Lecture 8: Small-Step Operational Semantics and Type Soundness

Programming Languages (H)

Simon Fowler & Michele Sevegnani

Semester 2, 2024/2025

sli.do 3400724

sli.do 3400724

Overview

- **Last time**
  - Types and Typechecking
- **This lecture:**
  - A refresher on inductive definitions & proof by induction
  - Type soundness for big-step semantics
  - Limitations of big-step semantics
  - Small-step semantics
  - Type soundness for small-step semantics

sli.do 3400724

So, our program typechecks. Why should we care?

- Throughout my lectures so far, I've hammered home the importance of formal definitions
  - Reduced ambiguity, ability to spot corner cases, less verbosity, but most importantly the **ability to do mathematical reasoning**
- Last lecture we looked at typing rules and typecheckers. Intuitively, typechecking rules out a lot of programming errors, which is great.
- But what does this actually **mean** in practice? Which errors? Can typechecking get us any formal guarantees?

sli.do 3400724

Type Soundness, Informally

Core idea: “If a program is well-typed, then it shouldn’t run into any errors when it runs”

Let’s see how we can try to write this using our two judgements…

sli.do 3400724

$$\boxed{M \Downarrow V}$$

Big-step operational semantics

Expression M evaluates to value V

$$\boxed{\Gamma \vdash M : A}$$

Typing rules

Expression M has type A under environment Γ

"If a program is well-typed, then it'll always evaluate down to a value."

sli.do 3400724

$$\boxed{M \Downarrow V}$$

Big-step operational semantics

Expression M evaluates to value V

$$\boxed{\Gamma \vdash M : A}$$

Typing rules

Expression M has type A under environment Γ

If $\cdot \vdash M : A$ then there exists some V such that $M \Downarrow V$ and $\cdot \vdash V : A$.

Proof by Induction

Refresher: Induction Proofs

- Before we start, **I'm not expecting you write induction proofs in the exam**, but induction is important in PL theory / for understanding the concepts
- Proof by induction is a proof technique for proving properties over inductively-defined structures
- Induction allows us to **assume the property holds** for any subexpressions

Natural Numbers

Base case

$0 \in \mathbb{Z}$

Inductive case

$x + 1 \in \mathbb{Z}, \quad \text{if } x \in \mathbb{Z}$

Binary Trees

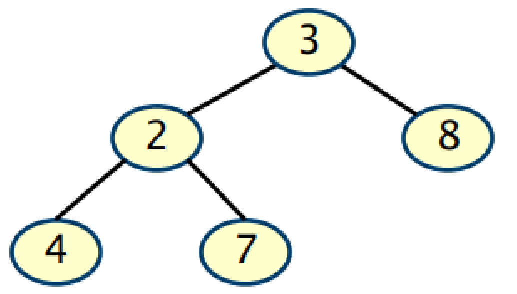
Base case

`leaf` ∈ `BinaryTree`

Inductive case

`node(t1, x, t2)` ∈ `BinaryTree`

if `t1`, `t2` ∈ `BinaryTree` and $x \in \mathbb{Z}$



sli.do 3400724

Structural Induction

- **Structural** induction allows us to do induction over an inductively-defined data structure by reasoning about the cases individually.
- Example: the **size** of a binary tree is recursively defined as follows

```
size(leaf) = 1
size(node(t1, x, t2)) = 1 + size(t1) + size(t2)
```

- We want to prove that **the size of every binary tree is odd**
  - Remember from AF2 that we can write even numbers as 2k (for some k), and odd numbers as `2k + 1`
  - We can use our **induction hypothesis** to reason that the size of each subtree is odd

sli.do 3400724

The size of all binary trees is odd

1) **Begin by stating the proof technique:**

"We proceed by induction on the structure of t"

2) **Next, write out the cases** (in this case, ways of constructing a tree)

3) **Prove each case,** making use of the IH if necessary

4) Once all cases are proved, we have proved that the theorem holds

Case `t = leaf`

`size(leaf) = 1`, which is odd as required

Case `t = node(t1, x, t2)`

`size(node(t1, x, t2)) = 1 + size(t1) + size(t2)`

By the induction hypothesis:

- `size(t1)` is odd, so can be written $2j + 1$
- `size(t2)` is odd, so can be written $2k + 1$

So we have that `1 + size(t1) + size(t2)` can be written as:

$$\begin{aligned} &1 + (2j + 1) + (2k + 1) \\ &= 1 + (2j + 2k + 2) \\ &= 2(j + k + 1) + 1 \end{aligned}$$

which is odd, as required

sli.do 3400724

Our languages are all inductively-defined!

We've defined all of our languages using BNF, but that is in fact an inductive definition, meaning **we can reason about our languages using induction**! Compare the two styles for L_{Arith}…

```
Integers n
Operators ⊙   ::= + | - | * | /
Terms L, M, N ::= n
                | L ⊙ M
```

$n \in \mathbb{Z}$

$n \in$ `Terms`

$L \odot M \in$ `Terms if:`

- $L \in$ `Terms`
- $\odot \in$ `{ +, -, *, /}`
- $M \in$ `Terms`

sli.do 3400724

Rule Induction

- We have defined all of our formal definitions using **inference rules**
- Roughly speaking, **rule induction** allows us to assume that a property holds for all premises of a rule, and we then use this information to show that the property holds for the conclusion
    - If we do this for all rules, then we have proved the property

sli.do 3400724

Rule Induction Example

We can define the set of even positive integers E using the following rules:

$$\frac{}{0 \in E} \qquad \frac{n \in E}{n + 2 \in E}$$

Theorem: If $n \in E$, then $n + n \in E$

Proof: By rule induction on the derivation of $n \in E$

Case 1:

$$\frac{}{0 \in E}$$

0 + 0 = 0, so the property holds immediately

Hutton, G. Programming language semantics: It's easy as 1,2,3. JFP 33(9), 2023.

sli.do 3400724

Rule Induction Example

We can define the set of even positive integers E using the following rules:

$$\frac{}{0 \in E} \qquad \frac{n \in E}{n+2 \in E}$$

Case 2:

$$\frac{n \in E}{n+2 \in E}$$

We want to show that $(n + 2) + (n + 2) \in E$
By the induction hypothesis, $n + n \in E$
Thus we can show:

$$\cfrac{\cfrac{n+n \in E}{(n+n)+2 \in E}}{((n+n)+2)+2 \in E}$$

Rearranging we arrive at $(n + 2) + (n + 2) \in E$, as required.

Proving Big-Step Type Soundness

sli.do 3400724

Proving Type Soundness for L_{Arith}: A Problem

- Our type soundness statement says that if an expression is well-typed (under an empty environment), then it can evaluate to a value of the same type
- However, what about the following case?

$$\boxed{(1 + 2)\ /\ 0 \Downarrow ???}$$

- Since division by zero is undefined, the expression **does not evaluate to a value** and is a **counterexample** (our interpreters would throw an exception)
- There are a few ways of fixing this (adding exceptions into the language, adding a special case with a default value) – but for simplicity we'll just remove division

sli.do 3400724

Proving Type Soundness: Overall Approach

Theorem: If $\cdot \vdash M : A$ then there exists some V such that $M \Downarrow V$ and $\cdot \vdash V : A$.

To prove this, our approach is as follows:

- Proceed by rule induction on the derivation of $\cdot \vdash M : A$
- For each case, we can **assume** that any subexpressions are typed according to the typing rule
- By the induction hypothesis, we can show that any subexpression will evaluate to a value
- Using this information, we'll use the corresponding big-step rule to show that the whole expression evaluates to a value

sli.do 3400724

Proving Type Soundness for $\mathrm{L}_{\mathrm{Arith}}$ without division

Theorem: If $\cdot \vdash M : A$ then there exists some V such that $M \Downarrow V$ and $\cdot \vdash V : A$.

Proof: By induction on the derivation of $\cdot \vdash M : A$.

Case $\cdot \vdash n : \mathrm{Int}$

It follows immediately that $\dfrac{}{n \Downarrow n}$, as required

sli.do 3400724

Proving Type Soundness for L_{Arith} without division

Case $\cdot \vdash M \odot N : \text{Int}$

Assumption: $\dfrac{\cdot \vdash M : \text{Int} \quad \cdot \vdash N : \text{Int}}{\cdot \vdash M \odot N : \text{Int}}$

By the induction hypothesis, there exist V, W such that $M \Downarrow V$ and $N \Downarrow W$ and both $\cdot \vdash V : \text{Int}$ and $\cdot \vdash W : \text{Int}$.

Consider the evaluation rule for binary operators: $\dfrac{M \Downarrow V \quad N \Downarrow W}{M \odot N \Downarrow V \mathbin{\widehat{\odot}} W}$

Since {+, -, *} are total functions on integers, $V \mathbin{\widehat{\odot}} W$ is defined, and therefore by the above evaluation rule, $M \odot N \Downarrow V \mathbin{\widehat{\odot}} W$ with $\cdot \vdash V \mathbin{\widehat{\odot}} W : \text{Int}$ as required.

sli.do 3400724

Limitations of Big-Step Semantics

The big-step type soundness property is very strong:

Theorem: If $\cdot \vdash M : A$ then there exists some V such that $M \Downarrow V$ and $\cdot \vdash V : A$.

This **does not** hold for L_{Rec} as it requires that every L_{Rec} term terminates – which is not the case, for example

```
(rec f(x). f x) true
```

But naturally we want to show some form of type soundness property holds for recursive languages!

The answer: reason about evaluation **step-by-step**

Small-Step Operational Semantics

sli.do 3400724

Small-Step Operational Semantics

$$\boxed{M \Downarrow V}$$

Big-step operational semantics

Expression M evaluates to value V

$$\boxed{M \longrightarrow N}$$

Small-step operational semantics

Expression M takes a reduction step to **expression** N

We write $M \longrightarrow^* N$ to mean M takes zero or more reduction steps to N

sli.do 3400724

Small-Step Rules for L_{Arith}

$$\frac{}{V \odot W \longrightarrow V \mathbin{\widehat{\odot}} W}$$

If both arguments are values, then we can apply the actual operation

$$\frac{M \longrightarrow M'}{M \odot N \longrightarrow M' \odot N}$$

$$\frac{M \longrightarrow M'}{V \odot M \longrightarrow V \odot M'}$$

Otherwise, we need to use the **congruence rules**

The first says that if we have an expression $M \odot N$ and M can take a step to M', then the whole expression can take a step to $M' \odot N$.

The second allows reduction of the second subexpression, if the first is already a value.

The rules enforce a **left-to-right evaluation order**

sli.do 3400724

Evaluating $(1 + (2 * 3)) + (4 * 5)$

$$\dfrac{\dfrac{\overline{2 * 3 \longrightarrow 6}}{1 + (2 * 3) \longrightarrow 1 + 6}}{(1 + (2 * 3)) + (4 * 5) \longrightarrow (1 + 6) + (4 * 5)}$$

First step: We can't evaluate either addition, but we **can** evaluate multiplication

$$\dfrac{\overline{1 + 6 \longrightarrow 7}}{(1 + 6) + (4 * 5) \longrightarrow 7 + (4 * 5)}$$

Second step: We can now evaluate 1 + 6 to 7

sli.do 3400724

Evaluating $\big(1 + (2 * 3)\big) + (4 * 5)$

$$\frac{4 * 5 \longrightarrow 20}{7 + (4 * 5) \longrightarrow 7 + 20}$$

Third step: evaluate the multiplication

$$\frac{}{7 + 20 \longrightarrow 27}$$

Final step: both operands are values, so perform the addition, and we're done

sli.do 3400724

Small-Step Rules for L_{If}

$$\frac{}{\texttt{if}\ \texttt{true}\ \texttt{then}\ M\ \texttt{else}\ N \longrightarrow M}$$

$$\frac{}{\texttt{if}\ \texttt{false}\ \texttt{then}\ M\ \texttt{else}\ N \longrightarrow N}$$

We have **two** main small-step rules for conditionals

The first takes a step to the `then` branch if the test is the value `true`

The second takes a step to the `else` branch if the test is the value `false`

$$\frac{L \rightarrow L'}{\texttt{if}\ L\ \texttt{then}\ M\ \texttt{else}\ N \longrightarrow \texttt{if}\ L'\ \texttt{then}\ M\ \texttt{else}\ N}$$

We need a congruence rule to evaluate the test if it isn't yet a value

sli.do 3400724

Small-Step Rules for $\mathrm{L}_{\mathrm{Rec}}$

$$\frac{}{(\lambda x.\, M)\; V \longrightarrow M\,\{\, V \,/\, x \}}$$

When evaluating a function application where the function is a value, reduce to the function body (with argument substituted for parameter)

$$\frac{}{(\,\mathbf{rec}\; f\,(x).\, M)\; V \longrightarrow M\,\{\,(\mathbf{rec}\; f\,(x).\, M \,/\, f,\; V \,/\, x \}}$$

Recursion is similar, but we also need to substitute in a copy of the function

$$\frac{M \longrightarrow M'}{M\;N \longrightarrow M'\;N} \qquad \frac{M \longrightarrow M'}{V\;M \longrightarrow V\;M'}$$

Finally, the congruence rules allow us to reduce the function first, and then the argument

sli.do 3400724

Equivalence of Big- and Small-Step Semantics (1)

- Big-step and small-step semantics are different ways of saying the same thing, each with advantages and disadvantages
  - **Big-step**: Closer to how we write interpreters, but cannot reason easily about nonterminating expressions
  - **Small-step**: More fine-grained reasoning power, but (usually) further away from how we would implement a language
- My own sub-area, concurrent PL design, exclusively uses small-step semantics as we need to reason about individual communication actions between processes
- Luckily, we can prove that these two styles are equivalent, and get the best of both worlds

sli.do 3400724

Equivalence of Big- and Small-Step Semantics (2)

There's a corresponding small-step reduction sequence for every big-step derivation

If $M \Downarrow V$, then $M \longrightarrow^* V$.

Proof is by rule induction on the derivation of $M \Downarrow V$ (exercise if you're interested!)

There's a corresponding big-step derivation for every small-step reduction sequence from an expression to a value

If $M \longrightarrow^* V$, then $M \Downarrow V$.

Proof is by structural induction on M and inspection of the small-step rules (exercise if you're interested!)

Type Soundness

sli.do 3400724

Type Soundness for Small-Step Semantics

We can now specify a more general type soundness property that we can use on L_{Rec}!

If a process is well typed, then it is either already a value, or it can take a step while staying well typed

Preservation

Reduction doesn't change the result type or introduce type errors

Progress

Well typed processes don't get "stuck"

sli.do 3400724

Type Soundness for Small-Step Semantics

We can now specify a more general type soundness property that we can use on L_{Rec}!

If $\cdot \vdash M : A$, then either M is a value V, or there exists some N such that $M \longrightarrow N$ and $\cdot \vdash N : A$.

Preservation

If $\Gamma \vdash M : A$ and there exists some N such that $M \longrightarrow N$, then $\Gamma \vdash N : A$.

Progress

If $\cdot \vdash M : A$, then either M is a value V, or there exists some N such that $M \longrightarrow N$

Preservation

If $\Gamma \vdash M : A$ and there exists some N such that $M \longrightarrow N$, then $\Gamma \vdash N : A$.

sli.do 3400724

Preservation for $\mathrm{L}_{\mathrm{Arith}}$

It's easiest to prove preservation by induction on the derivation of $M \longrightarrow N$, so we need to consider all of the reduction rules

Preservation

If $\Gamma \vdash M : A$ and there exists some N such that $M \longrightarrow N$, then $\Gamma \vdash N : A$.

sli.do 3400724

$$\frac{}{V \odot W \longrightarrow V \mathbin{\widehat{\odot}} W}$$

Assumption: $\dfrac{\Gamma \vdash V : \mathrm{Int} \quad \Gamma \vdash W : \mathrm{Int}}{\Gamma \vdash V \odot W : \mathrm{Int}}$

As $\odot \in \{+, -, *\}$, we know that $V \mathbin{\widehat{\odot}} W$ is of type Int, as required

$$\frac{M \longrightarrow M'}{M \odot N \longrightarrow M' \odot N}$$

Assumptions: $\dfrac{\Gamma \vdash M : \mathrm{Int} \quad \Gamma \vdash N : \mathrm{Int}}{\Gamma \vdash M \odot N : \mathrm{Int}}$ **and** $M \longrightarrow M'$

By the induction hypothesis, $\Gamma \vdash M' : \mathrm{Int}$, so we can show:

$$\frac{\Gamma \vdash M' : \mathrm{Int} \quad \Gamma \vdash N : \mathrm{Int}}{\Gamma \vdash M' \odot N : \mathrm{Int}}$$ as required.

$$\frac{M \longrightarrow M'}{V \odot M \longrightarrow V \odot M'}$$

Assumptions: $\dfrac{\Gamma \vdash V : \mathrm{Int} \quad \Gamma \vdash M : \mathrm{Int}}{\Gamma \vdash V \odot M : \mathrm{Int}}$ **and** $M \longrightarrow M'$

By the induction hypothesis, $\Gamma \vdash M' : \mathrm{Int}$, so we can show:

$$\frac{\Gamma \vdash V : \mathrm{Int} \quad \Gamma \vdash M' : \mathrm{Int}}{\Gamma \vdash V \odot M' : \mathrm{Int}}$$ as required.

sli.do 3400724

Preservation for L_{If}

$$\frac{}{\textbf{if}\ \texttt{true}\ \textbf{then}\ M\ \textbf{else}\ N \longrightarrow M}$$

(the case for 'false' is similar)

Assumption: $$\frac{\Gamma \vdash \texttt{true} : \mathrm{Bool} \quad \Gamma \vdash M : A \quad \Gamma \vdash N : A}{\Gamma \vdash \textbf{if}\ \texttt{true}\ \textbf{then}\ M\ \textbf{else}\ N : \mathrm{A}}$$

It follows immediately from our assumption that $\Gamma \vdash M : A$ as required.

$$\frac{L \to L'}{\textbf{if}\ L\ \textbf{then}\ M\ \textbf{else}\ N \longrightarrow \textbf{if}\ L'\ \textbf{then}\ M\ \textbf{else}\ N}$$

Assumptions: $$\frac{\Gamma \vdash L : \mathrm{Bool} \quad \Gamma \vdash M : A \quad \Gamma \vdash N : A}{\Gamma \vdash \textbf{if}\ L\ \textbf{then}\ M\ \textbf{else}\ N : \mathrm{A}}$$ **and** $L \longrightarrow L'$

By the induction hypothesis, $\Gamma \vdash L' : \mathrm{Int}$, so we can show:

$$\frac{\Gamma \vdash M : \mathrm{Bool} \quad \Gamma \vdash M : A \quad \Gamma \vdash N : A}{\Gamma \vdash \textbf{if}\ L'\ \textbf{then}\ M\ \textbf{else}\ N : \mathrm{A}}$$ as required.

Progress

If $\cdot \vdash M : A$, then either M is a value V, or there exists some N such that $M \longrightarrow N$

sli.do 3400724

Progress for $\mathrm{L}_{\mathrm{Arith}}$

We need to prove progress by induction on the **typing derivation,** so need to consider all the typing rules

Progress

If $\cdot \vdash M : A$, then either M is a value V, or there exists some N such that $M \longrightarrow N$

sli.do 3400724

$$\frac{}{\Gamma \vdash n : \mathrm{Int}}$$

This case follows immediately, since n is already a value.

$$\frac{\Gamma \vdash M : \mathrm{Int} \quad \Gamma \vdash N : \mathrm{Int}}{\Gamma \vdash M \odot N : \mathrm{Int}}$$

By the induction hypothesis:

- either M is a value, or there exists some N such that $M \longrightarrow M'$
- either N is a value, or there exists some N such that $N \longrightarrow N'$.

So we have three cases we need to consider:

- $M \odot N$, where $M \longrightarrow M'$ -- so we can reduce by rule 2
- $V \odot N$, where $N \longrightarrow N'$ -- so we can reduce by rule 3
- $V \odot W$, so we can reduce by rule 1

1
$$\frac{}{V \odot W \longrightarrow V \mathbin{\widehat{\odot}} W}$$

2
$$\frac{M \longrightarrow M'}{M \odot N \longrightarrow M' \odot N}$$

3
$$\frac{M \longrightarrow M'}{V \odot M \longrightarrow V \odot M'}$$

sli.do 3400724

Progress for L_{If}

(Omitting the value cases as they are similar to integer literals)

$$\frac{\Gamma \vdash L : \mathrm{Bool} \quad \Gamma \vdash M : A \quad \Gamma \vdash N : A}{\Gamma \vdash \mathbf{if}\ L\ \mathbf{then}\ M\ \mathbf{else}\ N : \mathrm{A}}$$

By the IH, we know that either L is a value, or $L \longrightarrow L'$

If L is a value V, then since $\cdot \vdash V : \mathrm{Bool}$ it must be the case that either V = true or V = false and we can reduce by (1) or (2)

If $L \longrightarrow L'$ then we can reduce by (3)

1 $$\frac{}{\mathbf{if}\ \mathtt{true}\ \mathbf{then}\ M\ \mathbf{else}\ N \longrightarrow M}$$

2 $$\frac{}{\mathbf{if}\ \mathtt{false}\ \mathbf{then}\ M\ \mathbf{else}\ N \longrightarrow N}$$

3 $$\frac{L \rightarrow L'}{\mathbf{if}\ L\ \mathbf{then}\ M\ \mathbf{else}\ N \longrightarrow \mathbf{if}\ L'\ \mathbf{then}\ M\ \mathbf{else}\ N}$$

sli.do 3400724

Type Soundness for L_{Rec} (non-examinable)

- The proofs for L_{Lam} and L_{Rec} are similar
- The main difference is that we need a **substitution lemma** that shows that an expression remains well typed after substitution

If $\Gamma, x: A \vdash M : B$ and $\Gamma \vdash V : A$, then $\Gamma \vdash M\{V/x\}$

- I will add the full proof to Moodle for those who are interested

sli.do 3400724

Conclusion

- **This lecture:**
  - Lots of induction
  - Type soundness for big-step semantics
  - Small-step semantics
  - Type soundness via preservation and progress
- (Phew! You'll be happy to know that that's the most mathematical part of the course done – no more induction)
- **Want to write your own typechecker? Join me in the lab this afternoon!**