The session consisted of two presentations, one by Prof. Bjorn Hartmann and the other by Prof. Rajeev Alur. Below are notes from the presentations.

Prof. Hartmann: Evaluating high-level program specifications

- goal of evaluation is to see if the tools that we built have practical benefits for their users
- simple way to evaluate:
  - build the system
  - recruit users and assign tasks
  - compare performance of the new tool with that of the existing ones
- challenges with this approach:
  - might be too late to make changes to the tool by the time the evaluation is done
  - hence, start evaluations early!
  - there can be huge individual differences in programmer productivity with using the new tool (up to 10x)
  - this dwarfs the effects of new tool
  - the best tools are the ones that enable users to do something new, so there might be no prior art to compare with
- there are various types of evaluation techniques:
  - formative: prototype / mock up with specific cases output, walk a potential user with some sample scenarios, simulate the output of the new tool (using synthesis or grad student), and solicit feedback
  - automated: usability measures computed by software (e.g., precision / recall, as in data mining)
  - inspection: ask fields experts to come in and solicit feedback
  - formal: use cognitive psychology to simulate a person and evaluate the new tool
    - unfortunately, we have only piecewise knowledge of how humans work
  - empirical user studies: can be further divided into two types
    - qualitative: record how users use the tool
    - quantitative: measure some aspect of the interface. e.g., time to complete the given tasks, average number of errors, user’s ratings, etc
- how studies are usually run in reality
  - demonstrate that it works (existence proof)
  - measure some qualitative observations
  - bring external users in to self report
- case study: proton: multitouch gestures as regular expressions
  - in event-driven programming, finger touches are modeled like mouse clicks as event handlers
  - unfortunately, this blows up the state machine
  - proton proposes to use regex to describe a series of touch events, and attach callbacks at appropriate places
the high level goal is to derive from declarative spec (a visual language similar to guitar tablature) to generate recognition code, and we can also use static analysis to detect conflicts

how proton is evaluated:

- build prototype for existence proof
- one potential user study is to ask users build applications with / without proton and measure their productivity
  - did not do that as it will take too time to teach users event-driven programming
- instead, measured the learning curve in terms of time needed to learn how to use proton to recognize some gestures and compared that to the time taken to understand the iOS code that does the same

suggestions for ExCAPE

- ideal would be for each application area to identify 1 grad student who is interested in UI aspects to interface with Prof. Hartmann
- even during the early stages start to think about what kind of experiments to run to convince others that the tool / UI is the right one to build

questions from the audience

- how can we counter the “no experts” problem for a given application domain?
  - estimate the learning curve by throwing away the results from performing initial tasks and measuring the later ones instead
  - would be ideal to teach a class around your tool and study them via assignments and exams
- how can we discover what the users really need?
  - observe actual users during user studies rather than straightly relying on users’ explicit feedback (sometimes they might not even know what they want!)
  - do studies early
  - shadow actual users for a couple of days while using the tool and ask them why they do certain things
- are there ways to simulate the users rather than running actual studies?
  - not really, the cognitive dimensions of humans are not really operable

Prof. Alur: Proposal for synthesis competition

- inspired by SAT and SMT competitions that have led to impressive advances in tools, we should start a competition for synthesis
  - in the SAT / SMT community measurable progress (i.e., speed in solving benchmarks) has been made every year since starting the competitions
- but it might be hard to come up with a common computational problem for synthesis
  - it should capture the “modern” view of synthesis and API
  - should be based on logic and not in a particular language or application
  - need to have a good set of benchmarks to measure performance

- proposed computational problem
  - background: first order theory with typed variables
  - inputs:
    - structure: set $X$ of typed variables and set $P$ of single step (i.e., straight
line code) transition functions $T(X, X')$ expressed by control flow graph in SSA form

- local specification: $\psi(X, X')$ in temporal logic or assertions
- global specification: initialization $\text{Init}$ and set $\Phi$ of executions, the point is to delineate what executions are feasible
  - output:
    - find a synthesized program $T^*(X, X)$ such that:
      - $T^*(X,X')$ is in $P$
      - $T^*(X,X')$ implies $\psi(X,X')$
    - executions with $\text{Init}$ and using $T^*$ is contained in $\Phi$

- $\Phi$ can be specified using the following:
  - invariants of reachable states
  - asserts in C program
  - formulas in LTL
  - well-known properties such as deadlock free, race free, etc.

- structure is a partial program and can be specified using the following:
  - program with holes with a grammar of expressions to fill up the holes
  - loop scaffold, variables, and a grammar
  - program states and message types (for instance using EFSMs as in protocol designs, or encoding state with boolean variables as in finite state controllers)

- local specification is a specification of what program does in a single step.
  - for instance, if program currently in state $S$ and REQ received, then increment shares from 1 to 2 and send ACK message

- questions from the audience
  - what about the optimality of the generated solution?
    - can consider the best worst-case behavior and can also include some sort of bounds in the global specification
  - suggestion: limit $\Phi$ specifications to invariants and assertions only
    - we will identify which formats are the most feasible later on
  - how can we judge which solution is the best?
    - there were different suggestions from the audience, including:
      - speed
      - generality
      - can have both speed and generality awards
      - bring application domain experts as judges, or perhaps solicit computational problems from them
  - An issue with the SAT / SMT competition was raised where researchers focus too much on optimizing the benchmarks rather than improving science.
    - the suggestion was to first plan the competition to be around speed and then evolve that later on (say to generality / usability), based on the current bottleneck in the synthesis field.