APRON case study: Subway speed regulation system

1 Presentation

This example is extracted from an actual proposal for an automatic subway. It concerns a (simplified version of a) speed regulation system avoiding collision. Each train detects beacons that are placed along the track, and receives the "second" from a central clock. Ideally, a train should encounter one beacon each second. So the space left between beacons rules the speed of the train.

Now, each train T_i adjusts its speed as follows: Let $\#\mathrm{b}_i$ and $\#\mathrm{s}$ be respectively the number of encountered beacons and the number of received seconds (since the second signal is instantly broadcast, $\#\mathrm{s}$ is the same for all trains).

> when $\#\mathrm{b}_i \geq \#\mathrm{s} + 10$, the train notices it is early, and puts on the brake as long as $\#\mathrm{b}_i > \#\mathrm{s}$. Continuously braking makes the train stop before encountering 10 beacons.
>
> when $\#\mathrm{b}_i \leq \#\mathrm{s} - 10$, the train is late, and will be considered late as long as $\#\mathrm{b}_i < \#\mathrm{s}$. A late train signals it to the central clock, which does not emit the "second" as long as at least one train is late.

2 Some results with LRA

[HPR97] gives the results of Linear Relation Analysis applied to Lustre programs simulating systems with one and two trains. The Lustre compiler was used first to build an explicit control automaton (i.e., all boolean variables were compiled into the control structure). Fig. 1 shows the results for one train. For two trains, the analysis shows that the difference $\#\mathrm{b}_1 - \#\mathrm{b}_2$ of the number of beacons encountered by each train remains in the interval $[-29, +29]$. So, if they are initially separated by more than 29 beacons, no collision can occur.

3 A generic program

We give the program in pseudo-C, parameterized by the number of trains. The state of each train belongs to an enumerated type {ONTIME, ONBRAKE, STOPPED, LATE}.

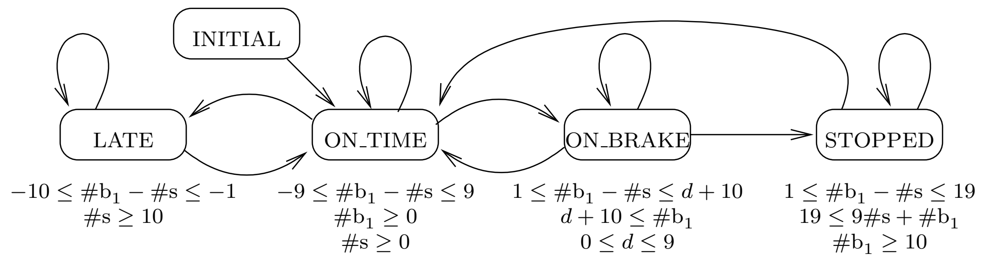

Figure 1: Result of the subway example, with one train

```
// Initializations
int nbbeacons[nbtrains], nbbeaconsonbrake[nbtrains];
state state[nbtrains];
int nbsec = 0;
for(i=0; i<nbtrains; i++){ nbbeacons[i]=0; state[i]=ONTIME; }
// Infinite loop
for(;;){
   // The central clock
   // if no train is late, it may send a second
   bool timeruns = 1;
   for(i=0; i<nbtrains; i++) if(state[i]==LATE){timeruns = 0; break;}
   if(timeruns) if(random()) nbsec++;
   // The trains
   for(i=0; i<nbtrains; i++){
      // if not stopped, may perceive a beacon
      if(state[i]<>LATE)  if(random()) nbbeacons[i]++;
      int diff = nbbeacons[i] - nbsec;
      // The automaton
      switch(state[i]){
         case ONTIME :{
            if(diff >= 10) {
              state[i] = ONBRAKE; nbbeaconsonbrake[i]=0;
            } break;
         }
         case ONBRAKE : { nbbeaconsonbrake[i]++;
            if(diff < 0) state[i] = ONTIME;
            if(nbbeaconsonbrake[i] >=10) state[i] = STOPPED;
            break;
         }
         case STOPPED : {
            if(diff < 0 ) state[i] = ONTIME; break;
         }
         case LATE : {
            if(diff >0)  state[i] = ONTIME;
            break;
         }
      } // automaton
   } // loop on trains
} // infinite loop
```

References

[HPR97] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.