Automated Assessment of UML Activity Diagrams

What is so special about UML Activity Diagrams? - While automated assessment of several UML diagram types like class diagrams or object diagrams can be done by pure static analysis, activity diagrams (as well as some other types of UML diagrams) offer an additional dimension: They have dynamic semantics.

Why are dynamic semantics interesting? - Just as in programming exercises, where test cases are executed on a solution to judge its correctness, diagrams with dynamic semantics can be “executed” as well to generate feedback to students. Since this is hardly impossible to do by hand, it’s a good use case for automated assessment.

Techniques

Our implementation uses a three-step approach:
1. According to the dynamic semantics of UML Activity Diagrams, which are defined as token flows, both the student’s solution and the sample solution are executed.
2. The resulting traces are serialized to rule out the effects of parallel activities.
3. A trace comparison based on Ukkonen’s algorithm [1] is used to find out whether the student’s solution produces the similar results as the sample solution.

Preliminary Results

The approach has currently been used in experimental evaluations based on exercise submissions from a bachelor degree course. Preliminary results include the following findings:

⇒ The approach is able to identify correct solutions that use different approaches than the sample solution, e.g. using interrupt sections instead of decision nodes. This is particularly due to using an alignment algorithm for approximate patterns instead of doing strict matching.
⇒ The approach is efficient enough to allow quick feedback to the students.
⇒ The approach is weak in face of mismatches in activity names. Using approximate matching here again is likely to solve this problem.

Example

Consider one sample solution (top right figure) and two student’s solutions to a modeling exercise. Students were asked to model activities concerned with accepting an invitation and joining a party. Each solution produces three different possible traces, so the comparison between a student’s solution and the sample solution requires the computation of nine trace alignments. For each trace from the sample solution that does not find a perfect match, feedback can be generated.

Sample comparisons
First trace of the sample solution is
Start -> Receive invitation -> n/a - {no time} -> n/a -> Refuse invitation -> End
and finds a perfect match in the first solution. The second solution produces trace
Start -> Receive invitation -> n/a - {no time} -> Reject invitation -> End
which is no perfect match due to the missing merge node, but still close enough to the sample solution to trigger no feedback.

The second trace of the sample solution is
Start -> Receive invitation -> n/a - {else} -> Accept invitation -> n/a - {get ill} -> n/a -> Refuse invitation -> End
This has no perfect match in neither of the solutions. However, the first solution produces
Start -> Receive invitation -> n/a - {else} -> Accept invitation - get ill -> n/a -> Refuse invitation -> End
which has a high proximity to the sample trace although the diagram uses a completely different syntactical construct.