Application of Verification Methods to Specifications of Signalling Equipment

Natsuki TERADA

Senior Researcher,

Signalling Systems Laboratory, Signalling and Transport Information Technology Division

Takashi TOYAMA

Researcher,

Formal methods have been expected to increase the reliability of software, including that of signalling systems. In order to further the use of formal methods in the railway signalling domain, it would be beneficial to show examples of models of various signalling equipment using formal methods. As some examples, the specification of automatic block systems for single lines was modeled using formal specification languages, and the effectiveness of the model was verified by means of theorem proving and satisfiability problem solver. For theorem proving, B-method was used, which is characterized by theorem proving and stepwise refinement. B-method yields a very powerful result as far as safety is concerned, but it requires a high level of effort. On the other hand, satisfiability problem solvers can easily find truth of the proposition composed with many variables, but some restrictions apply in terms of the proposition domain.

Keywords: software, formal methods, automatic block system for single lines, B-method, SMT solver

1. Introduction

Microelectronics is widely used for signalling equipment, following the development of techniques for achieving failsafe hardware properties. Microelectronics equipment is expected to progress more. This means that ensuring that software is safe is now an even greater priority.

Many techniques are used with the software for signalling systems, such as software diversity, single thread, etc., to ensure failsafe. However, safety cannot be ensured if the program makes a mistake.

Formal methods were looked at to increase the reliability of software [1, 2]. Formal methods are techniques for increasing the quality of systems by describing the system specifications using formal specification languages based on mathematics and logics, and by verifying the specifications mathematically and logically. Figure 1 shows an example of development using formal methods. When formal specification languages are used, one benefit is that it is possible to remove ambiguity from the specifications, and to validate them by means of rapid prototyping during the early stage of the development. When theorem proving and model checking are used, it is possible to ensure that there are no failures or mistakes. Moreover, it is possible to decrease mistakes in the programs, when they are translated from verified specifications.

The application of formal methods in the railway sector is very common outside Japan [3-5], but within Japan such usage remains rare. It is considered that in order to further the use of formal methods in the railway signalling domain, it would be beneficial to show examples of models of various signalling equipment using formal methods. It is also important for the example to be used to show the model verification process. As some examples of modelling, automatic block systems were modeled for single lines using B-method [6] (a formal method) and the models were verified by means of theorem proving. An SMT (Satisfiability Modulo Theories) solver [7] was also applied to the model verification, which decides the result of satisfiability problem.

Fig. 1 Development of software with formal methods

ability Modulo Theories) solver [7] was also applied to the model verification, which decides the result of satisfiability problem.

2. Automatic block systems for single lines

Interlocking systems and automatic train protection systems are considered typical representatives of signalling systems. However, automatic block systems for single lines were selected as the object of modeling, because they are composed of relays like interlocking systems, and they are simpler than interlocking systems. In fact, they are not independent of interlocking systems, but form part of them. In the standard connection diagrams for interlocking [8, 9], there are several pages for automatic block systems. Three kinds of system were modeled – automatic block systems for single lines, simplified automatic block systems, and semi-automatic block systems. They commonly use traffic direction control circuits. The focus is not on verification of the connection diagrams, but on verification of the functions. The connection diagrams were therefore not translated, but the functions of the automatic block systems were modeled.

Among the functions of automatic block systems, the

traffic direction control circuit and the traffic direction levers are the most important. The following concepts commonly apply for the three systems mentioned above:

- (1) When the directions of the levers at both neighboring stations agree, the traffic direction control circuit is composed in such a way that the power is supplied from the arrival station.
- (2) Once the traffic direction control circuit is composed, the change of direction at the arrival station is invalid.
- (3) Once the traffic direction control circuit is composed, it is locked while the departure route is set or a train is present between the stations. In such cases, the change of the direction at the departure station is also invalid.

Safety requirements were verified such that both the stations would not allow trains to depart at the same time, and changing direction at the arrival station would be invalid.

3. Verification of automatic block systems for single lines using B-method

3.1 B-method

B-method [6], which was used on this occasion, focuses on the generation of verified code. It is characterized by stepwise refinement and the proving of theorems automatically generated from the specifications. However, it is difficult to describe specifications using B, so VDM [10] was also used, which is another formal method. VDM focuses on the formal description of systems and validation. The specification by VDM++, an object-oriented variation of VDM, was first described. Then it was translated manually into B.

Both VDM++ and B can be used to describe invariants, which are the conditions held at any time among system variables. When the requirement is verified that “both the stations do not allow trains to depart at the same time,” the requirement is described as an invariant and it should be proved that the invariant is held. When it is verified that the “direction from the arrival station is invalid,” it should be proved that the state of direction circuit does not change even if the direction lever is changed. The conditions for proving the integrity of the systems are referred to as “proof obligations” and they are generated automatically from the specifications according to the rules. Indeed, the “verification” means discharging proof obligations.

3.2 VDM model of the simplified automatic block system

As a first step, simplified automatic block systems were modeled using VDM++. With this system, the track between the neighboring stations consists of only one block. Only one train can occupy the track, and the existence of the train can be detected with a device called the track circuit. In the model the system is divided into traffic direction control circuit and station equipment, both of which are described as class. The following were described in the station equipment class: “direction,” corresponding to the traffic direction relay; “directlock,” corresponding

to the relay for locking the traffic direction; “directlever,” corresponding to the traffic direction lever; and so on.

The traffic direction control circuit is composed of two pairs of circuits. One pair is the circuit for coordinating direction levers, and the other is the circuit for checking the presence of a train between the stations and for indicating a proceed signal. In the traffic direction control circuit class, the variables “directlever” at both the stations and “existence” of a train between the stations were described.

Following the description of these variables, operations were described corresponding to handling the lever, and to the train movements.

3.3 B model of simplified automatic block systems

As a next step, the specification was translated into B. Although there is a difference in structuring between VDM++ and B, a smaller number of changes are required in the system structure when each class of VDM++ corresponds to a B machine. But there are some small differences in expression. For example, it is possible to describe sequential operations in VDM++, while the same does not apply for the abstract machines of B. On the other hand, postcondition of operation is not available in B as it is in VDM++. The specification was described taking into consideration the above restriction. An example B description is shown in Fig.2.

```
xx <-- DownSet =
  ANY dir WHERE
    dir : DIRECTION &
      ((direction = up & directlock = unlock)
        => dir = down) &
      ((direction = down or directlock = lock)
        => dir = direction) &
    transition (direction, directlock, oppositedirect,
      dir, directlock, oppositedirect)
  THEN
    Set (routelever, routelock, down, dir, directlock,
      oppositedirect, existence) ||
    xx := dir
  END;
```

Fig. 2 A part of B model

3.4 Refinement of B model and verification

With B-method, an abstract model is first described, and then the model is refined stepwise. During the refinement, the operation is described in such a way that the result of the operation satisfies the condition described in the parent module, and the result is more decisive than the parent module. Finally, the specification is refined in such a way that it is possible to translate it directly into programming code. The final refinement is referred to as the “implementation.”

At each stage of refinement, proof obligations are generated. Most of them are discharged automatically, and the remainder should be discharged interactively. In this paper all of the proof obligations are discharged. Table 1

Table 1 Numbers of proofs (simplified automatic block system)

Module	Number of lines in B	Number of proof obligations	Automatically proved	Interactively proved
Departure station	209	99	85	14
Arrival station	209	99	85	14
Direction circuit*	60	21	17	4
Whole system	207	243	195	48
Total		462	382	80

*: Traffic direction control circuit

shows the number of proof obligations for each module, from the abstract machine to the implementation. The numbers indicate the complexity of specifications. In this paper, 83% of proof obligations are automatically discharged.

3.5 Modeling automatic block system for single lines and verification

As a next step, standard automatic block systems for single lines were modeled. In this system, the track between the stations is divided into some blocks, which is different from the simplified one. More than one train can run on the track between the stations when each of them runs on a separate block. The track circuits are equipped to carry signal information. To display this function, transmitters and receivers are equipped for both directions. The power supply of transmitter and receiver is turned on and off according to the applicable direction. The traffic direction control circuit is locked when one of the track circuits detects a train. Such control of signals is included in the model.

The model of the traffic direction control circuit does not differ significantly from the simplified systems. On the other hand, the model used for the control of signal aspects results in complexity. The track circuits were therefore modeled into a separate class. The numbers of proof obligations are shown in Table 2. All of the proof obligations are discharged. However, the complexity in the track circuit

model results in a large increase in the proof obligations for the whole model. A small increase in the proof obligations for the station equipment and the circuit direction is caused by the small increase of track circuits related with the lock of the direction.

3.6 Modeling semi-automatic block systems and verification

In the semi-automatic block systems, there is no track circuit between neighboring stations. Instead, a pair of short track circuits, called CT and OT, are equipped at the entrances of the stations. A relay called FSR is installed so that the direction is locked while a train departs from CT of the departure station until it reaches CT of the arrival station. The state of FSR depends on the sequence of track circuit changes. The traffic direction control circuit is composed of one pair of lines. When a train reaches the arrival station, the polarity of the power supply to the traffic direction control circuit reverses to notify of arrival at the departure station. Moreover, in order to handle unusual situations, such as regression or mis-departure of a train, a release button for regression, a substitution lever for entrance signals, and a release lever for mis-departure are provided, and procedures for handling them are specified. To ensure that there is no train between the neighboring stations, the number of relays is increased.

In the model, track circuits of CT, OT, platform track, and on route are described. The train position, which is

Table 2 Numbers of proofs (automatic block system)

Module	Number of lines in B	Number of proof obligations	Automatically proved	Interactively proved
Departure station	217	183	159	24
Arrival station	217	157	133	24
Control circuit	69	48	42	6
Track circuits	118	471	257	214
Whole system	333	9160	8435	725
Total		10019	9026	993

Table 3 Numbers of proofs (semi-automatic block system)

Module	Number of lines in B	Number of proof obligations	Automatically proved	Interactively proved
Station	604	11370	9763	1607
Direction circuit	103	112	85	27
Whole system	342	25989	25482	507
Total		37471	35330	2141

expressed as the combination of relays, is expressed as an enumerated set as follows:

Position = inexistence | home | route | CT | OTCT | OT | approach | arrival

Please note that “home” means that a train stays at the platform track and OTCT means that a train is detected at both CT and OT. The above-mentioned buttons are included in the model, while closure of track is excluded from the model.

The nature of train position changes was explored for each train position, with reference to the standard wire diagram. There are some relay movements not recognized as usual movements. There were two available options: one was to avoid them by precondition of operations, and the other was to specify the new position for them. The latter option was selected. As a result, it was necessary to split the cases according to the state of track circuits and train position.

The numbers of the generated proof obligations are shown in Table 3. It can be seen that there are many proof obligations for both of the station model and the whole model. It appears that this is because case splits are used many times for track circuit states. Once case split is used, proof obligations are doubled, because the result of the operation must hold invariants for each case. It takes several hours to check all of the proof obligations, even automatically.

The numbers of lines of specifications are also shown in Table 3. The number of lines for the station is 604 in semi-automatic block systems, while there are 217 lines for departure and arrival stations respectively in the automatic block system.

3.7 Summary of verification in B

The specifications of automatic block systems were verified using B. Even in the most complicated semi-automatic block systems, all of the proof obligations were discharged. However, when many variables and case splits are used, the number of proof obligations increases, which results an increase in the time required for verification.

4. Verification using SAT/SMT solver

One excellent advantage of the B-method is that verified codes can be obtained. However, when many variables are used, the number of proof obligations increases, which makes verification difficult. When systems that involve

more state variables than the automatic block systems are specified, other solutions are required. Model checking is one option, in which a system is described as a state transition system, and the properties of the system are checked exhaustively. However, model checking is associated with state explosion problems. The decision was made to opt for an SAT/SMT solver, which is related to model checking but is a different method.

4.1 SAT/SMT solvers

An SAT (SATisfiability problem) solver is a program that decides whether or not a combination of values that satisfies a proposition of Boolean variables exists. For example, suppose the following proposition is true:

$$(A \vee B) \wedge (A \vee \bar{B}) \wedge (\bar{A} \vee \bar{B}) \quad (1)$$

When A is true and B is false, (1) turns out to be true and it is found that a combination of A and B values exists. According to the theory, SAT is classified into NP (Non-deterministic Polynomial Time) complete, so there is no algorithm that can decide any problems in polynomial time. However, by applying heuristics, there are programs that decide practical SAT within a practical time. Recently, the performance of SAT has rapidly progressed. SAT with more than 10^6 variables can be solved.

In SAT solvers, only Boolean variables are acceptable, and the proposition is given in Conjunctive Normal Form (CNF), in which disjunctions of variables and/or negation of variable are conjunct. On the other hand, in SMT (SAT Modulo Theories) solvers, the range of proposition is extended to first-order predicate logic, integers and/or functions.

SAT/SMT solvers are used as a part of proof engines and model checkers. First application of formal verification to interlocking systems in Sweden [3] is based on an SAT solver.

As mentioned above, with B method it becomes harder to implement verification when the number of variables increases even in small applications. On the other hand, surprisingly many variables can be handled with an SAT/SMT solver. An attempt was therefore made to verify the specifications for automatic block systems using an SAT/SMT solver.

4.2 Formulation of verification using an SMT solver

With B-method, proof obligations are generated to check that invariants are held when operations are ap-

plied. When a specification is refined, the results of the refined operations have to satisfy the condition described in the predecessors. As SAT/SMT solvers only decide problems, but do not find all the possible combinations, the method for verifying the specifications must be formulated.

The variables for describing a system are given as $x_1, x_2, \dots, x_n$. Suppose that an invariant $I(x_1, x_2, \dots, x_n)$ is true at any time. The precondition of an operation is given as $P(x_1, x_2, \dots, x_n) = \text{true}$ and the results of the operation are given as $x'_1, x'_2, \dots, x'_n$.

To show that $x'_1, x'_2, \dots, x'_n$ satisfy the invariant for all possible combinations of $x_1, x_2, \dots, x_n$ that satisfy the invariant, it is sufficient to prove that there is no possible combination that makes $I(x_1, x_2, \dots, x_n)$ and $P(x_1, x_2, \dots, x_n)$ true, and $I(x'_1, x'_2, \dots, x'_n)$ false. Therefore, when $I(x_1, x_2, \dots, x_n)$ is true, $I(x'_1, x'_2, \dots, x'_n)$ is always true. That is, when the initial values of $x_1, x_2, \dots, x_n$ satisfy the invariant, the result of the operation always satisfies the invariant.

When an operation is given with the postcondition that $F(x_1, x_2, \dots, x_n, x'_1, x'_2, \dots, x'_n)$ is true, it must be shown that for each possible combination that makes $P(x_1, x_2, \dots, x_n)$ true, a combination must exist that makes $F(x_1, x_2, \dots, x_n, x'_1, x'_2, \dots, x'_n)$ true. In this case, suppose a combination of functions $f_1, f_2, \dots, f_n$ that satisfies $x'_1 = f_1(x_1, x_2, \dots, x_n)$, $x'_2 = f_2(x_1, x_2, \dots, x_n)$, $\dots$, $x'_n = f_n(x_1, x_2, \dots, x_n)$. When there is no combination of $x_1, x_2, \dots, x_n, f_1, f_2, \dots, f_n$ that makes $I(x_1, x_2, \dots, x_n)$ and $P(x_1, x_2, \dots, x_n)$ true, and $F(x_1, x_2, \dots, x_n, f_1, f_2, \dots, f_n) = F(x_1, x_2, \dots, x_n, x'_1, x'_2, \dots, x'_n)$ false, the operation is proved to be applicable.

The following fact should be noted. In the case of B method, operation can be non-deterministic in the abstract machines. In this case, the existence of one possible combination of variables that satisfies the invariant in the range of the result of the operation is sufficient for the feasibility of the operation. However, any combination of variables that satisfies the condition described in the operation but does not hold the invariant can be the solution of SAT because in that case $I(x_1, x_2, \dots, x_n)$ is true and $I(x'_1, x'_2, \dots, x'_n)$ is false. This makes the verification undecided. To avoid this situation, the result of the operation must hold the invariant. The same is true when operations are written with postconditions.

4.3 Actual verification

Yices [11] (Version 1), an SMT solver, was used. Two types of languages are available in Yices. One is based on SMT-LIB, common with many SMT solvers. The other is specific to Yices. In this paper, the latter was adopted because an enumerated type, referred to as “scalar type,” is available. A variable of scalar type is declared as follows:

```
(define-type DIRECTION (scalar up down))
```

Here, a DIRECTION type that takes the value of up or down is declared. Then, by declaring

```
(define dir-o::DIRECTION)
```

a variable dir-o of type DIRECTION can be declared.

Like other SMT solvers, Yices uses Polish notation, in which the operator is placed first, then the operands follow. For example, the proposition $(A > B) \Rightarrow ((A = C) \wedge (B = D))$ is described as follows:

```
(=> (> A B) (and (= A C) (= B D)))
```

Variables are extracted from B specifications. Types and variables to be defined before and after operations are

applied. In Yices, the specification is given in a single module while the specification is structured in several modules in B.

After the declaration of variables, invariants are described. In Yices, conditions between variables are referred to as “context,” and given with assert or assertion. For example, declaration

```
(assert (= dir-o up))
```

restricts dir-o to being always up. assert can be used as many times as possible. Therefore, the invariant can be split into several assertions. As part of the verification process a variable, t1 was defined that shows the truth of the invariant as follows:

```
(define t1:: bool (and (not (and (= dir-o down) (= dir-d up) ...))
```

In this case, t1 is equal to the formula following the first and. The variable t2 was also defined that shows the truth of the invariant after the operation. Then, the following assertion

```
(assert (and (= t1 true) (= t2 false)))
```

specifies to find a solution that satisfies the precondition but not the postcondition. When a relationship is described between variables before and after an operation is applied, check command starts searching solutions. If there is no solution, “unsat” is displayed. The following command specifies to show an evidence that satisfies all of assertions:

```
(!set-evidence true)
```

When an evidence is shown, it means a counterexample to violate the invariants.

Twenty-four variables were used in simplified automatic systems, and 66 variables in semi-automatic block systems. For each of the system verifications it takes less than one second for each operation. It takes a few seconds to check all 40 of operations. When there are some mistakes, a counterexample will be found or the search will be timed out. In this case, the description should be modified and tried again. The next verification will also be finished in a few seconds.

At first the number of blocks was specified as arbitrary for automatic block systems. In this case, quantifiers were used. Then, the search was not finished in one hour. Following this, the number of blocks was simplified to three and the quantifiers were removed. The search was finished in a few seconds. The number of variables used in the specification is 60 while it is 36 before the modification. This result shows that the difficulty of the verification is not dependent on the number of variables, but on the type of proposition. It is considered that the time is not increased even if the number of blocks is increased. However, it takes a long time to describe the specification.

5. Comparison between SMT solvers and B-methods

During the theorem proving with B, tens of thousands of proof obligations are generated, which require several weeks to be solved. In contrast, verification with SMT solvers is complete in just a few seconds. If there are some mistakes, re-verification is required. The time required for re-verification using an SMT solver is considerably less than the time required for the B-method. This means

that SMT solvers are effective as far as verification is concerned. Current SAT solvers are supposed to handle millions of variables. When interlocking systems with hundreds of routes are considered, the number of variables is considered to be from thousands to tens of thousands, which is within the range of SAT solvers.

Model checking is used as a method for verifying interlocking systems [5]. However, model checking is associated with state explosion problems when there are many variables. For example, when the on and off states are considered for each route, there are about a thousand states for ten routes, and a million states for twenty routes. On the other hand, it may be possible to check some properties with SAT/SMT solvers, but further investigation is required.

It is noticed that SAT/SMT solvers essentially only decide SAT. When the proposition is given incorrectly, an incorrect conclusion will be drawn. Therefore, automatic translation from specifications to input to SAT/SMT solvers is required in order to apply SAT/SMT solvers to actual equipment.

As the verification of automatic block systems shows, SMT solvers can handle quantifiers, but they offer a reduced level of performance. SAT/SMT solvers have good points and bad points. As for automatic block systems and interlocking systems, which are originally constructed with relays, SAT/SMT solvers seem to be effective. However, more consideration is required for calculation of the brake pattern and the track database in which numbers and sets are used. In this case B is more effective.

B is also effective for yielding verified code. A method must be selected taking the characteristics into account.

6. Conclusions

Theorem proving was carried out using B-method, and automatic verification of automatic block systems for single lines was performed via SMT solver. With B-method, numbers and sets can be used as components of specifications, and verified codes are available, while it takes a long time to finish verification especially when many variables are used. On the other hand, SMT solvers can handle many

variables, while natural numbers can be handled well. It would be beneficial to apply both methods to various other systems, and also to examine the application of various other methods.

References

- [1] Terada, N. and Fukuda, M., "Application of Formal Methods to the Railway Signalling Systems," *Quarterly Report of RTRI*, Vol. 43, No. 4, pp. 169–174, 2002.
- [2] Terada, N., "Application of Formal Methods to Signalling Systems Using Techniques Based on Stepwise Refinement," *Quarterly Report of RTRI*, Vol. 49, No.3, pp. 168–172, 2008.
- [3] Borälv, A., "Case Study: Formal Verification of a Computerized Railway Interlocking," *Formal Aspects of Computing*, Vol.10, pp. 338–360, 1988.
- [4] Behm, P., Benoit, P., Faivre, A. and Meynadier, J-M, "Météor: A Successful Application of B in a Large Project," *Proc. of FM'99*, Vol. I, LNCS 1708, pp. 369–387, 1999.
- [5] Haxthausen, A. E., Kjær, A. A, and Le Bliguet, M., "Formal Development of a Tool for Automated Modeling and Verification of Relay Interlocking Systems," *Proceedings of FME 2011*, LNCS 6664, pp. 118–132, 2011.
- [6] Abrial, J-R, *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, 1998.
- [7] Biere, A., Heule, M. J. H., van Haaren, H. and Walsh, T.(eds): *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, Vol. 185, IOS Press, 2009.
- [8] *Standard Connection Diagrams for Interlocking (2nd Edition)*, Japan Railway Electrical Engineering Association, 2002 (In Japanese).
- [9] *Standard Connection Diagrams for Interlocking (CTC and ARC) (2nd Edition)*, Japan Railway Electrical Engineering Association, 2000 (In Japanese).
- [10] Fitzgerald, J., and Larsen, P. G., *Modelling Systems – Practical Tools and Techniques in Software Development*, Cambridge University Press, 1998.
- [11] <http://yices.csl.sri.com>