A Revisionist History of Denotational Semantics

Stephen Brookes

Carnegie Mellon University

Domains XIII
July 2018

Denotational Semantics

Compositionality Principle

> *The meaning of a complex expression is determined by the meanings of its constituent expressions and the rules to combine them.*

Foundational references:

- Christopher Strachey, Dana Scott (1971)
  Toward a Mathematical Semantics for Computer Languages
- Strachey (1966): *Towards a Formal Semantics*

Historical precursors include:

- Boole (1815–1864), **Frege** (1845–1925)

The Denotational Template

To give a semantics:

- Find a **suitable** mathematical universe of *meanings*
- Define, **by structural induction**, a semantic function that maps phrases to their meanings, or *denotations*

Criteria for *suitability*: meanings should be

- **abstract** from machine details
- **expressive** of program behavior
- capable of **compositional** definition

Scott developed *domain theory* to underpin this approach, e.g.

- Scott (1970): *Outline of a Mathematical Theory of Computation*

Snapshots from History

Some landmarks

- Sequential imperative programs
  - *partial functions* from states to states
- Low-level machine code
  - *continuation semantics*
- Simply-typed functional programs
  - cartesian-closed categories, *continuous functions*
- Sequential functional programs
  - concrete domains, *sequential functions*, *game semantics*
- Algol-like programs
  - functor categories, *possible-world semantics*
- Concurrent programs
  - *resumptions*, *trace sets*, *event structures*, ...

A common thread

- domains, recursion as *fixed-point*

Strategy

Start with a programming language and a notion of *behavior*, develop a semantics to support *compositional reasoning*.

Criteria for success:

- **Adequacy** (minimum pre-requisite)
  semantic equivalence implies *behavioral indistinguishability*
- **Full abstraction** (stronger, harder to achieve)
  semantic equivalence $=$ *behavioral indistinguishability*

By definition, an **adequate denotational** semantics supports compositional reasoning about program behavior.

Lessons

Depending on behavior and programming language, it may be hard to find a **suitable** semantics capable of **compositional** definition.

- Algol-like language: *functor category semantics*
  - naturality conditions express key behavioral features
- PCF: long path *via* concrete domains to *game semantics*
  - sequential functions are hard to characterize
- Concurrency: need to include *non-sequential* traces
  - executions of $c_1 \| c_2$ not definable compositionally
  - semantics must account for interference between threads

Revisiting concurrency

- Early semantics assumed **sequential consistency** (SC):

  > *. . . the result of any execution is the same as if the operations of all the processors were executed in some sequential order. . .* *Lamport 1979*

- A program denotes a *set of traces*, with parallel composition as *interleaving*, sequential composition as *concatenation*.
  - Trace sets form a complete lattice
  - Semantic constructs denote *monotone* functions
  - Recursion, and *fair interleaving*, as *greatest* fixed point

Assessment

- At the time it was hard enough to deal with concurrency, without trying to handle procedures. . .
  - Later we developed *Parallel Algol* and a trace-based possible-worlds semantics
- Similarly for concurrency + pointers, mutable state
  - Later came *Concurrent Separation Logic* and *action traces*

Trace semantics is robust and adaptable, but inherently SC. . .

> *"Of the many forms of false culture, a premature converse with abstractions is perhaps the most likely to prove fatal . . . "*
>
> *Boole, 1859.*

Reassessment

> *. . . achieving sequential consistency may not be worth the price of slowing down the processors.* *Lamport 1979*

- Modern multi-processors do not guarantee SC behavior, and instead provide a *relaxed memory model*
  - reads may see stale values, because of buffering or caches
  - writes may get re-ordered, for optimized performance
- There is a wide range of *memory models*

  SC, TSO, PSO, RA, . . .

  offering a variety of behavioral guarantees
- *Trace semantics* is **not adequate**, except for SC

Beyond trace semantics

- For a denotational account of relaxed memory, we must abandon SC and embrace "*true concurrency*".
- In ongoing work with Ryan Kavanagh, we develop a *partial-order* semantic framework
  - Replace traces (linear orders) with pomsets (partial orders)
- A recent burst of progress in similar vein, e.g. Jeffrey and Riely (2016), using *event structures*, has similar aims.
- We regard our pomset approach as a natural evolutionary successor to trace semantics. . .

Denotational semantics should be more relaxed.

Actions

Our semantic framework builds on a set of *actions*, interpreted as having an *effect* on *state*.

Here we use the following grammar for actions:

$$\begin{array}{lcll} \lambda & ::= & \delta & \text{idle} \\ & | & i{=}v & \text{read} \\ & | & i{:=}v & \text{write} \end{array}$$

Can incorporate synchronization primitives, such as locks, actions tagged with memory levels (*na*, *at*, *rel*, *acq*,...), and *fence* instructions used to limit relaxation.

- Let $\mathcal{A}$ be the set of actions.

Action Pomsets

cf. Pratt 1986

An action pomset $(P, <, \Phi)$ is a partial order $(P, <)$ with a labelling function $\Phi : P \to \mathcal{A}$.

Pomsets are *equivalent* if they are *order-isomorphic*.

Let $\mathbf{Pom}(\mathcal{A})$ be the set of all action pomsets.

Parallel Composition

- Actions of P_1 and P_2 are independent

$$(P_1, <_1) \parallel (P_2, <_2) \;=\; (P_1 \uplus P_2, <_1 \uplus <_2)$$

Sequential Composition

- Actions of P_1 before actions of P_2

$$(P_1, <_1); (P_2, <_2) \;=\; (P_1 \uplus P_2,\; <_1 \uplus <_2 \cup P_1 \times P_2)$$

Relaxation

Rigidity

A memory model M has a *rigidity relation* $\lhd_M \subseteq \mathcal{A} \times \mathcal{A}$.
An ordered pair (p_1, p_2) can be M-relaxed *unless* $p_1 \lhd_M p_2$.

- SC allows no relaxations: $\lhd_{SC} = \mathcal{A} \times \mathcal{A}$
- TSO allows *write/read* relaxation on distinct locations:
  $p_1 \lhd_{TSO} p_2$ iff $\mathit{loc}(p_1) = \mathit{loc}(p_2)$ or $\neg(\mathit{IsWrite}\, p_1 \;\&\; \mathit{IsRead}\, p_2)$
- PSO allows *write/read* and *write/write* relaxation:
  $p_1 \lhd_{PSO} p_2$ iff $\mathit{loc}(p_1) = \mathit{loc}(p_2)$ or $\neg \mathit{IsWrite}\, p_1$

Relaxed Composition

- Actions of P_1 before actions of P_2, *modulo* M-relaxation

$$(P_1, <_1) ;_M (P_2, <_2) = (P_1 \uplus P_2,\ <_1 \uplus <_2 \cup \lhd_M \restriction (P_1 \times P_2))$$

Pomset Semantics

(parameterized by memory model)

$$\mathcal{P}_M : \mathbf{Com} \to \mathbb{P}(\mathbf{Pom}(\mathcal{A}))$$

is defined by structural induction:

$$\mathcal{P}_M(\mathbf{skip}) = \{\{\delta\}\}$$
$$\mathcal{P}_M(i{:=}e) = \{P; \{i{:=}v\} \mid (P, v) \in \mathcal{P}_M(e)\}$$
$$\mathcal{P}_M(c_1; c_2) = \{P_1 ;_M P_2 \mid P_1 \in \mathcal{P}_M(c_1), P_2 \in \mathcal{P}_M(c_2)\}$$
$$\mathcal{P}_M(c_1 \| c_2) = \{P_1 \| P_2 \mid P_1 \in \mathcal{P}_M(c_1), P_2 \in \mathcal{P}_M(c_2)\}$$
$$\mathcal{P}_M(\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2) = \mathcal{P}_M(b)_{tt}; \mathcal{P}_M(c_1)\ \cup\ \mathcal{P}_M(b)_{ff}; \mathcal{P}_M(c_2)$$
$$\mathcal{P}_M(\mathbf{while}\ b\ \mathbf{do}\ c) = (\mathcal{P}_M(b)_{tt}; \mathcal{P}_M(c))^*; \mathcal{P}_M(b)_{ff}$$

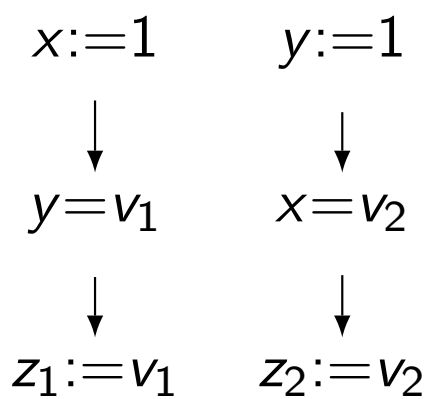
- We use ; to enforce data dependency or control dependency.
- We use $;_M$ to allow relaxation of program order.

Store Buffering

Pomset semantics

$$(x{:=}1; z_1{:=}y) \parallel (y{:=}1; z_2{:=}x)$$

- Each SC pomset has the form



  where $v_1, v_2 \in V_{int}$.
- For TSO: relax $(x{:=}1, y{=}v_1)$ and $(y{:=}1, x{=}v_2)$
- For PSO: also relax $(x{:=}1, z_1{:=}v_1)$ and $(y{:=}1, z_2{:=}v_2)$

Pomset Execution

(parameterized by memory model)

- We define the set $\mathcal{E}_M(P, <)$ of pomset *executions*.
- An M-execution is an ordering $<'$ that extends $<$ and fulfills the guarantees of memory model M.
  - SC: $<'$ is a *read-coherent* total order
  - TSO: $<'$ is a *read-coherent* total store order
  - PSO: $<'$ is a *read-coherent* per-location total store order
- Read-coherence:
  - (i) All initial reads of x see the same value v.
  - (ii) Non-initial reads of x get their value from a maximal write to x that precedes it.
- The input-output behavior of $(P, <')$ is $(pre(P), post(P))$, where $pre(P), post(P)$ are the states formed by the initial reads, maximal writes, respectively.

Results

Adequacy

For each memory model M we obtain a pomset semantics that is *compositional*, and adequate for describing execution.

> Executions cannot by themselves be defined compositionally. This was already the case for SC, requiring use of non-sequential traces.

Correctness

The input-output behavior of $\mathcal{P}_M(c)$ is *correct*:

- For SC: consistent with trace semantics

  read-coherent total order = sequentially executable trace
- For TSO, PSO: consistent with SPARC axioms

  read-coherent total store order = SPARC-TSO

  read-coherent per-location total store order = SPARC-PSO

Pomset behavior matches litmus test specifications...

Litmus Test 1

Store Buffering

SC ✗ TSO ✓ PSO ✓

$$(x{:=}1; z_1{:=}y) \parallel (y{:=}1; z_2{:=}x)$$

- Under TSO or PSO this program has an execution from

$$[x : 0, y : 0, z_1 : \_, z_2 : \_]$$

to

$$[x : 1, y : 1, z_1 : 0, z_2 : 0]$$

in which the reads see stale values.

- Under SC every execution ends with $z_1 = 1$ and/or $z_2 = 1$.

Litmus Test 2

Message Passing

SC ✓ TSO ✓ PSO ✗

$$(x{:=}37; y{:=}1) \parallel (\textbf{while } y = 0 \textbf{ do skip};\ z{:=}x)$$

- Under SC and TSO, every execution from $[x : 0, y : 0, z : \_]$ ends with $z = 37$.
- Not true under PSO, which can also end with $z = 0$.
- To ensure proper message-passing in PSO use a *fence* between $x{:=}37$ and $y{:=}1$, to prevent write/write relaxation.

Litmus Test 3

Independent Reads of Independent Writes

SC ✗ TSO ✗ PSO ✓

$$x{:=}1 \parallel y{:=}1 \parallel (z_1{:=}x;\ z_2{:=}y) \parallel (w_1{:=}y;\ w_2{:=}x)$$

- Under PSO there is an execution from

  $$[x : 0, y : 0, \ldots]$$

  to

  $$[x : 1, y : 1, z_1 : 1, z_2 : 0, w_1 : 1, w_2 : 0]$$

  in which the threads "see" the writes in different orders.
- Not true under SC and TSO, which guarantee a total order on all writes.

Litmus Test 4

Coherence

SC ✓ TSO ✓ PSO ✓

$$x{:=}1 \parallel x{:=}2 \parallel (z_1{:=}x;\ z_2{:=}x) \parallel (w_1{:=}x;\ w_2{:=}x)$$

- In all these memory models, there is no execution from

  $$[x : 0, y : 0, \ldots]$$

  ending with

  $$z_1 = 1, z_2 = 2, w_1 = 2, w_2 = 1.$$

- Writes to x appear in the same order, to all threads.
  All models guarantee to a per-location total store order.

Looking back, going forward

"We are not the first to advocate partial-order semantics."
Pratt 1986

- Pratt's *pomsets* were a "true concurrency" *process algebra*
  - "actions" as uninterpreted symbols, no state or execution
- Mostly concerned with abstract properties, e.g.
  A poset is representable as the set of its linearizations.
  - Not true for *executions*:

$$\mathcal{E}_M(P) \neq \bigcup \{\mathcal{E}_M(P') \mid P' \in \textsc{Lin}(P)\}$$

- Pioneering work by Petri, Greif, Mazurkiewicz, Winskel, ...
  - Petri nets, ..., event structures

Partial-order semantics provide an appropriate denotational setting for exploring relaxed memory...

Conclusions

- Denotational semantics is alive and kicking, as it turns 50...

> *"The estimation of a theory is not simply determined by its truth. It also depends upon the importance of its subject, and the extent of its applications, beyond which something must still be left to the arbitrariness of human opinion."*
>
> *George Boole, 1847.*

- It's useful to revisit, and question, established tradition...

> *"The one duty we owe to history is to rewrite it."*
>
> *Oscar Wilde*

HAPPY BIRTHDAY, DANA