VERIFICATION OF CONCURRENT SYSTEMS USING ACTL

ROBERT MEOLIC, TATJANA KAPUS, ZMAGO BREZOČNIK
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova ul. 17, SI-2000 Maribor, Slovenia
{meolic,kapus,brezocnik}@uni-mb.si

Contents:
1. Introduction
2. Action computation tree logic
3. Abbreviations of ACTL formulas
4. An example of using ACTL
5. Conclusion
VERIFICATION OF CONCURRENT SYSTEMS USING ACTL

Introduction

- A typical verification problem consists of formally establishing a relationship between a design and a specification.
- LTSs and process algebras (e.g. CCS, Lotos) are convenient tools for describing and reasoning about the behaviour of concurrent systems.
- Formal verification can be done
  - by equivalence checking of given two specifications,
  - by model checking.
- The model checking is roughly a technique for deciding whether the given finite structure defines a model of the given propositional temporal logic formula.
- Action computation tree logic (ACTL) is a mathematical notation, which is very suitable for property specification of systems described by LTSs or process algebras.
An example of concurrent system

**A crossing of a road and a railway:** The crossing consists of two barriers and train traffic lights.

- The barriers have to be kept down when the train crosses.
- The train can cross only if the traffic lights are green, otherwise it has to stop.
Labelled transition systems

Definition:
LTS is a triple $L = (S, \mathcal{A}, \delta)$ where:

- $S$ is a set of states;
- $\mathcal{A}$ is a finite, non-empty set of actions;
- $\delta$ is the transition relation.

A **path** from state $p_1$ is an infinite sequence of states $\pi = p_1, p_2, \ldots$ where:

$$(p_i, a_i, p_{i+1}) \in \delta, \ p_i \in S, \ a_i \in \mathcal{A}$$

In each LTS $L = (S, \mathcal{A}, \delta)$, exactly one state $p_0 \in S$ is chosen to be the initial state.

A **parallel composition** of LTSs $L_1, L_2, \ldots, L_n$ is defined as an LTS $||L_1, \ldots, L_n|| = (S', \mathcal{A}', \delta')$ where:

- \[ S' = S_{L_1} \times \ldots \times S_{L_n} \]
- \[ \mathcal{A}' = \mathcal{A}_{L_1} \cup \ldots \cup \mathcal{A}_{L_n} \]
- \[ (p_i, a, p'_i) \in \delta_{L_i} \land \forall j \neq i. (a \notin \mathcal{A}_{L_j}) \implies ((p_1, \ldots, p_i, \ldots, p_n), a, (p_1, \ldots, p'_i, \ldots, p_n)) \in \delta' \]
- \[ (p_i, a, p'_i) \in \delta_{L_i} \land (p_j, a, p'_j) \in \delta_{L_j} \implies ((p_1, \ldots, p_i, \ldots, p_j, \ldots, p_n), a, (p_1, \ldots, p'_i, \ldots, p'_j, \ldots, p_n)) \in \delta' \]
Action computation tree logic

Syntax:

\[ \chi ::= \text{true} \mid \text{false} \mid a \mid \neg \chi \mid \chi \land \chi' \]
\[ \varphi ::= \text{true} \mid \text{false} \mid \neg \varphi \mid \varphi \land \varphi' \mid E \gamma \mid A \gamma \]
\[ \gamma ::= [\varphi \{ \chi \} \cup \{ \chi' \} \varphi'] \mid [\varphi \{ \chi \} \cup \{ \chi' \} \varphi'] \]

ACTL formulas with basic ACTL operators:

\[ E[\varphi \{ \chi \} \cup \{ \chi' \} \varphi'] \]
\[ A[\varphi \{ \chi \} \cup \{ \chi' \} \varphi'] \]

ACTL formulas with derived ACTL operators:

\[ \text{EX}\{\chi\} \varphi = E[\text{true} \{\text{false}\} \cup \{\chi\} \varphi] \]
\[ \text{AX}\{\chi\} \varphi = A[\text{true} \{\text{false}\} \cup \{\chi\} \varphi] \]
\[ \text{EF}\{\chi\} \varphi = E[\text{true} \{\text{true}\} \cup \{\chi\} \varphi] \]
\[ \text{AF}\{\chi\} \varphi = A[\text{true} \{\text{true}\} \cup \{\chi\} \varphi] \]
\[ \text{EG} \varphi \{\chi\} = E[\varphi \{\chi\} \cup \{\text{false}\} \text{false}] \]
\[ \text{AG} \varphi \{\chi\} = A[\varphi \{\chi\} \cup \{\text{false}\} \text{false}] \]
Abbreviations of ACTL formulas

\[
<\chi> \varphi = \text{EX}\{\chi\} \varphi
\]
\[
[\chi] \varphi = \neg\text{EX}\{\chi\} \neg \varphi
\]
An example of using ACTL

We verified the model of a railway crossing.

LTSs Cross, Train, and Car were composed. The properties of the compound system were given in ACTL. We stated and verified the following properties:

- when the traffic lights turn red, the train will not cross unless the traffic lights turn green:
  \[ \text{AG} [\text{red}] \text{A} [\{\neg \text{trainCross}\} \text{U} \{\text{green}\}] \]

- when the barriers are open, the train will not cross unless the barriers lower:
  \[ \text{AG} [\text{open}] \text{A} [\{\neg \text{trainCross}\} \text{U} \{\text{close}\}] \]

- when a train is approaching the crossing, the train will not cross unless the barriers lower:
  \[ \text{AG} [\text{train}] \text{A} [\{\neg \text{trainCross}\} \text{U} \{\text{close}\}] \]

- when the barriers are close, they will not raise unless the train crosses:
  \[ \text{AG} [\text{close}] \text{A} [\{\neg \text{open}\} \text{U} \{\text{trainCross}\}] \]

- it never happens that both a car and a train are able to cross at the same time:
  \[ \neg \text{EF} (\text{EX} \{\text{trainCross}\} \land \text{EX} \{\text{carCross}\}) \]
Conclusion

- ACTL formulas are used for specifying properties of concurrent systems.
- ACTL is more suitable for specifying properties of concurrent systems described with LTSs or process algebra than other temporal logics (e.g. CTL).
- Model checking is more efficient method for system verification than equivalence testing.
- Drawback: model checking is not so appropriate when the complete behaviour of the system has to be verified. It is hard to determine whether the given ACTL formulas comprise all significant properties of the system or they do not.
- The presented work is a part of a project, where a tool for verification of concurrent systems is being developed. Our tool is based on BDDs and is already capable of checking observational equivalence, testing equivalence and ACTL model checking.
- In the future, our work will be focused on:
  - the introduction of normal form ACTL formulas,
  - extension of our tool to provide counterexamples.