RIPN system construction rules

Concerning the TR programmes

INT or REAL variables are considered
A section is included first, before the rules, where the following statements are made:

- **FACTS:** statement of facts, indicating whether they have parameters or not, e.g., open_clamp, veo_robot(INT, INT), all separated by ';'. They shall be added using remember, e.g. remember(veo_robot(5,6)), and deleted using forget, e.g. forget(open_clamp).
- **VARSINT:** declaration of integer variables separated by ';'.
- **VARSREAL:** declaration of real variables separated by ';'.
- **DISCRETE:** discrete actions, with or without parameters, e.g. open(INT, INT), close().
- **DURATIVE:** durative actions, with or without parameters, same syntax as discrete, e.g. rotate(), move(REAL).
- **TIMERS:** declaration of timers, e.g. timer1.
- **INIT:** initial values to integer or real variables using the symbol ":=". For example: x:=1, y:=5.66. If not specified, a value of 0 is assumed.

This is followed by the block delimited by the "<TR>" mark where the rules are specified, from highest priority (top) to lowest. The rules are constructed with the format:

```
<TR>
condition -> action(s) [changes in the BeliefStore, separated by ';']
```

Conditions are logical expressions with the usual operators (&&, ||, !) using BeliefStore variables and facts. Changes in the BeliefStore can be changes to variables with the usual arithmetic expressions (+,-,*,/) and/or adding or deleting facts (remember/forget). It is possible to use the wildcard character "_" in the facts both in the conditions and in the "forget" operation (which would delete all occurrences matching the indicated pattern).

Declared timers can be started ("start", with seconds in brackets), paused ("pause"), continued ("continue") or stopped ("stop"). When a timer ends, the fact ".end" is automatically added to the BeliefStore, can be used in the conditions and can be removed manually, "forget(t1.end)" in the case of a timer named "t1".

For example:

```
FACTS: seen(INT); done()
VARSINT: x; y;
VARSREAL: z
DISCRETE: open(INT, REAL)
DURATIVE: rotate(INT); close()
TIMERS: t1
INIT: x:= 0; z:=4.5

<TR>
seen(3) && x==2 && t1.end-> open(2, 4.5); rotate() [forget(seen(_)])
seen(_) && x==4 -> open(3, 6.6)
y<5 -> t1.continue(); close() [x:=3]; y<5 -> t1.continue(); close() [x:=3].
x> 1 -> rotate(3); close() [y:=4; remember(seen(3)); ]
x==0 -> t1.start(4)
True -> act1() [y:=2; z:=z+1].
```

The system is continuously evaluating the highest priority rule to be executed. There must always be at least one that meets the condition (hence the lowest one is usually set to True).

Durative actions are running as long as the rule that initiated them is active.

Discrete actions are triggered upon each activation of the rule. But it requires another rule to gain control and the first rule to re-activate for the execution of the discrete action to be triggered again. Operations on timers are considered discrete. Starting a timer that is already running removes the previous instance of the timer.

Concerning Petri Nets

The initial declaration section is the same as for the TR program. Each part of the system (Petri net or TR program), must declare what it uses in its context. If a variable/fact/action/timer is declared in both the Petri net and the TR program, it is because it is the same variable/fact/action/timer, so that changes made by one part of the system are visible from the other part.

After the part of the basic declarations, a Petri net part is included where places, transitions, arcs and initial marking are indicated. Places, transitions and arcs are separated by ";". An arc is indicated by "place->transition". If the arc is inhibiting, "-0>" is indicated. The initial marking must give a value (0 or 1) to each place.

```
FACTS:
VARSINT: x; y
VARSREAL:
INIT:
DISCRETE: open(INT, REAL)
DURATIVE:
TIMERS: t1

PLACES: p0; p1
TRANSITIONS: t0; t1
ARCS: p0->t0; t0->p1; p1->t1; t1->p0
```

```
INITMARKING: (1,0)

<PN>
p0: [open(3,4); t1.start(8); x:=1].
t0: [] if(y==2)
p1: [x:=2]
```

After the declarations, a block delimited by the "<PN>" mark specifies the behaviour. The changes that are carried out in the BeliefStore are indicated for the appropriate places with the same syntax that we discussed for the TR programs but also adding the actions (durative or discrete) and the handling of the timers. The conditions follow the same considerations as for TR programs:

```
place_name: [operations on the BeliefStore or durative/discrete actions/timers,
separated by ";"] if(condition on the BeliefStore)
```

Not all places need to be listed in this section, only those where changes are made to the BeliefStore and/or actions are initiated. Durative actions are running while the place is marked. Discrete actions only when the place loses the marking and regains it (same analogy as with the TR programme rules).

The transitions we call input are also specified:

```
transition_name: [operations on BeliefStore, separated by ";"] if(condition on
BeliefStore)
```

Transitions that are not listed in the section are immediate calls and have priority over incoming calls. As long as the system has not triggered all the immediate ones enabled to do so (their entry locations are marked and the exit locations have no tokens) it will continue to evolve the network. Once all of the immediate ones have been triggered, the system is ready to evaluate the triggering of the incoming transitions, which are those listed in the <PN> block. The system will fire the transitions that have a condition that is met given the state of the BeliefStore. In these transitions only changes can be performed on the BeliefStore, no actions can be performed and no timers can be managed.

Regarding the outputs

The system provides for three output files whose names can be specified in the class included in MainUnified.java:

- One corresponding to a dump of the state of the BeliefStore by the Petri net and the TR programme, each at the completion of each execution cycle.

- One corresponding to the execution trace of the TR program, indicating the rules that are being activated and possible errors detected both in the loading of the TR program from the input file and in the execution.
- One corresponding to the evolution of the Petri net marking and possible incidents that may occur during implementation.

It is possible to have all outputs displayed by indicating it in the main program.

It is also possible to run only the Petri net or only the TR program for debugging purposes with outputs in the console, for this purpose, we have included two files with the main method for each part.

By default it is in main where the outputs resulting from the execution as a RIPN system are shown: discrete actions triggered and durative actions started or stopped. All other things that happen in the system are plotted in the files mentioned above because they do not correspond to the effect of the system execution. For a real system deployment, those messages that are received in main as the observer object that it is, must be passed on to the corresponding physical system devices and actuators, making the appropriate modifications to the code. Having decoupled it following the observer pattern, the change is as simple as passing a different observer to both the network and the TR program instead of being the object included in the main method.

Work in progress

It remains to be implemented the reception of events from the environment that enable the triggering of the transitions called input. In the current version there are no such events and the immediate transitions only depend on the fulfilment of the condition evaluated on the BelieStore.
This functionality is scheduled to be completed in the next two months (June 2025).

If you have any comments or questions, please write to pedro.sanchez@upct.es
Polytechnic University of Cartagena, Spain.