DisCo

A specification method for reactive systems

In a nutshell…

- DisCo (distributed co-operation) is a formal specification method for reactive systems
- Incorporates
  - specification language
  - methodology for creating specifications
  - tools for working with specifications
    - Animate execution, visualize execution histories, theorem prover/verifier
- Method has a formal basis (derived from TLA, Temporal Logic of Actions)

What it does

- What it does
  - Allows you to reason about objects, actions, and relations
  - Relatively abstract – formal object interfaces need not be defined
  - Level of abstraction can be refined step-wise towards an implementation
- What it doesn't do
  - Produce any code (implementation using the specification execution model may not even be practical)

Basic Methodology

- Start with basic behavior, and incrementally add detail until the specification is at the desired level
- Focus is on collective behavior—how objects cooperate
- Employs closed world principle—specification describes both system and its environment (though environment may be highly non-deterministic)

Superposition

- Specification is built of layers, which introduce new details about the system
- Each layer is superimposed on existing layers, introducing new elements to the system
- System is represented by the combination of all the layers
- Stepwise construction forces you to respect constraints in previous layers, guaranteeing preservation of safety properties

Joint Actions

- Based on join action approach of Back and Kurki-Suonio
- Systems consists of objects and actions:
  - Whenever the guard of an action true for some set of objects, that action can be executed for them. The action body defines a set of state changes for those objects
  - Action execution is atomic. Multiple action are executed in sequential order.
- Specification meaning is the set of execution sequences it allows.

Parallelism?

- Actions are atomic—they can't be stopped or interfered with
- Interleaving model is used—only one action is executed at a time
  - This is easier to reason about
  - Model of observation, not execution—if two actions do not refer to the same variables they can be executed in any order (or in parallel)

Specification Language

```
layer example is

  class node is
    value : integer;
  end;

  relation neighbors(left, right : node) : 1:1;

  action swap
  by l, r : node              -- participants
  when neighbors(l,r)         -- guard
     and l.value < r.value
  do
    l.value := r.value        -- body
    || r.value := l.value;
  end;
end;
```

Reactive Systems?

- Simplest model of a system would be as a function of inputs to outputs. If you care about the states inbetween, the system is said to be reactive. (e.g. constant user interaction)
- Although the sequence of states is potentially infinite, we can define to categories of system properties:
  - Safety – "bad things never happen"
  - Liveness – "something good will eventually occur"

Specification Validation

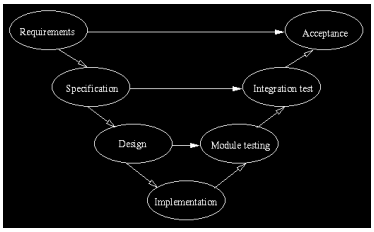
- DisCo has tool support for validation in which the specification is checked against its information requirements
- Includes animated simulation of specifications and graphical representation of execution scenarios
  - Makes it easy to find situations where safety properties (assertions) are violated
  - Since simulations are finite, liveness properties cannot be verified

Verification

- Verification establishes that the specification satisfies has some specific properties.
- Specifications can be verified formally using a theorem prover. A prototype tool supports mechanical verification using the PVS theorem prover.

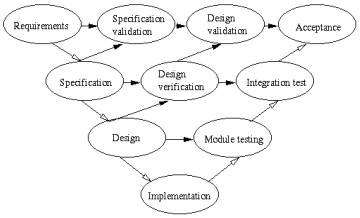
Wonderful

- So when would you use DisCo?
  - Using the waterfall model of LSD, DisCo would be used during the Requirements, Specification and Design steps (but not for Implementation, Testing or Maintainence)… almost.
- Why use it?
  - DisCo forces the specifier to be exact in early phases of the life cycle (beginning with a high level of abstraction). At later stages of development things are generally better understood and well-defined in the later stages.



If you did it perfectly the first time…

Yeah, right!



Better to verify things as you go...