CS720

Logical Foundations of Computer Science

Lecture 16: Small-step semantics

Tiago Cogumbreiro

Overview

- Introduction of small-step semantics
- Normalization of terms
- Relationship between small-step and big-step semantics.

Revisiting arithmetic semantics

A language with constants and a plus-operator:

$$t ::= n \mid t \oplus t$$

> How do we represent an evaluation function for a term, say $\mathtt{eval}(t)$?

Revisiting arithmetic semantics

A language with constants and a plus-operator:

$$t ::= n \mid t \oplus t$$

> How do we represent an evaluation function for a term, say $\mathrm{eval}(t)$?

$$\mathrm{eval}(n) = n$$

$$\mathrm{eval}(t_1 \oplus t_2) = \mathrm{eval}(t_2) + \mathrm{eval}(t_2)$$

> How do we represent $\mathrm{eval}(t)$ as a relation $t \Downarrow n$?

Revisiting arithmetic semantics

A language with constants and a plus-operator:

$$t ::= n \mid t \oplus t$$

> How do we represent an evaluation function for a term, say $\mathrm{eval}(t)$?

$$\mathrm{eval}(n) = n$$

$$\mathrm{eval}(t_1 \oplus t_2) = \mathrm{eval}(t_2) + \mathrm{eval}(t_2)$$

> How do we represent $\mathrm{eval}(t)$ as a relation $t \Downarrow n$?

$$\frac{}{n \Downarrow n} \qquad \frac{t_1 \Downarrow n_1 \qquad t_2 \Downarrow n_2}{t_1 \oplus t_2 \Downarrow n_1 + n_2}$$

Small-step operational semantics

The idea is to model computation similar to how a computer (or an abstract machine) would run.

We want to model the **smallest** step of computation (capture each tick the machine does).

In big step semantics, we are only interested in the ***outcome*** of a computation. In small step semantics, we are interested in how the ***execution*** unfolds.

A big-step semantics execution can be encoded as a sequence of multiple steps in small-step semantic.

$$\frac{}{n_1 \oplus n_2 \Rightarrow n_1 + n_2}\,(\text{P-const}) \qquad \frac{t_1 \Rightarrow t_1'}{t_1 \oplus t_2 \Rightarrow t_1' \oplus t_2}\,(\text{P-left})$$

$$\frac{t_2 \Rightarrow t_2'}{n_1 \oplus t_2 \Rightarrow n_1 \oplus t_2'}\,(\text{P-right})$$

Example

Step 1:

$$\dfrac{\dfrac{}{0 \oplus 3 \Rightarrow 3}\,(\text{P-const})}{(0 \oplus 3) \oplus (2 \oplus 4) \Rightarrow 3 \oplus (2 \oplus 4)}\,(\text{P-left})$$

Step 2:

$$\dfrac{\dfrac{}{2 \oplus 4 \Rightarrow 6}\,(\text{P-const})}{3 \oplus (2 \oplus 4) \Rightarrow 3 \oplus 6}\,(\text{P-right})$$

Step 3:

$$\dfrac{}{3 \oplus 6 \Rightarrow 9}\,(\text{P-const})$$

We may just write the short-hand notation:

$$(0 \oplus 3) \oplus (2 \oplus 4) \Rightarrow 3 \oplus (2 \oplus 4) \Rightarrow 3 \oplus 6 \Rightarrow 9$$

Abstracting a binary relation

> Notice how our small-step semantics always only has a unique "output". In such a case, we say that a relation is ***deterministic***. That is, a deterministic relation describes an ***injective function***.

Definition (deterministic relation). If $t_1 \Rightarrow t_2$ and $t_1 \Rightarrow t_2'$, then $t_2 = t_2'$.

UMass Boston

Deterministic relations

Theorem. $\Rightarrow$ is deterministic.

> Can you come up with a way to make $\Rightarrow$ non-deterministic?

Deterministic relations

Theorem. $\Rightarrow$ is deterministic.

> Can you come up with a way to make $\Rightarrow$ non-deterministic?

$$\frac{}{n_1 \oplus n_2 \Rightarrow n_1 + n_2}(\text{P-const}) \qquad \frac{t_1 \Rightarrow t_1'}{t_1 \oplus t_2 \Rightarrow t_1' \oplus t_2}(\text{P-left})$$

$$\frac{t_2 \Rightarrow t_2'}{\color{red}{t_1} \oplus t_2 \Rightarrow \color{red}{t_1} \oplus t_2'}(\text{P-right})$$

Deterministic relations

Theorem. $\Rightarrow$ is deterministic.

> Can you come up with a way to make $\Rightarrow$ non-deterministic?

$$\frac{}{n_1 \oplus n_2 \Rightarrow n_1 + n_2}\text{(P-const)} \qquad \frac{t_1 \Rightarrow t_1'}{t_1 \oplus t_2 \Rightarrow t_1' \oplus t_2}\text{(P-left)}$$

$$\frac{t_2 \Rightarrow t_2'}{t_1 \oplus t_2 \Rightarrow t_1 \oplus t_2'}\text{(P-right)}$$

$$(0 \oplus 3) \oplus (2 \oplus 4) \Rightarrow 3 \oplus (2 \oplus 4)$$
$$(0 \oplus 3) \oplus (2 \oplus 4) \Rightarrow (0 \oplus 3) \oplus 6$$

Small-step semantics as an abstract machine

We can think of the execution of an expression as:

1. The state of the machine is a term t
2. A step of the machine performs an atomic unit of computation (eg, evaluates one addition in a sub-expression)
3. The machine ***halts*** when it cannot perform any more steps.

Here is an inductive definition of value:

$$\frac{}{\mathrm{value}(n)}\,(\text{V-nat})$$

Exercise

Which of these are provable?

- $\mathrm{value}(10)$
- $\mathrm{value}(10 \oplus 2)$
- $\exists n, \mathrm{value}(n \oplus 10)$

Revisiting our small-step semantics

By convention we write a value (a halted state) as v.

$$\frac{}{n_1 \oplus n_2 \Rightarrow n_1 + n_2}\,(\text{P-const}) \qquad \frac{t_1 \Rightarrow t_1'}{t_1 \oplus t_2 \Rightarrow t_1' \oplus t_2}\,(\text{P-right})$$

$$\frac{\text{value}(v_1) \qquad t_2 \Rightarrow t_2'}{v_1 \oplus t_2 \Rightarrow t_1 \oplus t_2'}\,(\text{P-left})$$

Strong Progress

- Is our semantics always able to perform a step?

Strong Progress

- Is our semantics always able to perform a step? No. When a term is a value, then there is no rule that we can apply to perform a step.
- If the term is not a vale, is it always able to perform a step?

Strong Progress

- Is our semantics always able to perform a step? No. When a term is a value, then there is no rule that we can apply to perform a step.
- If the term is not a vale, is it always able to perform a step? Yes.

Theorem (Strong Progress). Given a single-step relation $(\Rightarrow)$ and a notion of value value. Any term t is either a value $\mathrm{value}(t)$ or it reduces $\exists t', t \Rightarrow t'$.

> - A language may not enjoy progress because we "forgot" to write a rule for a given command. Example: extend the grammar to include the minus-operator, but do not update the small-step semantics.
> - In concurrency theory, the notion of progress may capture the notion of deadlock freedom (ie, there is always a task that can perform an action, or all tasks are idle). Thus, many concurrent languages do not enjoy progress.
> - A language may not reduce because there are type-mismatch errors.

Thinking of the state in terms of steps

> If a language enjoys progress, then we can always perform a step until we reach a value.

- Can we perform a step on a value?
- What do we call a term where there are no further steps?

Normal Form

Normal form. A term that cannot perform any step (ie, make any progress).

$$\mathrm{nf}(t) := \neg\exists t', t \Rightarrow t'$$

> In our language, are all values in the normal form? Are all normal form terms a value?

Normal Form

Normal form. A term that cannot perform any step (ie, make any progress).

$$\mathrm{nf}(t) := \neg\exists t', t \Rightarrow t'$$

> In our language, are all values in the normal form? Are all normal form terms a value? **Yes!**

Theorem. $\mathrm{value}(t) \iff \mathrm{nf}(t)$.

This is a non-trivial result, depending on the language:

- One captures the notion of a halted state **syntactically** (a value)
- The other captures the notion of a halted state **semantically** (in terms of $\Rightarrow$)

Multi-Step Reduction

Our goal is to relate big-step and small-step semantics.

$$\frac{}{t \Rightarrow^\star t}(\text{R-refl}) \qquad \frac{t_1 \Rightarrow t_2 \qquad t_2 \Rightarrow^\star t_3}{t_1 \Rightarrow^\star t_3}(\text{R-step})$$

- This family of relations is also known as the (reflexive) **transitive-closure** of a relation.
- The multi-step reduction can be though of describing **all states that can be reached from a given starting state**.

Exercise

Recall the following propositions:

$$(0 \oplus 3) \oplus (2 \oplus 4) \Rightarrow 3 \oplus (2 \oplus 4) \Rightarrow 3 \oplus 6 \Rightarrow 9$$

> $(0 \oplus 3) \oplus (2 \oplus 4)$ reaches which terms?

Exercise

Recall the following propositions:

$$(0 \oplus 3) \oplus (2 \oplus 4) \Rightarrow 3 \oplus (2 \oplus 4) \Rightarrow 3 \oplus 6 \Rightarrow 9$$

$(0 \oplus 3) \oplus (2 \oplus 4)$ reaches which terms?

- $(0 \oplus 3) \oplus (2 \oplus 4) \Rightarrow^\star (0 \oplus 3) \oplus (2 \oplus 4)$
- $(0 \oplus 3) \oplus (2 \oplus 4) \Rightarrow^\star 3 \oplus (2 \oplus 4)$
- $(0 \oplus 3) \oplus (2 \oplus 4) \Rightarrow^\star 3 \oplus 6$
- $(0 \oplus 3) \oplus (2 \oplus 4) \Rightarrow^\star 9$

The normal form of a term

$$t \mathrel{\mathrm{nfof}} t' := t \Rightarrow^\star t' \land \mathrm{nf}(t')$$

> What is the normal form of $(0 \oplus 3) \oplus (2 \oplus 4)$?

The normal form of a term

$$t \; \mathrm{nfof} \; t' := t \Rightarrow^{\star} t' \wedge \mathrm{nf}(t')$$

> What is the normal form of $(0 \oplus 3) \oplus (2 \oplus 4)$?

$$(0 \oplus 3) \oplus (2 \oplus 4) \; \mathrm{nfof} \; 9$$

Definition (normalizing). We say that a relation, say $\Rightarrow$, is normalizing if, and only if, we can always find a normal form of t.

> Is $\Rightarrow$ normalizing?

The normal form of a term

$$t \operatorname{nfof} t' := t \Rightarrow^\star t' \land \operatorname{nf}(t')$$

> What is the normal form of $(0 \oplus 3) \oplus (2 \oplus 4)$?

$$(0 \oplus 3) \oplus (2 \oplus 4) \operatorname{nfof} 9$$

Definition (normalizing). We say that a relation, say $\Rightarrow$, is normalizing if, and only if, we can always find a normal form of t.

> Is $\Rightarrow$ normalizing? **Yes.**

Theorem. $\Rightarrow$ is normalizing.

Normalizing languages

In practice, a normalizing language is one where programs are ***guaranteed*** to terminate, by **design**.

> Do you know any normalizing language?

Normalizing languages

In practice, a normalizing language is one where programs are ***guaranteed*** to terminate, by **design**.

> Do you know any normalizing language?

Yes! Coq definitions are guaranteed to terminate. The Calculus of Constructions, which Coq implements, enjoys a normalizing semantics.

See also Dhall, a configuration programming language, for when you need **controlled** flexibility.

Can we relate small-step semantics

and big-step semantics now?

Relating small-step and big-step semantics

- **Theorem 1.** If $t \Downarrow n$, then $t \Rightarrow^\star n$.
- **Theorem 2.** If $t \ \mathrm{nfof}\ t'$, then $\exists n, t' = n \land t \Downarrow n$.

Suggestion: Regarding Theorem 2, you might want to prove first that if $t \ \mathrm{nfof}\ t'$, then $\exists n, t' = n$. And then show that, if $t \Rightarrow^\star n$, then $t \Downarrow n$.

UMass Boston

Workshop

```
Theorem step_deterministic:
  deterministic step.

Theorem strong_progress : forall t,
  value t \/ (exists t', t ==> t').

Lemma value_is_nf : forall v,
  value v -> normal_form step v.

Lemma nf_is_value : forall t,
  normal_form step t -> value t.

Theorem step_normalizing : (* By induction on [t]. *)
  normalizing step.  (* It is crucial to replace a nf by a value. *)
  (* P t1 t2 ==>* P (C n1) t2 ==>* P (C n1) (C n2) ==>* C (n1 + n2) *)
```

Summary

- Small-step semantics
- Deterministic relations
- Progress
- Normal forms
- Normalizing semantics
- Relating small-step and big-step semantics