SESF for Time-Constrained Programs

Shankar

April 14, 2014

Overview – 1

- Time-constrained program
  - statements are subject to time constraints
- Distributed program: local time constraints cause global effects
- Statement A can have two kinds of time constraints
  - only-within: A executed only within a given time interval
  - deadline: A executed within a given time
- Timing assumptions: time constraints enforced by platform
  - eliminate some evolutions of the program
- Timing properties: time constraints to be satsified by program

Overview – 2

- Let S be a time-constrained program
- S_τ: explicit-time version of S
  - "regular" program whose evolutions satisfy timing assumptions
  - "regular" reasoning/compositionality applies to S_τ
- S_τ is S with following added to environment
  - τnow: real-valued "current time" variable, readable by all
  - τage(δ): "ageing" function that increases τnow by δ
  - S_τ-transitions: S-transitions and ageing transitions
- Epoch variables: added to S to store times of step executions
- Timing assumptions: guards in epoch variables
  - only-within: guard on S-step
  - deadline: guard on τage(δ)
- Timing properties: assertions in epoch variables

Explict-time program of S

- Time-constrained program S
  - statements can simultaneously record value of τnow
  - statements can have "only-within" blocking conditions in τnow
  - S can have a deadline assumption, a predicate in τnow

- Explicit-time version of S

```
program Sτ(.)
  real τnow ← 0;
  sys ← startSystem(S(.));          // only-within: guards in S
  return mysid

  output τage(δ)
    oc {δ > 0 and <deadline[τnow | τnow + δ]>}
    τnow ← τnow + δ
    ic {true}
```

Implements and Compositionality

- Let A and B be time-constrained programs
- "A implements B" is now "A_τ implements B_τ"
- Compositionality theorem holds
- Program-based implements holds

- Typical timing assumptions for implementation programs
  - `delay(L,U)`: pass within `[L,U]` secs from arrival
  - `ddl{U}`: reach next timing assumption within `U`
  - `<blocking construct>(U)`:
    pass within `U` secs since arrival or since last unblocked