491 Knowledge Representation

The Action Language C+
Translation to logic program (ASP)

Marek Sergot
Department of Computing
Imperial College, London

November 2017 v1.1

Recap: Causal theories as logic programs

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

\[ L \leftarrow I_1 \land \ldots \land L_k \land \lnot L_{k+1} \land \ldots \land L_n \quad (k \geq 0, n \geq k) \tag{1} \]

or

\[ \bot \leftarrow I_1 \land \ldots \land L_k \land \lnot L_{k+1} \land \ldots \land L_n \quad (k \geq 0, n \geq k) \tag{2} \]

where \( L \) and every \( L_i \) is an atom (not necessarily Boolean).

Translate each such causal rule to the following logic program clauses, resp., constraints:

\[ L \leftarrow \lnot I_1, \ldots, \lnot L_k, \lnot L_{k+1}, \ldots, \lnot L_n \]
\[ \lnot \lnot I_1, \ldots, \lnot L_k, \lnot L_{k+1}, \ldots, \lnot L_n \]

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

Identify an interpretation \( I \) with the set of atoms satisfied by \( I \) (as usual). An interpretation \( I \) is a model of \( \Gamma \) iff \( I \) is an answer set of \( lp(\Gamma) \).

The above holds only for interpretations \( I \)—complete and consistent set of atoms—not for answer sets in general. (An answer set of \( lp(\Gamma) \) might not be an interpretation. We have to arrange that it is.)

And it holds for multi-valued signatures as well as Boolean ones—except that for multi-valued signatures we have to add rules to deal with more general form of atom. (For Boolean signatures this extra detail is not necessary.)

Multi-valued signatures

For multi-valued signatures it is still the case that every definite causal theory can be written equivalently as a set of causal rules of the forms (1) and (2). Now these rules will take the form

\[ c = v \leftarrow c_1 = v_1 \land \ldots \land c_k = v_k \land \lnot (c_{k+1} = v_{k+1}) \land \ldots \land \lnot (c_n = v_n) \quad (k \geq 0, n \geq k) \]

or

\[ \bot \leftarrow c_1 = v_1 \land \ldots \land c_k = v_k \land \lnot (c_{k+1} = v_{k+1}) \land \ldots \land \lnot (c_n = v_n) \quad (k \geq 0, n \geq k) \]

These causal rules are translated, respectively, to the following logic program clauses

\[ c = v \leftarrow \lnot (c_1 = v_1), \ldots, \lnot (c_k = v_k), \lnot c_{k+1} = v_{k+1}, \ldots, \lnot c_n = v_n \]

and constraints

\[ \bot \leftarrow \lnot (c_1 = v_1), \ldots, \lnot (c_k = v_k), \lnot c_{k+1} = v_{k+1}, \ldots, \lnot c_n = v_n \]

We need to add rules to define negated atoms of the form \( \lnot (c_i = v_i) \), and rules to ensure that a constant \( c \) can only have at most one value in \( dom(c) \). Effectively these rules will express the additional constraints described under ‘Reduction to Boolean signature’ earlier.

There are several ways to do this. Some are more convenient than others in the context of translating C+ to logic programs. The method is routine but rather long-winded. It is easiest to explain by means of an example. It should be obvious how to generalise.

Example

Consider a constant \( c \) with \( dom(c) = \{1, 2, 3\} \).

To define the three \( \lnot (c = v) \) literals, \( v \in dom(c) \), we need 6 rules \( \{n(n-1) \} \) rules when the constant has \( n \) possible values:

\[ \lnot (c = 1) \leftarrow c = 2 \]
\[ \lnot (c = 1) \leftarrow c = 3 \]

\[ \lnot (c = 2) \leftarrow c = 1 \]
\[ \lnot (c = 2) \leftarrow c = 3 \]

\[ \lnot (c = 3) \leftarrow c = 1 \]
\[ \lnot (c = 3) \leftarrow c = 2 \]

Assuming we have the usual (built-in) constraint that an answer set does not contain complementary literals, these six rules effectively capture half of the ‘Reduction to Boolean signature’, that \( c \) cannot have more than one value at a time:

\[ \lnot (c = 1 \land c = 2) \land \lnot (c = 1 \land c = 3) \land \lnot (c = 2 \land c = 3) \]

It remains to capture that \( c \) must have at least one of its possible three values:

\[ c = 1 \lor c = 2 \lor c = 3 \]

Since the three values are mutually exclusive, we can do this by ‘shifting’:

\[ c = 1 \leftarrow \lnot c = 2, \lnot c = 3 \]
\[ c = 2 \leftarrow \lnot c = 1, \lnot c = 3 \]
\[ c = 3 \leftarrow \lnot c = 1, \lnot c = 2 \]
Equivalently (easier) we can write the above rules using the negated atoms just defined:

\[
\begin{align*}
c &= 1 & \iff & \neg (c = 1) \\
c &= 2 & \iff & \neg (c = 2) \\
c &= 3 & \iff & \neg (c = 3)
\end{align*}
\]

Or we can do it with an ASP ‘choice rule’ and a cardinality constraint:

\[
1 \{ \ c = 1, \ c = 2, \ c = 3 \} \ \iff
\]

For application to \( C^+ \), the last part, whether by ‘shifting’ or choice rule, will be done by exogeneity laws.

**Application to \( C^+ \)**

I will use the variable \( T \) in the logic program to range over time indices. In the clingo program, the length \( m \) of paths will be represented by the clingo constant \( \text{maxT} \) whose value is specified when clingo is invoked.

I am going to keep the \( [\cdot] \) notation for time-stamped formulas. Obviously this is not valid clingo syntax but it is much easier to read. I will leave details of how to represent time-stamped atoms in clingo syntax for the moment.

**Static laws**

\[
\text{caused } F \text{ if } G_1 \land \ldots \land G_n \\
F[i] \iff G_1[i] \land \ldots \land G_n[i] \quad (i \in 0..m)
\]

\[
F[T] \iff \neg \bar{G}_1[T], \ldots, \neg \bar{G}_n[T] \quad (T \in 0..m)
\]

**Constraints:**

\[
\text{caused } \bot \text{ if } G_1 \land \ldots \land G_n \\
\bot \iff G_1[i] \land \ldots \land G_n[i] \quad (i \in 0..m)
\]

\[
\iff \neg \bar{G}_1[T], \ldots, \neg \bar{G}_n[T] \quad (T \in 0..m)
\]

**Action dynamic laws**

\[
\text{caused } \alpha \text{ if } G_1 \land \ldots \land G_n \\
\alpha[i] \iff G_1[i] \land \ldots \land G_n[i] \quad (i \in 0..m-1)
\]

\[
\alpha[T] \iff \neg \bar{G}_1[T], \ldots, \neg \bar{G}_n[T] \quad (T \in 0..m-1)
\]

**Fluent dynamic laws**

\[
caus \text{ed } F \text{ if } G_1 \land \ldots \land G_k \ \text{after} \ H_1 \land \ldots \land H_n \\
F[i+1] \iff G_1[i+1] \land \ldots \land G_k[i+1] \land H_1[i] \land \ldots \land H_n[i] \\
\quad (i \in 0..m-1)
\]

\[
F[T+1] \iff \neg \bar{G}_1(T+1), \ldots, \neg \bar{G}_k(T+1), \neg \bar{H}_1[T], \ldots, \neg \bar{H}_n[T] \\
\quad (T \in 0..m-1)
\]

By splitting sets, and assuming that answer sets do indeed represent interpretations, that is equivalent to:

\[
F[T+1] \iff \neg \bar{G}_1(T+1), \ldots, \neg \bar{G}_k(T+1), H_1[T], \ldots, H_n[T] \\
\quad (T \in 0..m-1)
\]

(This syntactically simpler version does not perform more efficiently, in general.)

**Exogeneity laws**

For every simple (as opposed to ‘statically determined’) fluent constant \( f \) and every \( v \in \text{dom}(f) \):

\[
f[0] = v \iff f[0] = v
\]

For every action constant \( a \), every \( v \in \text{dom}(a) \):

\[
a[i] = v \iff a[i] = v \quad (i \in 0..m-1)
\]

**Example:** Suppose fluent \( f \) has domain \( \{a, b, c\} \). First we must deal with the multivalued signature. We need to define the \( \neg(f[T] = v) \) literals at all time stamps \( T \):

\[
\neg(f[T] = a) \iff f[T] = b \quad (T \in 0..m)
\]

\[
\neg(f[T] = a) \iff f[T] = c \quad (T \in 0..m)
\]

\[
\neg(f[T] = b) \iff f[T] = a \quad (T \in 0..m)
\]

\[
\neg(f[T] = b) \iff f[T] = c \quad (T \in 0..m)
\]

\[
\neg(f[T] = c) \iff f[T] = a \quad (T \in 0..m)
\]

\[
\neg(f[T] = c) \iff f[T] = b \quad (T \in 0..m)
\]
Now the exogeneity laws for $f$ in the initial state:

\[
\begin{align*}
f[0] = a & \iff \neg(f[0] = a) \\
f[0] = b & \iff \neg(f[0] = b) \\
f[0] = c & \iff \neg(f[0] = c)
\end{align*}
\]

Alternatively, using a choice rule and cardinality constraint:

\[
1 \{ f[0] = a, f[0] = b, f[0] = c \} 1 \iff
\]

Apparently the choice rule method is much more efficient.

Obviously for a Boolean constant everything is much simpler. (Examples later.)

\begin{example}
For an action constant $a$ with domain $\{1, 2, 3\}$ (say) the exogeneity laws would be expressed either by ‘shifting’:

\[
\begin{align*}
a[T] = 1 & \iff \neg(a[T] = 1) \quad (T \in 0..m-1) \\
a[T] = 2 & \iff \neg(a[T] = 2) \quad (T \in 0..m-1) \\
a[T] = 3 & \iff \neg(a[T] = 3) \quad (T \in 0..m-1)
\end{align*}
\]

or by means of a choice rule and cardinality constraint:

\[
1 \{ a[T] = 1, a[T] = 2, a[T] = 3 \} 1 \iff \quad (T \in 0..m-1)
\]

Obviously we still need the usual rules to deal with the multi-valued signature:

\[
\begin{align*}
\neg(a[T] = 1) & \iff a[T] = 2 \quad (T \in 0..m-1) \\
\neg(a[T] = 1) & \iff a[T] = 3 \quad (T \in 0..m-1) \\
\neg(a[T] = 2) & \iff a[T] = 1 \quad (T \in 0..m-1) \\
\neg(a[T] = 2) & \iff a[T] = 3 \quad (T \in 0..m-1) \\
\neg(a[T] = 3) & \iff a[T] = 1 \quad (T \in 0..m-1) \\
\neg(a[T] = 3) & \iff a[T] = 2 \quad (T \in 0..m-1)
\end{align*}
\]

\end{example}

Example

Here is the example used earlier to illustrate construction of a literal completion. All fluent constants and action constants are Boolean in this example.

\begin{center}
\begin{tabular}{ll}
toggle causes on if & \neg on \\
toggle causes & \neg on if on \\
load causes loaded & \\
inertial on & inertial loaded \\
\end{tabular}
\end{center}

Without abbreviations:

\begin{center}
\begin{tabular}{ll}
caused on after & toggle \land \neg on \\
caused & \neg on after toggle \land on \\
caused loaded after load & \\
caused & \neg on if on after \neg on \\
caused loaded if loaded after loaded & \\
caused & \neg loaded if \neg loaded after \neg loaded \\
\end{tabular}
\end{center}

Causal theory:

\begin{center}
\begin{tabular}{ll}
on[i+1] \iff & toggle[i] \land \neg on[i] \\
\neg on[i+1] \iff & toggle[i] \land on[i] \\
loaded[i+1] \iff & load[i] \\
on[i+1] \iff & on[i+1] \land on[i] \\
\neg on[i+1] \iff & \neg on[i+1] \land \neg on[i] \\
loaded[i+1] \iff & loaded[i+1] \land loaded[i] \\
\neg loaded[i+1] \iff & \neg loaded[i+1] \land \neg loaded[i] \\
\end{tabular}
\end{center}

Together with exogeneity laws (not shown, but shown in the logic program.)

I will show the logic program in clingo syntax. Since all constants in the example are Boolean, we don’t need all the extra rules to deal with multivalued constants.

For a concrete representation, I will write $on[T]$ as $on(T)$, and similarly for the other time-stamped constants in the example.

The clingo constant maxT represents the maximum time stamp (length of paths to be constructed).
Logic program:

\[
\begin{align*}
on(T+1) & :- \neg \text{toggle}(T), \neg \text{on}(T), T=0..\text{maxT}-1. \\
\neg\text{on}(T+1) & :- \neg \text{on}(T), \neg \text{on}[T], T=0..\text{maxT}-1. \\
\text{loaded}(T+1) & :- \neg \text{load}(T), T=0..\text{maxT}-1. \\
\neg\text{loaded}(T+1) & :- \neg \text{loaded}(T), T=0..\text{maxT}-1. \\
\%	ext{ inertial on} \\
on(T+1) & :- \neg \text{on}(T+1), \neg \text{on}[T], T=0..\text{maxT}-1. \\
\neg\text{on}(T+1) & :- \neg \text{on}(T+1), \neg \text{on}[T], T=0..\text{maxT}-1. \\
\%	ext{ inertial loaded} \\
\text{loaded}(T+1) & :- \neg \text{load}(T+1), \neg \text{load}[T], T=0..\text{maxT}-1. \\
\neg\text{loaded}(T+1) & :- \neg \text{loaded}(T+1), \neg \text{load}[T], T=0..\text{maxT}-1. \\
\%	ext{ exogeneity} \\
1 \{ \text{on}(0), \neg \text{on}(0) \} 1. \\
1 \{ \text{loaded}(0), \neg \text{load}(0) \} 1. \\
1 \{ \text{toggle}(0), \neg \text{toggle}(T) \} 1 : T=0..\text{maxT}-1. \\
1 \{ \text{load}(0), \neg \text{load}(T) \} 1 : T=0..\text{maxT}-1.
\end{align*}
\]

More examples to follow.

Correctness check

The correspondence between the answer sets of a logic program \(lp(Γ^m_D)\) and the models of the causal theory \(Γ^m_D\) (and hence the paths of length \(m\) in the transition system defined by action description \(D\)) holds only when the answer set is an interpretation — a consistent and complete evaluation of all fluent and action constants at each time index.

Consistency is guaranteed because of the way that \(\neg (c = v)\) literals are defined. For completeness, it is necessary to ensure that the action description gives a value to every fluent at every time index. Completeness of the valuation for (exogenous) action constants is guaranteed by the exogeneity laws. For fluents, the exogeneity laws hold only for time index 0.

In practice, action descriptions have the desired property. All fluents that are inertial or which have a specified default value, for instance, will have a value at every time index. To be on the safe side (rarely necessary) it is possible to add a suitable set of constraints.

For every fluent \(f\) with domain \(\{v_1, \ldots, v_n\}\) add the constraints:

\[
\bot \leftarrow \neg f[T] = v_1, \ldots, \neg f[T] = v_n \quad (T \in 1..m)
\]

(The case \(T = 0\) is guaranteed by the exogeneity laws for \(f\).)

Computational tasks

The answer sets of \(lp(Γ^m_D)\) represent the states/ transitions/paths of length \(m\) in the transition system defined by \(D\). For computational tasks (prediction, temporal interpolation, planning, …) we want to pick out those answer sets satisfying some specific set of properties (initial state, goal state, etc).

We know, e.g. from the Coursework, that in order to pick out answer sets (transitions, paths) satisfying specific properties at given time indices we must add constraints to the logic program not facts (clauses with empty bodies).

For example: in Kautz’s stolen car problem where the car is known to be in the car park at times 0 and 3 (say) and known not to be in the car park at time 6 (say), we add constraints:

\[
\begin{align*}
:- \text{not } \text{p}(0). \\
:- \text{not } \text{p}(3). \\
:- \text{p}(6). 
\end{align*}
\]

This is a little hard to read.

It is much easier to read and understand written like this (recommended method!):

\[
\begin{align*}
\text{observations} & :- \\
\text{p}(0), \text{p}(3), \neg \text{p}(6). \\
:- \text{not } \text{observations}. 
\end{align*}
\]

(Because of the exogeneity laws, fluent formulas at time index 0— but only at time index 0— can be written as facts rather than as constraints. That is a detail.)

In the Yale Shooting Problem, the victim is alive at time 0. The gun is loaded at time 0. Waiting occurs (at time 1), and then the gun is shot at time 2.

\[
\begin{align*}
:- \text{not } \text{alive}(0). \\
:- \text{not } \text{load}(0). \\
:- \text{not } \text{wait}(1). \\
:- \text{not } \text{shoot}(2). 
\end{align*}
\]

Again, the following is the recommended method of expressing such constraints, and is much easier to read (and get right):

\[
\begin{align*}
\text{ysp}\_\text{constraints} & :- \\
\text{alive}(0), \text{load}(0), \text{wait}(1), \text{shoot}(2). \\
:- \text{not } \text{ysp}\_\text{constraints}. 
\end{align*}
\]
In this problem we are interested to know whether it is possible, given the sequence of events ysp constraints, that \( \text{alive}(3) \). We are therefore interested in paths of length 3 (at least) and we add the further constraint:

\[
\text{:-} \, \text{alive}(3).
\]

That eliminates all answer sets where the victim is not alive at time 3. Alternatively, because all answer sets represent consistent and complete interpretations, we could have used the constraint:

\[
\text{:-} \, \neg \text{alive}(3).
\]

Example  In the *Monkey and Bananas* problem, there is a monkey at location \( l_1 \), a box at location \( l_2 \), and some bananas at location \( l_3 \). Initially the monkey does not have the bananas; the goal is that the monkey does have the bananas. I omit the details of the action description: essentially the monkey can reach the bananas only by standing on the box. It can walk to where the box is, push the box to where the bananas are, climb on to the box, grasp the bananas.

Formulation of the action description is not difficult. For present purposes, I show only how to formulate the initial state and the goal. I will use clingo syntax. 

\[
\text{val(loc(monkey),0,l1)},
\]

represents that the value of fluent \( \text{loc(monkey)} \) at time stamp 0 is \( l_1 \), etc. (There are other possibilities.) Fluent \( \text{hasBananas} \) is Boolean.

We are looking for some value of \( \text{maxT} \) such that:

\[
\begin{align*}
\text{monkey_plan} :&= \\
&\text{val(loc(monkey),0,l1)}, \\
&\text{val(loc(box),0,l2)}, \\
&\text{val(loc(bananas),0,l3)}, \\
&\neg \text{hasBananas}(0), \\
&\text{hasBananas}(	ext{maxT}).
\end{align*}
\]

That could also be written as a set of separate constraints:

\[
\begin{align*}
\text{:-} \, \neg \text{val(loc(monkey),0,l1)}. \\
\text{:-} \, \neg \text{val(loc(box),0,l2)}. \\
\text{:-} \, \neg \text{val(loc(bananas),0,l3)}. \\
\text{:-} \, \neg \text{hasBananas}(0). \\
\text{:-} \, \text{hasBananas}(	ext{maxT}).
\end{align*}
\]

I suggest that the recommended method (first version) is clearer to read.
(ii) The (Boolean) action switch changes the value of fluent status from on to off and from off to on.

\[
\begin{align*}
\text{switch causes } & \text{status} = \text{on if } \text{status} = \text{off} \\
\text{switch causes } & \text{status} = \text{off if } \text{status} = \text{on} \\
\text{caused } & \text{status} = \text{on after switch } \land \text{status} = \text{on} \\
\text{caused } & \text{status} = \text{off after switch } \land \text{status} = \text{off} \\
\text{status}_{[i+1]} & = \text{on } \iff \text{switch}_{[i]} \land \text{status}_{[i]} = \text{off} \quad (i \in 0..m-1) \\
\text{status}_{[i+1]} & = \text{off } \iff \text{switch}_{[i]} \land \text{status}_{[i]} = \text{on} \quad (i \in 0..m-1)
\end{align*}
\]

In clingo syntax:

\[
\begin{align*}
\text{status}(T+1,\text{on}) & : - \text{not -switch}(T), \text{not -status}(T,\text{off}), \text{T=0..maxT-1}. \\
\text{status}(T+1,\text{off}) & : - \text{not -switch}(T), \text{not -status}(T,\text{on}), \text{T=0..maxT-1}. \\
\end{align*}
\]

The exogeneity laws:

\[
1 \{\text{switch}(T), -\text{switch}(T)\} 1 : = T=0..\text{maxT-1}.
\]

Because switch(T) is Boolean we do not need to include a definition of -switch(T):

\[
-\text{switch}(T) : = \text{not switch}(T), \text{T=0..maxT-1}.
\]

(The exogeneity laws subsume it.) It is not needed but it does not hurt to include it.

(iii) The (Boolean) action open is not executable when status = off.

\[
\begin{align*}
\text{nonexecutable open if } & \text{status} = \text{off} \\
\text{caused } & \text{on after open } \land \text{status} = \text{on} \\
\text{on } & \iff \text{open}_{[i]} \land \text{status}_{[i]} = \text{off} \quad (i \in 0..m-1)
\end{align*}
\]

The logic program:

\[
\begin{align*}
& : - \text{not -open}(T), \text{not -status}(T,\text{off}), \text{T=0..maxT-1}. \\
\end{align*}
\]

Equivalently, but no more efficiently:

\[
\begin{align*}
& : - \text{open}(T), \text{status}(T,\text{off}), \text{T=0..maxT-1}. \\
\end{align*}
\]

Exogeneity laws:

\[
1 \{\text{open}(T), -\text{open}(T)\} 1 : = T=0..\text{maxT-1}.
\]

Again, because open(T) is Boolean, the exogeneity laws already take care of defining -open(T).

---

From Exam 2007

In any given state, a certain (spring-loaded) door is either open or closed (but not both). Let the Boolean fluent closed represent that the door is closed and \( \neg \text{closed} \) that it is open.

(i) The (Boolean) action of pushing the door causes it to become open if is closed; pushing the door is not possible (executable) if the door is open.

\[
\begin{align*}
\text{push causes } & \neg \text{closed if closed} \\
\text{nonexecutable push if } & \neg \text{closed} \\
\text{caused } & \neg \text{closed after push } \land \neg \text{closed} \\
\text{caused } & \text{closed after push } \land \neg \text{closed} \\
\text{caused } & \neg \text{closed[i+1]} \iff \text{push}[i] \land \neg \text{closed}[i] \quad (i \in 0..m-1) \\
\text{closed[T+1]} & : - \neg \text{closed(T+1)}, \text{not -closed}(T), \text{T=0..maxT-1}. \\
\text{closed[T+1]} & : - \text{not -closed(T+1)}, \text{not -closed}(T), \text{T=0..maxT-1}.
\end{align*}
\]

All constants are Boolean. The exogeneity laws will take care of defining \( \neg \text{push}(T) \) but exogeneity laws for the fluent closed are only for time index 0. So we add:

\[
\begin{align*}
& \neg \text{closed}(T) : = \text{not closed}(T), \text{T=0..maxT}. \\
\end{align*}
\]

(ii) If the door is closed, it remains closed by default (‘inertia’); if it is open, it will be closed in the next state, by default.

\[
\begin{align*}
\text{caused closed if closed after close } & \text{ (inertial)} \\
\text{caused closed if closed after } & \neg \text{closed } \text{ (not inertial)} \\
\text{closed[i+1]} & \iff \text{closed}[i+1] \land \neg \text{closed}[i] \quad (i \in 0..m-1) \\
\text{closed[T+1]} & : - \neg \text{closed(T+1)}, \text{not -closed}(T), \text{T=0..maxT-1}. \\
\text{closed(T+1)} & : - \text{not -closed(T+1)}, \text{not -closed}(T), \text{T=0..maxT-1}.
\end{align*}
\]

The same effect could also be obtained simply as

\[
\begin{align*}
\text{caused closed if close } & \text{T} \\
\text{closed} & \neg \text{-closed(T+1)}, \text{T=0..maxT-1}.
\end{align*}
\]

If the door is also closed by default in the initial state, then even more simply:

\[
\begin{align*}
\text{default closed} \\
\text{caused closed if close} \\
\text{closed} & \neg \text{-closed(T)}, \text{T=0..maxT}.
\end{align*}
\]
Example

(Used earlier in the C+ notes) There are three agents a, b, c. Each has a car. There are
three locations: home, work, pub.

Fluent symbols:

\[ \text{loc}(x) = p: \text{agent } x \text{ is at location } p \]
\[ \text{car}(x) = p: \text{agent } x \text{'s car is at location } p \]

Action symbols:

\[ \text{walk}(x) = \text{dest}: x \text{ walks to } \text{dest} \]
\[ \text{drive}(x) = \text{dest}: x \text{ drives to } \text{dest} \]

The domain of \( \text{walk}(x) \) and \( \text{drive}(x) \) are 'destinations' not locations:
\[ \text{dom}(\text{walk}(x)) = \text{dom}(\text{drive}(x)) = \{ \text{home, work, pub, none} \} \]
\[ \text{drive}(x) = p \text{ when } \text{loc}(x) = p \text{ means that } x \text{ drives around and ends up back where he/she} \]
\[ \text{started. And similarly for } \text{walk}(x). \]

In the following \( x \) ranges over the agents and \( p, p' \) over the locations:

\[
\begin{align*}
\text{inertial } \text{loc}(x) \\
\text{inertial } \text{car}(x) \\
\text{walk}(x) = p \text{ causes } \text{loc}(x) = p \\
\text{drive}(x) = p \text{ causes } \text{loc}(x) = p \\
\text{drive}(x) = p \text{ causes } \text{car}(x) = p \\
\text{nonexecutable } \text{drive}(x) = p \land \text{walk}(x) = p' \\
\text{nonexecutable } \text{drive}(x) = p \text{ if } \text{loc}(x) \neq \text{car}(x)
\end{align*}
\]

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

\[
\text{nonexecutable } \text{drive}(x) = p \text{ if } \text{loc}(x) = p' \land
\neg(\text{car}(x) = p') \quad (\text{for all locations } p, p')
\]

In clingo:

\[
\begin{align*}
\text{agent(a). location(home).} \\
\text{agent(b). location(work).} \\
\text{agent(c). location(pub).} \\
\text{destination(none).} \\
\text{destination(X) :- location(X).}
\end{align*}
\]

In what follows the chosen clingo representation of multi-valued fluents and action
costants should be clear. (It is not the only possible one.) Notice that in this example
the definition of negated atoms can be expressed succinctly using general rules.

\[
\begin{align*}
\text{%- ---- fluents ----} \\
\text{%- definitions of -val(…)} \\
\text{-val(loc(X),T,P) :- agent(X), location(P),} \\
\text{val(loc(X),T,Q), location(Q), P \neq Q,} \\
T=0..\text{maxT}. \quad \text{% note} \\
\text{-val(car(X),T,P) :- agent(X), location(P),} \\
\text{val(car(X),T,Q), location(Q), P \neq Q,} \\
T=0..\text{maxT}. \quad \text{% note} \\
\text{% exogeneity of fluents} \\
% (There is a way of expressing these more succinctly in clingo.) \\
1 \{\text{val(loc(X),0,home),val(loc(X),0,work),val(loc(X),0,pub)}\} 1
:= \text{agent(X).} \\
1 \{\text{val(car(X),0,home),val(car(X),0,work),val(car(X),0,pub)}\} 1
:= \text{agent(X).} \\
\text{% ---- action constants ----} \\
\text{-val(drive(X),T,D) :- agent(X), destination(D),} \\
\text{val(drive(X),T,Dx), destination(Dx), D \neq Dx,} \\
T=0..\text{maxT-1}. \quad \text{% note} \\
\text{-val(walk(X),T,D) :- agent(X), destination(D),} \\
\text{val(walk(X),T,Dx), destination(Dx), D \neq Dx,} \\
T=0..\text{maxT-1}. \quad \text{% note} \\
\text{% exogeneity of actions} \\
1 \{\text{val(drive(X),T,home),val(drive(X),T,work),} \\
\text{val(drive(X),T,pub),val(drive(X),T,none)}\} 1
:= \text{agent(X), T=0..\text{maxT-1}.} \\
1 \{\text{val(walk(X),T,home),val(walk(X),T,work),} \\
\text{val(walk(X),T,pub),val(walk(X),T,none)}\} 1
:= \text{agent(X), T=0..\text{maxT-1}.}
\end{align*}
\]

\[
\begin{align*}
\text{%- ---- fluents ----} \\
\text{%- definitions of -val(…)} \\
\text{-val(loc(X),T,P) :- agent(X), location(P),} \\
\text{val(loc(X),T,Q), location(Q), P \neq Q,} \\
T=0..\text{maxT}. \quad \text{% note} \\
\text{-val(car(X),T,P) :- agent(X), location(P),} \\
\text{val(car(X),T,Q), location(Q), P \neq Q,} \\
T=0..\text{maxT}. \quad \text{% note} \\
\text{% exogeneity of fluents} \\
% (There is a way of expressing these more succinctly in clingo.) \\
1 \{\text{val(loc(X),0,home),val(loc(X),0,work),val(loc(X),0,pub)}\} 1
:= \text{agent(X).} \\
1 \{\text{val(car(X),0,home),val(car(X),0,work),val(car(X),0,pub)}\} 1
:= \text{agent(X).} \\
\text{% ---- action constants ----} \\
\text{-val(drive(X),T,D) :- agent(X), destination(D),} \\
\text{val(drive(X),T,Dx), destination(Dx), D \neq Dx,} \\
T=0..\text{maxT-1}. \quad \text{% note} \\
\text{-val(walk(X),T,D) :- agent(X), destination(D),} \\
\text{val(walk(X),T,Dx), destination(Dx), D \neq Dx,} \\
T=0..\text{maxT-1}. \quad \text{% note} \\
\text{% exogeneity of actions} \\
1 \{\text{val(drive(X),T,home),val(drive(X),T,work),} \\
\text{val(drive(X),T,pub),val(drive(X),T,none)}\} 1
:= \text{agent(X), T=0..\text{maxT-1}.} \\
1 \{\text{val(walk(X),T,home),val(walk(X),T,work),} \\
\text{val(walk(X),T,pub),val(walk(X),T,none)}\} 1
:= \text{agent(X), T=0..\text{maxT-1}.}
\end{align*}
\]

\[
\begin{align*}
\text{%- ---- fluents ----} \\
\text{%- definitions of -val(…)} \\
\text{-val(loc(X),T,P) :- agent(X), location(P),} \\
\text{val(loc(X),T,Q), location(Q), P \neq Q,} \\
T=0..\text{maxT}. \quad \text{% note} \\
\text{-val(car(X),T,P) :- agent(X), location(P),} \\
\text{val(car(X),T,Q), location(Q), P \neq Q,} \\
T=0..\text{maxT}. \quad \text{% note} \\
\text{% exogeneity of fluents} \\
% (There is a way of expressing these more succinctly in clingo.) \\
1 \{\text{val(loc(X),0,home),val(loc(X),0,work),val(loc(X),0,pub)}\} 1
:= \text{agent(X).} \\
1 \{\text{val(car(X),0,home),val(car(X),0,work),val(car(X),0,pub)}\} 1
:= \text{agent(X).} \\
\text{% ---- action constants ----} \\
\text{-val(drive(X),T,D) :- agent(X), destination(D),} \\
\text{val(drive(X),T,Dx), destination(Dx), D \neq Dx,} \\
T=0..\text{maxT-1}. \quad \text{% note} \\
\text{-val(walk(X),T,D) :- agent(X), destination(D),} \\
\text{val(walk(X),T,Dx), destination(Dx), D \neq Dx,} \\
T=0..\text{maxT-1}. \quad \text{% note} \\
\text{% exogeneity of actions} \\
1 \{\text{val(drive(X),T,home),val(drive(X),T,work),} \\
\text{val(drive(X),T,pub),val(drive(X),T,none)}\} 1
:= \text{agent(X), T=0..\text{maxT-1}.} \\
1 \{\text{val(walk(X),T,home),val(walk(X),T,work),} \\
\text{val(walk(X),T,pub),val(walk(X),T,none)}\} 1
:= \text{agent(X), T=0..\text{maxT-1}.}
\end{align*}
\]
% ---- causal laws ----
% inertial loc(x)
% inertial car(x)
val(loc(X),T+1,P) :- not -val(loc(X),T+1,P), not -val(loc(X),T,P),
agent(X), location(P), T=0..maxT-1.
val(car(X),T+1,P) :- not -val(car(X),T+1,P), not -val(car(X),T,P),
agent(X), location(P), T=0..maxT-1.
% walk(x)=p causes loc(x)=p
% drive(x)=p causes loc(x)=p
% drive(x)=p causes car(x)=p
val(loc(X),T+1,P) :- not -val(walk(X),T,P),
agent(X), location(P),
T=0..maxT-1.
val(loc(X),T+1,P) :- not -val(drive(X),T,P),
agent(X), location(P),
T=0..maxT-1.
val(car(X),T+1,P) :- not -val(drive(X),T,P),
agent(X), location(P),
T=0..maxT-1.
% nonexecutable drive(x)=p & walk(x)=p'
:- not -val(drive(X),T,P), not -val(walk(X),T,Q),
agent(X), location(P), location(Q),
T=0..maxT-1.
% nonexecutable drive(x)=p if loc(x) != car(x)
% equivalently
% nonexecutable drive(x)=p if loc(x)=p' & ~(car(x)=p')
:- not -val(drive(X),T,P),
not -val(loc(X),T,Q), not val(car(X),T,Q),
agent(X), location(P), location(Q),
T=0..maxT-1.

Example query
Suppose a walks home at time 0, then (at time 1) drives to the pub where he meets b. At
time 3 a walks home. At time 4 b is at work.
The recommended way of expressing the constraints (because it is clearer):

story_1 :-
val(walks(a),0,home).
val(drives(a),1,pub).
val(loc(b),2,pub).
val(walks(a),3,home).
val(loc(b),4,work).
:- not story_1.