Compositional Semantics for Shared-Variable Concurrency

Mikhail Svyatlovskiy, Shai Mermelstein, Ori Lahav

TEL AVIV UNIVERSITY

PLDI
COPENHAGEN 2024

Denotational semantics for shared-memory concurrency

- **Motivation**
  - Understand a *piece of code*: [C]
  - Justify local compiler transformations:

    $$\mathrm{Csrc} \rightsquigarrow \mathrm{Ctgt}: \forall P, s0, s\ .\ \langle P[\mathrm{Ctgt}], s0\rangle \downarrow s \Rightarrow \langle P[\mathrm{Csrc}], s0\rangle \downarrow s$$

- **Desired properties**
  - Compositionality: e.g., [C1 || C2] is determined from [C1] and [C2]
  - Adequacy: [Csrc] $\supseteq$ [Ctgt] implies Csrc $\rightsquigarrow$ Ctgt
  - Full abstraction: Csrc $\rightsquigarrow$ Ctgt implies [Csrc] $\supseteq$ [Ctgt]

- **Scope:** shared-memory concurrency
  - much more challenging than sequential programs
    - x:=1 ; x:=x+1 $\leftrightsquigarrow$ x:=2 in sequential programs
    - but only x:=1 ; x:=x+1 $\rightsquigarrow$ x:=2 in concurrent programs

Trace-Based Approach [Brookes '96]

- The semantics of a command C is the set of possible sequences of memory-to-memory transitions, interrupted by environment transformations

  e.g., [x:=4;y:=5] ∋ <[x,y,z↦0,0,0],[x,y,z↦4,0,0]>; <[x,y,z↦1,2,3],[x,y,z↦1,5,3]>

- Compositional, adequate, fully abstract

- Observation: Full abstraction assumes an **AWAIT** command that checks the values and changes the state atomically.
- **AWAIT** is impractical since it updates multiple variables, e.g.,

AWAIT(x=2 /\ y=2) then (x:=4; y:=5)

Observation: no full-abstraction without AWAIT

C = x:=1; y:=1; assume [(x=2 & y!=2) or (x!=2 & y=2)]; assume (x=y=2)
D = x:=1; y:=1; assume (x=y=2)

- $C \rightsquigarrow D$ does not hold (context P = **AWAIT** (x=y=1) then (x:=2; y:=2) || —)
- Indeed, in Brookes model, $[C] \supseteq [D]$ does not hold (trace <[x,y↦0,0],[x,y↦1,1]>; <[x,y↦2,2],[x,y↦2,2]>)
- $C \rightsquigarrow D$ is valid without **AWAIT**

- Our goal: compositional, adequate, fully abstract semantics *for a language without **AWAIT***

Our main contribution: a novel denotational semantics

- Compositional & adequate
- Full abstraction (limited):
  - Fully abstract assuming atomic **SNAPSHOT**

    e.g., SNAPSHOT(x=2 /\ y=2) atomically reads x=2 & y=2

  - Without **SNAPSHOT,** fully abstract for *loop-free* commands
- Fully mechanized in Coq

Our traces: an example

Initial store: partial map from local variables to values

$<s, \theta, \underline{\overline{W}(y,1) ; W(x,1) ; \overline{W}(x,2)}>$

Initial state: map from shared variables to values

Chronicle: sequence of actions

- $W(x, v)$: a component write
- $\overline{W}(x,v)$: an environment write

Denotations

- Denotations are sets of traces.
- Defined in two levels:

	Concrete semantics [C]	**Abstract semantics [[C]]**
Definition	inductive definition	closure of [C] by rewrite rules
Compositionality	✓	✓
Adequacy	✓	✓
Full abstraction	✗	✓ (limited)

Concrete Denotations

- [C] is a set of traces
  - [x:=1] = { <s, θ, e1 ; W(x,1) ; e2> |
    s ∈ State , θ ∈ Store, e1, e2 ∈ EnvChro }
  - [x:=x+1] = { <s, θ, e1 ; e2 ; W(x, e1(s)(x)+1) ; e3> |
    s ∈ State , θ ∈ Store, e1, e2, e3 ∈ EnvChro }

STORE

$$\frac{e_1, e_2 \in \mathrm{EnvChro} \qquad \alpha = \mathtt{W}(x, \theta(E))}{\langle s, \theta, e_1 \cdot \alpha \cdot e_2 \rangle \in \lfloor x := E \rfloor}$$

Sequential Composition

[C1 ; C2]: concatenate the traces of C1 with traces of C2

- Same initial stores
- Final state of C1 equals the initial state of C2
- Example:

  <[x,y↦0,0], θ, W(x, 1)> ; <[x,y↦1,0], θ, W(y, 2)> =

  <[x,y↦0,0], θ, W(x, 1) ; W(y, 2)>

SEQ

$$\frac{t_1 \in \lfloor C_1 \rfloor \qquad t_2 \in \lfloor C_2 \rfloor}{t_1 \,;\, t_2 \in \lfloor C_1 \,;\, C_2 \rfloor}$$

Parallel Composition

[C1 || C2]: synchronize the traces of C1 with traces of C2

- same initial states and stores
- action matching:
  - W(x, v) || $\overline{W}$(x, v) = W(x, v)
  - $\overline{W}$(x, v) || $\overline{W}$(x, v) = $\overline{W}$(x, v)
  - other cases are undefined
- Example: <[x,y↦0,0], θ, W(x, 1) ; $\overline{W}$(y, 2) ; $\overline{W}$(x,3)> ||
  <[x,y↦0,0], θ, $\overline{W}$(x, 1) ; W(y, 2) ; $\overline{W}$(x,3)> =
  <[x,y↦0,0], θ, W(x, 1) ; W(y, 2) ; $\overline{W}$(x,3)>

PAR

$$\frac{t_1 \in \lfloor C_1 \rfloor \qquad t_2 \in \lfloor C_2 \rfloor}{t_1 \parallel t_2 \in \lfloor C_1 \parallel C_2 \rfloor}$$

Concrete semantics

- We inductively define [C] as a set of traces
- This semantics is **compositional**
- This semantics is **adequate**
  - Key lemma: <C,s0> $\downarrow$ s iff $\langle s, \theta, c\rangle \in$ [C] s.t. c(s0)=s and c $\in$ CmpChro
- It supports **a wide variety of contextual refinements**:
  - Structural transformations (e.g., C1 || C2 $\leadsto$ C1; C2)
  - Reordering/introduction/elimination of local operations
  - Introduction/elimination of redundant reads
- Full abstraction is still a challenge:
  this semantics does not support transformations that modify the sequence of writes
  - e.g., x:=1; x:=x+1 $\leadsto$ x:=2
  - but <[x,y$\mapsto$0,0],θ,Wx2> $\in$ [x:=2] \ [x:=1; x:=x+1]

Abstract semantics: rewrite rules

- To make the semantics fully abstract we close the concrete sets of traces under *rewrite rules*.

[[C]] = closure([C])

closure([C]) = { c' | c ⇴* c' }

- For example,
    - x:=1; x:=x+1 ⇝ x:=2
    - <[x,y↦0,0],θ,W(x,2)> ∈ [x:=2] \ [x:=1; x:=x+1]
    - but we will have: <[x,y↦0,0],θ,W(x,2)> ∈ closure([x:=1; x:=x+1])

Coalesce rule

- compress a block of component writes into one write that has the same effect:

  $\langle s, \theta, c1 ; m1 ; W(x,v) ; m2 ; c2\rangle \twoheadrightarrow \langle s, \theta, c1 ; W(x,v) ; c2\rangle$

  provided that:

  1) m1,m2 are component chronicles

  2) (c1 ; m1 ; W(x,v) ; m2)(s) = (c1 ; W(x,v))(s)

- Example: x:=1; x:=x+1 $\rightsquigarrow$ x:=2
- Another example: let a:=y in (y:=1; x:=1; y:=a) $\rightsquigarrow$ x:=1

Environment-coalesce rule

- Similar compress, when we have one environment write in the middle

  $\langle s, \theta, c1 ; m1 ; \overline{W}(x,v) ; m2 ; c2 \rangle \twoheadrightarrow \langle s, \theta, c1 ; \overline{W}(x,v) ; c2 \rangle$ provided that:
  1) m1,m2 are component chronicles
  2) (c1 ; m1 ; W(x,v) ; m2)(s) =(c1 ; W(x,v))(s)
  3) (c1 ; m1)(s)(x) = c1(s)(x)

- Example: C= let a = y in (y := 3 ; if x ≠ 2 then (if x = 2 then y := a)) $\rightsquigarrow$
  if x ≠ 2 then (if x ≠ 2 then y := 3) else y := 3 = D
  - *Operational reasoning:* if a concurrent thread writes x := 2 between the IFs, then the overall effect of both C and D is skip.
    Otherwise, the overall effect is y := 3.
  - *Denotational reasoning:* in [C] we have either $\langle s, \theta, W(y,3) \rangle$ or $\langle s, \theta, W(y,3) ; \overline{W}(x,2) ; W(y, s(y)) \rangle \twoheadrightarrow \langle s, \theta, \overline{W}(x,2) \rangle$ (accompanied with some environment actions)

Remaining rewrite rules

- Delete-redundant: remove an action with no effect

  ⟨s, θ, c1 ; W(x,v) ; c2⟩ ⇴ ⟨s, θ, c1 ; c2⟩ provided that c1(s)(x) = v
- Example: let a:=x in (x:=a; C) ⇝ let a:=x in C

- Add-redundant: add an action with no effect

  ⟨s, θ, c1 ; c2⟩ ⇴ ⟨s, θ, c1 ; W(x,v) ; c2⟩ provided that c1(s)(x) = v
- Example: skip ⇝ FAA(x,0)

Compositionality argument

- We want to show: [C1] ⊆ [[C1’]] ⇒ [C1 || C2] ⊆ [[C1’ || C2]]
- Given t ∈ [C1 || C2], we need to show t ∈ [[C1’ || C2]]
- We have: t = t1 || t2, t1 ∈ [C1], t2 ∈ [C2]
- We also obtain: t1 ∈ [[C1’]], t1’ ∈ [C1’] , t1’ ⇴* t1
- We need to find some t’ ∈ [C1’ || C2] such that t’ ⇴* t
- This is equivalent to find some t1’’ ∈ [C1’] and t2’ ∈ [C2] such that t’ = t1’’ || t2’

Solution: find some invariants of []-semantics to obtain t2’, and possible rewrite rules should be dual to these invariants

Example of a dual rule

- Invariant: t ∈ [C], t ⇝ t’ ⇒ t’ ∈ [C]
- Duality: t1 ⇴ t2 if and only if dual(t2) ⇝ dual(t1)
- Dual trace: dual(⟨s, θ, a1 ; a2 ; … ; an ⟩) = ⟨s, θ, d(a1) ; d(a2) ; … ; d(an) ⟩
    - d(W(x, v)) = $\overline{W}$(x, v) d($\overline{W}$(x, v)) = W(x, v)
- Coalesce rule: ⟨s, θ, c1 ; m1 ; W(x,v) ; m2 ; c2⟩ ⇴ ⟨s, θ, c1 ; W(x,v) ; c2⟩ provided that:
  1) m1,m2 are component chronicles
  2) (c1 ; m1 ; W(x,v) ; m2)(s) = (c1 ; W(x,v))(s)
- Disperse rule: ⟨s, θ, c1 ; $\overline{W}$(x,v) ; c2⟩ ⇝ ⟨s, θ, c1 ; e1 ; $\overline{W}$(x,v) ; e2 ; c2⟩ provided that:
  1) e1,e2 are environment chronicles
  2) (c1 ; e1 ; $\overline{W}$(x,v) ; e2)(s) = (c1 ; $\overline{W}$(x,v))(s)

Full abstraction argument - example

Csrc = x:=1; y:=1; x:=1 $\rightsquigarrow$? x:=1; y:=1 = Ctgt

$\langle[x,y\mapsto0,0], \theta, W(x, 1) ; \overline{W}(x, 2) ; W(y, 1)\rangle = t \in [Ctgt] \setminus [[Csrc]]$

P[—] = snapshot([x,y↦0,0]); snapshot([x,y↦1,0]); let a = XCHG(x, 2) in (assume (a=1)); snapshot([x,y↦2,0]); snapshot([x,y↦2,1]) || —

We have <P[Ctgt], s0> →* <skip, [x,y↦2,1]>

But if <P[Csrc], s0> →* <skip, [x,y↦2,1]>, then t ∈ [[Csrc]], a contradiction.

- For loop-free commands, we use (sufficiently many) repeated reads instead of snapshot.

Conclusion

- New denotational semantics
- Compositional, adequate, (limited) fully abstract

- More in the paper:
  - Support Read-Modify-Writes
  - Non-deterministic assignments/cycles
  - Example showing that we don't have full abstraction without snapshots but with loops
  - Local rewrites

- Future work:
  - A fully abstract semantics without snapshot command
  - Semantics for termination-sensitive refinement (with infinite fair executions)
  - Weak memory models
  - Higher-order languages

Thank you!