The UNITY Model

Wolfgang Schreiner
Research Institute for Symbolic Computation
Johannes Kepler University, Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

Example: Scheduling a Meeting

- Problem:
  - Find the earliest meeting time acceptable to every member of a group of three people.
- For each member F, G, H, a time function f, g, h.
  - $f(t) \geq t$
  - The smallest time $r \geq t$ where F can meet.
  - *com*(t) := $(t = f(t) = g(t) = h(t))$.
- Formal specification:
  - Given: functions f, g, h over *Nat* where $f(t) \geq t \wedge g(t) \geq t \wedge h(t) \geq t$ $\wedge\ f(f(t)) = f(t) \wedge g(g(t)) = g(t)$ $\wedge\ h(h(t)) = h(t)$ and some z with *com*(z).
  - Find: $r = \mathit{min}\{t \leq z \mid \mathit{com}(t)\}$ (within a finite number of execution steps).

Operational Viewpoint

Find scheme for processes.

- Round-table algorithm:
  - Pass letter with proposed time t. F sets time to $f(t)$ and passes letter. After one round without change, algorithm terminates.
- Central coordinator:
  - Coordinator broadcasts maximum t, F sends $f(t)$ back. When coordinator receives identical values, algorithm terminates.
- Divide and conquer:
  - Divide persons into two groups, find meeting points for each group. When times differ, maximum time used for repeating steps.
- Bidding scheme:
  - Auctioneer calls out time t; F can raise bid to $f(t)$. Algorithm terminates, when bid value cannot be raised any more.

UNITY Viewpoint

Specification as starting point for design.

- Problem specification.
- Program design.
- Design refinement.
- Mapping to target architecture.

Alternative refinements or alternative programs possible.

A Simple Program

Program *P1*
  assign r := *min*$\{u \mid (0 \leq u \leq z) \wedge$ *com*$(u)\}$
end $\{$*P1*$\}$

- Directly suggested by specification.
- Correctness obvious.
- Efficiency low:
  - Iterative check of $0 \leq u \leq z$.
  - Sequential complexity $O(z)$.
  - Parallel complexity $O(\log z)$ with $O(z)$ processors.

Observation: com(u) need not be checked for $t \leq u < f(t)$.

Another Program

Program *P2*
  initially $r = 0$
  assign $r := f(r)$ [] $r := g(r)$ [] $r := h(r)$
end $\{$*P2*$\}$

- Non-deterministic selection of assignments.
- Fairness: no assignment is indefinitely blocked.

Is program correct?

Proof of Correctness

1. Invariant holds at all execution points:
   - **invariant** $0 \leq r$
     $\wedge$ $\langle$for all u where $0 \leq u < r$ :: $\neg$*com*$(u)\rangle$
   - Invariant is true initially.
   - Invariant is true after $r := f(r)$.
2. *FP* $:= (r = f(r) \wedge r = g(r) \wedge r = h(r))$.
   - Fixed point reached when $:=$ becomes $=$.
   - *FP* implies *com*(r).
   - (1) and (2) prove that when *P2* reaches fixed point, r is the minimum meeting time.
3. *P2* always reaches fixed point:
   - $\neg$*FP* $\wedge\, r = k$ implies $r > k$ at some later point.
   - (1) shows that r cannot increase beyond z.
   - Eventually *FP* must hold.

UNITY provides a full proof theory.

Mapping the Program to Machines

- von Neumann machine:
  - Repeated sequence of assignments until fixed point.
  - $r := f(r);\ r := g(r);\ r := h(r).$
- Assume computation of f is more efficient:
  - Apply f twice as often.
  - $r := f(r);\ r := g(r);\ r := f(r);\ r := h(r).$
- Multiprocessor mapping:
  - Partition statements among processors.
  - Each processor executes one of f, g, h.

Each UNITY programs describes a family of implementations!

Yet Another Solution

Program *P3*
  initially $r = 0$
  assign $r :=$ *max*$(f(r), g(r), h(r))$
end {varP3}

- Spirit of central-coordinator scheme.
- Suitable starting point for parallel solution.
- $O(\log N)$ steps with $O(N)$ processors.
  - N number of persons.

The UNITY Notation

Program *program-name*
  declare *declare-section*
  always *always-section*
  initially *initially-section*
  assign *assign-section*
end

- Structure:
  - *declare-section* declares program variables.
  - *always-section* defines functional relationships.
  - *initially-section* defines initial variable values.
  - *assign-section* contains assignment statements.
- Execution:
  - Initial state as specified by *initially-sectioon*.
  - In each step, one assignment is executed.
  - Non-deterministic (but fair) selection of statements.
  - Program reaches *fixed point* when state is not changed any more.

Enumerated Assignments

- List of assignment statements:
  - x, y, z := 0, 1, 2
  - x, y := 0, 1 || z := 2
  - x := 0 || y := 1 || z := 2
- Case analysis:
  - x := -1 if $y < 0$
$\sim$ 0 if $y = 0$
  - Nondeterministic choice.
- Examples:
  - x, y := y, x.
  - $x := y$ if $y \geq 0$
$\sim$ -y if $y \leq 0$.
  - s, i := $s + A[i]$, $i + 1$ if $i < N$
  - $C[k]$:= *min*($A[i]$, $B[j]$)
|| $k := k + 1$
|| i := i+1 if $A[i] \leq B[j]$
|| j := j+1 if $A[i] \geq B[j]$

Quantified Assignments

- Parallel variable quantification:
  - $\langle || i \colon 0 \leq i \leq N :: A[i] := B[i] \rangle$
  - $A[0] := B[0] \; || \; A[1] := B[1] \; || \; \ldots \; || \; A[N] := B[N]$
- Examples:
  - $\langle || i \colon 0 \leq i \leq N :: A[i] := \mathit{max}(A[i], B[i]) \rangle$
    $\langle || i \colon 0 \leq i \leq N :: A[i] := B[i] \text{ if } A[i] < B[i] \rangle$
    $\langle || i \colon 0 \leq i \leq N \wedge A[i] < B[i] :: A[i] := B[i] \rangle$
  - $\langle || i{,}j \colon 0 \leq i \leq N \wedge 0 \leq j \leq N ::$
    $U[i,j] := 0 \text{ if } i \neq j$
    $\sim 1 \text{ if } i = j \rangle$

Assign-section

- List of program statements.
- Non-deterministic selection.
- Quantification possible.
- Examples:
  - $\langle[]\ i{,}j\text{: } 0 \leq i \leq N \wedge 0 \leq j \leq N ::$
    $U[i, j] := 0 \text{ if } i{\neq}j$
    $\sim 1 \text{ if } i = j\rangle$
  - $\langle||\ i{,}j\text{: } 0 \leq i \leq N \wedge 0 \leq j \leq N \wedge i{\neq}j ::$
    $U[i, j] := 0\rangle$
    $[]\ \langle||\ i\text{: } 0 \leq i \leq N :: U[i, j] := 1\rangle$
  - $\langle[]\ i\text{: } 0 \leq i \leq N :: U[i, j] := 1$
    $[]\ \langle||\ j\text{: } 0 \leq j \leq N \wedge i{\neq}j :: U[i, j] := 0\rangle\ \rangle$
- Quantification condition must not depend on program variables.
  - Numer of statements is fixed.

Initially-section

- Syntax similar to assign-section.
  - $:=$ is replaced by $=$.
- Initial values of some variables.
  - Function of initial values of other variables.
- Set of equations must be *proper*:
  1. Variable appears at most once on a left side.
  2. No circular variable dependencies.
- Examples:
  - **initially** $N = 3$
    $[]\ \langle || \ k\colon\ 0 \leq k < N :: \ A[N-k] = k \rangle$
  - **initially** $B[0] = 0 \ || \ N = 2 \ []$
    $\langle []\ i\colon\ 0 < i \leq N :: \ A[i] = B[i-1] \ []\ B[i] = A[i] \rangle$
- Section defines *initial condition*:
  - Replace $||$ and $[]$ by $\wedge$.
  - Replace $(x = e_0$ if $b_0 \sim \ldots \sim e_n$ if $b_n)$
    by $(b_0 \Rightarrow (x = e_0) \wedge \ldots \wedge (b_n \Rightarrow (x = e_n))$.

Always-section

- Define program variables as functions of other variables.
- Same syntax as in initially-section.
- Variables on left side are *transparent*.
  - Must not appear on left side of any initialization or assignment.
- Same restrictions as in initially-section.
- Example:
  - **initially** $a = 0$ [] $b = 1$
    always $s = a+b$

Quantified Expression

- Quantification over arbitrary functions:
  - Existential quantification:
    $\langle \vee\ i\colon 0 \leq i \leq N :: b[i] \rangle$
  - Minimum element:
    $\langle \textit{min}\ i\colon 0 \leq i \leq N :: A[i] \rangle$
  - Sum of elements:
    $\langle +\ i\colon 0 \leq i \leq N :: A[i] \rangle$
  - Number of elements smaller than $A[j]$:
    $\langle +\ i\colon 0 \leq i \leq N \wedge A[i] < A[j] :: 1 \rangle$

Examples

- Scheduling a common meeting time

Program *P2*
  declare r: *integer*
  initially $r = 0$
  assign $\langle [] \; i\text{: } 0 \leq i < N :: r := f_i(r)\rangle$
end $\{$*P2*$\}$

- Semaphore for N processes:

Program *semaphore*
  declare g: *integer*
  initially $g = 1$
  assign
    $\langle [] \; i\text{: } 0 \leq i < N ::$
      $g, p[i] := g - 1,$ *false* if $g > 0 \land p[i]$
    $[] \; g, v[i] := g + 1,$ *false* if $v[i]\rangle$
end $\{$*semaphore*$\}$

Sorting

Program *sort1*
  assign
    $\langle [] \; i\colon 0 \leq i < N ::$
      $A[i], A[i+1] := A[i+1], A[i]$ if $A[i] > A[i+1] \rangle$
end $\{$*sort1*$\}$

Program *sort2*
  assign
    $\langle \| \; i\colon 0 \leq i < N \wedge$ *even*(i) $::$
      $A[i], A[i+1] := A[i+1], A[i]$ if $A[i] > A[i+1] \rangle$
  $[] \; \langle \| \; i\colon 0 \leq i < N \wedge$ *odd*(i) $::$
      $A[i], A[i+1] := A[i+1], A[i]$ if $A[i] > A[i+1] \rangle$
end $\{$*sort2*$\}$

Program *sort3*
  assign
  $[] \; \langle [] \; j\colon 0 \leq j \leq 1 ::$
    $\langle \| \; i\colon 0 \leq i < N \wedge j = i$ *mod* $2 ::$
      $A[i], A[i+1] := A[i+1], A[i]$ if $A[i] > A[i+1] \rangle \rangle$
end $\{$*sort3*$\}$