

Structural Semantics

We initially defined dynamic semantics of $\mathcal{L}\{\text{num str}\}$ using a single-step transition relation $e \mapsto e'$.

1. Primitive instructions.
2. Search rules.

Evaluation is the restriction of multi-step evaluation to values:
 $e \mapsto^* v$.

1

Contextual Semantics

A variant on structural semantics, differing only in style of presentation, is called **contextual semantics**:

1. Instruction transitions.
2. Evaluation contexts.

Evaluation produces values, with the judgement $e \text{ val}$ as for structural semantics.

2

Instruction Transitions

$$\frac{m+n = p \text{ nat}}{\text{plus}(\text{num}[m]; \text{num}[n]) \rightsquigarrow \text{num}[p]}$$

$$\frac{s^*t = u \text{ str}}{\text{cat}(\text{str}[s]; \text{str}[t]) \rightsquigarrow \text{str}[u]}$$

$$\frac{}{\text{let}(e_1; x.e_2) \rightsquigarrow [e_1/x]e_2}$$

Similar rules for multiplication and string length.

3

Evaluation Contexts

Judgement $\mathcal{E} \text{ ectxt}$ determines location of the next instruction to execute in a larger expression. That location is marked by a "hole" (\circ).

$$\frac{}{\circ \text{ ectxt}}$$

$$\frac{\mathcal{E}_1 \text{ ectxt}}{\text{plus}(\mathcal{E}_1; e_2) \text{ ectxt}}$$

$$\frac{e_1 \text{ val} \quad \mathcal{E}_2 \text{ ectxt}}{\text{plus}(e_1; \mathcal{E}_2) \text{ ectxt}}$$

Similar rules for multiplication

4

Evaluation Contexts

$$\frac{\mathcal{E}_1 \text{ ectxt}}{\text{cat}(\mathcal{E}_1; e_2) \text{ ectxt}}$$

$$\frac{e_1 \text{ val} \quad \mathcal{E}_2 \text{ ectxt}}{\text{cat}(e_1; \mathcal{E}_2) \text{ ectxt}}$$

Similar rules for string length

$$\frac{\mathcal{E}_1 \text{ ectxt}}{\text{let}(\mathcal{E}_1; x.e_2) \text{ ectxt}}$$

5

Evaluation Contexts: Filling Holes

Execution contexts are templates, instantiated by filling holes with instructions to be executed. Judgement $e' = \mathcal{E}\{e\}$ says that expression e' results from filling the hole in \mathcal{E} with expression e .

$$\overline{e = \circ\{e\}}$$

$$\frac{e_1 = \mathcal{E}_1\{e\}}{\text{plus}(e_1; e_2) = \text{plus}(\mathcal{E}_1; e_2)\{e\}}$$

$$\frac{e_1 \text{ val } e_2 = \mathcal{E}_2\{e\}}{\text{plus}(e_1; e_2) = \text{plus}(e_1; \mathcal{E}_2)\{e\}}$$

Similar rules for multiplication

6

Evaluation Contexts: Filling Holes

$$\frac{e_1 = \mathcal{E}_1\{e\}}{\text{cat}(e_1; e_2) = \text{cat}(\mathcal{E}_1; e_2)\{e\}}$$

$$\frac{e_1 \text{ val } e_2 = \mathcal{E}_2\{e\}}{\text{cat}(e_1; e_2) = \text{cat}(e_1; \mathcal{E}_2)\{e\}}$$

Similar rules for string length

$$\frac{e_1 = \mathcal{E}_1\{e\}}{\text{let}(e_1; x.e_2) = \text{let}(\mathcal{E}_1; x.e_2)\{e\}}$$

7

Contextual Semantics

A single rule defines contextual semantics:

$$\frac{e = \mathcal{E}\{e_0\} \quad e_0 \rightsquigarrow e'_0 \quad e' = \mathcal{E}\{e'_0\}}{e \mapsto e'}$$

Thus, transition $e \mapsto e'$ consists of:

1. Decomposing e into evaluation context and instruction
2. Execution of instruction
3. Replacing instruction by result of its execution at same spot

8

Contextual Semantics: Example Derivation

A contextual semantics derivation of :

$$\text{plus}(\text{plus}(\text{num}[1]; \text{num}[2]); \text{num}[4]) \mapsto \text{plus}(\text{num}[3]; \text{num}[4])$$

We abbreviate $\text{plus}(\text{num}[1]; \text{num}[2])$ by $p(1, 2)$, $\text{num}[3]$ by 3, etc.

$$\frac{\frac{\overline{p(1, 2) = \circ\{p(1, 2)\}}}{p(p(1, 2), 4) = p(\circ, 4)\{p(1, 2)\}} \quad \frac{1+2 = 3 \text{ nat}}{p(1, 2) \rightsquigarrow 3} \quad \frac{\overline{3 = \circ\{3\}}}{p(3, 4) = p(\circ, 4)\{3\}}}{p(p(1, 2), 4) \mapsto p(3, 4)}$$

9

Correspondence Between Structural and Contextual Semantics

Structural and contextual semantics define the same transition relation.

Writing $e \mapsto_s e'$ for the transition relation defined by structural semantics and $e \mapsto_c e'$ for the one defined by contextual semantics:

Theorem 1

$e \mapsto_s e'$ iff $e \mapsto_c e'$.

10

Advantages of Contextual Semantics

Superficial: Simple formulation, leaving decompositions implicit:

$$\frac{e_0 \rightsquigarrow e'_0}{\mathcal{E}\{e_0\} \mapsto \mathcal{E}\{e'_0\}}$$

Deeper: Transition judgements apply only to closed expressions of **fixed** type, while structural semantics transitions necessarily defined over expressions of **every** type.

11

Structural Semantics

Advantages of structural semantics (and its variants):

- Scales well to richer languages.
- Supports crisp meta-theory such as safety proof.

12

Structural Semantics

Disadvantages of structural semantics:

- Evaluation is defined only indirectly.
- Individual steps are often not of interest.

13

Evaluation Semantics

A more direct approach is to define an evaluation relation $e \Downarrow v$ directly by an inductive definition.

- Avoids details of finding next step.
- Avoids indirection of structural semantics definition.

The rules defining the evaluation relation for $\mathcal{L}\{\text{num str}\}$ follow.

14

Evaluation Semantics: Primitive Values

$$\frac{}{\text{num}[n] \Downarrow \text{num}[n]}$$

$$\frac{}{\text{str}[s] \Downarrow \text{str}[s]}$$

15

Evaluation Semantics: Primitive Operations

$$\frac{e_1 \Downarrow \text{num}[n_1] \quad e_2 \Downarrow \text{num}[n_2] \quad n_1 + n_2 = n \text{ nat}}{\text{plus}(e_1; e_2) \Downarrow \text{num}[n]}$$

Similar rule for multiplication.

$$\frac{e_1 \Downarrow \text{str}[s_1] \quad e_2 \Downarrow \text{str}[s_2] \quad s_1 \hat{\ } s_2 = s \text{ str}}{\text{cat}(e_1; e_2) \Downarrow \text{str}[s]}$$

$$\frac{e \Downarrow \text{str}[s] \quad |s| = n \text{ str}}{\text{len}(e) \Downarrow \text{num}[n]}$$

16

Evaluation Semantics: Primitive Operations

$$\frac{[e_1/x]e_2 \Downarrow v_2}{\text{let}(e_1; x.e_2) \Downarrow v_2}$$

Since $[e_1/x]e_2$ is not a sub-expression of $\text{let}(e_1; x.e_2)$, these rules are **not** syntax-directed!

17

Evaluation Semantics: Example Derivation

An evaluation semantics derivation of :

$$\text{plus}(\text{plus}(\text{num}[1]; \text{num}[2]); \text{num}[4]) \Downarrow \text{num}[7]$$

We abbreviate $\text{plus}(\text{num}[1]; \text{num}[2])$ by $p(1, 2)$, $\text{num}[3]$ by 3, etc.

$$\frac{\frac{\frac{1 \Downarrow 1}{\text{p}(1, 2)} \quad \frac{2 \Downarrow 2}{3} \quad 1+2 = 3 \text{ nat}}{\text{p}(1, 2) \Downarrow 3} \quad \frac{4 \Downarrow 4}{3+4 = 7 \text{ nat}}}{\text{p}(\text{p}(1, 2), 4) \Downarrow 7}$$

18

Induction on Evaluation

Since the evaluation judgement is inductively defined, there is an associated principle of induction, called **induction on evaluation**.

To prove that $\mathcal{P}(e \Downarrow v)$ for some property \mathcal{P} , it suffices to prove that \mathcal{P} is closed under the rules defining the evaluation judgement.

1. $\mathcal{P}(\text{num}[n] \Downarrow \text{num}[n])$ and $\mathcal{P}(\text{str}[s] \Downarrow \text{str}[s])$.
2. Assuming \mathcal{P} holds for each of the premises of an evaluation rule, show that it holds for the conclusion as well.

19

Induction on Evaluation

Note that induction on evaluation is **not** the same as structural induction, because the evaluation rules are not syntax-directed!

It is, however, sufficient for proving properties such as the following:

Lemma 2

If $e \Downarrow v$ then $v \text{ val}$.

This is easily proved by rule induction on the rules defining evaluation.

20

Correspondence Between Structural and Evaluation Semantics

How does the evaluation semantics relate to the structural semantics?

Theorem 3

For all closed expressions e and values v , $e \mapsto^* v$ iff $e \Downarrow v$.

How can we prove this theorem?

21

Correspondence

Evaluation can be seen as a description of a transition path from expression to value.

Lemma 4

If $e \Downarrow v$, then $e \mapsto^* v$.

This can be proved by rule induction on the rules defining evaluation.

22

Correspondence

The converse is trickier!

Conjecture 5

If $e \mapsto e'$ then $e \Downarrow e'$.

This is a **not even sensible** since e' must be a value if $e \Downarrow e'$!

The problem remains, even if we replace \mapsto by \mapsto^* .

23

Correspondence

Lemma 6

If $e \mapsto^* v$ then $e \Downarrow v$.

Recall: multi-step evaluation is inductively defined!

$$\frac{}{v \mapsto^* v} \quad \frac{e \mapsto e' \quad e' \mapsto^* v}{e \mapsto^* v}$$

24

Correspondence

Therefore it suffices to show:

- If $e \mapsto e'$ and $e' \Downarrow v$, then $e \Downarrow v$.

That is, we need only show that evaluation is closed under head expansion (reverse execution)!

Proceed by induction on the rules defining the transition judgement.

25

Type Safety And Evaluation Semantics

It is easy to state and prove an analogue of the Preservation Theorem for evaluation semantics:

Theorem 7

If $e : \tau$ and $e \Downarrow v$, then $v : \tau$.

This may be proved directly by induction on the rules defining evaluation

26

Type Safety And Evaluation Semantics

Is there an analogue of the Progress Theorem? How can we capture the idea of making progress?

Conjecture 8 (FALSE)

If $e : \tau$ then either e is a value or there exists a value v such that $e \Downarrow v$.

This asks for too much! It's not even true, since there are well-typed expressions that loop forever.

What to do?

27

Type Safety And Evaluation Semantics

Standard approach: reduce progress to preservation for an **instrumented** semantics.

- Add new evaluation rules that check for run-time type errors.
- Prove that these rules can never be used when evaluating a well-typed program.

28

Type Safety And Evaluation Semantics

It's kind of backwards, but

- It works.
- It's standard.
- It leads to an interesting idea.

29

Checked Errors

Define a judgement $e \uparrow$ stating that the expression e **goes wrong** when executed. For example:

$$\frac{}{\text{plus}(\text{str}[s]; e_2) \uparrow} \quad \frac{e_1 \text{ val}}{\text{plus}(e_1; \text{str}[s]) \uparrow}$$

Similar rules for each primitive construct of $\mathcal{L}\{\text{num str}\}$

30

Checking Type Errors

Idea: turn **type** errors into **checked** errors.

- Explicitly check for ill-typed situations.
- Previously these were left undefined (stuck).

31

Safety With Checked Errors

Theorem 9

If $e \uparrow$ then there is no τ such that $e : \tau$.

Corollary 10

If $e : \tau$ then $\neg(e \uparrow)$.

That is, well-typed programs do not “go wrong”.

32

Safety With Checked Errors

Proof is by induction on the evaluation rules:

For example, consider the rule

$$\frac{}{\text{plus}(\text{str}[s]; e_2) \uparrow}$$

We observe that $\text{str}[s] : \text{str}$ and hence $\text{plus}(\text{str}[s]; e_2)$ is ill-typed.

33

Safety for Evaluation vs Structural Semantics

Evaluation semantics:

- We must add a judgement $e \uparrow$ only to show it is irrelevant for well-typed programs
- We must be sure to add an evaluation rule for each “stuck” state.
- No way to ensure we have added enough
- Proof only shows that the ones we’ve added cannot arise, but cannot prove we’ve covered all possible cases.

34

Safety for Evaluation vs Structural Semantics

Structural semantics:

- No behavior specified for ill-typed programs, so ill-typed programs get “stuck” without explicit intervention – no need to specify $e \uparrow$ judgements!
- Corresponds more closely to implementation – compiler needn’t generate checks for run-time type errors and ill-typed programs have no meaning.
- Hence execution is more efficient and language definition is simpler.

35

Cost Semantics

A structural semantics provides an abstract **time complexity** for programs.

- Time complexity of an expression = number of steps to reach a value.

Can also consider **space complexity** as a **cost measure** of programs.

36

Cost Semantics

A **cost semantics** is an evaluation semantics indexed by a **cost measure**.

$$e \Downarrow^k v \text{ iff } e \Downarrow v \text{ with cost } k.$$

Costs are **integers** representing the number of steps required to evaluate the expression.

37

Cost Semantics for Expressions

Numbers:

$$\frac{}{\text{num}[n] \Downarrow^0 \text{num}[n]}$$

$$\frac{e_1 \Downarrow^{k_1} \text{num}[n_1] \quad e_2 \Downarrow^{k_2} \text{num}[n_2]}{\text{plus}(e_1; e_2) \Downarrow^{k_1+k_2+1} \text{num}[n_1 + n_2]}$$

Similarly for multiplication

38

Cost Semantics for Expressions

Strings:

$$\frac{}{\text{str}[s] \Downarrow^0 \text{str}[s]}$$

$$\frac{e_1 \Downarrow^{k_1} \text{str}[s_1] \quad e_2 \Downarrow^{k_2} \text{str}[s_2]}{\text{cat}(e_1; e_2) \Downarrow^{k_1+k_2+1} \text{str}[s_1 \hat{ } s_2]}$$

Similarly for string length

39

Cost Semantics for Expressions

Let Expressions:

$$\frac{[e_1/x]e_2 \Downarrow^{k_2} v_2}{\text{let}(e_1; x.e_2) \Downarrow^{k_2+1} v_2}$$

40

Relation to Structural Semantics

Define $e \mapsto^k e'$ by induction on $k \geq 0$:

- $e \mapsto^0 e$;
- if $e \mapsto e' \mapsto^k e''$, then $e \mapsto^{k+1} e''$.

Theorem 11

For any closed expression e and closed value v of the same type, $e \Downarrow^k v$ iff $e \mapsto^k v$.

41

Abstract and Concrete Costs

Are the costs assigned realistic?

- Unit cost for arithmetic? (Typical assumption.)
- Unit cost for string concatenation? (Typical.)
- Unit cost for substitution? (Less obvious.)

42

Environment Semantics

All semantics so far use substitution to implement binding, but that's not how it's done in practice.

In practice, variable bindings are recorded in some auxiliary data structure.

This can be elegantly modeled using hypothetical judgements, in an approach called **environment semantics**.

- Hypotheses of the form $x \Downarrow v$
- Set Θ of hypotheses, no x occurring twice, called an **environment**.

43

Environment Semantics Judgements

$$\frac{}{\Theta, x \Downarrow v \vdash x \Downarrow v}$$

$$\frac{\Theta \vdash e_1 \Downarrow \text{num}[n_1] \quad \Theta \vdash e_2 \Downarrow \text{num}[n_2]}{\Theta \vdash \text{plus}(e_1; e_2) \Downarrow \text{num}[n_1+n_2]}$$

Similarly for multiplication

$$\frac{\Theta \vdash e_1 \Downarrow \text{str}[s_1] \quad \Theta \vdash e_2 \Downarrow \text{str}[s_2]}{\Theta \vdash \text{cat}(e_1; e_2) \Downarrow \text{str}[s_1 \hat{\ } s_2]}$$

Similarly for string length

$$\frac{\Theta \vdash e_1 \Downarrow v_1 \quad \Theta, x \Downarrow v_1 \vdash e_2 \Downarrow v_2}{\Theta \vdash \text{let}(e_1; x.e_2) \Downarrow v_2}$$

44

Environment Semantics: Example Derivation

An environment semantics derivation of :

$$\Theta \vdash \text{plus}(\text{plus}(\text{num}[1]; \text{num}[2]); \text{num}[4]) \Downarrow \text{num}[7]$$

We abbreviate $\text{plus}(\text{num}[1]; \text{num}[2])$ by $p(1, 2)$, $\text{num}[3]$ by 3, etc.

$$\frac{\frac{\Theta \vdash 1 \Downarrow 1 \quad \Theta \vdash 2 \Downarrow 2}{\Theta \vdash p(1, 2) \Downarrow 3} \quad \Theta \vdash 4 \Downarrow 4}{\Theta \vdash p(p(1, 2), 4) \Downarrow 7}$$

45

Correspondence Between Environment and Evaluation Semantics

Theorem 12

$x_1 \Downarrow v_1, \dots, x_n \Downarrow v_n \vdash e \Downarrow v$ iff $[v_1, \dots, v_n/x_1, \dots, x_n]e \Downarrow v$.

46

Summary

- **Contextual semantics** provides an alternative presentation of structural semantics, emphasizing the next location of evaluation.
- **Evaluation semantics** is a direct specification of evaluation that suppresses intermediate states.
- Evaluation semantics generalizes naturally to a **cost semantics** that accounts for execution time.
- **Environment semantics** models a more realistic treatment of variable binding.

47