Formal Semantics III: Designing Rules (Part D)

CAS CS 320: Principles of Programming Languages

Thursday, April 4, 2024

NEW MATERIAL,

NOT IN PRECEDING LECTURES

Applying The Evaluation Rules To Our "Toy" Language Augmented With Variables

called "VarLang" in these slides where we use "p/S" to denote a configuration instead of "(S,p)"

Operational Semantics of VarLang

```
<prog>   ::= <com> ; <prog> | []
<com>    ::= Push <term> | Pop | Swap | Add | Let <var> | Quit
<term>   ::= <int> | <var>
<var>    ::= variables
<int>    ::= integers
```

VarLang is a stack manipulating language with the ability to bind variables. When designing its operational semantics, we must account for the bindings between variables and values.

```
<state> ::= <prog>/<stack>/<env> | ERROR
<stack> ::= <int> :: <stack> | []
<env>   ::= (<var> ↦ <int>) :: <env> | []
```

Operational Semantics of VarLang

```
<state> ::= <prog>/<stack>/<env> | ERROR
<stack> ::= <int> :: <stack> | []
<env>   ::= (<var> ↦ <int>) :: <env> | []
```

Environment examples:

- []
- `(x ↦ 1) :: []`
- `(y ↦ 3) :: (x ↦ 1) :: []`
- `(y ↦ 3) :: (x ↦ 1) :: (w ↦ 4) :: []`

Operational Semantics of VarLang

```
<state> ::= <prog>/<stack>/<env> | ERROR
<stack> ::= <int> :: <stack> | []
<env>   ::= (<var> ↦ <int>) :: <env> | []
```

We include the environment as a part of VarLang's reduction relation.

$$P/S/E \to Q/R/F$$

This relation states that program P with stack S and environment E reduces to program Q with stack R and environment F.

Operational Semantics of VarLang

$$\frac{n \in \mathbb{Z}}{\text{Push } n\ ;\ p/S/E \rightarrow p/(n :: S)/E}\ \text{push-int}$$

$$\frac{update(E, v, n) = F}{\text{Let } v\ ;\ p/(n :: S)/E \rightarrow p/S/F}\ \text{let-ok}$$

$$\frac{v \in var \qquad fetch(E, v) = n}{\text{Push } v\ ;\ p/S/E \rightarrow p/(n :: S)/E}\ \text{push-var}$$

$$\frac{}{\text{Let } v\ ;\ p/[\,]/E \rightarrow \text{ERROR}}\ \text{let-error}$$

$$\frac{v \in var \qquad fetch(E, v) = \bot}{\text{Push } v\ ;\ p/S/E \rightarrow \text{ERROR}}\ \text{push-error}$$

$fetch : env \times var \rightarrow \mathbb{Z} \cup \{\bot\}$

$update : env \times var \times \mathbb{Z} \rightarrow env$

$fetch$ and $update$ are meta-functions which exist outside of VarLang. They manipulate the environment in the expected way.

Operational Semantics of VarLang

Example: reduction of Push 1; Let x; Push x; Push x; [] in an empty stack and environment.

(1) $$\frac{1 \in \mathbb{Z}}{\text{Push 1; Let x; Push x; Push x; [] / [] / []} \rightarrow \text{Let x; Push x; Push x; [] / (1 :: []) / []}}\ \text{push-int}$$

Operational Semantics of VarLang

Example: reduction of Push 1; Let x; Push x; Push x; [] in an empty stack and environment.

$$\frac{1 \in \mathbb{Z}}{\text{Push 1; Let x; Push x; Push x; [] / [] / []} \rightarrow \text{Let x; Push x; Push x; [] / (1 :: []) / []}}\ \text{push-int} \quad (1)$$

$$\frac{update([], x, 1) = (\text{x} \mapsto 1) :: []}{\text{Let x; Push x; Push x; [] / (1 :: []) / []} \rightarrow \text{Push x; Push x; [] / [] / (x} \mapsto \text{1) :: []}}\ \text{let-ok} \quad (2)$$

Operational Semantics of VarLang

Example: reduction of Push 1; Let x; Push x; Push x; [] in an empty stack and environment.

$$\text{(1)}\quad \frac{1 \in \mathbb{Z}}{\text{Push 1; Let x; Push x; Push x; [] / [] / []} \rightarrow \text{Let x; Push x; Push x; [] / (1 :: []) / []}}\ \text{push-int}$$

$$\text{(2)}\quad \frac{update([], x, 1) = (\text{x} \mapsto 1) :: []}{\text{Let x; Push x; Push x; [] / (1 :: []) / []} \rightarrow \text{Push x; Push x; [] / [] / (x} \mapsto \text{1) :: []}}\ \text{let-ok}$$

$$\text{(3)}\quad \frac{x \in var \qquad fetch((\text{x} \mapsto 1) :: [], x) = 1}{\text{Push x; Push x; [] / [] / (x} \mapsto \text{1) :: []} \rightarrow \text{Push x; [] / (1 :: []) / (x} \mapsto \text{1) :: []}}\ \text{push-var}$$

Operational Semantics of VarLang

Example: reduction of Push 1; Let x; Push x; Push x; [] in an empty stack and environment.

(1) $$\frac{1 \in \mathbb{Z}}{\text{Push 1; Let x; Push x; Push x; [] / [] / []} \rightarrow \text{Let x; Push x; Push x; [] / (1 :: []) / []}}\ \text{push-int}$$

(2) $$\frac{update([], x, 1) = (\text{x} \mapsto 1) :: []}{\text{Let x; Push x; Push x; [] / (1 :: []) / []} \rightarrow \text{Push x; Push x; [] / [] / (x} \mapsto \text{1) :: []}}\ \text{let-ok}$$

(3) $$\frac{x \in var \qquad fetch((\text{x} \mapsto 1) :: [], x) = 1}{\text{Push x; Push x; [] / [] / (x} \mapsto \text{1) :: []} \rightarrow \text{Push x; [] / (1 :: []) / (x} \mapsto \text{1) :: []}}\ \text{push-var}$$

(4) $$\frac{x \in var \qquad fetch((\text{x} \mapsto 1) :: [], x) = 1}{\text{Push x; [] / (1 :: []) / (x} \mapsto \text{1) :: []} \rightarrow \text{[] / (1 :: 1 :: []) / (x} \mapsto \text{1) :: []}}\ \text{push-var}$$

Operational Semantics of VarLang

Example: reduction of Push 1; Let x; Push x; Push x; [] in an empty stack and environment.

Compose together single step reductions via the transitive rule for multi-step.

$$\overline{}\ \text{reflexive}$$

$$(4)\quad [\,]\ /\ (1 :: 1 :: [\,])\ /\ (x \mapsto 1) :: [\,] \to^* [\,]\ /\ (1 :: 1 :: [\,])\ /\ (x \mapsto 1) :: [\,]$$

$$\overline{}\ \text{transitive}$$

$$(3)\quad \text{Push x; } [\,]\ /\ (1 :: [\,])\ /\ (x \mapsto 1) :: [\,] \to^* [\,]\ /\ (1 :: 1 :: [\,])\ /\ (x \mapsto 1) :: [\,]$$

$$\overline{}\ \text{transitive}$$

$$(2)\quad \text{Push x; Push x; } [\,]\ /\ [\,]\ /\ (x \mapsto 1) :: [\,] \to^* [\,]\ /\ (1 :: 1 :: [\,])\ /\ (x \mapsto 1) :: [\,]$$

$$\overline{}\ \text{transitive}$$

$$(1)\quad \text{Let x; Push x; Push x; } [\,]\ /\ (1 :: [\,])\ /\ [\,] \to^* [\,]\ /\ (1 :: 1 :: [\,])\ /\ (x \mapsto 1) :: [\,]$$

$$\overline{}\ \text{transitive}$$

$$\text{Push 1; Let x; Push x; Push x; } [\,]\ /\ [\,]\ /\ [\,] \to^* [\,]\ /\ (1 :: 1 :: [\,])\ /\ (x \mapsto 1) :: [\,]$$

(THIS PAGE INTENTIONALLY LEFT BLANK)