Question 1 (based on a fragment from 2007 exam)

How are the following expressed as static and/or fluent dynamic laws of the action language $C+$? (Write the laws in the general form, without using any $C+$ abbreviations such as ‘inertial’ and ‘causes’.) In each case, what is the translation to (time-stamped) causal rules?

1. The default value of fluent $f$ is $v$.
2. The fluent $g$ is inertial. ($g$ is not necessarily Boolean.)
3. There is no state in which a particular (spring-loaded) door is both open and closed (though it is always one or the other).
4. The action of pushing the door causes it to become open if it is closed; pushing the door is not possible (executable) if the door is open.
5. 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.

Question 2 (Yale Shooting Problem, with targets)

Consider the following variant of the ‘Yale Shooting Problem’ (YSP). There are actions of loading a gun, aiming the gun at a person $x$, who then becomes the ‘target’, and shooting (i.e., pulling the trigger). Shooting a loaded gun when it is aimed at $x$ results in $x$ being dead (not alive). Shooting the gun also unloads it. The gun can be shot when it is not loaded but that has no effect, whether the target is alive or not.

Formulate this version of the YSP as an action description in the language $C+$.

Use a multivalued fluent $\text{target}=x$. You should allow for the possibility that there is no target (e.g., by including $\text{none}$ in the domain of $\text{target}$). For the action of aiming, you can either use Boolean action constants $\text{aim}(x)$ or a single multi-valued action constant $\text{aim}$ whose domain is the same as that of $\text{target}$. (It doesn’t make much difference.)

Assume for simplicity that targets do not move. Aiming the gun at $x$ means pointing it at $x$. And loading it means ensuring it is loaded: one can ‘load’ a gun that is already loaded.

Further: a gun cannot be loaded and aimed at the same time; a gun cannot be shot and aimed at the same time.

How does one formulate a query to determine whether the sequence of actions $\text{aim at } a$, $\text{load}$, $\text{aim at } b$, $\text{shoot}$ results in fluent $\text{alive}(b)$ being false when executed in an initial state where the gun is not loaded?

Suppose the (anonymous) shooter is happy if (and only if) everyone is dead.

How would you get a ‘plan’ for the goal where the shooter is happy?

Use a Boolean statically determined fluent $\text{happy}$.

Question 3 (Yale Shooting Problem, with lights)

Suppose there is also a light (represented by the Boolean fluent $\text{on}$).

- Shooting a gun when it is loaded kills the target, but only if the light is on. If the light is off, it is possible to shoot the gun but it will have no effect on the target, loaded or not.
- It is possible to load the gun while the light is off but not possible to aim the gun while the light is off.
- Whenever the light is off, there is no target ($\text{target}=\text{none}$).

Consider two variations: there is a Boolean action $\text{toggle}$: it switches the light to on if it is off, and to off if it is on.

Another possibility: the status of the light is exogenous.

Question 4 (Yale Shooting Problem, trigger-happy)

Suppose the person with the gun is trigger-happy: whenever the gun is loaded, this person shoots, whether there is currently a target or not.

Question 5

Make up your own variations. For instance . . .

Yale Shooting Problem, with moving targets. Suppose that targets can move. There are (Boolean) action constants $\text{move}(x)$. They represent that $x$ moves (it does not matter where).

What happens if the target moves? What happens if the target moves as the gun is being shot?
The above are abbreviations for the fluent dynamic laws:

\[
\text{caused } \text{door} = \text{open} \text{ if } \top \text{ after push} \land \text{door} = \text{closed}
\]

\[
\text{caused } \bot \text{ if } \top \text{ after push} \land \text{door} = \text{open}
\]

which translate to (for paths of length \(m\)):

\[
door[i+1] = \text{open} \iff \text{push}[i] \land \text{door}[i] = \text{closed} \quad (0 \leq i < m)
\]

\[
\bot \iff \text{push}[i] \land \text{door}[i] = \text{open} \quad (0 \leq i < m)
\]

(5) 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.

One way, writing down what is stated above:

\[
\text{caused } \text{door} = \text{closed} \text{ if } \text{door} = \text{closed after } \text{door} = \text{closed}
\]

\[
\text{caused } \text{door} = \text{closed} \text{ if } \text{door} = \text{closed after } \text{door} = \text{open}
\]

which translate to (for paths of length \(m\)):

\[
door[i+1] = \text{closed} \iff \text{door}[i+1] = \text{closed} \land \text{door}[i] = \text{closed} \quad (0 \leq i < m)
\]

\[
door[i+1] = \text{closed} \iff \text{door}[i+1] = \text{closed} \land \text{door}[i] = \text{open} \quad (0 \leq i < m)
\]

You might guess that the above are equivalent to

\[
door[i+1] = \text{closed} \iff \text{door}[i] = \text{closed} \lor (\text{door}[i] = \text{closed} \land \text{door}[i] = \text{open}) \quad (0 \leq i < m)
\]

and hence to

\[
door[i] = \text{closed} \iff \text{door}[i] = \text{closed} \quad (0 < i \leq m)
\]

If you thought that, you would be correct. Since we also have the ‘exogeneity’ law for the simple fluent \text{door} at timestamp 0:

\[
door[0] = \text{closed} \iff \text{door}[0] = \text{closed}
\]

we get

\[
door[i] = \text{closed} \iff \text{door}[i] = \text{closed} \quad (0 \leq i \leq m)
\]

which in C+ is just

\[
\text{default } \text{door} = \text{closed} \quad \text{(caused } \text{door} = \text{closed if } \text{door} = \text{closed)}
\]

The exam question did not expect such a detailed answer of course. Either of

\[
\text{caused } \text{door} = \text{closed if } \text{door} = \text{closed after } \text{door} = \text{closed}
\]

\[
\text{caused } \text{door} = \text{closed if } \text{door} = \text{closed after } \text{door} = \text{open}
\]

or

\[
\text{default } \text{door} = \text{closed} \quad \text{(caused } \text{door} = \text{closed if } \text{door} = \text{closed)}
\]

would be enough, without further comment.
Question 2 In what follows $x$ ranges over agent names $a, b, c, \ldots$. 

Fluent constants: alive($x$), loaded (Boolean), target (domain: $a, b, c, \ldots$ none).

Action constants: load, shoot, aim($x$) (all Boolean).

Instead of many Boolean constants aim($x$) you might prefer to use a single constant aim with domain the same as target: $a, b, c, \ldots$ none.

- inertial target
- inertial loaded
- inertial alive($x$)

Instead of inertial alive($x$) you might prefer to say that alive($x$) persists by default but not alive($x$) persists, and no default about it:

caused alive($x$) if alive($x$) after alive($x$)
caused ~alive($x$) after ~alive($x$) (once you are dead, you are dead)

It doesn’t make any difference in this example. (Because there is no action which brings a dead $x$ back to life.)

And the rest:

- load causes loaded
- aim($x$) causes target=$x$(or aim=$x$ causes target=$x$)
- shoot causes ~alive($x$) if loaded $\land$ target=$x$
- shoot causes ~loaded
- nonexecutable load $\land$ aim($x$) (or nonexecutable load $\land$ aim=$x$)
- nonexecutable shoot $\land$ shoot (or nonexecutable shoot $\land$ aim=$x$)

You might perhaps prefer to write the shoot causes ... rule like this:

- shoot causes ~alive($x$) if loaded $\land$ target=$x$ $\land$ ~alive($x$)

Personally I don’t like these references to ‘causes’ anyway. If you write it out as a fluent dynamic law in full you get

caused ~alive($x$) if $\top$ after loaded $\land$ target=$x$

which makes sense whether or not you add the extra $\land$ ~alive($x$) condition to the body. (It makes no difference in this example.)

Note that: use of a single multi-valued action constant aim with the same domain as target automatically builds in the constraint that one can’t aim at two different targets simultaneously. I chose to use (Boolean) action constants aim($x$) instead. You might think that we would therefore need to add some further constraints, viz:

nonexecutable aim($x$) $\land$ aim($y$) if $x \neq y$

It would not be wrong to do this — indeed, I included them in my first attempt — but then I noticed that they are unnecessary. This is because aim($x$) causes target=$x$ and aim($y$) causes target=$y$; but target can’t have two different values simultaneously, so there can’t be a transition with aim($x$) and aim($y$) for any $x \neq y$.

For the query: let $YSP_4$ be the causal theory corresponding to paths of length 4 in the C+ action description above. We want to know whether

\[ \text{comp}(YSP_4) \models \text{aim}(a)[0] \land \text{aim}(b)[0] \land \text{load}[0] \land \text{aim}(a)[1] \land \text{aim}(b)[2] \land \text{shoot}[3] \rightarrow \text{alive}[4] \]

that is, whether

\[ \text{comp}(YSP_4) \cup \{ \text{aim}(a)[0] \land \text{aim}(b)[0] \land \text{load}[0] \land \text{aim}(a)[1] \land \text{aim}(b)[2] \land \text{shoot}[3] \land \text{alive}[4] \} \]

is satisfiable. (If it is not satisfiable then the answer to the query is ‘yes’; if it is satisfiable then the answer to the query is ‘no’ because we have a counter-example.)

If we ask whether

\[ \text{comp}(YSP_4) \cup \{ \text{aim}(a)[0] \land \text{aim}(b)[0] \land \text{load}[0] \land \text{aim}(a)[1] \land \text{aim}(b)[2] \land \text{shoot}[3] \land \text{~alive}[4] \} \]

is satisfiable we find all paths at which $b$ is not alive after the third transition.

If we ask whether

\[ \text{comp}(YSP_4) \cup \{ \text{aim}(a)[0] \land \text{aim}(b)[0] \land \text{load}[0] \land \text{aim}(a)[1] \land \text{aim}(b)[2] \land \text{shoot}[3] \} \]

is satisfiable we discover (amongst other things) whether alive($b$) is true or false at time 4. I tried the above query in sCCalc. All paths of length 4 had ~alive($b$)[4] as expected, but there were many such paths, depending on how many potential targets there are. For instance, with dom(target) = $\{a, b, c, \text{none}\}$ there are two paths: one in which $c$ is alive at time 0 and then throughout, and one in which $c$ is not alive at time 0, and throughout. With another target $d$ we get four paths, with another target $e$ we get eight paths, and so on.

To get rid of all these combinations I defined a (statically determined) Boolean fluent:

- default all_alive
- caused ~all_alive if ~alive($x$)

Defined this way, all_alive does not depend on how many potential targets there are: $x$ ranges over $a, b, c, \ldots$.

Now I ask for models of

\[ \text{comp}(YSP_4) \cup \{ \text{all_alive}[0] \land \text{aim}(a)[0] \land \text{load}[1] \land \text{aim}(b)[2] \land \text{shoot}[3] \} \]

sCCalc produced just one path, no matter how many potential targets were specified, with ~alive($b$)[4] in each.
Planning example: happy is a Boolean statically determined fluent constant.

\[
\text{default \ happy} \\
\text{caused } \neg \text{happy if alive}(x)
\]

As for \(\forall x \text{alive} \), this definition does not depend on how many potential targets there are: \(x\) ranges over \(a, b, \ldots\) (\(\text{happy}\) is not equivalent to \(\forall x \text{alive} \)).

For the ‘plan’, pick some maximum length \(m\) for runs/paths/traces and then ask iteratively whether:

\[
\text{comp}(\Gamma_k^{\text{YS}}) \cup \{\neg \text{loaded}[0] \wedge \text{happy}[k]\}
\]

is satisfiable, for \(k = 0 \ldots m\).

I tried this in iCCalc, with a guessed value of \(m = 10\), and with \(\text{dom}(\text{target}) = \{a, b, c, \text{none}\}\). iCCalc produced four plans of length 0 — obvious in hindsight. All of them had happy (everyone dead) in the initial state, and differed only in the value of \(\text{target}\) in the initial state.

So then I asked for models of

\[
\text{comp}(\Gamma_k^{\text{YS}}) \cup \{\text{all}_{\text{alive}}[0] \wedge \neg \text{loaded}[0] \wedge \text{happy}[k]\}
\]

iteratively, for \(k = 0 \ldots 10\). With \(\text{dom}(\text{target}) = \{a, b, c, \text{none}\}\) as before this produced 24 solutions at time=8; again, many differed simply in the value of \(\text{target}\) in the initial state. Again, obvious in hindsight. Also obvious in hindsight is that none of these plans had \(\text{target} = \text{none}\) in the initial state — \(\text{target} = \text{none}\) in the initial state would force one more action to aim the gun, making the plan 9 steps long instead of 8.

So then finally I tried finding models of

\[
\text{comp}(\Gamma_k^{\text{YS}}) \cup \{\text{target}[0] = \text{none} \wedge \text{all}_{\text{alive}}[0] \wedge \neg \text{loaded}[0] \wedge \text{happy}[k]\}
\]

iteratively, for \(k = 0 \ldots 10\). Now, for targets \(a, b, c\), there are 48 solutions, of length 9. Why so many solutions? Partly it is because the targets can be shot in any order: with three targets \(a, b, c\) to kill, there are 6 possible permutations. The other variation is whether, for each victim, one first loads the gun then aims, or first aims and then loads the gun. To see the effect suppose we add one further constraint:

\[
\text{caused shoot if loaded}
\]

This is an action dynamic law which says that the gun is fired whenever (as soon as) \(\text{loaded}\) is true. One could also write instead (equivalently):

\[
\text{nonexecutable } \neg \text{shoot if loaded}
\]

With this extra constraint, and targets \(a, b, c\), there are indeed 6 solutions, all of length 9. With targets \(a, b, c, d\), I got 24 = 4! solutions, of length 12. With targets \(a, b, c, d, e\), I got 120 = 5! solutions, of length 15. And so on.

(Naturally I wouldn’t expect you to do these calculations without access to something like iCCalc. Many of these points — for instance how to formulate the initial state — only become obvious when we try it and see what we get.)

**Question 3** \(x\) ranges over agent names \(a, b, c, \ldots\). The signature is as in Question 2, but we add a fluent constant \(\text{light}\) (domain: \(\text{on, off}\)). (Or, if you prefer, use a Boolean fluent \(\text{on}\). It doesn’t make any difference.)

\[
\text{inertial target} \\
\text{inertial loaded} \\
\text{inertial alive}(x) \\
\text{inertial light} \\
\text{load causes loaded} \\
\text{aim}(x) \text{ causes target} = x \\
\text{shoot causes } \neg \text{alive}(x) \text{ if loaded } \wedge \text{target} = x \wedge \text{light} = \text{on} \%
\]

\text{modified} \%

we don’t need that shooting with light off has no effect.

\[
\text{nonexecutable load } \wedge \text{aim}(x) \\
\text{nonexecutable load } \wedge \text{shoot} \\
\text{nonexecutable aim}(x) \text{ if light} = \text{off}
\]

Now for first version we add a (Boolean) action constant \(\text{toggle}\) and

\[
\text{toggle causes light} = \text{on} \quad \text{if light} = \text{off} \\
\text{toggle causes light} = \text{off} \quad \text{if light} = \text{on}
\]

For the second version, instead of the action \(\text{toggle}\) we make \(\text{light}\) ‘exogenous’: its value varies from state to state but we do not specify how — that is outside the system being modelled. \(\text{exogenous light}\) is shorthand for the pair of \(C+\) laws:

\[
\text{caused light} = \text{on} \quad \text{if light} = \text{on} \\
\text{caused light} = \text{off} \quad \text{if light} = \text{off}
\]

And we have to get rid of \(\text{inertial light}\) (otherwise it would never change from state to state.)

**Question 4** (‘trigger-happy’)

This was already done in the discussion of Question 2. Add either the action dynamic law

\[
\text{caused shoot if loaded}
\]

or equivalently (in this example) the fluent dynamic law

\[
\text{nonexecutable shoot if } \neg \text{loaded} \quad (\text{shoot causes } \bot \text{ if } \neg \text{loaded})
\]
Question 5 (‘moving targets’)

There are many possible variations. What I had in mind was something like this:

\[ \text{move}(x) \text{ causes } target = \text{none} \text{ if } target = x \]

The above does not deal with the case where the current target \( x \) moves, and in moving exposes some other target \( y \) standing behind. Not does it deal with the possibility that \( y \) moves into the line of fire.

So perhaps, instead of the above:

\[ \text{move}(x) \text{ may cause } target = y \] \text{ (for all } y \in \text{dom}(target), \text{ including } y = \text{none})

that is, in full:

\[ \text{target} = y \text{ if } target = y \text{ after } \text{move}(x) \]

Similarly, the effects of shooting have to be adjusted. We could try, for example

\[ \text{shoot} \land \neg \text{move}(x) \text{ causes } \neg \text{alive}(x) \text{ if } \text{loaded} \land target = x \]

that is, in full:

\[ \text{caused} \neg \text{alive}(x) \text{ if } \top \text{ after } \text{shoot} \land \neg \text{move}(x) \land \text{loaded} \land target = x \]

But again that does not deal with the possibility that someone else moves into the line of fire, or is exposed when \( x \) moves.

So perhaps better (\( y \) ranges over \( a, b, \ldots ):\)

\[ \text{caused} \text{ no-one-moves if } \text{no-one-moves} \]
\[ \text{caused} \neg \text{no-one-moves if } \text{move}(x) \]
\[ \text{shoot} \land \neg \text{no-one-moves} \text{ causes } \neg \text{alive}(x) \text{ if } \text{loaded} \land target = x \]
\[ \text{shoot} \land \text{move}(y) \text{ may cause } \neg \text{alive}(x) \text{ if } \text{loaded} \land \text{alive}(x) \]

that is, in full:

\[ \text{caused} \neg \text{alive}(x) \text{ if } \top \text{ after } \text{shoot} \land \neg \text{no-one-moves} \land \text{loaded} \land target = x \]
\[ \text{caused} \neg \text{alive}(x) \text{ if } \neg \text{alive}(x) \text{ after } \text{shoot} \land \text{move}(y) \land \text{loaded} \land \text{alive}(x) \]

You can, if you prefer, add another condition \( \neg \text{alive}(x) \) to the first of the laws above, and in the second write \( \neg \text{no-one-moves} \) in place of \( \text{move}(y) \).