Semantics of Parallel Programming Languages

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

Nondeterminism and Concurrency

- Deterministic programs
  - Same input always produces same output.
  - Evaluation strategy need not be unique.
  - Example: side effect-free arithmetic addition.
- Nondeterministic programs
  - Several evaluation strategies allowed.
  - Different strategies lead to different outputs.
  - Bad design: addition *with* side effects.
  - Acceptable: simultaneous requests to a server.
- Parallel evaluation
  - Noninterfering parallelism: simultaneous evaluation of subparts does not interact.
  - Interfering parallelism: interaction exists, relative speeds affect final result.
  - Concurrent languages: interfering parallelism.

Can sequential semantics be generalized?

Denotational Semantics

- Deterministic program semantics
  - $F[[P]] = d$
  - Valuation F maps program P into denotation d.
- Nondeterministic and concurrent program semantics
  - $F[[P]] = \{d_0, d_1, \ldots, d_n\}$
  - Denotation contains all possible results d_i.
  - d_i is element of the *powerdomain* $P(D)$.

Powerdomain construction required.

Powerdomains

Domain of sets of elements.

- Powerdomain builder.
  - $P(A) = \{X \subseteq A\}$.
- Smallest element:
  - $\{\}$: $P(A)$.
- Singleton sets:
  - $\{a\} \in P(A)$, for $a \in A$.
- Binary union:
  - $M \cup N \in P(A)$, for $M, N \in P(A)$.
- Disassembly operation:
  - $f^+ : P(A) \to P(B)$, for $f : A \to P(B)$.
  - $f^+(M) = \cup\{f(m) | m \in M\}$.

Basics for modeling non-determinism.

Multichoice Commands

- Imperative languages:
  - Sequential execution of commands.
  - Order of updates on store is important.
  - Sequencing not neccessary only for conditional command: **if** B **then** C_1 **else** C_2.
- Multichoice **cases** command

  cases
  B_1: C_1;
  B_2: C_2;
  ...
  B_n: C_n;
  end

  - Generalization of conditional command.
  - C_i executed when B_i evaluates to *true*.

What if in a multichoice more than one condition B_i holds?

The Guarded Command Language

- Traditionally: sequential test of choices.
  - Select *first* C_i where B_i holds.
- Dijkstra: non-deterministic choice.
  - Choose *any* C_i for which B_i holds.
- *Guarded command language*:
  - Assignment language.
  - Non-deterministic conditional command.
  - Non-deterministic multi-test loop.

Tool for studying non-determinism!

Denotational Semantics

I. Syntactic Domains:
$\mathsf{C} \in$ Command
$\mathsf{G} \in$ Guarded-command
$\mathsf{E} \in$ Expression
$\mathsf{B} \in$ Boolean-expression
$\mathsf{I} \in$ Identifier

II. Abstract Syntax:
$\mathsf{C} ::= \mathsf{C}_1;\ \mathsf{C}_2 \mid \mathsf{I} := \mathsf{E} \mid$ **if** G **fi** $\mid$ **do** G **od**
$\mathsf{G} ::= \mathsf{G}_1\ [\!]\ \mathsf{G}_2 \mid \mathsf{B} \rightarrow \mathsf{C}$
$\mathsf{E} ::= \ldots$
$\mathsf{B} ::= \ldots$

Example
do
   A mod 2 $= 0 \rightarrow$ A $:=$ A$/2$
 [] A mod 3 $= 0 \rightarrow$ A $:=$ A$/3$
 [] A mod 5 $= 0 \rightarrow$ A $:=$ A$/5$
od

Denotational Semantics

III. Semantic algebras:

Tr . . . truth values
Id . . . identifiers
Nat . . . natural numbers

$\mathit{Store} = \mathit{Id} \rightarrow \mathit{Nat}$
update: $\mathit{Id} \rightarrow \mathit{Nat} \rightarrow \mathit{Store} \rightarrow \mathit{Store}$
$\quad \mathit{update}\ i\ v\ s = [i \rightarrow v]s$

$\mathit{Answer} = P(\mathit{Tr} \times \mathit{Store})$
$\mathit{no\text{-}answer}$: Answer
$\quad \mathit{no\text{-}answer} = \{\}$
join: $\mathit{Answer} \rightarrow \mathit{Answer} \rightarrow \mathit{Answer}$
$\quad a_1\ \mathit{join}\ a_2 = a_1 \cup a_2$
then: $(\mathit{Store} \rightarrow \mathit{Answer}) \rightarrow (\mathit{Store} \rightarrow \mathit{Answer})$
$\quad \rightarrow (\mathit{Store} \rightarrow \mathit{Answer})$
$\quad (f_1\ \mathit{then}\ f_2)(s) = (g^+ \circ f_1)(s)$
$\quad$ where $g(t, s) = t \rightarrow f_2(s)\ []\ (t, s)$

Denotational Semantics

IV. Valuation functions:

C: Command $\rightarrow$ *Store* $\rightarrow$ *Answer*
C[[C$_1$; C$_2$]]s = (**C**[[C$_1$]] *then* **C**[[C$_2$]])s
C[[I := E]]s = (*true*, *update* [[I]] (**E**[[E]]s) s)
C[[**if** G **fi**]]s = **T**[[G]]s $\rightarrow$ **G**[[G]]s [] (*false*, s)
C[[**do** G **od**]]s = **T**[[G]]s $\rightarrow$
    (**G**[[G]] *then* **C**[[**do** G **do**]])(s) [] (*true*, s)

T: Guarded-command $\rightarrow$ *Store* $\rightarrow$ *Tr*
T[[G$_1$ [] G$_2$]]s = (**T**[[G$_1$]]s) *or* (**T**[[G$_2$]]s)
T[[B $\rightarrow$ C]]s = **B**[[B]]s

G: Guarded-command $\rightarrow$ *Store* $\rightarrow$ *Answer*
G[[G$_1$ [] G$_2$]]s = (**G**[[G$_1$]]s) *join* (**G**[[G$_2$]]s)
G[[B $\rightarrow$ C]]s = **B**[[B]]s $\rightarrow$ **C**[[C]]s [] *no-answer*

E:Expression $\rightarrow$ *Store* $\rightarrow$ *Nat*
B:Boolean-expr $\rightarrow$ *Store* $\rightarrow$ *Tr*

Valuation Functions

- *Answer* is set of (*tag*, *store*) pairs.
  - *tag*=*true*: store is valid.
  - *tag*=*false*: error has occured.
- $(f_1$ *then* $f_2)(s)$ sequences commands.
  - f_1 operates on s producing set of stores.
  - Each answer store is passed through f_2 producing an answer set.
  - The answer sets are unioned.
- **C** represents non-determinism.
  - **T** determines if at least one of the *guards* holds.
  - Conditional: failure of all guards causes abortion.
  - Loop: failure of all guards causes exit.
  - **G** joins the effects of all those commands whose guards hold.

Example

$\mathsf{C}_0 = \textbf{if } \mathsf{G}_1 \; [] \; \mathsf{G}_2 \textbf{ fi}$
$\mathsf{G}_1 = \mathsf{X} \geq 0 \rightarrow \mathsf{Y} := 1$
$\mathsf{G}_2 = \mathsf{X} = 0 \rightarrow \mathsf{Y} := 0$

$\mathbf{C}[[\mathsf{C}_0]]s = \ldots$

Case: *access* $[[\mathsf{X}]]\ s =$ *zero*
$\ldots = \{\ (\mathit{true}, s_1)\ \}$ where
$\quad s_1 =$ *update* $[[\mathsf{Y}]]$ *one* s

Case: *access* $[[\mathsf{X}]]\ s >$ *zero*
$\ldots = \{\ (\mathit{true}, s_1), (\mathit{true}, s_2)\ \}$ where
$\quad s_1 =$ *update* $[[\mathsf{Y}]]$ *one* s
$\quad s_2 =$ *update* $[[\mathsf{Y}]]$ *zero* s

Case: *access* $[[\mathsf{X}]]\ s <$ *zero*
$\ldots = \{\ (\mathit{false}, s)\ \}$

Concurrency

Modelling of interfering parallelism

- Assignments operate concurrently on store.
- Result is the set of possible result stores.
- Powerdomain constructions required.
- *Resumption semantics* (Plotkin):
  - Central: concurrent evaluation of C_1 and C_2.
  - Problem: how to model concurrent interaction?
  - Idea: *interleave* atomic steps of C_1 and C_2.

Concurrency modeled by interleaving!

Abstract Syntax

P $\in$ Program
C $\in$ Command
E $\in$ Expression
B $\in$ Boolean-expr
I $\in$ Identifier

P ::= C.
C ::= I := E | C_1; C_2 | C1 || C2
| **if** B **then** C_1 **else** C_2 | **while** B **do** C

- Parallel evaluation operator ||.
- Assignment is *non-interruptable*.
  - Exclusive access to store.
- Interleaving semantics
  - Evaluation of $C_1||C_2$ interleaves assignments of C_1 with those of C_2.
  - Semantics of $C_1||C_2$ generates set of results of all possible interleavings.

Examples

- [[X := X+1]]
  - Atomic update of cell [[X]].
- [[X := X+2; X := X-1]]
  - Command may be interrupted after evaluation of first assignment.
  - *Not* semantically equivalent to [[X := X+1]]
- [[(X := X+2; X := X-1) || X := 3]]
  - X := X+2; X := X-1; X := 3.
  - X := X+2; X := 3; X := X-1.
  - X := 3; X := X+2; X := X-1.

Many interleavings possible!

Examples

- [[(**if** X=0 **then** Y:=1 **else** Y:=2) || X := X+1]]
  - Initial value of X is *zero*:
    test X = 0; Y := 1; X := X+1
    test X = 0; X := X+1; Y := 1
    X := X+1; test X=0; Y:=2
  - Initial value of X is positive:
    test X = 0; Y := 2; X := X+1
    test X = 0; X := X+1; Y := 2
    X := X+1; test X = 0; Y := 2

Sequence of actions of one command influenced by interleaving of other command!

Examples

- [[(X:=1; **while** X¿0 **do** Y:=Y+1) || X :=0]]
  - X:=0; X:=1; test X>0; Y:=Y+1; test X>0; Y:=Y+1; ...
  - X:=1; X:=0; test X>0
  - X:=1; test X>0; X:=0; Y:=Y+1; test X>0.
  - X:=1; test X>0; Y:=Y+1; X:=0; test X>0.
  - X:=1; test X>0; Y:=Y+1; test X>0; X:=0; Y:=Y+1; test X>0.
  - ...
- [[X:=0]] eventually occurs.
- Evaluation sequence is then called *fair*.
- In unfair sequence, [[X:=0]] never occurs.
- Resumption semantics models both fair and unfair sequences.

Semantic Algebras

State = (*Tr* $\times$ *Store* $\times$ *Res*)
Res = Store $\rightarrow P($*State*$)$

step: (*Store* $\rightarrow$ *Store*) $\rightarrow$ *Res*
  step$(f, s) = \{$ (*true*, $f(s)$, _) $\}$
pause: Res $\rightarrow$ *Res*
  pause$(r, s) = \{$ (*false*, s, r) $\}$
seq: *Res* $\times$ *Res* $\rightarrow$ *Res*
  $(r_1$ *seq* $r_2)(s) = (f^+ \circ r_1)(s)$
    where $f(t, s, r) = t \rightarrow$
      $\{$ (*false*, s, r_2) $\}$ [] $\{$ (*false*, s, r*seq*r_2) $\}$
par: *Res* $\times$ *Res* $\rightarrow$ *Res*
  $(r_1$ *par* $r_2)(s) = (r_1$ *then* $r_2)(s) \cup (r_2$ *then* $r_1)(s)$
    where *then*: *Res* $\times$ *Res* $\rightarrow$ *Res*
    $(r_1$ *then* $r_2)(s) = (f^+ \circ r_1)(s)$
      where $f(t, s, r) = t \rightarrow \{$ (*false*, s, r_2) $\}$ []
      $\{$ (*false*, s, r *then* r_2), (*false*, s, r_2 *then* r) $\}$
flatten: *Res* $\rightarrow$ *Store* $\rightarrow P($*Store*$)$
  flatten $r\ s = (f^+ \circ r)(s)$
    where $f(t, s, r) = t \rightarrow \{s\}$ [] (*flatten* r)(s)

Valuation Functions

$\mathbf{P}$: Program $\rightarrow$ *Store* $\rightarrow P(\textit{Store})$
$\mathbf{P}[[\mathrm{C}]] = \textit{flatten}(\mathbf{C}[[\mathrm{C}]])$

$\mathbf{C}$: Command $\rightarrow$ *Res*
$\mathbf{C}[[\mathrm{I} := \mathrm{E}]] = \textit{step}(\lambda s.\ \textit{update}\ [[\mathrm{I}]]\ (\mathbf{E}[[\mathrm{E}]]s)\ s)$
$\mathbf{C}[[\mathrm{C}_1;\ \mathrm{C}_2]] = \mathbf{C}[[\mathrm{C}_1]]\ \textit{seq}\ \mathbf{C}[[\mathrm{C}_2]]$
$\mathbf{C}[[\mathrm{C}_1\ ||\ \mathrm{C}_2]] = \mathbf{C}[[\mathrm{C}_1]]\ \textit{par}\ \mathbf{C}[[\mathrm{C}_2]]$
$\mathbf{C}[[\mathbf{if}\ \mathrm{B}\ \mathbf{then}\ \mathrm{C}_1\ \mathbf{else}\ \mathrm{C}_2]](\mathrm{s}) = \mathbf{B}[[\mathrm{B}]]s \rightarrow$
$\quad (\textit{pause}\ (\mathbf{C}[[\mathrm{C}_1]])\ s)\ []$
$\quad (\textit{pause}\ (\mathbf{C}[[\mathrm{C}_2]])\ s)$
$\mathbf{C}[[\mathbf{while}\ \mathrm{B}\ \mathbf{do}\ \mathrm{C}]](\mathrm{s}) = \mathbf{B}[[\mathrm{B}]]s \rightarrow$
$\quad (\textit{pause}\ (\mathbf{C}[[\mathrm{C}]]\ \textit{seq}\ \mathbf{C}[[\mathbf{while}\ \mathrm{B}\ \mathbf{do}\ \mathrm{C}]])\ s)\ []$
$\quad (\textit{step}\ (\lambda s.\ s)\ s)$

Resumptions

- *Resumption* = denotation of a command in the context of interfering parallelism.
- Set of interruptable evaluation sequences.
- r resumption, s store.
- Single step r:
  - Result $r(s)$ contains a single store s'.
- Sequence r:
  - Result $r(s)$ is application of fist step.
  - Pair (store s', resumption r').
  - r' describes the remaining steps to be done.
- Resumption of complete program is flattened into the set of possible evaluation sequences.

Tool for modeling interruptions!

Summary

- Non-determinism is modeled by powerdomain constructions.
- Interfering parallelism is modeled by resumption semantics.
- Problem: constructions are far too complex!
- Function notation is ill-suited for multivalued objects.
- Denotational semantics is insufficient.

We look for more natural ways of describing parallelism!