T \models_1 (\text{loc}(a)[0] = N \land \neg \text{go}(a)[0]) \rightarrow \text{loc}(a)[1] = N$
- $T \models_2 (\text{loc}(a)[0] = N \land \text{go}(b)[0] \land \text{go}(a)[1]) \rightarrow \text{loc}(a)[2] = N$
- $T \models_m (\text{loc}(a)[i] = N \land \text{loc}(a)[i+2] = N) \rightarrow (\text{go}(a)[i] \leftrightarrow \text{go}(a)[i+1])$ for all $0 \leq i \leq m-2$
- $T \models_m (\text{loc}(a)[i] = S \land \text{loc}(b)[i] = S) \rightarrow (\neg \text{go}(a)[i] \land \text{go}(b)[i])$ for all $0 \leq i \leq m-1$

(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 $T$ and a query language of signature $(\sigma^f, \sigma^a)$:

- 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 $\sigma^f$ and $\alpha_i$ are formulas of $\sigma^a$.

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

Or in other words is it the case that

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

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 $m$?

We want to know: is there a possible run/path $\pi$ through the transition system $T$ such that

$$T, \pi \models_m F[0] \land \alpha_0[0] \land \alpha_1[1] \land \ldots \land \alpha_{m-1}[m-1] \land G[m]$$

**‘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

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

And as before, checking whether there is a possible path/run $\pi$ of $T$ such that

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

deals with the variant of the problem in which we want to know whether it is possible that initially $F$.

**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) $\epsilon_0, \epsilon_1, \ldots, \epsilon_{k-1}$ such that there is a path/run $s_0 \epsilon_0 s_1 \ldots s_{k-1} \epsilon_{k-1} s_k$ of $T$ 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 $\pi$ of $T$ such that:

$$T, \pi \models F[0] \land G[k]$$

If there is such a path $\pi = s_0 \epsilon_0 s_1 \ldots s_{k-1} \epsilon_{k-1} s_k$ then it contains a representation of the plan: $\epsilon_0, \epsilon_1, \ldots, \epsilon_{k-1}$.

(The reference here is to events $\epsilon_i$ because we want fully specified actions in the plan. In the other problems, a formula $\alpha_i$ represents a partially specified event/transition.)

**But note** This is often called planning in the literature but it’s not really planning. There is more to planning that just finding a suitable sequence of events $\epsilon_0, \epsilon_1, \ldots, \epsilon_{k-1}$ that gets us from the initial state to the goal state. For instance, some of these $\epsilon_i$ 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 $\epsilon_i$ 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 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. In principle it could also be solved, by iterating various tests as in the planning problem to cover in some systematic fashion single gaps of one missing event, single gaps of $j$ missing events, multiple gaps of single and $j$ missing events, and so on, and so on, in all combinations. Clearly this method is not feasible without some further restrictions on possible gaps because of the (infinitely many) combinations to be considered.
The Action Description Language C+

The language 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 C+ is a set of C+ laws that define a transition system of a certain kind.

Syntax

An action signature is a (non-empty) set $\sigma^f$ of fluent constants and a (non-empty) set $\sigma^a$ of action constants.

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

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.

The expressions of C+:

- **Static laws**
  
  $\text{caused } F \text{ if } G$

  where $F$ and $G$ are fluent formulas (i.e., formulas of signature $\sigma^f$).

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

- **Fluent dynamic laws**
  
  $\text{caused } F \text{ if } G \text{ after } \psi$

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

  Informally, in a transition $(s, \varepsilon, s')$, formulas $F$ and $G$ are evaluated at $s'$ (the resulting state), fluent atoms in $\psi$ are evaluated at $s$ (i.e., in the state immediately before the transition), and action atoms in $\psi$ are evaluated on the transition $\varepsilon$ 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 set of fluent dynamic laws can be written equivalently as laws of the form

$\text{caused } F \text{ if } G \text{ after } H \land \alpha$

where $H$ is a fluent formula (no action constants) and $\alpha$ is an action formula (no fluent constants).

- **Action dynamic laws**

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

  where $\alpha$ is an action formula (i.e., a formula of signature $\sigma^a$) and $\psi$ is any formula of signature $\sigma^f \cup \sigma^a$.

  Two special cases:

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

  (Every transition/event of type $\beta$ is also a transition/event of type $\alpha$.)

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

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

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

Various abbreviations

- $\alpha$ causes $F$ if $G = F$ if $\top$ after $\alpha \land G$
- nonexecutable $\alpha$ if $\psi = \bot$ if $\top$ after $\alpha \land \psi$ ($= \alpha$ causes $\bot$ if $\psi$)
- default $F = F$ if $F$ (why ‘default’? — explained later)

Default persistence (‘inertia’) of fluents is not a built-in feature of the C+ language. One specifies explicitly which fluents are ‘inertial’ by means of a C+ law of the form

$\text{inertial } f$

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

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

How this form of rule works to express default persistence of $f = v$ will become clearer when we look at the semantics of C+ laws.
Definite action descriptions are the ones of most practical significance.

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:

\[
\begin{align*}
toggle &\text{ causes on if } \neg \text{on} \\
toggle &\text{ causes } \neg \text{on if on}
\end{align*}
\]

These are shortand for the following fluent dynamic laws:

\[
\begin{align*}
on &\text{ if } T \after toggle \land \neg \text{on} \\
\neg \text{on} &\text{ if } T \after toggle \land \text{on}
\end{align*}
\]

Or, instead of having a Boolean fluent on one might prefer to have a fluent constant light with values on and off.

The effects of the toggle action would then be expressed:

\[
\begin{align*}
toggle &\text{ causes light }= \text{on } \text{if } \text{light }= \text{off} \\
toggle &\text{ causes light }= \text{off } \text{if } \text{light }= \text{on}
\end{align*}
\]

Abbreviations

The language $\mathcal{C}^+$ provides various abbreviations, such as causes, inertial, and nonexecutable. Here are the most common. (We won’t bother with the full list.)

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>default $F$</td>
<td>$F$ if $F$</td>
</tr>
<tr>
<td>default $F$ if $G$</td>
<td>$F$ if $F \land G$</td>
</tr>
<tr>
<td>inertial $f$</td>
<td>$f = v$ if $f = v$ after $f = v$ for all $v \in \text{dom}(f)$</td>
</tr>
<tr>
<td>$\alpha$ causes $G$</td>
<td>$G$ if $T$ after $\alpha$</td>
</tr>
<tr>
<td>$\alpha$ causes $G$ if $\psi$</td>
<td>$G$ if $T$ after $\alpha \land \psi$ (or: $\alpha$ causes $\bot$)</td>
</tr>
<tr>
<td>nonexecutable $\alpha$ if $\psi$</td>
<td>$\bot$ if $T$ after $\alpha \land \psi$ (or: $\alpha$ causes $\bot$ if $\psi$)</td>
</tr>
<tr>
<td>$\alpha$ may cause $G$</td>
<td>$G$ if $G$ after $\alpha$</td>
</tr>
<tr>
<td>$\alpha$ may cause $G$ if $\psi$</td>
<td>$G$ if $G$ after $\alpha \land \psi$</td>
</tr>
</tbody>
</table>

The last of these, may cause, is used for specifying the effects of non-deterministic actions. (Examples later.)

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

Semantics

(The rationale behind these definitions is far from obvious. It should make sense if you remember that the language $\mathcal{C}^+$ has its roots in the formalism of ‘nonmonotonic causal theories’. It is NOT NECESSARY to memorize the definitions in this section. It is not even necessary to read them.)

Consider an action description $D$.

- A state is an interpretation of $\sigma^I$ (the fluent constants) that satisfies $G \rightarrow F$ for every static law $F$ if $G$ in $D$.
- A transition label (or event) is an interpretation of $\sigma^A$ (the action constants).
- A transition is a triple $(s, \varepsilon, s')$ where $s$ and $s'$ are states and $\varepsilon$ is a transition label/event. $s$ is the initial state of the transition and $s'$ is the resulting state. A transition defined by an action description $D$ must satisfy the following additional constraints.

\[
\begin{align*}
T_{static}(s) &= \{ F \mid F \text{ if } G \in D \text{ if } s \models G \} \\
E(s, \varepsilon, s') &= \{ F \mid F \text{ if } G \text{ after } \psi \text{ is in } D, s' \models G, s \cup \varepsilon \models \psi \}
\end{align*}
\]

Definite action descriptions are the ones of interest.

When $D$ is definite, $(s, \varepsilon, s')$ is a transition defined by $D$ (or simply: a transition of $D$) iff $s \in S$ and

- $s = T_{static}(s)$
- $s' = T_{static}(s') \cup E(s, \varepsilon, s')$

Definite action descriptions are the ones of most practical significance.
Example (first, one without any static laws)

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

In the diagram, transition labels 'load' and 'toggle' are shorthand for \{load, ¬toggle\} and \{¬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 \{¬load, ¬toggle\} (‘null’ event).

Here, the label \( \text{it} \) is shorthand for \{load, toggle\} and \( \text{null} \) is shorthand for the ‘null’ event \{¬load, ¬toggle\}.

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

\[ \bot \text{ if } \top \text{ after } \neg \text{load} \land \neg \text{toggle} \]

for which there is a standard abbreviation in C+:

\[ \text{nonexecutable } \neg \text{load} \land \neg \text{toggle} \]

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

\[ \text{nonexecutable } \text{load} \land \text{toggle} \]

Example (‘Yale Shooting Problem’)

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

(\(\star\)) 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 \land 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 \{¬shoot, ¬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.

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.

Because of the static laws, there are only four states in the transition system and not $2^3 = 8$. The diagram does not show the transitions with labels \{toggle, win, ¬lose\} and \{toggle, ¬win, lose\}.

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, ¬lose\} and \{toggle, ¬win, lose\}.

Because of the static laws, there are only four states in the transition system and not $2^3 = 8$. The diagram does not show the transitions with labels \{toggle, win, ¬lose\} and \{toggle, ¬win, lose\}.

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 rich, happy, on; Boolean action constants win, lose, toggle.
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

$caused happy$ if $rich$

eliminates certain combinations.

The language $C+$ 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 (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

- inertial $AtWork$
- inertial $CarInGarage$
- $go$ causes $AtWork$
- nonexecutable $go$ 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:

$go$ may cause $\neg CarInGarage$ if $CarInGarage$

This is an abbreviation for the dynamic law

$\neg CarInGarage$ if $\neg CarInGarage$ after $CarInGarage \land go$

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

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.)
Alternatively, we could distinguish between walking to work and driving to work. Let us have two Boolean action constants \( \text{walk} \) and \( \text{drive} \) to represent walking and driving to work respectively. The action description

\[
\begin{align*}
\text{inertial } & \text{AtWork} \\
\text{walk } & \text{causes } \text{AtWork} \\
\text{drive } & \text{causes } \text{AtWork} \\
\text{drive } & \text{causes } \neg \text{CarInGarage} \text{ if } \text{CarInGarage} \\
\text{nonexecutable } & \text{walk } \text{if } \text{AtWork} \\
\text{nonexecutable } & \text{drive } \text{if } \text{AtWork} \\
\text{nonexecutable } & \text{drive } \text{if } \neg \text{CarInGarage} \\
\end{align*}
\]

defines the following transition system (‘null’ events \( \{\neg \text{walk}, \neg \text{drive}\} \) omitted):

\[
\begin{array}{c}
\neg \text{AtWork} \\
\neg \text{CarInGarage} \\
\text{AtWork} \\
\text{CarInGarage} \\
\end{array}
\quad
\begin{array}{c}
\neg \text{walk} \\
\text{walk} \\
\neg \text{drive} \\
\text{drive} \\
\end{array}
\quad
\begin{array}{c}
\text{AtWork} \\
\text{CarInGarage} \\
\neg \text{raining} \\
\text{raining} \\
\end{array}
\]

The first two \( \text{causes} \) laws could be replaced by the (equivalent) law: \( (\text{walk} \lor \text{drive}) \text{ causes } \text{AtWork} \)

We could also represent that \( \text{walk} \) and \( \text{drive} \) are both kinds of \( \text{go} \) by means of action dynamic laws:

\[
\begin{align*}
\text{caused } \text{go } & \text{if } \text{walk} \\
\text{caused } \text{go } & \text{if } \text{drive}
\end{align*}
\]

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

\[
\begin{align*}
\text{nonexecutable } & \text{walk } \land \neg \text{go} \quad (\text{walk } \land \neg \text{go } \text{causes } \bot) \\
\text{nonexecutable } & \text{drive } \land \neg \text{go} \quad (\text{drive } \land \neg \text{go } \text{causes } \bot)
\end{align*}
\]

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

\[
\text{nonexecutable } \text{go } \land \neg \text{walk } \land \neg \text{drive}
\]

This would not change the form of the transition system shown above except to replace transition labels ‘walk’ and ‘drive’ by \( \{\text{go}, \text{walk}\} \) and \( \{\text{go}, \text{drive}\} \) respectively.

Notice that the transition label \( \{\text{go}, \text{walk}\} \) cannot distinguish between two concurrent but unrelated actions \( \text{go} \) and \( \text{walk} \) 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 \( \text{raining} \). We express that \( \text{raining} \) is exogenous by adding the following pair of static laws:

\[
\begin{align*}
\text{raining } & \text{if } \text{raining} \\
\neg \text{raining } & \text{if } \neg \text{raining}
\end{align*}
\]

Here is a fragment of the transition system obtained:

\[
\begin{array}{c}
\neg \text{AtWork} \\
\neg \text{CarInGarage} \\
\text{AtWork} \\
\text{CarInGarage} \\
\text{raining} \\
\neg \text{raining} \\
\text{walk} \\
\text{drive} \\
\text{AtWork} \\
\text{CarInGarage} \\
\neg \text{raining} \\
\text{walk} \\
\text{drive} \\
\end{array}
\]

The pair of static laws for \( \text{raining} \) above may also be written more concisely in \( \mathcal{C}^+ \) as:

\[
\text{exogenous } \text{raining}
\]

In general, for a fluent constant \( f \), the abbreviation

\[
\text{exogenous } f
\]

stands for the set of static laws \( f = v \text{ if } f = v, \) for every \( v \in \text{dom}(f) \).
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:
\[ \text{dom}(\text{walk}(x)) = \text{dom}(\text{drive}(x)) = \{\text{home, work, pub, none}\}. \]

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 iCCALC.

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 ∧ walk(x) = p’
- nonexecutable drive(x) = p if loc(x) ≠ 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) ≠ car(x) in the last line is valid syntax in CCALC and iCCALC.

The last line is shorthand for the following C+ laws:

\[ \text{nonexecutable } \text{drive}(x) = p \text{ if } \text{loc}(x) = p’ ∧ \neg(\text{car}(x) = p’). \]

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 C+?

Here is the earlier C+ action description.

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

\[
\begin{align*}
&\text{inertial loaded} \\
&\text{inertial alive} \\
&\text{load causes loaded} \\
&\text{shoot causes } \neg \text{alive if loaded} \\
&\text{shoot causes } \neg \text{loaded} \\
&\text{nonexecutable shoot } \land \text{ load} \\
&\text{nonexecutable wait } \land \text{ shoot} \\
&\text{nonexecutable wait } \land \text{ load} \\
&\bot \text{ after } \neg \text{wait } \land \neg \text{ shoot } \land \neg \text{ load}
\end{align*}
\]

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 C+?

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

How to express this? Add another causal law:

\[ \text{wait may cause } \neg \text{ loaded} \]
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 $\Gamma^D_m$ such that the models of $\Gamma^D_m$ 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 $i$th 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 $i$th state of the history (path) to the $(i+1)$th state.

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

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

- **Static law**
  
  caused $F$ if $G$ \quad $\implies$ \quad $F[i] \Leftarrow G[i]$ \quad ($i \in 0 \ldots m$)

- **Fluent dynamic law**
  
  caused $F$ if $G$ after $\psi$ \quad $\implies$ \quad $F[i+1] \Leftarrow G[i+1] \land \psi[i]$ \quad ($i \in 0 \ldots m-1$)

- **Action dynamic law**
  
  caused $a$ if $\psi$ \quad $\implies$ \quad $a[i] \Leftarrow \psi[i]$ \quad ($i \in 0 \ldots m-1$)

We also require the following exogeneity laws:

- For every *simple* fluent constant $f$ and every $v \in \text{dom}(f)$:
  
  $f[0] = v \iff f[0] = v$

- For every action constant $a$, every $v \in \text{dom}(a)$:
  
  $a[i] = v \iff a[i] = v$ \quad ($i \in 0 \ldots m-1$)

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

The exogeneity laws are necessary because without them there could be no models of the causal theory $\Gamma^D_0$.

Why? Because if $c$ is a constant in the signature of some causal theory $\Gamma$, and there are no rules $c = v \iff \ldots$ in $\Gamma$, then $\Gamma$ has no models. (See comment in the background notes on causal theories and Question 1(1) in the associated tutorial exercises.)

Note that there are no exogeneity laws for *statically determined fluent constants* — their values are determined by the values of the simple fluents.

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

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>default $F$</td>
<td>$F[i] \Leftarrow F[i]$</td>
</tr>
<tr>
<td>default $F$ if $G$</td>
<td>$F[i] \Leftarrow F[i] \land G[i]$</td>
</tr>
<tr>
<td>inertial $f$</td>
<td>$f[i+1] = v \iff F[i+1] = v \land F[i] = v$ \quad for all $v \in \text{dom}(f)$</td>
</tr>
<tr>
<td>$\alpha$ causes $G$</td>
<td>$G[i+1] \Leftarrow \alpha[i] \land \psi[i]$</td>
</tr>
<tr>
<td>$\alpha$ may cause $G$ if $\psi$</td>
<td>$G[i+1] \Leftarrow \alpha[i] \land \psi[i]$</td>
</tr>
<tr>
<td>nonexecutable $\alpha$</td>
<td>$\bot \Leftarrow \alpha[i]$</td>
</tr>
<tr>
<td>nonexecutable $\alpha$ if $\psi$</td>
<td>$\bot \Leftarrow \alpha[i] \land \psi[i]$</td>
</tr>
</tbody>
</table>

Clearly any interpretation $X$ of the signature of $\Gamma^D_m$ can be written in the form

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

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

So, here is the key point . . .

Models of $\Gamma^D_m$ \quad \iff \quad paths/histories of length $m$ in $D$

In particular, $\Gamma^D_0$ represents paths of length 1 of $D$, i.e., the *transitions* of the transition system described by $D$.

$\Gamma^D_0$ represents paths of length 0, i.e., the *states* of the transition system described by $D$.
Example

\[
on[i+1] \leftrightarrow (\text{toggle}[i] \land \neg\text{on}[i]) \lor (\text{on}[i+1] \land \text{on}[i])
\]
\[
\neg\text{on}[i+1] \leftrightarrow (\text{toggle}[i] \land \text{on}[i]) \lor (\neg\text{on}[i+1] \land \neg\text{on}[i])
\]
\[
\text{loaded}[i+1] \leftrightarrow \text{load}[i] \lor (\text{loaded}[i+1] \land \text{loaded}[i])
\]
\[
\neg\text{loaded}[i+1] \leftrightarrow \neg\text{loaded}[i] \lor (\neg\text{loaded}[i+1] \land \neg\text{loaded}[i])
\]
\[
\text{on}[0] \leftrightarrow \text{on}[0]
\]
\[
\neg\text{on}[0] \leftrightarrow \neg\text{on}[0]
\]
\[
\text{loaded}[0] \leftrightarrow \text{loaded}[0]
\]
\[
\neg\text{loaded}[0] \leftrightarrow \neg\text{loaded}[0]
\]
\[
\text{toggle}[i] \leftrightarrow \text{toggle}[i]
\]
\[
\neg\text{toggle}[i] \leftrightarrow \neg\text{toggle}[i]
\]
\[
\text{load}[i] \leftrightarrow \text{load}[i]
\]
\[
\neg\text{load}[i] \leftrightarrow \neg\text{load}[i]
\]

\[\{ \text{tie}\} \] \hspace{1cm} (**) \hspace{1cm} \{ \text{tie}\}

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

Some versions of C+ require an explicit declaration

\[\text{exogenous a}\]

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)

\[\text{on}[0] \leftrightarrow \bot, \quad \neg\text{on}[0] \leftrightarrow \bot\]

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

Computational tasks

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

The ‘Causal Calculator’

Given an action description \(D\) of signature \((\sigma', \sigma^a)\), non-negative integer \(m\), and query \(\psi\) of the time-stamped signature \(\sigma_m\), CCALC

- performs the translation of \(D\) to \(\Gamma_m^D\),
- constructs \(\text{comp}(\Gamma_m^D)\),
- invokes a standard propositional sat-solver to find (classical) models of \(\text{comp}(\Gamma_m^D) \cup \psi\), 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.

(http://www.cs.utexas.edu/users/tag/cc)
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.

(We also have our own (re-)implementation of CCalc, called iCCalc. iCCalc supports a range of other action languages based on extensions of C+, as well as other features such as connection to temporal logic model checkers and model checkers for other, more exotic logics.)

iCCalc also supports the alternative implementation route, via logic programs and answer sets. That turns out to work much better, and much faster than the literal completion. Details coming at the end. The literal completion method is included here for completeness, and its historical importance.

**Examples of computational tasks**

This is how the computational tasks summarised earlier look when formulated in C+.

**Prediction** Given an action description $D$ of signature $(\sigma^*, \sigma^+)$:

- 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 $\sigma^*$ and $\alpha_i$ are formulas of $\sigma^+$.

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

In other words, we want to know whether

$$\text{comp}(\Gamma_m^D) \models (F[0] \land \alpha_0[0] \land \alpha_1[1] \land \cdots \land \alpha_{m-1}[m-1] \rightarrow G[m])$$

We can use a sat-solver to check whether

$$\text{comp}(\Gamma_{m+1}^D) \cup \{ F[0] \land \alpha_0[0] \land \alpha_1[1] \land \cdots \land \alpha_{m-1}[m-1] \land \neg G[m] \}$$

is satisfiable.

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 $m$? In other words, is there a possible run/path through the transition system such that $G$ holds at its final state?

We check whether

$$\text{comp}(\Gamma_m^D) \cup \{ F[0] \land \alpha_0[0] \land \alpha_1[1] \land \cdots \land \alpha_{m-1}[m-1] \land G[m] \}$$

is satisfiable. If satisfiable, a propositional sat-solver will return all models.

**‘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

$$\text{comp}(\Gamma_m^D) \models (\alpha_0[0] \land \alpha_1[1] \land \cdots \land \alpha_{m-1}[m-1] \land G[m] \rightarrow F[0])$$

We check whether

$$\text{comp}(\Gamma_m^D) \cup \{ \alpha_0[0] \land \alpha_1[1] \land \cdots \land \alpha_{m-1}[m-1] \land G[m] \land \neg F[0] \}$$

is satisfiable. And as before, checking whether

$$\text{comp}(\Gamma_m^D) \cup \{ \alpha_0[0] \land \alpha_1[1] \land \cdots \land \alpha_{m-1}[m-1] \land G[m] \land F[0] \}$$

is satisfiable deals with the variant of the problem in which we want to know whether it is possible that initially $F$.

**‘Planning’** (it’s not really 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 $s_0 \varepsilon_0 s_1 \cdots s_{k-1} \varepsilon_{k-1} s_k$ 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 $m$:

$$\text{comp}(\Gamma_k^D) \cup \{ F[0] \land G[k] \} \quad \text{is satisfiable}$$

The sat-solver returns all models, and these contain a representation of the plan: $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{k-1}$.

(The reference here is to events $\varepsilon$, because we want fully specified actions in the plan.)

And remember: this is not really planning.
In a separate set of notes (‘Addendum’).

References


