Model Checking: A Formal Method for Safety Assurance of Logistic Systems

Robert Meolic, Tatjana Kapus, Zmago Brezočnik
Faculty of Electrical Engineering and Computer Science
Smetanova 17, SI-2000 Maribor, Slovenia
{meolic,kapus,brezocnik}@uni-mb.si

ABSTRACT

Model checking is a formal method widely used in computer science for verification of concurrent systems, e.g. communication protocols. The method requires that a system is given with a graph and properties of the system are given as a set of logic formulas. Model checking is an algorithmic method and it can be carried out fully automatically by fixpoint calculation. This paper describes how to use model checking for safety assurance of logistic systems.

1 Introduction

Safety assurance of logistic systems is a multilayer problem. In general, it can be partitioned into two parts:

- the design of traffic networks that enable safe traffic flow realisation and
- the care of all subjects involved in the traffic.

Although the behaviour of traffic participants is somehow standardised by traffic rules, it is not always predictable in praxis. Therefore, particular attention is paid to the design of traffic networks, which have to prevent all participants from dangerous situations. With the great progress in computer science many new efficient tools turn up, that can serve for safety assurance of logistic systems [10]. Moreover, the problems which have to be solved here are not very different than the problems that appear in ensuring the correctness of various concurrent systems in computer science, electronics, and telecommunications.

Concurrent system is a system composed of two or more components which are concurrently executed and communicate with each other. Having a road traffic in mind, this is a usual approach, e.g. the components are cars, pedestrians, traffic signalling etc. To use computer methods for reasoning about the system, a behaviour of each component should be described by a formal mathematical notation. More realistic and precise description of the system leads to a more realistic and useful results. Many statistical methods for planning various kind of traffic have been proposed, which are used for its rationalisation. However, statistical methods can not be used trustworthy for safety assurance, because they take into account an average traffic stream. Dangerous situations, incidents and road accidents are neither "average" events nor a consequence of an "average" behaviour of traffic participants.

In last years, a formal method called *model checking* became popular for verification of concurrent systems [2, 4, 6]. The method requires that a system is given with a graph, which describes the system behaviour in terms of states and actions. This approach can be easily applied for designing road traffic networks. For example, traffic lights are implemented as finite state machines and their specifications can be directly used for model checking. In comparison with statistical formulas, a graph is much more transparent, understandable and supports more precise specifications. Each component of the given system can be represented with its own graph, which enables modelling of specialities and exceptions, for example a driver ignoring traffic regulations.

Using model checking, *the properties* of the system are given as a set of logic formulas. The requirements are expressed as propositions, the validity of which can be checked in the given system, for example: if the traffic lights are green for the pedestrians, then they are red for the cars. Model checking is automatic and does not need an interaction with the user. Thus, the user can concentrate in specifying the model and properties.

In this paper we present model checking, where an *action computation tree logic* (ACTL) is used for specifying the properties of the system [1, 3, 5, 9]. First, we briefly describe how to model a concurrent system. Then, ACTL syntax is presented and instructions for creating ACTL formulas are given. In section 4, we describe an example of formal verification of *a crossing of a road and a railway* with ACTL model checking. The system is complex enough, so that the capacity of formal methods can be shown. Namely, the given model of crossing has a hidden error, which cannot be easily found without a formal verification.

2 How to model a concurrent system

We will model the behaviour of concurrent systems with a simple process algebra similar to CCS [1]. The system will be described by a set of asynchronously executed communicating processes. Each process executes input and

output actions and transits between states. The action is an input action if its name terminates by '?', and it is an output action if its name terminates by '!'. Two actions whose names differ only in the terminating symbol, e.g. $a?$ and $a!$, are called complementary actions. The processes synchronise with each other by simultaneously performing complementary actions. Each process has exactly one initial state (double-circled in the figures) where it starts. The sequence of transitions which the process can perform is called a *path*.

An example of a process is in Figure 1. It represents a simple crossing of a road and a railway. A train can approach the crossing (action **train!**) and then cross it (action **trainCross!**). Also, a car can approach the crossing (action **car!**) and then cross it (action **carCross!**). The process consists of 6 states and 10 transitions.

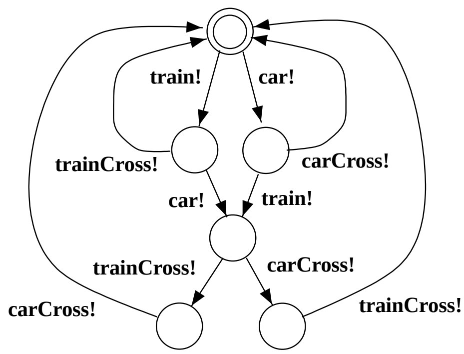

Figure 1: A simple description of a crossing

If the system consists of two or more processes, they must be composed together before the formal verification is carried out. The compound process represents the behaviour of the whole system. In the stage of process composition we distinguish actions which serve for synchronisation of processes (internal actions) and actions which are used for communication with the environment (external actions). We will type external actions bold.

3 How to create ACTL formulas

ACTL is a propositional logic extended with path quantifiers and temporal operators. ACTL formulas may include:

- constants *true* and *false*,
- *action variables* $a!, a?, \ldots$,
- standard boolean operators $\wedge$, $\vee$, and $\neg$,
- *temporal operators* **X** ("for the next transition"), **F** ("for some transition in the future"), **G** ("for all transitions in the future"), **U** ("until"), $\underline{\mathbf{U}}$ ("unless"), and
- *path quantifiers* **E** ("there exists a path") and **A** ("for all paths").

By definition, χ, φ, and γ are syntactically correct *action formula*, *state formula* (also called ACTL formula), and *path formula*, respectively, iff they satisfy the following syntactic rules:

$$\begin{aligned}
\chi &::= \; \mathit{true} \mid \mathit{false} \mid a \mid \neg\chi \mid \chi \wedge \chi' \mid \chi \vee \chi' \\
\varphi &::= \; \mathit{true} \mid \mathit{false} \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \mathbf{E}\,\gamma \mid \mathbf{A}\,\gamma \\
\gamma &::= \; \mathbf{X}\{\chi\}\,\varphi \mid \mathbf{F}\{\chi\}\,\varphi \mid \mathbf{G}\,\varphi\{\chi\} \mid \\
&\qquad [\varphi\{\chi\}\,\mathbf{U}\,\{\chi'\}\varphi'] \mid [\varphi\{\chi\}\,\underline{\mathbf{U}}\,\{\chi'\}\varphi']
\end{aligned}$$

An action for which ACTL formula χ is valid is called χ-action. A state where ACTL formula φ is valid is called φ-state. The transition from state p to state q where action formula χ is valid for the action executed during this transition and ACTL formula φ is valid in state q is called (χ, φ)-*transition*.

Let π be a path in the process. Then, the meaning of temporal operators can be explained as follows (see Figure 2).

- Formula $\mathbf{X}\{\chi\}\,\varphi$ is valid on path π iff the first transition on the path is (χ, φ)-transition.
- Formula $\mathbf{F}\{\chi\}\,\varphi$ is valid on path π iff there exists a (χ, φ)-transition on the path.
- Formula $\mathbf{G}\,\varphi\{\chi\}$ is valid on path π iff ACTL formula φ is valid in the first state of this path and all transitions on the path are (χ, φ)-transitions.
- Formula $[\varphi\,\{\chi\}\,\mathbf{U}\,\{\chi'\}\,\varphi']$ is valid on path π iff ACTL formula φ is valid in the first state of this path and the path begins with a finite sequence of (χ, φ)-transitions followed by a (χ', φ')-transition.
- Formula $[\varphi\,\{\chi\}\,\underline{\mathbf{U}}\,\{\chi'\}\,\varphi']$ is valid on path π iff formula $[\varphi\,\{\chi\}\,\mathbf{U}\,\{\chi'\}\,\varphi']$ is valid on this path or formula $\mathbf{G}\,\varphi\{\chi\}$ is valid on this path.

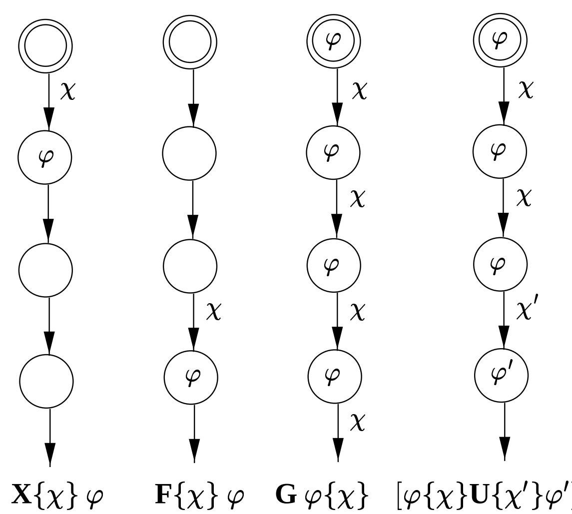

Figure 2: The meaning of temporal operators

In ACTL formulas each temporal operator is immediately preceded by a path quantifier forming thus an *ACTL operator*. When path quantifier **E** is used, we require that the property expressed by the formula is valid for at least one path starting in the given state of the process. Otherwise, when path quantifier **A** is used, we require that the property expressed by the formula is valid for all paths starting in the given state of the process.

An ACTL formula can be nested in another one. Moreover, the constant *true* can be omitted in many cases:

$$\begin{aligned} \mathbf{E}\,[\{\chi\}\,\mathbf{U}\,\{\chi'\}\,\varphi'] &= \mathbf{E}[true\,\{\chi\}\,\mathbf{U}\,\{\chi'\}\,\varphi']\,, \\ \mathbf{A}\,[\varphi\,\mathbf{U}\,\varphi'] &= \mathbf{A}[\varphi\,\{true\}\,\mathbf{U}\,\{true\}\,\varphi'] \end{aligned}$$

There are also two widely accepted abbreviations of ACTL operators:

$$\begin{aligned} <\chi>\varphi &= \mathbf{EX}\{\chi\}\,\varphi \\ [\,\chi\,]\,\varphi &= \neg\mathbf{EX}\{\chi\}\,\neg\varphi \end{aligned}$$

The meaning of ACTL formula $<\chi>\varphi$ is the same as the meaning of ACTL formula $\mathbf{EX}\{\chi\}\,\varphi$. ACTL formula $[\,\chi\,]\,\varphi$ is valid in the given state of the process iff each transition from this state leads to a φ-state whenever the action executed during the transition is χ-action.

In praxis, some experiences are necessary to form usable ACTL formulas. To help in this work, many patterns can be introduced, which serve as templates. Here are some simple but useful patterns.

- Action $a!$ may be executed in the future: $\mathbf{EF}\ \{a!\}$
- Action $a!$ will be executed in the future: $\mathbf{AF}\ \{a!\}$
- Action $a!$ will be executed infinitely often in the future: $\mathbf{AG\ AF}\ \{a!\}$
- After action $a!$ is executed, action $b!$ will be executed: $\mathbf{AG}\ [a!]\ \mathbf{AF}\ \{b!\}$
- After action $a!$ is executed, action $b!$ will never be executed: $\mathbf{AG}\ [a!]\ \mathbf{AG}\ \{\neg b!\}$
- Action $b!$ may be executed before action $a!$ is executed: $\mathbf{E}\,[\{\neg a!\}\,\mathbf{U}\,\{b!\}]$
- Action $a!$ will not be executed unless action $b!$ is executed: $\mathbf{A}\,[\{\neg a!\}\,\mathbf{U}\,\{b!\}]$

4 An example of ACTL model checking

We described and verified a crossing of a road and a railway [8]. The crossing consists of two barriers and train traffic lights. The barriers have to be closed when the train crosses. The train can cross only if the traffic lights are green, otherwise it has to stop.

A simple process representing a crossing of a road and a railway is already shown in Figure 1. However, that description is not very useful because it does not describe the behaviour of the participants separately and therefore it represents more or less only evident and expected scenarios. A more adequate approach is to describe the system as a composition of three processes representing the crossing, car, and train (Figure 3).

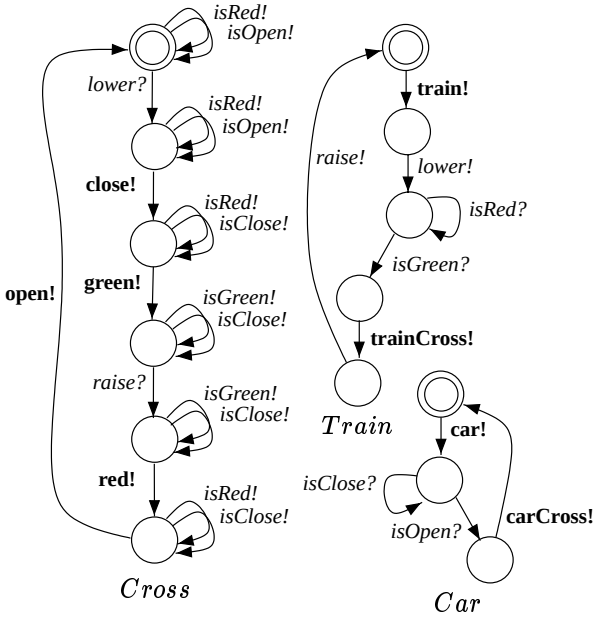

Figure 3: The crossing of a road and a railway

Process $Cross$ represents a crossing composed of barriers and train traffic lights. Actions *isRed!*, *isGreen!*, *isOpen!*, and *isClose!* inform a car and a train about the state of the barriers and train traffic lights. Actions *lower?* and *raise?* are used to change the state of the barriers. Actions **red!**, **green!**, **close!**, and **open!** signal a change of the state of the barriers and train traffic lights to the environment.

Process $Train$ and process Car represent a train and a car, respectively. Actions **train!** and **car!** denote that a train or a car is approaching the crossing. Actions **trainCross!** and **carCross!** denote that a train or a car is crossing. Other actions are used to communicate with process $Cross$.

We stated and verified the following ACTL formulas that assure the safety of a crossing of a road and a railway:

- when the traffic lights turn red, the train will not cross unless the traffic lights turn green:
  $\mathbf{AG}\,[\,\mathbf{red!}\,]\,\mathbf{A}\,[\{\neg\mathbf{trainCross!}\}\,\mathbf{U}\,\{\mathbf{green!}\}]$
- when the barriers are open, the train will not cross unless the barriers lower:
  $\mathbf{AG}\,[\,\mathbf{open!}\,]\,\mathbf{A}\,[\{\neg\mathbf{trainCross!}\}\,\mathbf{U}\,\{\mathbf{close!}\}]$
- when a train is approaching the crossing, the train will not cross unless the barriers lower:
  $\mathbf{AG}\,[\,\mathbf{train!}\,]\,\mathbf{A}\,[\{\neg\mathbf{trainCross!}\}\,\mathbf{U}\,\{\mathbf{close!}\}]$
- when the barriers are close, they will not raise unless the train crosses:
  $\mathbf{AG}\,[\,\mathbf{close!}\,]\,\mathbf{A}\,[\{\neg\mathbf{open!}\}\,\mathbf{U}\,\{\mathbf{trainCross!}\}]$

- it never happens that both a car and a train are able to cross at the same time:

$$\neg \mathbf{EF}\,(\mathbf{EX}\,\{\mathbf{trainCross!}\} \wedge \mathbf{EX}\,\{\mathbf{carCross!}\})$$

We verified ACTL formulas with our own tool EST [7], which is based on binary decision diagrams. The results of model checking show a dangerous situation which can arrise in the system. Namely, the last stated ACTL formula is not valid in the initial state of the compound system. A path showing why the particular ACTL formula is not valid is called *counterexample*. In our case, the counterexample is a path where a car drives into the crossing when barriers are open, but then it does not leave the crossing (it does not perform action **carCross!**). Because the process $Cross$ does not examine, whether the car has leaved the dangerous area, the traffic lights may turn green before the car leaves the crossing and therefore a collision can appear.

5 Conclusion

Safety assurance of logistic systems consists of various problems similar to many others which arise in ensuring the correctness of concurrent systems. This paper presents how a verification method called model checking can be successfully applied for reasoning about the safety of traffic flow.

Model checking is a typical formal method where a relationship between a design and a specification of a system is formally established. It is obvious that more detailed description of the system design leads to more accurate results. However, adding new details into the graph is not very complicated and if an appropriate approach is used, the description of large systems can be very clear and understandable, too. This is a great advantage of model checking over many other methods.

References

[1] C. Bernardeschi et al. A Formal Verification Environment for Railway Signaling System Design. *Formal Methods In System Design*, 12(2):139–161, 1998.

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992. Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, 1990, 428–439.

[3] R. Mateescu. Formal Description and Analysis of a Bounded Retransmission Protocol. In *Proceedings of the COST 247 International Workshop on Applied Formal Methods in System Design*, pages 98–113, Maribor, Slovenia, 1996.

[4] K. L. McMillan. *Symbolic Model Checking*. PhD thesis, Carnegie Mellon University, May 1992. Technical report CMU-CS-92-131.

[5] R. Meolic. Checking correctness of concurrent systems behaviour. Master's thesis, Faculty of Electrical Engineering and Computer Science, Maribor, November 1999. In Slovene.

[6] R. Meolic, T. Kapus, and Z. Brezočnik. Exploring Properties of a Bounded Retransmission Protocol with VIS. *Journal of Computing and Information Technology CIT*, 7(4):311–321, 1999.

[7] R. Meolic, T. Kapus, and Z. Brezočnik. The Efficient Symbolic Tools Package. In *Proceedings of the 8th International Conference Software, Telecommunications and Computer Networks (SoftCOM 2000), Split, Croatia*, October 2000. To be published.

[8] R. Meolic, T. Kapus, and Z. Brezočnik. Verification of concurrent systems using ACTL. In M. H. Hamza, editor, *Applied informatics: proceedings of the IASTED international conference AI'2000, Innsbruck, Austria*, pages 663–669, Anaheim, Calgary, Zürich, February 2000. IASTED/ACTA Press.

[9] R. De Nicola, A. Fantechi, S. Gnesi, and G. Ristori. An action based framework for verifying logical and behavioural properties of concurrent systems. *Computer Networks and ISDN Systems*, 25:761–778, 1993. In *Proceedings of 3rd Workshop on Computer Aided Verification*, 1991.

[10] Virtual Library. Formal Methods. `http://www.comlab.ox.ac.uk/archive/formal-methods.html`.