A Closed-loop Model-based Design Approach Based On Automatic Verification and Transformation

Kun Zhang
Jonathan Sprinkle
```java
public void testFoo() throws Exception {
    String answer = 42;
}
```

Type mismatch: cannot convert from int to String

1 quick fix available:

- Change type of 'answer' to 'int'

Press 'F2' for focus
```java
void foo() throws IOException {
    FileReader reader = new FileReader("file");
    int ch;
    while ((ch = reader.read()) != -1) {
        System.out.println(ch);
        reader.read();
    }
}
```
```java
private List<String> fList;
public void method() {
    fList.add("A string");
}

Add type parameters to 'List'
Infer Generic Type Arguments...

'Ctrl+Enter' to fix 2 problems of same category in file

... public class E1 {
    private List<String> fList;
    public void method() {
    ...
Constraints: prevent known incorrect from being created

Error message:

- Paradigm Violation! Cannot create within parent Model
  FastUAVStateTransitionDiagram of type StateTransitionDiagram an object with kind StateTransitionDiagram
- Paradigm violation! Cannot make connection for source: 1 (State) to destination: 2 (State) in aspect StateTransition within parent: FastUAVStateTransitionDiagram (StateTransitionDiagram)
Error: Your composed state model violates concurrency rules that guarantee avoiding race conditions.  
1 quick fix available....
• As shown in the figure, we propose to close the loop of model based design procedure by:
  – (i) incorporating behavioral constraints into the DSML
  – (ii) automating the verification process
  – (iii) generating model transformations based on a transformation library constructed in advance
  – (iv) running those model transformations automatically

• Behavioral constraints are always interpreted into the verification code in order to automate the verification.
• Verification results are then fed into the transformation generator, and the generator outputs a transformation solution.
• The loop will keep going on until all constraints are satisfied.
• FSM is a common tool for behavioral modeling of discrete systems. We approach the process of closing the loop by exploring the process as applied to automating the distributed FSM modeling.

The metamodel of the FSM DSML. The dashed rectangle encloses the part required for FSM modeling, the rest is for modeling behavioral constraints.
A deterministic FSM is a 5-tuple $(A, S, s_0, \delta, F)$, where $A$ is the input alphabet, $S$ is a finite set of states, $s_0 \in S$ is the initial state, $\delta : S \times A \rightarrow S$ is the state transition function, and $F$ is the set of accepting states.

$S$, $s_0$ and $\delta$ are defined structurally in the model. Behavioral requirements constrain on $F$ and $A$. 

A set of FSMs, where $i$ is the integer index representing a specific FSM, and $k$ is the discrete time step.

The verification engine outputs the verification result $V_k$. $V_k$ provides adequate information for tracking a single problem node $N_k$. After performing the transformation $T_k$ on ${FSM_i}_k$
Overview of the Closed-loop Automation

- (i) Incorporation of Constraints in DSML
  - Two aspects of problems, deadlock and interactively behavioral inconsistency, are taken into consideration. Constraints are expressed in the DSML.

- (ii) Verification Synthesis
  - The idea of constructing verification is to generate the Promela code according to the FSMs and the constraints. The FSMs model is translated into the code framework. Constraints will be interpreted into logic statements containing assertions or printing clauses, on which problem tracing relies.

- (iii) Transformation Solutions
  - Design problem-specific transformation solutions.

- An illustrative example is shown as follows.
• Constraint 1:
  • A constraint is given by the ‘Accepting State Mark’, which will invalidate the trail
    – (State 1 → State 2 → State A)
  • and allow only 2 trails,
    – (State 1 → State A → State2 → StateB)
  • and
    – (StateA → State1 → State2 → State B).

A model example showing two concurrent state machines. The squares represent states. The arrow within a square denotes a transmitted event during the execution of the container state. The small shadowy rectangle is the event required for firing the attached transition. The dashed gray box contains the set of accepting states, which are added if constraints on F are required.
Suppose the modeler’s intent is to let State 1 happen before State A. The activity model in the 3rd column of the figure is used to specify such constraint.

- Two trails
  - (State 1 → State A → State2 → StateB)
  - (StateA → State1 → State2 → State B)
- pass the Constraint 1.
- But the later one will be filtered out by the Constraint 2.
When it comes to verification, FSMs and the constraints are translated into Promela code.

The generated Promela code must be consistent in behavior with the generated application code (e.g., the way they handle events).

Then SPIN will run the Promela code in verification mode. If a constraint violation detected, SPIN stops and generates the trail log. We then feed the Promela code with the log into SPIN to recur the violation with details printed out. Based on the details, we can infer the problem node that causes the violation.
Based on the above 2 types of constraints, we list the possible problems in the order of processing priorities: (i) the event required for a transition does not exist in the model; (ii) the event will not happen after the occurrence of its receptor; (iii) a circular wait exists; (iv) a behavioral trail breaks the activity model constraint. These problems are the problem nodes, and each node maintains references to its respective transformation solutions.

The trail (State 1 $\rightarrow$ State 2 $\rightarrow$ State A) will cause deadlock since Event 1 is released before the transition between State A and B can receive it, and the (ii) node will be responsible. A corresponding solution example for this problem is shown below.
• After running the automation loop, the final result is shown in the figure.
• A transitory state is added at the very beginning in Task2, and the transition to State A will happen only after the State 1 has been executed. State B will never be reached unless State 2 is executed and the Event 1 has been transmitted. Thus, the automation loop produces the model that sticks to the trail (State 1 → State A → State 2 → State B) as desired.
• The example to apply our research is a model-based design work for configurable sensor network in river environment. The domain background is that, a group of drifters, equipped with propellers and sensors, are released into the river for the purpose of real-time water quality monitoring.

• The approach is to design the DSML utilizing FSM concepts as language structure and using domain concepts as events.
• Results
  – Apply complex constraints to correcting models (instead of preventing incorrect models from being built)
  – Requires simulation or verification engines
  – Analogous to Eclipse’s “Did you mean…” function

• Future Work
  – Application to larger concurrent state models
  – Introduction of new verification engines and tools
  – Integration of continuous-time constraints to modify structure of the models

“Self-Reconfigurable Sensors in River Environments” NSF CNS-0930919, with Sonia Martinez (UC San Diego) and Alex Bayen (UC Berkeley)

“CAREER: Domain-Specific Modeling Techniques for Cyber-Physical Systems” NSF CNS-1253334

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or AFOSR.