491 Knowledge Representation

The Action Language $\mathcal{C}+$

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

$$L \Leftarrow L_1 \wedge \ldots \wedge L_k \wedge \neg L_{k+1} \wedge \ldots \neg L_n \qquad (k \geq 0, n \geq k) \tag{1}$$

or

$$\bot \Leftarrow L_1 \wedge \ldots \wedge L_k \wedge \neg L_{k+1} \wedge \ldots \neg L_n \qquad (k \geq 0, n \geq k) \tag{2}$$

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:

$$L \leftarrow \ not\ \neg L_1,\ \ldots,\ not\ \neg L_k,\ not\ L_{k+1},\ \ldots,\ not\ L_n$$
$$\leftarrow \ not\ \neg L_1,\ \ldots,\ not\ \neg L_k,\ not\ L_{k+1},\ \ldots,\ not\ L_n$$

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 Γ 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

$$c = v \ \Leftarrow\ c_1 = v_1 \wedge \ldots \wedge c_k = v_k \wedge \neg(c_{k+1} = v_{k+1}) \wedge \ldots \wedge \neg(c_n = v_n) \qquad (k \geq 0, n \geq k)$$

or

$$\bot \ \Leftarrow\ c_1 = v_1 \wedge \ldots \wedge c_k = v_k \wedge \neg(c_{k+1} = v_{k+1}) \wedge \ldots \wedge \neg(c_n = v_n) \qquad (k \geq 0, n \geq k)$$

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

$$c = v \ \leftarrow\ not\ \neg(c_1 = v_1), \ldots, not\ \neg(c_k = v_k),\ not\ c_{k+1} = v_{k+1}, \ldots, not\ c_n = v_n$$

and constraints

$$\bot \ \leftarrow\ not\ \neg(c_1 = v_1), \ldots, not\ \neg(c_k = v_k),\ not\ c_{k+1} = v_{k+1}, \ldots, not\ c_n = v_n$$

We need to add rules to define negated atoms of the form $\neg(c_1 = v_1)$, 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 $\mathcal{C}+$ 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 $\neg(c = v)$ literals, $v \in dom(c)$, we need 6 rules ($n(n-1)$ rules when the constant has n possible values):

$$\begin{array}{lll} \neg(c=1) \leftarrow c=2 & \neg(c=2) \leftarrow c=1 & \neg(c=3) \leftarrow c=1 \\ \neg(c=1) \leftarrow c=3 & \neg(c=2) \leftarrow c=3 & \neg(c=3) \leftarrow c=2 \end{array}$$

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:

$$\neg(c=1 \wedge c=2) \wedge \neg(c=1 \wedge c=3) \wedge \neg(c=2 \wedge c=3)$$

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

$$c=1 \vee c=2 \vee c=3$$

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

$$\begin{array}{l} c=1 \leftarrow\ not\ c=2,\ not\ c=3 \\ c=2 \leftarrow\ not\ c=1,\ not\ c=3 \\ c=3 \leftarrow\ not\ c=1,\ not\ c=2 \end{array}$$

Equivalently (easier) we can write the above rules using the negated atoms just defined:

$$\begin{aligned} c=1 \leftarrow\ & \mathit{not}\, \neg(c=1) \\ c=2 \leftarrow\ & \mathit{not}\, \neg(c=2) \\ c=3 \leftarrow\ & \mathit{not}\, \neg(c=3) \end{aligned}$$

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

$$1\ \{\, c=1,\ c=2,\ c=3\,\}\ 1 \leftarrow$$

For application to $\mathcal{C}+$, the last part, whether by 'shifting' or choice rule, will be done by exogeneity laws.

Application to $\mathcal{C}+$

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 `maxT` whose value is specified when `clingo` is invoked.

I am going to keep the [ ] 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

$$\begin{aligned} &\mathsf{caused}\ F\ \mathsf{if}\ G_1 \wedge \ldots \wedge G_n \\ &F[i] \Leftarrow G_1[i] \wedge \ldots \wedge G_n[i] \qquad (i \in 0\,..\,m) \\ F[T] \leftarrow\ & \mathit{not}\ \overline{G_1}[T],\ \ldots,\ \mathit{not}\ \overline{G_n}[T] \qquad (T \in 0\,..\,m) \end{aligned}$$

Constraints:

$$\begin{aligned} &\mathsf{caused}\ \bot\ \mathsf{if}\ G_1 \wedge \ldots \wedge G_n \\ &\bot \Leftarrow G_1[i] \wedge \ldots \wedge G_n[i] \qquad (i \in 0\,..\,m) \\ \leftarrow\ & \mathit{not}\ \overline{G_1}[T],\ \ldots,\ \mathit{not}\ \overline{G_n}[T] \qquad (T \in 0\,..\,m) \end{aligned}$$

Action dynamic laws

$$\begin{aligned} &\mathsf{caused}\ \alpha\ \mathsf{if}\ G_1 \wedge \ldots \wedge G_n \\ &\alpha[i] \Leftarrow G_1[i] \wedge \ldots \wedge G_n[i] \qquad (i \in 0\,..\,m{-}1) \\ \alpha[T] \leftarrow\ & \mathit{not}\ \overline{G_1}[T],\ \ldots,\ \mathit{not}\ \overline{G_n}[T] \qquad (T \in 0\,..\,m{-}1) \end{aligned}$$

Fluent dynamic laws

$$\begin{aligned} &\mathsf{caused}\ F\ \mathsf{if}\ G_1 \wedge \ldots \wedge G_k\ \mathsf{after}\ H_1 \wedge \ldots \wedge H_n \\ &F[i{+}1] \Leftarrow G_1[i{+}1] \wedge \ldots \wedge G_k[i{+}1] \wedge H_1[i] \wedge \ldots \wedge H_n[i] \qquad (\ i \in 0\,..\,m{-}1) \\ F(T{+}1) \leftarrow\ & \mathit{not}\ \overline{G_1}(T{+}1),\ \ldots,\ \mathit{not}\ \overline{G_k}(T{+}1),\ \mathit{not}\ \overline{H_1}[T],\ \ldots,\ \mathit{not}\ \overline{H_n}[T] \qquad (\ T \in 0\,..\,m{-}1) \end{aligned}$$

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

$$F(T{+}1) \leftarrow\ \mathit{not}\ \overline{G_1}(T{+}1),\ \ldots,\ \mathit{not}\ \overline{G_k}(T{+}1),\ H_1[T],\ \ldots,\ H_n[T] \qquad (T \in 0\,..\,m{-}1)$$

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

Constraints:

$$\begin{aligned} &\mathsf{caused}\ \bot\ \mathsf{if}\ G_1 \wedge \ldots \wedge G_k\ \mathsf{after}\ H_1 \wedge \ldots \wedge H_n \\ &\bot \Leftarrow G_1[i{+}1] \wedge \ldots \wedge G_k[i{+}1] \wedge H_1[i] \wedge \ldots \wedge H_n[i] \qquad (\ i \in 0\,..\,m{-}1) \\ \leftarrow\ & \mathit{not}\ \overline{G_1}(T{+}1),\ \ldots,\ \mathit{not}\ \overline{G_k}(T{+}1), \mathit{not}\ \overline{H_1}[T],\ \ldots,\ \mathit{not}\ \overline{H_n}[T] \qquad (\ T \in 0\,..\,m{-}1) \end{aligned}$$

or (simpler, but no more efficient version):

$$\leftarrow\ \mathit{not}\ \overline{G_1}(T{+}1),\ \ldots,\ \mathit{not}\ \overline{G_k}(T{+}1),\ H_1[T],\ \ldots,\ H_n[T] \qquad (T \in 0\,..\,m{-}1)$$

Exogeneity laws

For every *simple* (as opposed to 'statically determined') fluent constant f and every $v \in dom(f)$:

$$f[0]=v \Leftarrow f[0]=v$$

For every action constant a, every $v \in dom(a)$:

$$a[i]=v \Leftarrow a[i]=v \qquad (i \in 0\,..\,m{-}1)$$

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:

$$\begin{aligned} \neg(f[T]=a)\ &\leftarrow\ f[T]=b \qquad (T \in 0..m) \\ \neg(f[T]=a)\ &\leftarrow\ f[T]=c \qquad (T \in 0..m) \\ \neg(f[T]=b)\ &\leftarrow\ f[T]=a \qquad (T \in 0..m) \\ \neg(f[T]=b)\ &\leftarrow\ f[T]=c \qquad (T \in 0..m) \\ \neg(f[T]=c)\ &\leftarrow\ f[T]=a \qquad (T \in 0..m) \\ \neg(f[T]=c)\ &\leftarrow\ f[T]=b \qquad (T \in 0..m) \end{aligned}$$

Now the exogeneity laws for f in the initial state:

$$\begin{aligned} f[0]=a &\leftarrow \mathit{not}\,\neg(f[0]=a) \\ f[0]=b &\leftarrow \mathit{not}\,\neg(f[0]=b) \\ f[0]=c &\leftarrow \mathit{not}\,\neg(f[0]=c) \end{aligned}$$

Alternatively, using a choice rule and cardinality constraint:

$$1\ \{\ f[0]=a,\ f[0]=a,\ f[0]=a\ \}\ 1\ \leftarrow$$

Apparently the choice rule method is much more efficient.

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

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

$$\begin{aligned} a[T]=1 &\leftarrow \mathit{not}\,\neg(a[T]=1) && (T\in 0..m-1) \\ a[T]=2 &\leftarrow \mathit{not}\,\neg(a[T]=2) && (T\in 0..m-1) \\ a[T]=3 &\leftarrow \mathit{not}\,\neg(a[T]=3) && (T\in 0..m-1) \end{aligned}$$

or by means of a choice rule and cardinality constraint:

$$1\ \{\ a[T]=1,\ a[T]=2,\ a[T]=3\ \}\ 1\ \leftarrow \qquad (T\in 0..m-1)$$

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

$$\begin{aligned} \neg(a[T]=1) &\leftarrow a[T]=2 && (T\in 0..m-1) \\ \neg(a[T]=1) &\leftarrow a[T]=3 && (T\in 0..m-1) \\ \neg(a[T]=2) &\leftarrow a[T]=1 && (T\in 0..m-1) \\ \neg(a[T]=2) &\leftarrow a[T]=3 && (T\in 0..m-1) \\ \neg(a[T]=3) &\leftarrow a[T]=1 && (T\in 0..m-1) \\ \neg(a[T]=3) &\leftarrow a[T]=2 && (T\in 0..m-1) \end{aligned}$$

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.

$$\begin{aligned} &\mathit{toggle}\ \mathsf{causes}\ \mathit{on}\ \mathsf{if}\ \neg\mathit{on} \\ &\mathit{toggle}\ \mathsf{causes}\ \neg\mathit{on}\ \mathsf{if}\ \mathit{on} \\ &\mathit{load}\ \mathsf{causes}\ \mathit{loaded} \\ &\mathsf{inertial}\ \mathit{on} \\ &\mathsf{inertial}\ \mathit{loaded} \end{aligned}$$

Without abbreviations:

$$\begin{aligned} &\mathsf{caused}\ \mathit{on}\ \mathsf{after}\ \mathit{toggle}\wedge\neg\mathit{on} \\ &\mathsf{caused}\ \neg\mathit{on}\ \mathsf{after}\ \mathit{toggle}\wedge\mathit{on} \\ &\mathsf{caused}\ \mathit{loaded}\ \mathsf{after}\ \mathit{load} \\ &\mathsf{caused}\ \mathit{on}\ \mathsf{if}\ \mathit{on}\ \mathsf{after}\ \mathit{on} \\ &\mathsf{caused}\ \neg\mathit{on}\ \mathsf{if}\ \neg\mathit{on}\ \mathsf{after}\ \neg\mathit{on} \\ &\mathsf{caused}\ \mathit{loaded}\ \mathsf{if}\ \mathit{loaded}\ \mathsf{after}\ \mathit{loaded} \\ &\mathsf{caused}\ \neg\mathit{loaded}\ \mathsf{if}\ \neg\mathit{loaded}\ \mathsf{after}\ \neg\mathit{loaded} \end{aligned}$$

Causal theory:

$$\begin{aligned} &\mathit{on}[i{+}1] \Leftarrow \mathit{toggle}[i]\wedge\neg\mathit{on}[i] \\ &\neg\mathit{on}[i{+}1] \Leftarrow \mathit{toggle}[i]\wedge\mathit{on}[i] \\ &\mathit{loaded}[i{+}1] \Leftarrow \mathit{load}[i] \\ &\mathit{on}[i{+}1] \Leftarrow \mathit{on}[i{+}1]\wedge\mathit{on}[i] \\ &\neg\mathit{on}[i{+}1] \Leftarrow \neg\mathit{on}[i{+}1]\wedge\neg\mathit{on}[i] \\ &\mathit{loaded}[i{+}1] \Leftarrow \mathit{loaded}[i{+}1]\wedge\mathit{loaded}[i] \\ &\neg\mathit{loaded}[i{+}1] \Leftarrow \neg\mathit{loaded}[i{+}1]\wedge\neg\mathit{loaded}[i] \end{aligned}$$

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 $\mathit{on}[T]$ 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:

```
 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.

 % inertial on
 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.

 % inertial loaded
 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.

% exogeneity

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.
```

More examples to follow.

Correctness check

The correspondence between the answer sets of a logic program $lp(\Gamma^D_m)$ and the models of the causal theory Γ^D_m (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:

$$\bot \leftarrow \ not\, f[T]=v_1, ..., not\, f[T]=v_n \qquad (T \in 1..m)$$

(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:

```
:- not p(0).
:- not p(3).
:- p(6).
```

This is a little hard to read.

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

```
observations :-
   p(0), p(3), -p(6).

:- not observations.
```

(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.

```
:- not alive(0).
:- not load(0).
:- not wait(1).
:- not shoot(2).
```

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

```
ysp_constraints :-
   alive(0), load(0), wait(1), shoot(2).

:- not ysp_constraints.
```

In this problem we are interested to know whether it is possible, given the sequence of events `ysp_constraints`, that `alive(3)`. We are therefore interested in paths of length 3 (at least) and we add the further constraint:

```
:- -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:

```
:- not alive(3).
```

Example In the *Monkey and Bananas* problem, there is a monkey at location `l1`, a box at location `l2`, and some bananas at location `l3`. 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. `val(loc(monkey),0,l1)` represents that the value of fluent `loc(monkey)` at time stamp 0 is `l1`, etc. (There are other possibilities.) Fluent `hasBananas` is Boolean.

We are looking for some value of `maxT` such that:

```
monkey_plan :-
  val(loc(monkey),0,l1),
  val(loc(box),0,l2),
  val(loc(bananas),0,l3),
  -hasBananas(0),
  hasBananas(maxT).

:- not monkey_plan.
```

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

```
:- not val(loc(monkey),0,l1).
:- not val(loc(box),0,l2).
:- not val(loc(bananas),0,l3).
:- not -hasBananas(0).
:- not hasBananas(maxT).
```

I suggest that the recommended method (first version) is clearer to read.

Examples

From Exam 2004

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

inertial *status*

caused $status = on$ if $status = on$ after $status = on$

caused $status = off$ if $status = off$ after $status = off$

$status[i{+}1] = on \Leftarrow status[i{+}1] = on \wedge status[i] = on \qquad (i \in 0..m{-}1)$

$status[i{+}1] = off \Leftarrow status[i{+}1] = off \wedge status[i] = off \qquad (i \in 0..m{-}1)$

I am going to write the logic program using `clingo` syntax. For a concrete representation, `status(T,V)` represents that fluent *status* has value `V` at time index `T`.

```
status(T+1,on)  :- not -status(T+1,on), not -status(T,on),
                   T=0..maxT-1.
status(T+1,off) :- not -status(T+1,off), not -status(T,off),
                   T=0..maxT-1.
```

Or equivalently (but no more efficiently):

```
status(T+1,on)  :- not -status(T+1,on), status(T,on),
                   T=0..maxT-1.
status(T+1,off) :- not -status(T+1,off), status(T,off),
                   T=0..maxT-1.
```

We also need:

```
-status(T,on)  :- status(T,off), T=0..maxT.
-status(T,off) :- status(T,on),  T=0..maxT.
```

Note the range of time index values `T`.

Since *status* is two-valued we could have used a Boolean fluent `on(T)` and used `-on(T)` instead of `status(T,off)`. That representation would be shorter (but harder to read).

The required *exogeneity laws* are:

```
1 {status(0,on), status(0,off)} 1.
```

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

$$
\begin{aligned}
&\mathit{switch} \text{ causes } \mathit{status}=\mathit{on} \text{ if } \mathit{status}=\mathit{off}\\
&\mathit{switch} \text{ causes } \mathit{status}=\mathit{off} \text{ if } \mathit{status}=\mathit{on}\\
&\text{caused } \mathit{status}=\mathit{on} \text{ after } \mathit{switch} \wedge \mathit{status}=\mathit{off}\\
&\text{caused } \mathit{status}=\mathit{off} \text{ after } \mathit{switch} \wedge \mathit{status}=\mathit{on}\\
&\mathit{status}[i{+}1]=\mathit{on} \Leftarrow \mathit{switch}[i] \wedge \mathit{status}[i]=\mathit{off} \qquad (i \in 0..m{-}1)\\
&\mathit{status}[i{+}1]=\mathit{off} \Leftarrow \mathit{switch}[i] \wedge \mathit{status}[i]=\mathit{on} \qquad (i \in 0..m{-}1)
\end{aligned}
$$

In `clingo` syntax:

```
status(T+1,on)  :- not -switch(T), not -status(T,off),
                   T=0..maxT-1.
status(T+1,off) :- not -switch(T), not -status(T,on),
                   T=0..maxT-1.
```

The exogeneity laws:

```
1 {switch(T), -switch(T)} 1 :- T=0..maxT-1.
```

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

```
-switch(T) :- not switch(T), 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 $\mathit{status}=\mathit{off}$.

$$
\begin{aligned}
&\text{nonexecutable } \mathit{open} \text{ if } \mathit{status}=\mathit{off}\\
&\text{caused } \bot \text{ after } \mathit{open} \wedge \mathit{status}=\mathit{off}\\
&\bot \Leftarrow \mathit{open}[i] \wedge \mathit{status}[i]=\mathit{off} \qquad (i \in 0..m{-}1)
\end{aligned}
$$

The logic program:

```
:- not -open(T), not -status(T,off), T=0..maxT-1.
```

Equivalently, but no more efficiently:

```
:- open(T), status(T,off), T=0..maxT-1.
```

Exogeneity laws:

```
1 {open(T), -open(T)} 1 :- T=0..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 $\neg\mathit{closed}$ 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.

$$
\begin{aligned}
&\mathit{push} \text{ causes } \neg\mathit{closed} \text{ if } \mathit{closed}\\
&\text{nonexecutable } \mathit{push} \text{ if } \neg\mathit{closed}\\
&\text{caused } \neg\mathit{closed} \text{ after } \mathit{push} \wedge \mathit{closed}\\
&\text{caused } \bot \text{ after } \mathit{push} \wedge \neg\mathit{closed}\\
&\neg\mathit{closed}[i{+}1] \Leftarrow \mathit{push}[i] \wedge \mathit{closed}[i] \qquad (i \in 0..m{-}1)\\
&\bot \Leftarrow \mathit{push}[i] \wedge \neg\mathit{closed}[i] \qquad (i \in 0..m{-}1)
\end{aligned}
$$

```
-closed(T+1) :- not -push(T), not -closed(T), T=0..maxT-1.

:- not -push(T), not closed(T), T=0..maxT-1.

% exogeneity fluent closed
1 {closed(0), -closed(0)} 1.

% exogeneity action push
1 {push(T), -push(T)} 1 :- T=0..maxT-1.
```

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

```
-closed(T) :- not closed(T), T=0..maxT.
```

(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.

$$
\begin{aligned}
&\text{caused } \mathit{closed} \text{ if } \mathit{closed} \text{ after } \mathit{closed} \qquad \textit{(inertial)}\\
&\text{caused } \mathit{closed} \text{ if } \mathit{closed} \text{ after } \neg\mathit{closed} \qquad \textit{(not inertial)}\\
&\mathit{closed}[i{+}1] \Leftarrow \mathit{closed}[i{+}1] \wedge \mathit{closed}[i] \qquad (i \in 0..m{-}1)\\
&\mathit{closed}[i{+}1] \Leftarrow \mathit{closed}[i{+}1] \wedge \neg\mathit{closed}[i] \qquad (i \in 0..m{-}1)
\end{aligned}
$$

```
closed(T+1) :- not -closed(T+1), not -closed(T), T=0..maxT-1.
closed(T+1) :- not -closed(T+1), not closed(T), T=0..maxT-1.
```

The same effect could also be obtained simply as

$$\text{caused } \mathit{closed} \text{ if } \mathit{closed} \text{ after } \top$$

```
closed(T+1) :- not -closed(T+1), T=0..maxT-1.
```

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

$$
\begin{aligned}
&\text{default } \mathit{closed}\\
&\text{caused } \mathit{closed} \text{ if } \mathit{closed}
\end{aligned}
$$

```
closed(T) :- not -closed(T), T=0..maxT.
```

Example

(Used earlier in the $\mathcal{C}+$ notes) 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 $dest$
The domain of $walk(x)$ and $drive(x)$ are 'destinations' not locations:
$dom(walk(x)) = dom(drive(x)) = \{home, work, pub, none\}$.
$drive(x)=p$ when $loc(x)=p$ means that x drives around and ends up back where he/she started. And similarly for $walk(x)$.

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

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

The last line is shorthand for the following $\mathcal{C}+$ laws:

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

In `clingo`:

```
agent(a).          location(home).
agent(b).          location(work).
agent(c).          location(pub).

destination(none).
destination(X) :- location(X).
```

In what follows the chosen `clingo` representation of multi-valued fluents and action constants 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.


```
% ---- 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.

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.
```