

Denotational semantics

The method

- define syntax (*syntactic domains*)
- define *semantic domains*
- define *semantic functions*
- use *compositional* definitions

History, late 60s, 70s:

- Strachey, Scott (PRG/Oxford)
- VDM (Bekič, Lucas, Jones, Bjørner)
- Plotkin, Smyth, Stoy, Mosses, de Bakker, Tennent, Reynolds ...

Syntactic domains

Each syntactic category of the language forms a *syntactic domain*, which has as elements all the syntactic phrases in this category.

Semantic domains

Semantic domains capture the forms of the intended meanings (*denotations*) for syntactic phrases of the language. All the denotations live in semantic domains, but typically not all elements in semantic domains are denotable.

Semantic domains are defined from *basic domains* (**Int**, **Bool**) using *domain constructors*: product, (disjoint) sum, function spaces, etc.

There is a semantic domain for each key syntactic category of the language.

Semantic functions

For each syntactic category \mathbf{Cat} , define a *semantic function*

$$\mathcal{C}: \mathbf{Cat} \rightarrow \mathbf{CAT}$$

which assigns to the syntactic phrases $ph \in \mathbf{Cat}$ their *denotations* in the corresponding semantic domain \mathbf{CAT} :

$$\mathcal{C}[[ph]] \in \mathbf{CAT}$$

BTW: This defines a semantic equivalence: phrases $ph_1, ph_2 \in \mathbf{Cat}$ are *semantically equivalent* (equivalent w.r.t. the denotational semantics)

$$ph_1 \equiv_{\mathcal{D}\mathcal{S}} ph_2$$

whenever $\mathcal{C}[[ph_1]] = \mathcal{C}[[ph_2]]$.

Compositionality

Semantic functions are defined *compositionally*, so that the denotation of a phrase depends only on the denotations of its immediate components:

$$\mathcal{C}[\varphi(ph_1, \dots, ph_n)] = \Phi(\mathcal{C}[ph_1], \dots, \mathcal{C}[ph_n])$$

Such a *semantic clause* is given for each syntactic construct φ .

Key consequences:

STRUCTURAL INDUCTION

Congruence properties of the semantic equivalence

Homomorphism
property
lurking out

Denotational semantics for TINY

Syntactic domains

Num (**Var**) **Exp** **BExp** **Stmt**

Somewhat informally:

$N \in \mathbf{Num} ::= 0 \mid 1 \mid 2 \mid \dots$

$(x \in \mathbf{Var} ::= \dots)$

$e \in \mathbf{Exp} ::= N \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2$

$b \in \mathbf{BExp} ::= \mathbf{true} \mid \mathbf{false} \mid e_1 \leq e_2 \mid \neg b' \mid b_1 \wedge b_2$

$S \in \mathbf{Stmt} ::= x := e \mid \mathbf{skip} \mid S_1; S_2 \mid \mathbf{if } b \mathbf{ then } S_1 \mathbf{ else } S_2 \mid \mathbf{while } b \mathbf{ do } S'$

Denotational semantics for TINY

Semantic domains

Int (**Bool**) (**State**) **EXP** **BEXP** **STMT**

Int = $\{0, 1, -1, 2, -2, \dots\}$

Bool = $\{\text{tt}, \text{ff}\}$

State = **Var** \rightarrow **Int**

EXP = **State** \rightarrow **Int**

BEXP = **State** \rightarrow **Bool**

STMT = **State** \rightarrow **State**

Semantic functions:

\mathcal{N} : **Num** \rightarrow **Int**

\mathcal{E} : **Exp** \rightarrow **EXP**

\mathcal{B} : **BExp** \rightarrow **BEXP**

\mathcal{S} : **Stmt** \rightarrow **STMT**

STMT: *partial functions*

Semantic function definitions

- Use any mathematical constructions/functions/notations that make sense for our semantic domains.
- In particular, feel free to use basic operations on data in our semantic domains:

$$+, *, - : \mathbf{Int} \times \mathbf{Int} \rightarrow \mathbf{Int}$$
$$\leq : \mathbf{Int} \times \mathbf{Int} \rightarrow \mathbf{Bool}$$
$$\neg : \mathbf{Bool} \rightarrow \mathbf{Bool}$$
$$\vee, \wedge : \mathbf{Bool} \times \mathbf{Bool} \rightarrow \mathbf{Bool}$$

...

Remember though: **COMPOSITIONALITY!**

Some auxiliary notation

- *λ -notation*: $\lambda x:D.E$ stands for the function that maps any $d \in D$ to $E[d/x]$
- *identity*: $id_D = \lambda x:D.x$
- *function composition*: the composition of $f: D_1 \rightarrow D_2$ and $g: D_2 \rightarrow D_3$ is written as $f;g: D_1 \rightarrow D_3$
- *conditional*: $ifte_D: \mathbf{Bool} \times D \times D \rightarrow D$ is defined by

$$ifte_D(c, d_1, d_2) = \begin{cases} d_1 & \text{if } c = \mathbf{tt} \\ d_2 & \text{if } c = \mathbf{ff} \end{cases}$$

(the index D will often be omitted, when clear from the context)

- *indexing*: given any function $f: D_1 \times \dots \times D_n \rightarrow D$, for any domain I ,

$$\text{lift}^I(f): (I \rightarrow D_1) \times \dots \times (I \rightarrow D_n) \rightarrow (I \rightarrow D)$$

is defined as follows:

$$\text{lift}^I(f)(fd_1, \dots, fd_n) = \lambda i: I. f(fd_1(i), \dots, fd_n(i))$$

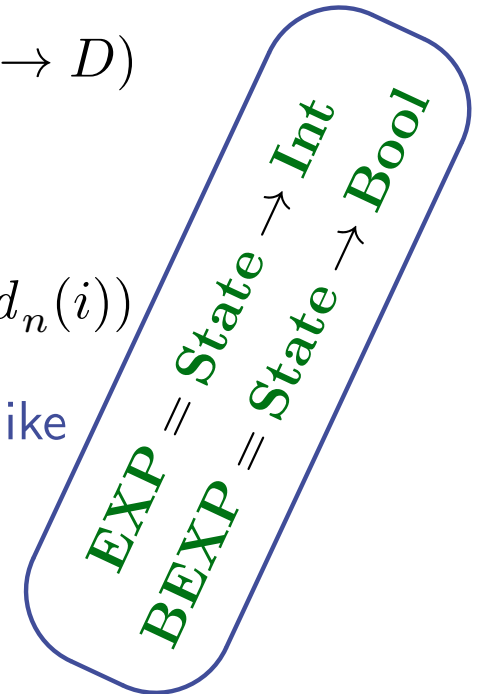
For instance, the conditional on state-dependent functions, like

$$\text{cond}: \mathbf{BEXP} \times \mathbf{EXP} \times \mathbf{EXP} \rightarrow \mathbf{EXP}$$

given explicitly by

$$\text{cond}(B, E_1, E_2)(s) = \text{ifte}_{\mathbf{Int}}(B(s), E_1(s), E_2(s)) = \begin{cases} E_1(s) & \text{if } B(s) = \mathbf{tt} \\ E_2(s) & \text{if } B(s) = \mathbf{ff} \end{cases}$$

may be defined as $\text{cond} = \text{lift}^{\mathbf{State}}(\text{ifte}_{\mathbf{Int}})$.



All these carry over
to partial functions as well

Currying

named after Haskell Curry
by Christopher Strachey

$$D_1 \times D_2 \times \cdots \times D_n \rightarrow D \cong D_1 \rightarrow (D_2 \rightarrow \cdots \rightarrow (D_n \rightarrow D) \cdots)$$

Notational conventions:

- $D_1 \rightarrow D_2 \rightarrow \cdots \rightarrow D_n \rightarrow D$ means $D_1 \rightarrow (D_2 \rightarrow \cdots \rightarrow (D_n \rightarrow D) \cdots)$
- for $f: D_1 \rightarrow D_2 \rightarrow \cdots \rightarrow D_n \rightarrow D$, $d_1 \in D_1$, $d_2 \in D_2$, \dots , $d_n \in D_n$,
 $f d_1 d_2 \cdots d_n$ means $(\cdots ((f d_1) d_2) \cdots) d_n$
- where as before $f d_1$ means $f(d_1)$, etc

Denotational semantics for TINY

Semantic clauses

$\mathcal{N}: \mathbf{Num} \rightarrow \mathbf{Int}$

$$\mathcal{N}[0] = 0$$

$$\mathcal{N}[1] = 1$$

$$\mathcal{N}[2] = 2$$

...

$\mathcal{E}: \mathbf{Exp} \rightarrow \mathbf{EXP}$, where $\mathbf{EXP} = \mathbf{State} \rightarrow \mathbf{Int}$

$$\mathcal{E}[N] s = \mathcal{N}[N] \quad \mathcal{E}[x] s = s x$$

$$\mathcal{E}[e_1 + e_2] s = \mathcal{E}[e_1] s + \mathcal{E}[e_2] s$$

$$\mathcal{E}[e_1 * e_2] s = \mathcal{E}[e_1] s * \mathcal{E}[e_2] s$$

$$\mathcal{E}[e_1 - e_2] s = \mathcal{E}[e_1] s - \mathcal{E}[e_2] s$$

$\mathcal{B}: \mathbf{BExp} \rightarrow \mathbf{BEXP}$, where $\mathbf{BEXP} = \mathbf{State} \rightarrow \mathbf{Bool}$

$$\mathcal{B}[\mathbf{true}] s = \mathbf{tt} \quad \mathcal{B}[\mathbf{false}] s = \mathbf{ff} \quad \mathcal{B}[\neg b] s = \neg(\mathcal{B}[b] s)$$

$$\mathcal{B}[e_1 \leq e_2] s = (\mathcal{E}[e_1] s \leq \mathcal{E}[e_2] s) \quad \mathcal{B}[b_1 \wedge b_2] s = (\mathcal{B}[b_1] s \wedge \mathcal{B}[b_2] s)$$

Denotational semantics for TINY

Semantic clauses

$\mathcal{N}: \mathbf{Num} \rightarrow \mathbf{Int}$

$$\mathcal{N}[0] = 0$$

$$\mathcal{N}[1] = 1$$

$$\mathcal{N}[2] = 2$$

...

$\mathcal{E}: \mathbf{Exp} \rightarrow \mathbf{EXP}$, where $\mathbf{EXP} = \mathbf{State} \rightarrow \mathbf{Int}$

$$\mathcal{E}[N] = \lambda s:\mathbf{State}.\mathcal{N}[N] \quad \mathcal{E}[x] = \lambda s:\mathbf{State}.s\ x$$

$$\mathcal{E}[e_1 + e_2] = \mathit{lift}^{\mathbf{State}}(+)(\mathcal{E}[e_1], \mathcal{E}[e_2])$$

$$\mathcal{E}[e_1 * e_2] = \mathit{lift}^{\mathbf{State}}(*) (\mathcal{E}[e_1], \mathcal{E}[e_2])$$

$$\mathcal{E}[e_1 - e_2] = \mathit{lift}^{\mathbf{State}}(-)(\mathcal{E}[e_1], \mathcal{E}[e_2])$$

$\mathcal{B}: \mathbf{BExp} \rightarrow \mathbf{BEXP}$, where $\mathbf{BEXP} = \mathbf{State} \rightarrow \mathbf{Bool}$

$$\mathcal{B}[\mathbf{true}] = \lambda s:\mathbf{State}.\mathbf{tt} \quad \mathcal{B}[\mathbf{false}] = \lambda s:\mathbf{State}.\mathbf{ff} \quad \mathcal{B}[\neg b] = \mathit{lift}^{\mathbf{State}}(\neg)(\mathcal{B}[b])$$

$$\mathcal{B}[e_1 \leq e_2] = \mathit{lift}^{\mathbf{State}}(\leq)(\mathcal{E}[e_1], \mathcal{E}[e_2]) \quad \mathcal{B}[b_1 \wedge b_2] = \mathit{lift}^{\mathbf{State}}(\wedge)(\mathcal{B}[b_1], \mathcal{B}[b_2])$$

Denotational semantics for TINY

Semantic clauses

$\mathcal{S}: \text{Stmt} \rightarrow \text{STMT}$, where $\text{STMT} = \text{State} \rightarrow \text{State}$

$$\mathcal{S}[[x := e]] s = s[x \mapsto \mathcal{E}[[e]] s]$$

$$\mathcal{S}[[\text{skip}]] s = s$$

$$\mathcal{S}[[S_1; S_2]] s = \mathcal{S}[[S_2]] (\mathcal{S}[[S_1]] s)$$

$$\mathcal{S}[[\text{if } b \text{ then } S_1 \text{ else } S_2]] s = \text{ifte}(\mathcal{B}[[b]] s, \mathcal{S}[[S_1]] s, \mathcal{S}[[S_2]] s)$$

$$\mathcal{S}[[\text{while } b \text{ do } S]] s = \text{ifte}(\mathcal{B}[[b]] s, \mathcal{S}[[\text{while } b \text{ do } S]] (\mathcal{S}[[S]] s), s)$$

Denotational semantics for TINY

Semantic clauses

$\mathcal{S}: \text{Stmt} \rightarrow \text{STMT}$, where $\text{STMT} = \text{State} \rightarrow \text{State}$

$$\mathcal{S}[[x := e]] = \lambda s:\text{State}.s[x \mapsto \mathcal{E}[[e]] s]$$

$$\mathcal{S}[[\text{skip}]] = id_{\text{State}}$$

$$\mathcal{S}[[S_1; S_2]] = \mathcal{S}[[S_1]]; \mathcal{S}[[S_2]]$$

$$\mathcal{S}[[\text{if } b \text{ then } S_1 \text{ else } S_2]] = \text{cond}(\mathcal{B}[[b]], \mathcal{S}[[S_1]], \mathcal{S}[[S_2]])$$

$$\mathcal{S}[[\text{while } b \text{ do } S]] = \text{cond}(\mathcal{B}[[b]], \mathcal{S}[[S]]; \mathcal{S}[[\text{while } b \text{ do } S]], id_{\text{State}})$$

Something wrong?

The clause for **while**:

$$\mathcal{S}[\mathbf{while} \ b \ \mathbf{do} \ S] = \mathit{cond}(\mathcal{B}[b], \mathcal{S}[S]; \mathcal{S}[\mathbf{while} \ b \ \mathbf{do} \ S], \mathit{id}_{\mathbf{State}})$$

is *not* compositional!

We "define": $???$ $\mathcal{S}[\mathbf{while} \ b \ \mathbf{do} \ S] = \Phi(\dots, \mathcal{S}[\mathbf{while} \ b \ \mathbf{do} \ S], \dots)$ $???$

We need *fixed-point definitions*

For $\Phi: (\mathbf{State} \rightarrow \mathbf{State}) \rightarrow (\mathbf{State} \rightarrow \mathbf{State})$, we want to use the clause

$$F = \Phi(F)$$

to define an "appropriate" $F: \mathbf{State} \rightarrow \mathbf{State}$ $???$

Potential problems with fixed-point definitions

Consider fixed-point definitions in $\mathbf{STMT} = \mathbf{State} \rightarrow \mathbf{State}$, as

$$F = \Phi(F)$$

- Does a fixed point always exist?

$$F = \lambda s:\mathbf{State}. \text{ifte}_{\mathbf{State}}(F(s) \text{ is not defined}, s, F(s)[x \mapsto F(s)(x) + 1])$$

Only some functionals Φ may be allowed

- If a fixed point exists, is it unique?

$$F = \lambda s:\mathbf{State}. F(s)[x \mapsto 2 * F(s)(x)]$$

(or even: $F = \lambda s:\mathbf{State}. F(s)$)

Some “best” fixed point must be chosen

The guiding fixed-point definition

Looking closer at the clause for **while**:

$$\mathcal{S}[\mathbf{while} \ b \ \mathbf{do} \ S] = \Phi(\mathcal{S}[\mathbf{while} \ b \ \mathbf{do} \ S])$$

where $\Phi: \mathbf{STMT} \rightarrow \mathbf{STMT}$ is defined as follows:

$$\Phi(F) = \mathit{cond}(\mathcal{B}[b], \mathcal{S}[S]; F, \mathit{id}_{\mathbf{State}})$$

Whatever fixed point we choose, we want it to be adequate for our operational intuitions; we want a denotation $\mathit{fix}(\Phi) \in \mathbf{STMT}$ that is a fixed point of Φ (so that $\Phi(\mathit{fix}(\Phi)) = \mathit{fix}(\Phi)$) and is adequate for the operational semantics of **while**, i.e., such that

$$\langle \mathbf{while} \ b \ \mathbf{do} \ S, s \rangle \Rightarrow^* s' \ \text{iff} \ \mathit{fix}(\Phi) \ s = s'$$

Right guess!

Suppose that we have such adequacy for S , i.e., $\langle S, s \rangle \Rightarrow^* s'$ iff $\mathcal{S}[[S]] s = s'$.

Right guess:

$\langle \text{while } b \text{ do } S, s \rangle \Rightarrow^* s'$ iff for some $n \geq 0$, $\Phi^n(\emptyset_{\text{State} \rightarrow \text{State}}) s = s'$

where $\emptyset_{\text{State} \rightarrow \text{State}} : \text{State} \rightarrow \text{State}$ is the function undefined everywhere,

$\Phi^0(\emptyset_{\text{State} \rightarrow \text{State}}) = \emptyset_{\text{State} \rightarrow \text{State}}$, and $\Phi^{n+1}(\emptyset_{\text{State} \rightarrow \text{State}}) = \Phi(\Phi^n(\emptyset_{\text{State} \rightarrow \text{State}}))$,

and $\Phi(F) = \text{cond}(\mathcal{B}[[b]], \mathcal{S}[[S]]; F, \text{id}_{\text{State}})$.

Proof: in a moment.

Conclusion

$\mathcal{S}[[\text{while } b \text{ do } S]] = \text{fix}(\Phi) = \bigcup_{n \geq 0} \Phi^n(\emptyset_{\text{State} \rightarrow \text{State}})$

This is well-defined, and yields the *least* fixed point of Φ .

Believe this for now;
more on this later.

while $\{sqr = (rt + 1)^2 \wedge rt^2 \leq n\}$ $sqr \leq n$ **do** $rt := rt + 1; sqr := sqr + 2 * rt + 1$

$$\Phi(F) = \text{cond}(\mathcal{B}[\![sqr \leq n]\!], \mathcal{S}[\![rt := rt + 1; sqr := sqr + 2 * rt + 1]\!]; F, id_{\text{State}})$$

$s(n, rt, sqr)$	$\Phi^0(\emptyset)(s)$	$\Phi^1(\emptyset)(s)$	$\Phi^2(\emptyset)(s)$	$\Phi^3(\emptyset)(s)$	$\Phi^4(\emptyset)(s)$	\dots	$\bigcup \Phi^n(\emptyset)(s)$
0, 0, 1	?	0, 0, 1	0, 0, 1	0, 0, 1	0, 0, 1	\dots	0, 0, 1
1, 0, 1	?	?	1, 1, 4	1, 1, 4	1, 1, 4	\dots	1, 1, 4
2, 0, 1	?	?	2, 1, 4	2, 1, 4	2, 1, 4	\dots	2, 1, 4
3, 0, 1	?	?	3, 1, 4	3, 1, 4	3, 1, 4	\dots	3, 1, 4
4, 0, 1	?	?	?	4, 2, 9	4, 2, 9	\dots	4, 2, 9
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots
8, 0, 1	?	?	?	8, 2, 9	8, 2, 9	\dots	8, 2, 9
9, 0, 1	?	?	?	?	9, 3, 16	\dots	9, 3, 16
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots

$$\Phi(F) = \text{cond}(\mathcal{B}[\text{sqr} \leq n], \mathcal{S}[\text{rt} := \text{rt} + 1; \text{sqr} := \text{sqr} + 2 * \text{rt} + 1]; F, \text{id}_{\text{State}})$$

$s(n, \text{rt}, \text{sqr})$	$\Phi^0(\emptyset)(s)$	$\Phi^1(\emptyset)(s)$	$\Phi^2(\emptyset)(s)$	$\Phi^3(\emptyset)(s)$	$\Phi^4(\emptyset)(s)$	\dots	$\bigcup \Phi^n(\emptyset)(s)$
0, 0, 1	?	0, 0, 1	0, 0, 1	0, 0, 1	0, 0, 1	\dots	0, 0, 1
1, 0, 1	?	?	1, 1, 4	1, 1, 4	1, 1, 4	\dots	1, 1, 4
1, 1, 4	?	1, 1, 4	1, 1, 4	1, 1, 4	1, 1, 4	\dots	1, 1, 4
2, 0, 1	?	?	2, 1, 4	2, 1, 4	2, 1, 4	\dots	2, 1, 4
2, 1, 4	?	2, 1, 4	2, 1, 4	2, 1, 4	2, 1, 4	\dots	2, 1, 4
3, 0, 1	?	?	3, 1, 4	3, 1, 4	3, 1, 4	\dots	3, 1, 4
3, 1, 4	?	3, 1, 4	3, 1, 4	3, 1, 4	3, 1, 4	\dots	3, 1, 4
4, 0, 1	?	?	?	4, 2, 9	4, 2, 9	\dots	4, 2, 9
4, 1, 4	?	?	4, 2, 9	4, 2, 9	4, 2, 9	\dots	4, 2, 9
4, 2, 9	?	4, 2, 9	4, 2, 9	4, 2, 9	4, 2, 9	\dots	4, 2, 9
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots
9, 0, 1	?	?	?	?	9, 3, 16	\dots	9, 3, 16
9, 1, 4	?	?	?	9, 3, 16	9, 3, 16	\dots	9, 3, 16
9, 2, 9	?	?	9, 3, 16	9, 3, 16	9, 3, 16	\dots	9, 3, 16
9, 3, 16	?	9, 3, 16	9, 3, 16	9, 3, 16	9, 3, 16	\dots	9, 3, 16
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots

$\langle \mathbf{while} \ b \ \mathbf{do} \ S, s \rangle \Rightarrow^* s' \implies$ for some $n \geq 0$, $\Phi^n(\emptyset_{\mathbf{State} \rightarrow \mathbf{State}}) s = s'$
 where $\Phi(F) = \mathit{cond}(\mathcal{B}[[b]], \mathcal{S}[[S]]; F, \mathit{id}_{\mathbf{State}})$

Proof

By induction on the length of the computation $\langle \mathbf{while} \ b \ \mathbf{do} \ S, s \rangle \Rightarrow^k s'$.

$k > 0$: Then $\langle \mathbf{while} \ b \ \mathbf{do} \ S, s \rangle \Rightarrow \gamma \Rightarrow^{k-1} s'$. By cases on this first step:

- $\mathcal{B}[[b]] \ s = \mathbf{ff}$ and $\gamma = s$. Then $s' = s$, and $\Phi(\emptyset_{\mathbf{State} \rightarrow \mathbf{State}}) s = s$. OK
- $\mathcal{B}[[b]] \ s = \mathbf{tt}$ and $\gamma = \langle S; \mathbf{while} \ b \ \mathbf{do} \ S, s \rangle \Rightarrow^{k-1} s'$. Then $\langle S, s \rangle \Rightarrow^{k_1} \hat{s}$ and $\langle \mathbf{while} \ b \ \mathbf{do} \ S, \hat{s} \rangle \Rightarrow^{k_2} s'$, for some $\hat{s} \in \mathbf{State}$ and $k_1, k_2 > 0$ with $k_1 + k_2 = k - 1$. Hence, $\mathcal{S}[[S]] \ s = \hat{s}$ and $\Phi^n(\emptyset_{\mathbf{State} \rightarrow \mathbf{State}}) \hat{s} = s'$ for some $n \geq 0$. Thus, $\Phi^{n+1}(\emptyset_{\mathbf{State} \rightarrow \mathbf{State}}) s = s'$. OK

BTW: This relies on $\langle S, s \rangle \Rightarrow^* s' \implies \mathcal{S}[[S]] \ s = s'$

$\langle \text{while } b \text{ do } S, s \rangle \Rightarrow^* s' \iff \text{for some } n \geq 0, \Phi^n(\emptyset_{\text{State} \rightarrow \text{State}}) s = s'$
 where $\Phi(F) = \text{cond}(\mathcal{B}[b], \mathcal{S}[S]; F, \text{id}_{\text{State}})$

Proof

By induction on $n \geq 0$, assuming $\Phi^n(\emptyset_{\text{State} \rightarrow \text{State}}) s = s'$.

$n > 0$: Then $\Phi^n(\emptyset_{\text{State} \rightarrow \text{State}}) s = \text{cond}(\mathcal{B}[b], \mathcal{S}[S]; \Phi^{n-1}(\emptyset_{\text{State} \rightarrow \text{State}}), \text{id}_{\text{State}}) s$.

- $\mathcal{B}[b] s = \mathbf{ff}$: then $\Phi^n(\emptyset_{\text{State} \rightarrow \text{State}}) s = s$, so $s' = s$, and also

$\langle \text{while } b \text{ do } S, s \rangle \Rightarrow s$. OK

- $\mathcal{B}[b] s = \mathbf{tt}$: then $\Phi^n(\emptyset_{\text{State} \rightarrow \text{State}}) s = \Phi^{n-1}(\emptyset_{\text{State} \rightarrow \text{State}}) (\mathcal{S}[S] s) = s'$.

Hence, $\langle \text{while } b \text{ do } S, \mathcal{S}[S] s \rangle \Rightarrow^* s'$, and since $\langle S, s \rangle \Rightarrow^* (\mathcal{S}[S] s)$, we get

$\langle \text{while } b \text{ do } S, s \rangle \Rightarrow \langle S; \text{while } b \text{ do } S, s \rangle \Rightarrow^* \langle \text{while } b \text{ do } S, \mathcal{S}[S] s \rangle \Rightarrow^* s'$.

OK

BTW: This relies on $\langle S, s \rangle \Rightarrow^* s' \iff \mathcal{S}[S] s = s'$

Adequacy of denotational semantics

Fact: For each statement $S \in \mathbf{Stmt}$ and states $s, s' \in \mathbf{State}$,

$$\langle S, s \rangle \Rightarrow^* s' \text{ iff } \mathcal{S}[[S]] s = s'$$

Proof:

“ \implies ” : By induction on the length of the computation $\langle S, s \rangle \Rightarrow^* s'$.

“ \impliedby ” : By structural induction on S .