491 Knowledge Representation

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

Marek Sergot
Department of Computing
Imperial College, London

November 2014 v1.0b

Recap: Causal theories as logic programs

Every definite causal theory $\Gamma$ can be written equivalently as a set of causal rules of the form

$$L \leftarrow L_1 \land \ldots \land L_k \land \neg L_{k+1} \land \ldots \land \neg L_n \quad \text{(k \geq 0, n \geq k)}$$

or

$$\bot \leftarrow L_1 \land \ldots \land L_k \land \neg L_{k+1} \land \ldots \land \neg L_n \quad \text{(k \geq 0, n \geq k)}$$

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:

$$L \leftarrow \neg L_1, \ldots, \neg L_k, \neg L_{k+1}, \ldots, \neg L_n$$

$$\bot \leftarrow \neg L_1, \ldots, \neg L_k, \neg L_{k+1}, \ldots, \neg L_n$$

For every multi-valued constant $c$ add, for every value $v \in \text{dom}(c)$, rules defining $\neg (c = v)$, as follows:

$$\neg (c = v) \leftarrow c = v' \quad \text{(for every } v' \in \text{dom}(c), v' \neq v)$$

(For Boolean constants, this last step can be simplified.)

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.)

Application to $C+$

Leaving aside details of how to represent time-stamped atoms for a moment, the translation of $C+$ to logic program is immediate.

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.

Static laws

caused $F$ if $G_1 \land \ldots \land G_n$.

$$F[i] \leftarrow G_1[i] \land \ldots \land G_n[i] \quad (i \in 0..m)$$

$$F(T) \leftarrow \neg \text{G}_1(T), \ldots, \neg \text{G}_n(T) \quad (T \in 0..m)$$

Constraints:

caused $\bot$ if $G_1 \land \ldots \land G_n$.

$$\bot \leftarrow G_1[i] \land \ldots \land G_n[i] \quad (i \in 0..m)$$

$$\bot \leftarrow \neg \text{G}_1(T), \ldots, \neg \text{G}_n(T) \quad (T \in 0..m)$$

Action dynamic laws

caused $\alpha$ if $G_1 \land \ldots \land G_n$.

$$\alpha[i] \leftarrow G_1[i] \land \ldots \land G_n[i] \quad (i \in 0..m)$$

$$\alpha(T) \leftarrow \neg \text{G}_1(T), \ldots, \neg \text{G}_n(T) \quad (T \in 0..m)$$

Fluent dynamic laws

caused $F$ if $G_1 \land \ldots \land G_k$ after $H_1 \land \ldots \land H_n$.

$$F[i+1] \leftarrow G_1[i+1] \land \ldots \land G_k[i+1] \land H_1[i] \land \ldots \land H_n[i] \quad (i \in 0..m-1)$$

$$F(T+1) \leftarrow \neg \text{G}_1(T+1), \ldots, \neg \text{G}_k(T+1), \neg \text{H}_1(T), \ldots, \neg \text{H}_n(T) \quad (T \in 0..m-1)$$

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

$$F(T+1) \leftarrow \neg \text{G}_1(T+1), \ldots, \neg \text{G}_k(T+1), H_1(T), \ldots, H_n(T) \quad (T \in 0..m-1)$$

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

caused ⊥ if \( G_1 \land \ldots \land G_k \) after \( H_1 \land \ldots \land H_n \)

\( \perp \iff G_1[i+1] \land \ldots \land G_k[i+1] \land H_1[i] \land \ldots \land H_n[i] \quad (i \in 0..m-1) \)

or (simpler, but no more efficient version):

\( \perp \iff \neg G_1(T+1), \ldots, \neg G_k(T+1), \neg H_1(T), \ldots, \neg H_n(T) \quad (T \in 0..m-1) \)

Exogeneity laws

For every sample (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 \( v \in \text{dom}(a) \):

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

Let us represent time-stamped Boolean atoms \( p[i] \) and \( \neg p[i] \) (shorthand for \( p[i] = \text{t} \) and \( p[i] = \text{f} \)) by the logic program (clingo) literals \( p(i) \) and \( \neg p(i) \) respectively.

Let us represent time-stamped multi-valued atoms \( c[i][v] = v \) by the logic program atoms \( \text{val}(c,i,v) \).

Example: Suppose fluent \( f \) has domain \( \{a, b, c\} \). The exogeneity laws for \( f \):

\[ f[0] = a \iff f[0] = a \]
\[ f[0] = b \iff f[0] = b \]
\[ f[0] = c \iff f[0] = c \]

would be translated, applying the general translation method, to:

\[
\text{val}(f,0,a) : \neg \text{val}(f,0,a).
\]
\[
\text{val}(f,0,b) : \neg \text{val}(f,0,b).
\]
\[
\text{val}(f,0,c) : \neg \text{val}(f,0,c).
\]

\[
\neg \text{val}(f,0,a) : \text{val}(f,0,b).
\]
\[
\neg \text{val}(f,0,a) : \text{val}(f,0,c).
\]
\[
\neg \text{val}(f,0,b) : \text{val}(f,0,a).
\]
\[
\neg \text{val}(f,0,b) : \text{val}(f,0,c).
\]
\[
\neg \text{val}(f,0,c) : \text{val}(f,0,a).
\]
\[
\neg \text{val}(f,0,c) : \text{val}(f,0,b).
\]

Cardinality constraints

The ASP cardinality constraint:

\[ k_{\min}\{L_1,\ldots,L_n\} k_{\max} \]

specifies that between \( k_{\min} \) and \( k_{\max} \) of the literals \( L_1,\ldots,L_n \) must belong to any answer set. As a special case

\[ 1\{L_1,\ldots,L_n\}1 \]

specifies that exactly one of \( L_1,\ldots,L_n \) must belong to any answer set.

Thus, the required exogeneity laws for fluent \( f \) at time index 0 can be specified succinctly (and much more efficiently) by writing (in clingo notation):

\[ 1 \{\text{val}(f,0,a), \text{val}(f,0,b), \text{val}(f,0,c)\} 1. \]

For Boolean fluents, the exogeneity laws

\[ f[0] \iff f[0] \]

\[ \neg f[0] \iff \neg f[0] \]

would be translated to the logic program clauses (clingo notation)

\[ f(0) : \neg \neg f(0) \]
\[ \neg f(0) : \neg \neg f(0) \]

They can be expressed more efficiently (apparently) by means of the cardinality constraint:

\[ 1 \{ f(0), \neg f(0) \} 1. \]

Example: For an action constant \( a \) with domain \( \{1,2,3\} \) (say) the exogeneity laws would be expressed (in clingo notation):

\[ 1 \{ \text{val}(a,T,1), \text{val}(a,T,2), \text{val}(a,T,3) \} 1 : - T = 0..\text{max}T-1. \]

\( T = 0..\text{max}T-1 \) is valid clingo syntax and will cause gringo to ground all rules with values of \( T \) between 0 and \( \text{max}T \). Here \( \text{max}T \) is a clingo constant representing the required length \( m \) of paths in the transition system.

It is much more efficient to make use of a feature of clingo (and most other ASP solvers) called cardinality constraints.
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{align*}
\text{toggle} & \text{ causes on if } \neg \text{on} \\
\text{toggle} & \text{ causes } \neg \text{on if on} \\
\text{load} & \text{ causes loaded} \\
\text{inertial} & \text{ on} \\
\text{inertial} & \text{ loaded}
\end{align*}

Without abbreviations:

\begin{align*}
\text{caused} & \text{ on after toggle } \land \neg \text{on} \\
\text{caused} & \text{ on after toggle } \land \text{on} \\
\text{caused} & \text{ loaded after load} \\
\text{caused} & \text{ on if on after on} \\
\text{caused} & \text{ on if } \neg \text{on after } \neg \text{on} \\
\text{caused} & \text{ loaded if loaded after loaded} \\
\text{caused} & \text{ } \neg \text{loaded if } \neg \text{loaded after } \neg \text{loaded}
\end{align*}

Causal theory:

\begin{align*}
on[i+1] & \iff \text{toggle}[i] \land \neg \text{on}[i] \\
\neg \text{on}[i+1] & \iff \text{toggle}[i] \land \text{on}[i] \\
\text{loaded}[i+1] & \iff \text{load}[i] \\
on[i+1] & \iff \text{on}[i+1] \land \text{on}[i] \\
\neg \text{on}[i+1] & \iff \neg \text{on}[i+1] \land \neg \text{on}[i] \\
\text{loaded}[i+1] & \iff \text{loaded}[i+1] \land \text{loaded}[i] \\
\neg \text{loaded}[i+1] & \iff \neg \text{loaded}[i+1] \land \neg \text{loaded}[i]
\end{align*}

Together with exogeneity laws (not shown)

Logic program:

\begin{verbatim}
on(T+1) :- not -toggle(T), not on(T), T=0..maxT-1.
-on(T+1) :- not -toggle(T), not -on(T), T=0..maxT-1.
loaded(T+1) :- not -load(T), T=0..maxT-1.
\end{verbatim}

\% inertial on
\begin{verbatim}
on(T+1) :- not -on(T+1), not -on(T), T=0..maxT-1.
-on(T+1) :- not on(T+1), not on(T), T=0..maxT-1.
\end{verbatim}

\% inertial loaded
\begin{verbatim}
loaded(T+1) :- not -loaded(T+1), not -loaded(T), T=0..maxT-1.
-loaded(T+1) :- not loaded(T+1), not loaded(T), T=0..maxT-1.
\end{verbatim}

\% exogeneity
\begin{verbatim}
1 { on(0), -on(0) } 1.
1 { loaded(0), -loaded(0) } 1.
1 { toggle(T), -toggle(T) } 1 :- T=0..maxT-1.
1 { load(T), -load(T) } 1 :- T=0..maxT-1.
\end{verbatim}

More examples to follow.

Correctness check

The correspondence between the answer sets of a logic program \(lp(\Gamma_m^D)\) and the models of the causal theory \(\Gamma_m^D\) (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 \(-\text{val}(\ldots)\) 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 \textit{inertial} or which have a specified \textit{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:

\begin{verbatim}
:- not \text{val}(f,T,v_1), \ldots, not \text{val}(f,T,v_n), T=1..maxT.
\end{verbatim}

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

The answer sets of $lp(\Gamma_D^m)$ 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:

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

Because all answer sets of the logic program represent interpretations (complete and consistent values for all fluent and action constants at all time indices), the above can also be expressed equivalently as:

\[
\begin{align*}
&\text{:=- -}p(0). \\
&\text{:=- -}p(3). \\
&\text{:=- } p(6).
\end{align*}
\]

This could also be written:

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

(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 (time 1), and then the gun is shot at time 2.

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

Alternatively, and perhaps more clearly:

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

Here we are interested in paths of length 3 because we want to know whether it is possible, given the sequence of events ysp_constraints, that alive(3).

In general, let $W$ be a formula expressing the properties of the answer sets we want ($W$ is for ‘want’). In the Yale Shooting Problem, for example, $W$ would be $\text{alive(0) \land load(0) \land wait(1) \land shoot(2)}$.

We eliminate all answer sets that do not satisfy $W$ by adding the constraint

\[
\text{:=- not } W
\]

In general however not $W$ will have to be transformed into clausal form (a conjunction of literals or negations of literals). We can get the same effect by the following means:

\[
\text{problem_constraints} \leftarrow W \\
\text{:=- not problem_constraints}
\]

For example: in the Monkey and Bananas problem, the monkey is initially at location $l_1$, a box is at location $l_2$, and the bananas are at location $l_3$. Initially the monkey does not have the bananas; the goal is that the monkey does have the bananas. We are looking for some value of maxT such that:

\[
\begin{align*}
\text{monkey_plan} &\leftarrow \text{val(loc(monkey),0,l1),} \\
&\text{val(loc(box),0,l2),} \\
&\text{val(loc(bananas),0,l3),} \\
&\text{-hasBananas(0),} \\
&\text{hasBananas(maxT).} \\
\text{:=- not monkey_plan.}
\end{align*}
\]

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

\[
\begin{align*}
\text{:=- not val(loc(monkey),0,l1).} \\
\text{:=- not val(loc(box),0,l2).} \\
\text{:=- not val(loc(bananas),0,l3).} \\
\text{:=- not -hasBananas(0).} \\
\text{:=- not hasBananas(maxT).}
\end{align*}
\]
Or equivalently again, because of the exogeneity of fluents at time index 0, as a mixture
of facts and constraints:

\[
\begin{align*}
\text{val(loc(monkey),0,11).} \\
\text{val(loc(box),0,12).} \\
\text{val(loc(bananas),0,13).} \\
\text{\neg hasBananas(0).}
\end{align*}
\]

\[\vdash \text{\neg hasBananas(maxT).}\]

**Examples**

From Exam 2004

(i) The fluent constant `status`, with two possible values `on` and `off`, is inertial.

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

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

Or equivalently (but no more efficiently):

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

(ii) The (Boolean) action `switch` changes the value of fluent `status` from `on` to `off` and
from `off` to `on`.

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

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

We define:

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

The exogeneity laws:

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

Note that because `switch(T)` is Boolean, the exogeneity laws subsume the definition
of `\neg switch(T)` given above. It could be removed (but it does not hurt to include it.)

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

\[
\begin{align*}
\text{nonexecutable open if status = off} \\
\text{caused \bot after open \land status = off} \\
\text{\bot \iff open[i] \land status[i] = off} & (i \in 0..m-1) \\
\vdash \neg -\text{open(T)}, \neg -\text{val(status,T,off)}, T=0..\text{maxT-1}.
\end{align*}
\]

or (equivalently, but no more efficiently)

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

The exogeneity laws:

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

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

The required exogeneity laws are:

\[
1 \{\text{val(status,0,on), val(status,0,off)}\} 1.
\]
From Exam 2007

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

- \( \text{push causes } \neg \text{closed if } \text{closed} \)
- \( \text{nonexecutable } \neg \text{closed if } \neg \text{closed} \)
- \( \text{caused } \neg \text{closed after } \text{push } \wedge \neg \text{closed} \)
- \( \text{caused } \bot \text{ after } \text{push } \wedge \neg \text{closed} \)
- \( \neg \text{closed}[i+1] \iff \text{push}[i] \wedge \neg \text{closed}[i] \quad (i \in 0..m-1) \)
- \( \bot \iff \text{push}[i] \wedge \neg \text{closed}[i] \quad (i \in 0..m-1) \)
- \( \neg \text{closed}(T+1) := \neg \text{push}(T), \neg \neg \text{closed}(T), T=0..\text{maxT}-1. \)

(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.

- \( \text{caused } \neg \text{closed if } \text{closed after } \text{closed} \) (inertial)
- \( \text{caused } \neg \text{closed if } \text{closed after } \neg \text{closed} \) (not inertial)
- \( \neg \text{closed}[i+1] \iff \text{closed}[i+1] \wedge \neg \text{closed}[i] \quad (i \in 0..m-1) \)
- \( \neg \text{closed}[i+1] \iff \text{closed}[i+1] \wedge \neg \neg \text{closed}[i] \quad (i \in 0..m-1) \)
- \( \neg \text{closed}(T+1) := \neg \neg \text{closed}(T+1), \neg \neg \text{closed}(T), T=0..\text{maxT}-1. \)

The same effect could also be obtained simply as

- \( \text{caused } \neg \text{closed if } \text{closed after } \top \) 
- \( \neg \text{closed}(T+1) := \neg \neg \text{closed}(T+1), T=0..\text{maxT}-1. \)

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

- \( \text{default } \neg \text{closed} \)
- \( \text{caused } \neg \text{closed if } \text{closed} \)
- \( \neg \text{closed}(T) := \neg \neg \text{closed}(T), T=0..\text{maxT}. \)

Example

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

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

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

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

- \( \text{inertial } \text{loc}(x) \)
- \( \text{inertial } \text{car}(x) \)
- \( \text{walk}(x)=p \text{ causes } \text{loc}(x)=p \)
- \( \text{drive}(x)=p \text{ causes } \text{loc}(x)=p \)
- \( \text{drive}(x)=p \text{ causes } \text{car}(x)=p \)
- \( \text{nonexecutable } \text{drive}(x)=p \wedge \text{walk}(x)=p' \)
- \( \text{nonexecutable } \text{drive}(x)=p \text{ if } \text{loc}(x) \neq \text{car}(x) \)

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

- \( \text{nonexecutable } \text{drive}(x)=p \text{ if } \text{loc}(x)=p' \wedge \neg (\text{car}(x)=p') \) (for all locations \( p,p' \))

In clingo:

- \( \text{agent}(a). \quad \text{location(home)}. \)
- \( \text{agent}(b). \quad \text{location(work)}. \)
- \( \text{agent}(c). \quad \text{location(pub)}. \)
- \( \text{destination(none)}. \)
- \( \text{destination}(X) := \text{location}(X) . \)
% ---- fluents ----
% definitions of -val(...)
-val(loc(X),T,P) :- agent(X), location(P),
  val(loc(X),T,Q),
  location(Q), P != Q,
  T=0..maxT. % note
-val(car(X),T,P) :- agent(X), location(P),
  val(car(X),T,Q),
  location(Q), P != Q,
  T=0..maxT. % note

% exogeneity of fluents
% (There is a way of expressing these more succinctly in clingo.)
1 {val(loc(X),0,home),val(loc(X),0,work),val(loc(X),0,pub)} 1 :-
  agent(X).
1 {val(car(X),0,home),val(car(X),0,work),val(car(X),0,pub)} 1 :-
  agent(X).

% ---- action constants ----
-val(drive(X),T,D) :- agent(X), destination(D),
  val(drive(X),T,Dx),
  destination(Dx), D != Dx,
  T=0..maxT-1. % note
-val(walk(X),T,D) :- agent(X), destination(D),
  val(walk(X),T,Dx),
  destination(Dx), D != Dx,
  T=0..maxT-1. % note

% exogeneity of actions
1 {val(drive(X),T,home),val(drive(X),T,work),
  val(drive(X),T,pub),val(drive(X),T,none)} 1 :-
  agent(X), T=0..maxT-1.
1 {val(walk(X),T,home),val(walk(X),T,work),
  val(walk(X),T,pub),val(walk(X),T,none)} 1 :-
  agent(X), T=0..maxT-1.

% ---- 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.

Add the constraints

\[
\text{:- } \text{val(walks(a),0,home).} \\
\text{:- } \text{val(drives(a),1,pub).} \\
\text{:- } \text{val(loc(b),2,pub).} \\
\text{:- } \text{val(walks(a),3,home).} \\
\text{:- } \text{val(loc(b),4,work).} 
\]

Equivalently (because of exogeneity of actions) we can instead use a mixture of facts and constraints:

\[
\text{val(walks(a),0,home).} \\
\text{val(drives(a),1,pub).} \\
\text{val(walks(a),3,home).} \\
\text{:- } \text{val(loc(b),2,pub).} \quad \text{% must be contraint} \\
\text{:- } \text{val(loc(b),4,work).} \quad \text{% must be contraint} 
\]

Or alternatively, the recommended method (because it is clearer):

\[
\text{story_1 :-} \\
\text{val(walks(a),0,home).} \\
\text{val(drives(a),1,pub).} \\
\text{val(loc(b),2,pub).} \\
\text{val(walks(a),3,home).} \\
\text{val(loc(b),4,work).} \\
\text{:- not story_1.} 
\]