1. Objective

Problem: Standard model checkers return single execution paths as counter-examples while CTL has branching counter-examples in general.

Approach: Generate branching counter-examples annotated with parts of (the negation of) the violated property.

Example: A buggy scheduler executing prioritized processes; at each step, the scheduler can execute a process or skip it.

- This model does not satisfy the property that the scheduler will never be able to run the process with priority $s_1$ unless it is the only possibility, expressed as $A[AX ~s_1 W AX (P_0 \lor P_1)]$ in CTL ($P_0$ means that process $x$ is running, $P_1$ means that no process is running).
- The full counter-example is branching, but model checkers return partial information.

This work is based on Clarke et al. and Rasse.

2. Tree-like Annotated Counter-Examples

TLACEs are full counter-examples for ACTL—and full witnesses for ECTL—in general; they completely explain a violation. A TLACE is defined by:

$$n := \text{node}([\{b\} ~\neg \psi], \{(E \pi \cdot p)^s\}, \{(A \pi)^s\})$$

$$p := (n^s) | (n^s, \text{loop}(n))$$

We are interested in consistent TLACEs that are adequate for a model $M$ and a property $\phi$, i.e., TLACEs that come from $M$ and correctly explain $\phi$.

3. Generating Counter-Examples

The generating algorithm works recursively on the structure of the formula; relies on sub-algorithms to extract paths from the model witnessing temporal formulas.

```
switch \phi do
    case ... case E[\psi_1 U \psi_2]
        \{s_1, ..., s_m \} \rightarrow E[\text{explain}(M, s, \psi_1, \psi_2)]
        \text{for } s_i \in \{s_1, ..., s_m\} \text{ do}
        \text{\{ } p \leftarrow \{ \text{explain}(M, s_i, \psi_2) \} \}
        \text{\{ } p \leftarrow \text{explain}(M, s_i, \psi_2) \}
        \text{\{ return node(\{ \text{explain}(M, s_0, \psi_0) \} \} } \}
    case ...
```

4. Tools

- NuSMV: modified to generate and export TLACEs.
- TLACE Visualizer: displaying and manipulating TLACEs.
- TLACEs displayed as graphs, the inspecting nodes: state information, annotations and branches).
- Folding/unfolding branches.
- Displaying nodes information along a path in the graph.

5. Conclusion and further work

Contributions:
- Formalization of branching annotated counter-examples for ACTL;
- Generating algorithm;
- Implementation in a symbolic model checker;
- Tool for visualizing and manipulating the counter-examples.

Further work:
- Extend the formalization to richer logics like epistemic temporal logics;
- Interactive generation of branches: explain only the part relevant for the user;
- Explain $A$ operators through interactive game: the user can try to show the satisfiability while the system shows him that it is impossible.

References


A. CTL Model Checking

**Model checking:** checks whether a (finite state) model satisfies a (temporal logic) property; if not, returns a counter-example.

**CTL:** branching temporal logic expressing properties about the execution tree of the model.

- $AX \phi$: all successors satisfy $\phi$
- $EG \phi$: there exists a path where all states satisfy $\phi$
- $A[\psi W \phi]$: for all paths, $\phi$ is true up to a state satisfying $\psi$

**ACTL:** only $A$ quantifiers and negations applied to atomic propositions.

Grascomp’s Day, November 3rd, 2011, Université Libre de Bruxelles, Brussels, Belgium.