Proceedings of ITC-CSCC 2000, Pusan, Korea

Specification and verification of a single-track railroad signaling in CafeOBJ

Takahiro Seino, Kazuhiro Ogata, and Kokichi Futatsugi
Graduate School of Information Science, JAIST
1-1 Asahidai, Tatsunokuchi, Ishikwa 923-1292, JAPAN
{t-seino, ogata, kokichi}@jaist.ac.jp

Summary. A signaling system for a single-track railroad has been specified in CafeOBJ. In this paper, we describe the specification of arbitrary two adjacent stations connected by a single line that is called a *two-station system*. The system consists of two stations, a railroad line (between the stations) that is also divided into some contiguous sections, signals and trains. Each object has been specified in terms of their behavior, and by composing the specifications with projection operators the whole specification has been described. A safety property that more than one train never enters a same section simultaneously has also been verified with CafeOBJ.

1 Introduction

Since key industrial systems such as railroad signaling systems and aviation control systems heavily affect people's lives, we must improve their safety as much as possible. We do not think that we can improve their safety in an ad hoc way because the systems are complex as well as huge. It is one possible approach to improving their safety that we formally specify the systems and verify some properties that the systems should have based on the formal specifications.

Formal specification languages in which we can formally specify systems and with which we can formally verify their properties have been proposed. CafeOBJ [4] is one of them. CafeOBJ allows us to specify state machines or objects of object-oriented systems in terms of their behavior.

We believe that case studies that we formally specify and verify some systems have to be done so that we can improve specification and verification techniques with formal specification languages such as CafeOBJ, and also make the languages easier to use. Therefore, as a case study we have done the following experiment. We have specified a kind of railroad signaling systems in CafeOBJ, and have formally verified the system has an important safety property based on the formal specification with the help of the CafeOBJ system.

Railroad systems usually adopt block systems so as to protect collisions between trains [9]. In block systems, railroad lines are partitioned into contiguous sections, in each of which at most one train is allowed to be. Railroad signaling systems are designed to aim at (semi-)automatically implementing block systems. We have dealt with a single-track railroad system that consists of a straight line on which more than one station is located. In this paper, we describe the specification of arbitrary two adjacent stations connected by a single line that is called a *two-station system* and the verification that no collision occurs.

The rest of the paper is organized as follows. Sect. 2 mentions CafeOBJ and how to specify systems in CafeOBJ and verify their properties with CafeOBJ. Sect. 3 describes two-station systems, their specification in CafeOBJ, and the verification with CafeOBJ that the systems have a safety property that more than one train never enters a same section simultaneously. In Sect. 4, we introduce some related works, and we finally conclude the paper in Sect. 5.

2 CafeOBJ in a nutshell

CafeOBJ [4] is a direct successor of OBJ3 [7] that is one of the best-known algebraic specification languages. One of the outstanding features of CafeOBJ is that we can specify state machines or objects naturally, which were supposed to be difficult to specify in algebraic specification languages. The point is hidden algebra [6], with which we specify objects in terms of their behavior. There are two kinds of sorts in hidden algebra: *hidden* and *visible* sorts. A hidden sort represents the state space of an object, and a visible one usual data such as integers. There are also two kinds of operations: *action* and *observation* operations. An action operation may change the state of an object, and the state of an object can be only observed with observation ones. We use *projection* operations to combine specifications for component systems and build a specification for a compound system.

We show a specification for *fields of radio buttons* as an example. Fig. 1 shows a field of radio buttons consisting of three buttons. We can use fields of radio buttons to exclusively choose one among the buttons. We first show the signature of a specification for but-

Figure 1: Fields of radio buttons.

tons from which fields of radio buttons are made:

```
op   init   : Bool -> Btn  -- initial state.
bops on off : Btn -> Btn   -- actions.
bop  on?    : Btn -> Bool  -- observation.
```

Btn is a hidden sort representing the state space of each button, and Bool is a (built-in) visible sort representing boolean values. Operator init takes a boolean value, representing the initial state of a button. Action operators on and off can select and disselect a button, respectively. Observation operator on? allows us to observe the state of a button, i.e. selected or disselected represented by true or false. We use equations to define what happens next after applying an action operator to a button. The equations for buttons are as follows:

```
eq on? (init (B:Bool)) = B .
eq on? (on (S:Btn))  = true .
eq on? (off (S:Btn)) = false .
```

B and S are variables whose sorts are Bool and Btn, respectively. The first equation means that the initial state of a button is what is given to the button as its argument. The second (or third) equation means that the state of a button is changed to true (or false), i.e. selected (or disselected), after applying on (or off) to the button.

We next show the signature of a specification for fields of radio-buttons:

```
op   init : -> RdBtn               -- initial state.
bop  on   : BtnID RdBtn -> RdBtn   -- action.
bop  on?  : BtnID RdBtn -> Bool    -- observation.
bop  btn  : BtnID RdBtn -> Btn     -- projection.
```

Hidden sort RdBtn represents the state space of fields of radio buttons. Visible sort BtnID represents IDs for each button. We use action operator on, observation operator on?, and projection operator btn to select, observe, and obtain a button whose ID is given as its first argument, respectively. Observation operator on? for fields of radio buttons is defined with observation operator on? for component buttons as follows:

```
eq  on? (BTN, R) = on? (btn (BTN, R)) .
```

where BTN and R are variables whose sorts are BtnID and RdBtn, respectively. The following equation means that every button of fields of radio buttons is initially disselected:

```
eq  btn (BTN, init) = init (false) .
```

The following two equations mean that if a button is selected, any other button is disselected:

Figure 2: UML object diagram for fields of radio-buttons.

```
ceq btn (BTN, on (BTN', R)) = on (btn (BTN, R))
    if  BTN == BTN' .
ceq btn (BTN, on (BTN', R)) = off (btn (BTN, R))
    if  BTN =/= BTN' .
```

We show the verification that a fields of radio buttons has the safety property that at most one button is selected. We suppose that there are at least two buttons in a fields of radio buttons. Since every button in any field of radio buttons is initially disselected from the specification, the safety property initially holds. Then all we have to do is that given any state in which the safety property holds, we show that the safety property also holds in each next state after applying any action operator to the state. There are two cases that the safety property holds: 1) no button is selected, and 2) only one button is selected. The state corresponding to the cases (1) and (2) are represented by rb1 and rb2, respectively. We suppose that b11 and b12 are arbitrary buttons in rb1, and rb1' represents the next state after selecting b11. The following proof score makes it possible to show that the safety property holds in rb1':

```
ops rb1 rb1' : -> RdBtn .
ops b11 b12  : -> BtnID .
eq on? (btn (b11, rb1)) = false .
eq on? (btn (b12, rb1)) = false .
eq rb1' = on (b11, rb1) .
red on? (b11, rb1') == true
    and on? (b12, rb1') == false .
```

The case (2) is divided into two cases that 2a) the selected button is selected again, and 2b) any disselected button is selected. We suppose that b21 and b22 are the selected button and an arbitrary disselected button in rb2, and rb2a' and rb2b' represent the next states after selecting b21 and b22, respectively. The case (2b) is also divided into two cases that there are two buttons, and more than two buttons. We suppose that b23 represents an arbitrary disselected button except for b22 in rb2 if there are more than two buttons. The following proof score makes it possible to show that the safety property holds in rb2a' and rb2b':

```
ops rb2 rb2a' rb2b' : -> RdBtn .
ops b21 b22 b23     : -> BtnID .
eq on? (btn (b21, rb2)) = true .
eq on? (btn (b22, rb2)) = false .
eq on? (btn (b23, rb2)) = false .
eq rb2a' = on (b21, rb2) .
eq rb2b' = on (b22, rb2) .
red on? (b21, rb2a') == true
    and on? (b22, rb2a') == false .
red on? (b21, rb2b') == false
    and on? (b22, rb2b') == true
    and on? (b23, rb2b') == false .
```

We have completed the verification that a fields of radio buttons has the safety property.

3 A single-track railroad system

We consider a two-station system shown in Fig. 3. The system has seven sections[1] $T_{n\,(n=1,\ldots,7)}$ and four signals $S_{n\,(n=1,\ldots,4)}$. A station consists of three sections: T_1, T_2, and T_3 for station A, and T_5, T_6, and T_7 for station B. A section has two properties: the number of trains in it and the direction. The direction has three possible values: L_{dir} (for left), R_{dir} (for right), and N_{dir} (for unspecified). A signal has two possible states: G (for green) and R (for red) with usual meanings.

Initially there are two trains C_2 and C_2 in the system as shown in Fig. 3, and every signal shows R. Besides, T_1 and T_6, T_2 and T_7, and T_3, T_4, and T_5 have R_{dir}, L_{dir}, and N_{dir}, respectively, in the initial state, and the directions of T_1, T_2, T_6, and T_7 cannot be changed.

Let us show one possible scenario that train C_1 reaches station B shown in Fig. 4:

1. Fig. 4 (a) shows the initial state.
2. It is confirmed whether the direction of T_4 is N_{dir}, and only if so, the direction is set to R_{dir}(see Fig. 4 (b)).
3. It is confirmed whether the direction of T_3 and T_4 is N_{dir} and R_{dir}, respectively, and only if so, the direction of T_3 is set to R_{dir}. It is confirmed whether both directions of T_3 and T_4 are R_{dir}, and there is no train on T_3 and T_4, and only if so, S_1 is changed to G from R (see Fig. 4 (c)).
4. It is confirmed whether S_1 is G, and only if so, C_1 is moved to T_3 from T_1 and S_1 is changed to R at the same time (see Fig. 4 (d)), and then C_1 is moved to T_4.
5. It is confirmed whether the direction of T_5 is N_{dir}, and only if so, it is set to R_{dir}. It is confirmed whether the direction of T_5 is R_{dir}, and there is no train on T_5 and T_6, and only if so, S_3 is changed to G from R (see Fig. 4 (e)).
6. It is confirmed whether S_3 is G, and only if so, C_1 is moved to T_5 from T_4 and T_3 is changed to R at the same time, and then C_1 is moved to T_6 (see Fig. 4 (f)).

In the above scenario, we have mentioned how objects such as S_1 change their states. We describe how to change the states of objects in more detail.

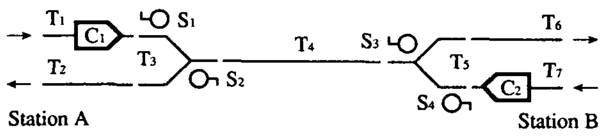

Figure 3: Two-station system.

[1]Each T_n may not actually correspond to a section, but in this paper it is regarded as a section for brevity.

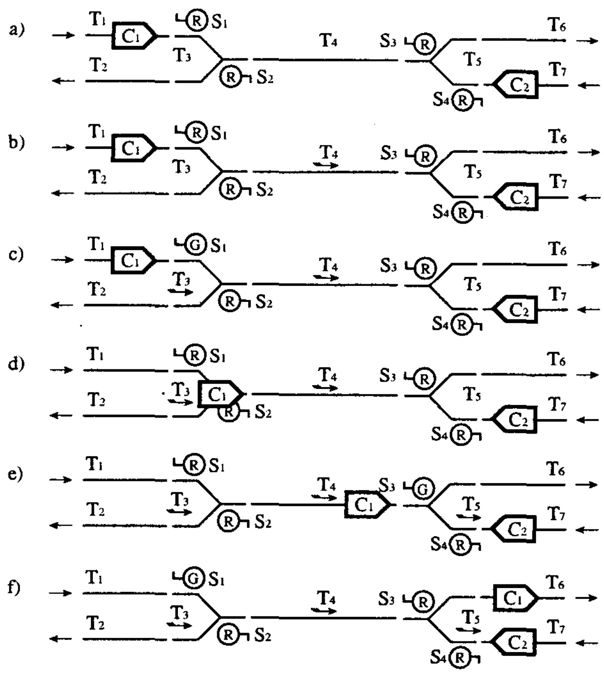

Figure 4: One possible scenario.

- The direction of T_4 can be set to either R_{dir} or L_{dir} only if it is N_{dir}. It can be set back to N_{dir} from R_{dir} (or L_{dir}) if the direction of T_3 (or T_5) is N_{dir}.
- The direction of T_3 can be set to R_{dir} (or L_{dir}) only if it is N_{dir} and the direction of T_4 is R_{dir} (or any value). It can be set back to N_{dir} only if there is no train on it. The direction of T_5 can be changed likewise.
- S_1 can be changed to G from R only if there is no train on both T_3 and T_4, and both direction of T_3 and T_4 are R_{dir}. If a train enters T_3, or the direction of T_3 is set back to N_{dir}, S_1 must be set back to R simultaneously. S_4 can be changed likewise.
- S_3 can be changed to G from R only if there is no train on both T_5 and T_6, and the direction of T_5 is R_{dir}. If a train enters T_5, or the direction of T_5 is set back to N_{dir}, S_3 must be set back to R simultaneously. S_2 can be changed likewise.

3.1 Specification

We have written the specification of the two-station system described above in CafeOBJ. Roughly speaking, the specification is a composite one of several specifications of components, i.e. trains and sections. Signals are represented by sections. For example, S_1 is done by T_3 and T_4. Components are synthesized according to the component-based specification in CafeOBJ [5]. The point is projection operations, with which the specification of a whole system can be written in terms of behavior of components. Fig. 2 shows the UML object diagram corresponding to our specification.

We show the main part of the specification of the two-station system:

```
op   init    : -> Sys                     -- initial state.
bop watch?  : SignalID Sys -> Signal     -- observation.
bop where?  : TrainID Sys -> TcID        -- observation.
bop reach   : TrainID Sys -> Sys         -- action.
bop leave   : TrainID Sys -> Sys         -- action.
bop move    : TrainID Sys -> Sys         -- action.
bop setdir  : TcID Dir Sys -> Sys        -- action.
op   train   : TrainID Sys -> Train       -- projection.
op   tc      : TcID Sys -> Tc             -- projection.
```

Sys is a hidden sort representing the state space of the two-station system, and Train and Tc are also hidden sorts representing the state spaces of a train and a section that are components of the system. The other sorts are visible ones. Bool represents the boolean values, TrainID, SignalID, and TcID represent IDs of trains, signals, and sections, respectively, and Signal and Dir represents values of signals and directions of sections, respectively.

Operator init represents the initial state of the two-station system. Operators watch? and where? are observation ones. watch? returns either R or G of the signal given as its first argument. where? returns the section where the train given as its first argument is. Operator reach, leave, move, and setdir are action ones. reach puts a train that runs from left to right (or from left to right) on T_1 (or T_7), which means that a train enters a station from a yard or the previous section of T_1 (or T_7). leave is the opposite one that removes a train from T_1 (or T_7). move moves a train to the next section. If the next section has a signal, the operator is enabled (or can change the system state) only if the signal is G. setdir sets a section (except for T_1, T_2, T_6, and T_7) to either L_{dir}, R_{dir}, or N_{dir}. The operators train and tc are projection ones that combine the specifications of trains and sections.

We describe how to define each operation with equations.

Action operator setdir only affects each section T_n in the two-station system. Each train C_n cannot be affected by setdir at all. So, it is very simple to define setdir for projection operator train as follows:

```
eq train (TR, setdir (TC, D, S)) = train (TR, S) .
```

The equation means that even if setdir sets a section TC in a system S to a direction D, a train TR does not change its state at all. On the other hand, setdir for projection operator tc is defined as follows:

```
ceq tc (TC, setdir (TC', D, S)) =
                              setdir (D, tc (TC, S))
    if  TC == TC'   and  setdir-cond (TC, D, S) .
ceq tc (TC, setdir (TC', D, S)) = tc (TC, S)
    if  TC =/= TC'
    or  not (setdir-cond (TC, D, S)) .
```

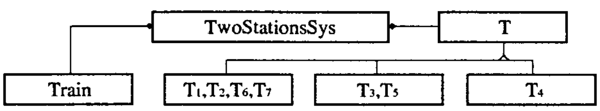

Figure 5: UML object diagram for two-station systems.

setdir on the left-hand side of each equation is an action operator for the whole system, and setdir on the right-hand side is an action operator for each section. The first equation means that if setdir tries to set a section TC' in a system S to a direction D provided that a condition setdir-cond is satisfied, the section TC' is actually set to the direction. The second equation means that even if setdir tries to set TC' in S to D, any other section TC does not change its state, and the section TC' does not change its state either unless the condition setdir-cond is satisfied.

The condition setdir-cond is defined for each section T_n. For sections t1, t2, t6, t7, and yard, the condition setdir-cond is always false as defined as follows:

```
op setdir-cond : TcID Dir Sys -> Bool
eq setdir-cond (t1, D, S) = false .
eq setdir-cond (t2, D, S) = false .
eq setdir-cond (t6, D, S) = false .
eq setdir-cond (t7, D, S) = false .
eq setdir-cond (yard, D, S) = false .
```

where tn and yard are constants representing a section, and the previous section of either T_1 or T_7, respectively. For t3, t4, and t5, the condition setdir-cond is defined as described earlier. The definition is as follows:

```
eq setdir-cond (t3, L, S) = dir? (tc (t3, S)) == N .
eq setdir-cond (t3, R, S) = dir? (tc (t3, S)) == N
                        and dir? (tc (t4, S)) == R .
eq setdir-cond (t3, N, S) = not
                              (exist? (tc (t3, S))) .
eq setdir-cond (t4, L, S) = dir? (tc (t4, S)) == N .
eq setdir-cond (t4, R, S) = dir? (tc (t4, S)) == N .
eq setdir-cond (t4, N, S) = (dir? (tc (t4, S)) == R
                        and dir? (tc (t3, S)) == N)
                        or  (dir? (tc (t4, S)) == L
                        and dir? (tc (t5, S)) == N) .
eq setdir-cond (t5, L, S) = dir? (tc (t5, S)) == N
                        and dir? (tc (t4, S)) == L .
eq setdir-cond (t5, R, S) = dir? (tc (t5, S)) == N .
eq setdir-cond (t5, N, S) = not
                              (exist? (tc (t5, S))) .
```

where constants L, R, and N represent L_{dir}, R_{dir}, and N_{dir}, respectively, and dir? and exist? are observation operators for sections T_n with which we can observe the direction of each section and confirm whether there exist trains on each section, respectively. For example, for the section t3 in a system S and the direction L, the condition setdir-cond is true if the direction of t3 in S is N.

Observation operator watch? obtaining the state of each signal is defined as follows:

```
ceq watch? (SG, S) = G  if signal-cond (SG, S) .
ceq watch? (SG, S) = R  if not (signal-cond (SG, S)) .
```

A signal SG is G (or R) if a condition signal-cond is satisfied (or not). The condition signal-cond is defined for each signal as follows:

```
op signal-cond : SignalID Sys -> Bool
eq signal-cond (s1, S) = exist? (tc (t3, S)) == false
                     and exist? (tc (t4, S)) == false
                     and dir? (tc (t3, S)) == R .
eq signal-cond (s2, S) = exist? (tc (t2, S)) == false
                     and exist? (tc (t3, S)) == false
                     and dir? (tc (t3, S)) == L .
eq signal-cond (s3, S) = exist? (tc (t5, S)) == false
                     and exist? (tc (t6, S)) == false
                     and dir? (tc (t5, S)) == R .
eq signal-cond (s4, S) = exist? (tc (t4, S)) == false
                     and exist? (tc (t5, S)) == false
                     and dir? (tc (t5, S)) == L .
```

where sn is a constant representing S_n. The above equations basically correspond to what we have described on behavior of each signal except that the direction of T_4 is not inspected. The reason why the inspection does not need to do is because if the direction of T_3 (or T_5) is R_{dir}(or L_{dir}), it is clear from the definition of `setdir-cond` that the direction of T_4 is also R_{dir}(or L_{dir}).

Action operator `move` for projection operator `train` is defined as follows:

```
ceq train (TR, move (TR', S)) = move (train (TR, S))
    if  TR == TR'
    and move-cond (where? (TR, S), TR, S) .
ceq train (TR, move (TR', S)) = train (TR, S)
    if  TR =/= TR'
    or  not (move-cond (where? (TR, S), TR, S)) .
```

`move` on the left-hand side of each equation is an action operator for the whole system, and `move` on the right-hand side is an action operator for each train component. The first equation means that if `move` tries to move a train `TR'` to the next section provided that a condition `move-cond` is satisfied, the train `TR'` is actually moved to the next section. The second equation means that even if `move` tries to move a train `TR'` to the next section, any other train does not move at all, and the train `TR'` does not move either unless `move-cond` is satisfied. Action operator `move` for projection operator `tc` is defined as follows:

```
ceq tc (TC, move (TR, S)) = enter (tc (TC, S))
    if  TC == where? (move (train (TR, S)))
    and  move-cond (where? (train (TR, S)), TR, S) .
ceq tc (TC, move (TR, S)) = leave (tc (TC, S))
    if  TC == where? (train (TR, S))
    and  move-cond (where? (train (TR, S)), TR, S) .
ceq tc (TC, move (TR, S)) = tc (TC, S)
    if  TC =/= where? (train (TR, S))
    or  TC =/= where? (move (train (TR, S)))
    or  not
       (move-cond (where? (train (TR, S)), TR, S)) .
```

where `enter` is an action operator for a section, meaning that a train has entered the section, and `where?` is an observation operator for a train observing the section on which there exists the train. The first (or second) equation means that if `move` tries to move a train `TR` in a system `S` provided that the condition `move-cond` is satisfied, the train `TR` enters the next of the section where `TR` is (or leaves the section where where `TR` is). The third equation means that even if `move` tries to move `TR` in `S`, no train enters and/or leaves any other section, and no train enters and/or leaves the section where `TR` is and the next section unless the condition `move-cond` is satisfied.

The condition `move-cond` is defined for each section as follows:

```
op move-cond : TcID TrainID Sys -> Bool
eq move-cond (t1, TRR, S) = watch? (s1, S) == G .
eq move-cond (t2, TRR, S) = false .
eq move-cond (t3, TRR, S) = true .
eq move-cond (t4, TRR, S) = watch? (s3, S) == G .
eq move-cond (t5, TRR, S) = true .
eq move-cond (t6, TRR, S) = false .
eq move-cond (t7, TRR, S) = false .
eq move-cond (yard, TRR, S) = false .
```

The above equations are good for a train moving from left to right. For example, a train on section T_1 can move to section T_3 if signal S_1 shows G, a train on section T_2 cannot move to section T_3 at any time, and a train on section T_3 can always move to section T_4, which are represented by the first, second, and third equations, respectively. The equations for a train moving from right to left can be defined as well.

Action operations `reach` and `leave` can be defined as `move`.

3.2 Verification

We have proved that the two-station system has a safety property that more than one train never enter a same section simultaneously. We describe the verification.

Basically we have used the same verification technique described in Sect. 2. In the two-station system, however, there are states such that although the states have the property, the property is not preserved in the next states after applying some action to the states. Therefore, we first find out such states, and then show that these states are not reachable from the initial state.

There are basically four cases corresponding to such cases. For the symmetry of the two-station system, however, only two cases should be considered. The two cases are (r1) and (r2) shown in Fig. 6. Suppose that there exist two trains moving left on T_2 and T_3, respectively, the two trains are on T_2 simultaneously if action operator `move` is applied to the train on T_3. Now we show that any state corresponding to the case (r1) is not reachable. Although there are more than one state that are predecessors of the states corresponding to the case (r1), we only need to consider the states corresponding to the case (r1') because any other previous state coincides with one of the states corresponding to the case (r1). Only applying `move` to the train on T_4 in the case (r1') could change a state corresponding to (r1') to a state corresponding to (r1). Therefore, we have only to show that such a transition cannot be happened. The following proof score can prove this:

```
ops c1 c2 : -> L-TrainID .
ops r1 r1' : -> Sys .
eq  where? (train (c1, r1')) = t2 .
eq  where? (train (c2, r1')) = t4 .
eq  dir? (train (c1, r1')) = L .
eq  dir? (train (c2, r1')) = L .
eq  howmany? (tc (t2, r1')) = s 0 .
eq  howmany? (tc (t3, r1')) = 0 .
eq  howmany? (tc (t4, r1')) = s 0 .
eq  r1 = move (c2, r1') .
red where? (c2, r1') == where? (c2, r1) .
```

Next let us consider the case (r2). Suppose that there exist a train moving right on T_3 and a train moving either left or right on T_4, the two trains are on T_4 simultaneously if action operator `move` is applied to the train on T_3. We can show that any state corresponding to the case (r2) is not reachable in the same way as the case (r1). In this case, there are two cases (r2a) and (r2b) corresponding to the states that are predecessors of the states corresponding to the case (r2). Moreover, we have to consider two cases (r2b') and (r2b") that are predecessors of the states corresponding to the case (r2b) because a state corresponding to the case (r2b) can be changed to a state corresponding to the case (r2). In this paper, we only show that any state corresponding to the case (r2b') is not reachable. The other three cases can be done likewise. The following proof score makes it possible to show that any state corresponding to the case (r2b') is not reachable:

```
op  c1  : -> L-TrainID .
op  c2  : -> R-TrainID .
ops r2b r2b' : -> Sys .
eq  where? (train (c1, r2b')) = t5 .
eq  where? (train (c2, r2b')) = t1 .
eq  dir? (train (c2, r2b')) = R .
eq  howmany? (tc (t1, r2b')) = s 0 .
eq  howmany? (tc (t3, r2b')) = 0 .
eq  howmany? (tc (t4, r2b')) = 0 .
eq  howmany? (tc (t5, r2b')) = s 0 .
eq  dir? (tc (t4, r2b')) = L .
eq  dir? (tc (t5, r2b')) = L .
eq  r2b = move (c2, r2b') .
red where? (c2, r2b') == where? (c2, r2b') .
```

We have completed the verification that the two-station system has the safety property.

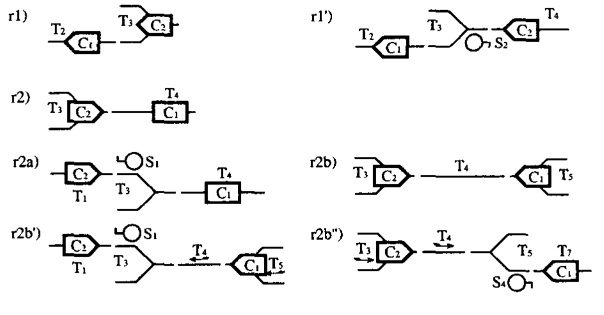

Figure 6: Unsafe but unreachable states.

4 Related Work

Block systems are the principal concept for safety assurance on railroad domain. In [3], Cichoki and Gorski describe a formal specification of railroad signaling systems in Z and show some safety propeties and hazards on the railroad signaling system by using FMEA (Failure Mode and Effect Analysis) analysis techniques.

In the railroad domain, to synthesize signals and branches are called *interlocking*, and each station needs an interlocking controller. There are many works of applying formal methods for interlocking design. For example, Morley models interlocking logic with higher order logic and implements his models and a model checker in Standard ML [8]. He proves full-automatically some safety properties about interlocking with the models and the model checker. But it is still difficult to prove properties interlocking for huge stations.

Bjørner et al model many functions in the railroad domain and describe their requirements as widely as possible. The domain models and requirement definitions are written both informally in English and formally in the RAISE Specification Language[1] [2].

5 Conclusion

We have briefly described the specification of a single-track railroad system in CafeOBJ, and the verification of its signaling system that no collision between trains occurs if trains run according to the signals.

References

[1] Bjørner, D., Braad, J. and Mogensen, K.: Models of railway systems: Domain. *Proc. of the 5th workshop on FMERAIL.* 1999.

[2] Bjørner, D., Braad, J. and Mogensen, K.: Models of railway systems: Requirements. *Proc. of the 5th workshop on FMERAIL.* 1999.

[3] Cichocki, T. and Gorski, J.: Safety assessment of computerized railway signaling equipment supprted by formal techniques. *Proc. of the 5th workshop on FMERAIL.* 1999.

[4] Diaconescu, R. and Futatsugi, K.: CafeOBJ report. World Scientific. 1998.

[5] Diaconescu, R., Futatsugi, K. and Iida, S.: Component-based algebraic specification and verification in CafeOBJ. *Proc. of FM'99.* LNCS 1709 Springer. (1999) 1644–1663.

[6] Goguen, J. and Malcolm, G.: A hidden agenda. *To appear in Theoretical Computer Science.* Also available as Technical Report CS97-538. Comp. Sci.&Eng. Dept. Univ. of Calif. at San Diego. 1997.

[7] Goguen, J., and Malcom, G.: Software engineering with OBJ – algebraic specification in action. Advances in formal methods Vol.2, Kluwer. 2000.

[8] Morley, M.: Safety assurance in interlocking design. Ph.D thesis of Univ. of Edinburgh. 1996.

[9] Yoshitake, I. and Akimoto, Y: An explanation of driving security facilities. Japan Railway Books inc. 1984.