<table>
<thead>
<tr>
<th>Title</th>
<th>Specification and verification of a single-track railroad signaling in CafeOBJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>SEINO, Takahiro; OGATA, Kazuhiro; FUTATSUGI, Kokichi</td>
</tr>
<tr>
<td>Citation</td>
<td>IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E84-A(6): 1471-1478</td>
</tr>
<tr>
<td>Issue Date</td>
<td>2001-06</td>
</tr>
<tr>
<td>Type</td>
<td>Journal Article</td>
</tr>
<tr>
<td>Text version</td>
<td>publisher</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/10119/3981">http://hdl.handle.net/10119/3981</a></td>
</tr>
<tr>
<td>Rights</td>
<td>Copyright (C) 2001 IEICE(許諾番号:07RB0175)</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The document describes the specification and verification of a single-track railroad signaling system using CafeOBJ. It includes technical details and methodologies used in the verification process.
Specification and Verification of a Single-Track Railroad Signaling in CafeOBJ*

Takahiro SEINO†, Kazuhiro OGATA†, and Kokichi FUTATSUGI†, Nonmembers

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 operations the whole specification has been described. A safety property that more than one train never enter a same section simultaneously has also been verified with CafeOBJ.

key words: CafeOBJ, formal methods, railroad signaling

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 that the systems have some desired properties based on the formal specifications.

Formal specification languages in which we can formally specify systems and with which we can formally verify that the systems have some properties have been proposed. CafeOBJ [4] is one of them. CafeOBJ allows us to specify state machines or objects of object-orientation 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 that 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 prevent 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 are 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. Section 2 mentions CafeOBJ and how to specify systems in CafeOBJ and verify that the systems have some properties with CafeOBJ. Section 3 describes the two-station system, its specification in CafeOBJ, and the verification with CafeOBJ that the system has a safety property that more than one train never enter 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 of object-orientation 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. In addition, components are synthesized according to the component-based specification in CafeOBJ [5]. We use projection operations to combine specifications of component systems and build a specification of a composite system.

2.1 How to Specify Concurrent Systems in CafeOBJ

We show a specification for fields of radio-buttons as an example. Figure 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. A
We first write a specification of buttons (corresponding to $\text{Btn}$ in Fig. 2), and then we use a projection operation to combine the specification of buttons and build a specification for fields of radio-buttons (corresponding to $\text{RdBtn}$ in Fig. 2).

We show the signature of a specification of buttons from which fields of radio-buttons are made:

- **$\text{op init}$**: $\text{Bool} \rightarrow \text{Btn}$ -- initial state.
- **$\text{bops on off}$**: $\text{Btn} \rightarrow \text{Btn}$ -- actions.
- **$\text{bop on?}$**: $\text{Btn} \rightarrow \text{Bool}$ -- observation.

A comment starts with `--` and terminates at the end of the line. $\text{Btn}$ is a hidden sort representing the state space of each button, and $\text{Bool}$ is a (built-in) visible sort representing boolean values. Operator $\text{init}$ takes a boolean value, representing an initial state of a button. Action operators $\text{on}$ and $\text{off}$ can select and disselect a button, respectively. Observation operator $\text{on?}$ allows us to observe the state of a button, i.e. selected or disselected. If a button is selected, $\text{on?}$ returns $\text{true}$, and otherwise $\text{on?}$ returns $\text{false}$. We use equations to define what happens next after applying an action operation to a button. The equations for buttons are as follows:

- $\text{eq on? (init (B)) = B .}$
- $\text{eq on? (on (S)) = true .}$
- $\text{eq on? (off (S)) = false .}$

$B$ and $S$ are variables whose sorts are $\text{Bool}$ and $\text{Btn}$, 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 $\text{true}$ (or $\text{false}$), i.e. selected (or disselected), after applying $\text{on}$ (or $\text{off}$) to the button.

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

- **$\text{-- initial state.}$**
- **$\text{op init}$**: $\text{BtnID} \rightarrow \text{RdBtn}$ -- action.
- **$\text{bop on}$**: $\text{BtnID} \rightarrow \text{Btn}$ -- observation.
- **$\text{op btn}$**: $\text{BtnID} \rightarrow \text{Btn}$ -- projection.

Hidden sort $\text{RdBtn}$ and visible one $\text{BtnID}$ represent the state space of fields of radio-buttons and IDs for buttons, respectively. Projection operator $\text{btn}$ is used to make fields of radio-buttons by combining buttons as components. More precisely, given a field of radio-buttons and an ID for a button in the field, $\text{btn}$ takes out the corresponding button out from the field, or projects the field onto (the axis of) the button.

Operator $\text{init}$ takes an ID as its argument, representing an initial state of a field of radio-buttons at which the button corresponding to the ID is selected and any other button is disselected. The following two (conditional) equations define this. Note that $\text{Btn}$ and $\text{BTN}'$ are variables whose sort is $\text{BtnID}$, and $R$ is a variable whose sorts are $\text{RdBtn}$.

- $\text{eq btn (BTN, init (BTN')) = init (true) if BTN == BTN'.}$
- $\text{eq btn (BTN, init (BTN')) = init (false) if BTN' /= BTN'.}$

$\text{init}$ on the left-hand sides is an operator on sort $\text{RdBtn}$, while $\text{init}$ on the right-hand sides is one on sort $\text{Btn}$. Action operator $\text{on}$ chooses one among the buttons, namely that it makes the chosen button selected and makes any other button disselected. The following two equations define this.

- $\text{eq btn (BTN, on (BTN', R)) = on (btn (BTN, R)) if BTN == BTN' .}$
- $\text{eq btn (BTN, on (BTN', R)) = off (btn (BTN, R)) if BTN' /= BTN' .}$

$\text{on}$ on the left-hand sides is an action operator on sort $\text{RdBtn}$, while $\text{on}$ on the right-hand sides is one on sort $\text{Btn}$. Observation operator $\text{on?}$ observes the state of a button given by its first argument in a field of radio-buttons given by its second argument. The following equation defines this.

- $\text{eq on? (BTN, R) = on? (btn (BTN, R)) .}$

$\text{on?}$ on the left-hand side is an observation operator on sort $\text{RdBtn}$, while $\text{on?}$ on the right-hand side is one on sort $\text{Btn}$. Let us consider the following expression, or term:

- $\text{on (1, on (3, on (2, init (1))))}$

where natural numbers are used to identify buttons. The term represents the state of a field of radio-buttons after some action operations are applied to the field. The initial state is that button 1 is selected and any other button is disselected, followed by choosing button 2, button 3, and again button 1. Let $r$ be the term. By applying $\text{on?}$ to $r$, we can observe the state of each button in that state. For example, $\text{on? (1, r)}$ is $\text{true}$ and $\text{on? (2, r)}$ is $\text{false}$.

### 2.2 How to Verify Concurrent Systems with CafeOBJ

We describe the verification of the claim that a field
of radio-buttons has the safety property that only one button is always selected. If a field of radio-buttons has one button, it is easy to show the claim. Hence, we suppose that there are at least two buttons in a field of radio-buttons. Since one and only button is initially selected in a field of radio-buttons, the claim is initially true. Hence, all we have to do is that given any state of a field of radio-buttons in which the claim holds, we show that the claim keeps holding after applying any action operation to the state. Let $rb$ be a state where the claim holds, namely that only one button is selected. Since there is one action operation in a field of radio-buttons, we examine whether the claim keeps holding after applying the action operation to $rb$. We should consider two cases: (1) the selected button is selected again, and (2) any disselected button is selected. Let $b_1$ and $b_2$ be the selected button and an arbitrary disselected button in $rb$, respectively, and let $rb_1$ and $rb_2$ be the next states after selecting $b_1$ and $b_2$, respectively. The case (2) is also divided into two cases that there are two buttons, and more than two buttons in $rb$. Let $b_3$ be an arbitrary disselected button except for $b_2$ in $rb$ if there are more than two buttons in $rb$. The following proof score makes it possible to show that the claim keeps holding in $rb_1$ and $rb_2$.

\[
\begin{align*}
\text{ops rb rb1 rb2 :-> RdBtn} . \\
\text{ops b1 b2 b3 :-> BtnID} . \\
\text{eq on? (btn(b1, rb)) = true} . \\
\text{eq on? (btn(b2, rb)) = false} . \\
\text{eq on? (btn(b3, rb)) = false} . \\
\text{eq rb1 = on (b1, rb)} . \\
\text{eq rb2 = on (b2, rb)} . \\
\text{red on? (b1, rb1) == true} . \\
\text{red on? (b1, rb2) == false} . \\
\text{red on? (b2, rb1) == true} . \\
\text{red on? (b2, rb2) == false} . \\
\end{align*}
\]

The equations on the third, fourth, and fifth lines mean that button $b_1$ is selected, and $b_2$ and $b_3$ representing an arbitrary disselected button are disselected in $rb$. The equations on the sixth and seventh lines define $rb_1$ and $rb_2$ as the next states after selecting $b_1$ and $b_2$, respectively. CafeOBJ command red reduces a term by regarding given equations as left-to-right rewrite rules. The term following the first (or second) red means that the claim is also true in $rb_1$ (or $rb_2$) as well. Both terms are reduced to true. We have completed the verification that a field 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 $T_n (n = 1, \ldots, 7)$ 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 neutral). A signal has two possible states: $G$ (for green) and $R$ (for red) with usual meanings.

Initially there are two trains $C_1$ 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. Figure 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 directions of $T_3$ and $T_4$ are $N_{dir}$ 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, $S_1$

Each $T_n$ may not actually correspond to a section, but in this paper it is regarded as a section for brevity.
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 $S_3$ 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.

- 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 describe the specification of the two-station system in CafeOBJ. As the specification of fields of radio-buttons, specifications of components, i.e. trains and sections, are first written, and then the specification of the two-station system is built by combining the specifications of the components. Signals are represented in terms of sections. For example, $S_1$ is represented by the states of $T_3$ and $T_4$. If both directions of $T_3$ and $T_4$ are $R_{dir}$, and there is no train on both $T_3$ and $T_4$, then this case means that $S_1$ is G, and otherwise the other cases mean that $S_1$ is R. Figure 5 shows the UML object diagram corresponding to our specification.

The signature of the specification of trains is as follows:

- initial state.

![Fig.5 UML object diagram for two-station systems.](image-url)
Operator \text{init} represents the initial state of the two-station system. Operators \text{watch} and \text{where} are observation ones. \text{watch} returns either R or G of the signal given as its first argument. \text{where} returns the section where the train given as its first argument is. Operator \text{reach}, \text{leave}, \text{move}, and \text{setdir} are action ones. \text{reach} puts a train running right (or left) 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). \text{leave} is the opposite one that removes a train from T_1 (or T_7). \text{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. \text{setdir} sets a section (except for T_1, T_2, T_6, and T_7) to either L_{dir}, R_{dir}, or N_{dir}. Operators \text{train} and \text{tc} are projection ones that combine the specifications of trains and sections to build the specification of the two-station system.

We describe how to define each operation in equations.

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

\[
\text{eq train} (\text{TR}, \text{setdir} (\text{TC}, D, S)) = \\
\text{train} (\text{TR}, S).
\]

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

\[
\text{ceq tc} (\text{TC}, \text{setdir} (\text{TC'}, D, S)) = \\
\text{setdir} (D, \text{tc} (\text{TC}, S)) \\
\text{if} \ TC == \text{TC'} \\
\text{and \text{setdir-cond} (TC, D, S)}. \\
\text{ceq tc} (\text{TC}, \text{setdir} (\text{TC'}, D, S)) = \\
\text{tc} (\text{TC}, S) \\
\text{if} \ TC !== \text{TC'} \\
\text{or \ not \ (\text{setdir-cond} (TC, D, S)}).
\]

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

Condition \text{setdir-cond} is defined for each section T_n. For sections t1, t2, t6, t7, and yard, condition \text{setdir-cond} is always false as defined as follows:

\[
\text{op setdir-cond} : \text{TcID Dir Sys} \rightarrow \text{Bool} \\
\text{eq setdir-cond} (t1, D, S) = \text{false}. \\
\text{eq setdir-cond} (t2, D, S) = \text{false}. \\
\text{eq setdir-cond} (t6, D, S) = \text{false}. \\
\text{eq setdir-cond} (t7, D, S) = \text{false}. \\
\text{eq setdir-cond} (yard, D, S) = \text{false}.
\]

where \text{t1} and \text{yard} are constants representing T_n and the previous section of either T_1 or T_7, respectively. For t3, t4, and t5, condition \text{setdir-cond} is defined as described earlier. The definition is as follows:

\[
\text{eq setdir-cond} (t3, L, S) = \\
\text{dir} (\text{tc} (t3, S)) == \text{N}.
\]

\[
\text{eq setdir-cond} (t3, R, S) = \\
\text{dir} (\text{tc} (t3, S)) == \text{N} \\
\text{and} \ dir (\text{tc} (t4, S)) == \text{R}.
\]

\[
\text{eq setdir-cond} (t3, N, S) = \\
\text{not (exist? (tc (t5, S)))}.
\]

\[
\text{eq setdir-cond} (t4, L, S) = \\
\text{dir} (\text{tc} (t4, S)) == \text{N}.
\]

\[
\text{eq setdir-cond} (t4, R, S) = \\
\text{dir} (\text{tc} (t4, S)) == \text{N}.
\]

\[
\text{eq setdir-cond} (t4, N, S) = \\
\text{(dir} (\text{tc} (t4, S)) == \text{R} \\
\text{and} \ dir (\text{tc} (t3, S)) == \text{N} \\
\text{or (dir} (\text{tc} (t4, S)) == \text{L} \\
\text{and dir} (\text{tc} (t5, S)) == \text{N})).
\]

\[
\text{eq setdir-cond} (t5, L, S) = \\
\text{dir} (\text{tc} (t5, S)) == \text{N} \\
\text{and dir} (\text{tc} (t4, S)) == \text{L}.
\]

\[
\text{eq setdir-cond} (t5, R, S) = \\
\text{dir} (\text{tc} (t5, S)) == \text{N}.
\]

\[
\text{eq setdir-cond} (t5, N, S) = \\
\text{not (exist? (tc (t5, S)))}.
\]

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

Observation operator \text{watch}? obtaining the state of each signal is defined as follows:

\[
\text{ceq watch}? (SG, S) = G \\
\text{if signal-cond (SG, S)}.
\]

\[
\text{ceq watch}? (SG, S) = R \\
\text{if not (signal-cond (SG, S))}.
\]
Signal SG is G (or R) if condition signal-cond is satisfied (or not). Condition signal-cond is defined for each signal as follows:

begin{verbatim}
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 Sn. The above equations basically correspond to what we have described on behavior of each signal except that the direction of T4 is not inspected. The reason why the inspection does not need is because if the direction of T3 (or T5) is Rdir (or Ldir), it is clear from the definition of setdir-cond that the direction of T4 is also Rdir (or Ldir).

Observation operation where? is defined simply as follows:

eq where? (TR, S) = where? (train (TR, S)) .

where? on the left-hand side is an observation operator on Sys, while where? on the right-hand side is one on Train.

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 on Sys, and move on the right-hand side is an action operator on Train. The first equation means that if move tries to move train TR' to the next section provided that condition move-cond is satisfied, train TR' is actually moved to the next section. The second equation means that even if move tries to move train TR' to the next section, any other train does not move at all, and 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? (train (TR, S))
  and move-cond (where? (TR, S), TR, S) .

ceq tc (TC, move (TR, S)) =
  exit (tc (TC, S))
  if TC == where? (train (TR, S))
  and move-cond (where? (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? (TR, S), TR, S)) .

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

Condition move-cond is defined for each section as follows:

begin{verbatim}
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 (y, TRR, S) = false .

where the above equations are good for a train running right. For example, a train on section T1 can move to section T3 if signal S1 shows G, a train on section T2 cannot move to section T3 at any time, and a train on section T3 can always move to section T4, which are represented by the first, second, and third equations, respectively. The equations for a train running left can be defined as well. Note that as described above, action operator move does not move a train on section T6 to the yard or the right section of T6, but action operator
leave does this. Action operators reach and leave may be considered as special versions of move. They can be defined as move.

3.2 Verification

We have proved that the two-station system has the 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 operation 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.

First let us consider the case (r1). Suppose that there exist two trains moving left on T2 and T3, respectively, the two trains are on T2 simultaneously if action operator move is applied to the train on T3. 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 previous state coincides with one of the states corresponding to the case (r1). Only applying move to the train on T4 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:

\[
\begin{align*}
\text{ops} & \ c1 \ c2 : \rightarrow \text{TrainID} . \\
\text{ops} & \ r1 \ r1' : \rightarrow \text{Sys} . \\
\text{eq} & \ \text{where}?: (\text{train} (c1, r1')) = t2 . \\
\text{eq} & \ \text{where}?: (\text{train} (c2, r1')) = t4 . \\
\text{eq} & \ \text{dir}?: (\text{train} (c1, r1')) = L . \\
\text{eq} & \ \text{dir}?: (\text{train} (c2, r1')) = L . \\
\text{eq} & \ r1 = \text{move} (c2, r1') . \\
\text{red} & \ \text{where}?: (c2, r1') = \text{where}?: (c2, r1) .
\end{align*}
\]

Constants c1 and c2 are IDs of two trains in the case (r1') of Fig. 6. Constant r1' represents a state corresponding the case (r1') of Fig. 6. State r1' is characterized with above equations. For example, the equation on the third line means that train c1 is on section t2 in state r1'. In the proof, we do not define any other components such as T5 that do not matter in the proof. Constant r1 represents the state after trying to apply action operation move to train c2 in state r1'. The term following red means that the section on which train c2 is in state r1 is the same as in state r1', namely that any state corresponding to the case (r1) is not reachable from the initial state. The term has been reduced to true.

Next let us consider the case (r2). Suppose that there exist a train moving right on T3 and a train moving either left or right on T4, the two trains are on T4 simultaneously if action operator move is applied to the train on T3. 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 (r2). That is, all that is needed is to show that any state corresponding to one of the three cases (r2a), (r2b') and (r2b") does not lead to any state corresponding to the case (r2). In this paper, we only show that any state corresponding to the case (r2b') does not lead to any state corresponding to the case (r2b), which implies that it does not lead to any state corresponding to the case (r2). The following proof score makes it possible to show this:

\[
\begin{align*}
\text{op} & \ c1 : \rightarrow \text{TrainID} . \\
\text{op} & \ c2 : \rightarrow \text{TrainID} . \\
\text{ops} & \ r2b \ r2b' : \rightarrow \text{Sys} . \\
\text{eq} & \ \text{where}?: (\text{train} (c1, r2b')) = t5 . \\
\text{eq} & \ \text{where}?: (\text{train} (c2, r2b')) = t1 . \\
\text{eq} & \ \text{dir}?: (\text{train} (c2, r2b')) = R . \\
\text{eq} & \ \text{exist}?: (\text{tc} (t1, r2b')) = \text{true} . \\
\text{eq} & \ \text{exist}?: (\text{tc} (t3, r2b')) = \text{false} . \\
\text{eq} & \ \text{exist}?: (\text{tc} (t4, r2b')) = \text{false} . \\
\text{eq} & \ \text{exist}?: (\text{tc} (t5, r2b')) = \text{true} . \\
\text{eq} & \ \text{dir}?: (\text{tc} (t4, r2b')) = L . \\
\text{eq} & \ \text{dir}?: (\text{tc} (t5, r2b')) = L . \\
\text{eq} & \ r2b = \text{move} (c2, r2b') . \\
\text{red} & \ \text{where}?: (c2, r2b') = \text{where}?: (c2, r2b') .
\end{align*}
\]
The other two cases can be done likewise.

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

4. Related Work

Block systems are the principal concept for safety assurance on the railroad domain. Cichoki and Gorski describe a formal specification of railroad signaling systems written in Z and show some safety properties and hazards on the systems with FMEA (Failure Mode and Effect Analysis) analysis technique [3]. The technique is a kind of methodology for system analysis with bottom up approach, and aims to identify and document anticipated faults of the components and their impact on the system external interfaces. Cichoki and Gorski indicate that some hazards cause failures of the lowest components (hardware) on their model, and show countermeasures for some cases on the hazards.

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 to interlocking design. For example, Morley models interlocking logic with higher order logic and implements his model and a model checker in Standard ML [8]. He designs a special language with which we can specify rails, signals and branches on a station and prove 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. They aim to illustrate what a railroad system is by decomposing the system into a number of components. The domain models and requirement definitions are written both informally in English and formally in the RAISE Specification Language [1], [2]. Their works are still in progress, and these papers are incomplete.

5. Conclusion

We have 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 with the help of the CafeOBJ system.

References