CS 476 – Programming Language Design

William Mansky
We’ve used inference rules and proof trees to describe the behavior of programs.

\[(1 + 2) + (3 + 4) : \text{int} \quad \text{if } 1 + 2 = 3 \text{ then } 2 \times 2 \text{ else } 7 \downarrow 4\]

We can also use them to prove properties about the language itself!

- If a program is has a type, then its result has the same type.
- Big-step and small-step semantics allow the same behaviors.
- The interpreter correctly implements the language semantics.
Programming Language Metatheory

• We can also use them to prove properties about the language itself!
  — If a program has a type, then its result has the same type
  — Big-step and small-step semantics allow the same behaviors
  — The interpreter correctly implements the language semantics

• The properties of the different systems that define the language are called its metatheory

• Metatheory is (part of) why we bother with all these systems!
  — We use a type system because it guarantees something about the execution of well-typed programs.
  — We write semantics because it gives us a way of telling whether an interpreter is correct.
Metatheory #1: Type Safety

• A type system is *safe* if well-typed programs don’t get stuck
  ― Stuck program = runtime error

• Progress: if $e$ is well-typed, then either $e$ is a value, or there is
  an $e'$ such that $e \rightarrow e'$

• Preservation: if $e : \tau$ and $e \rightarrow e'$, then $e' : \tau$

• Safety = progress + preservation

• This also means that if $e : \tau$ and $e \rightarrow \cdots \rightarrow v$, then $v : \tau$
Proving Progress: Rule Induction

• Progress: if $e$ is well-typed, then either $e$ is a value, or there is an $e'$ such that $e \rightarrow e'$
Proving Progress: Rule Induction

• Progress: if $e : \tau$ for any $\tau$, then either $e$ is a value, or there is an $e'$ such that $e \rightarrow e'$

• Proof: by induction on the proof that $e : \tau$. 

Expressions: Types

- **Types**: int, bool

- **Rules**:
  
  - \((n \text{ is a number})\)
    
    \[
    \frac{n : \text{int}}{n}
    \]

  - \((b \text{ is a boolean})\)
    
    \[
    \frac{b : \text{bool}}{b}
    \]

  - \(e_1 : \tau \quad e_2 : \tau\)
    
    \[
    \frac{e_1 = e_2 : \text{bool}}{e_1 = e_2}
    \]

  - \(e_1 : \text{int} \quad e_2 : \text{int}\)
    
    \[
    \frac{e_1 + e_2 : \text{int}}{e_1 + e_2}
    \]

  - \(e_1 : \text{bool} \quad e_2 : \text{bool}\)
    
    \[
    \frac{e_1 \text{ and } e_2 : \text{bool}}{e_1 \text{ and } e_2}
    \]

  - \(e : \text{bool} \quad e_1 : \tau \quad e_2 : \tau\)
    
    \[
    \frac{\text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}{\text{if } e \text{ then } e_1 \text{ else } e_2}
    \]
Proving Progress: Rule Induction

• Progress: if \( e : \tau \) for any \( \tau \), then either \( e \) is a value, or there is an \( e' \) such that \( e \rightarrow e' \)

• Proof: by induction on the proof that \( e : \tau \).

\[
\begin{array}{c}
\vdots \\
\hline
e_1 : \tau_1 \\
\hline
\hline
\vdots \\
\hline
\hline\end{array}
\]

\[
\begin{array}{c}
\vdots \\
\hline
e_2 : \tau_2 \\
\hline
\hline
\vdots \\
\hline
\hline\end{array}
\]

\[
e : \tau
\]

• Numerical induction:
  — We want to prove that for all \( n \), \( P(n) \) holds.
  — Base case: show that \( P(0) \) holds.
  — Inductive case: assume that \( P(k) \) holds for some \( k \), and show that \( P(k+1) \) holds.
Proving Progress: Rule Induction

• Progress: if $e : \tau$ for any $\tau$, then either $e$ is a value, or there is an $e'$ such that $e \rightarrow e'$

• Proof: by induction on the proof that $e : \tau$.

\[
\begin{array}{c c c}
\ldots & \ldots \\
\hline
e_1 : \tau_1 & e_2 : \tau_2 \\
\hline
\end{array}
\]

• Rule induction:
  — We want to prove that if $e : \tau$ then $P(e)$ holds.
  — Base case: show that if $e : \tau$ is proved immediately, then $P(e)$ holds.
  — Inductive case: if $e : \tau$ is proved by a rule, assume that $P(e_i)$ holds for each $e_i$ in the premises, and show that $P(e)$ holds.
Proving Progress: Base Cases

• Progress: if $e : \tau$ for any $\tau$, then either $e$ is a value, or there is an $e'$ such that $e \rightarrow e'$

\[
\frac{(n \text{ is a number})}{n : \text{int}} \quad e \text{ is a value!}
\]

\[
\frac{(b \text{ is a boolean})}{b : \text{bool}}
\]
Proving Progress: Inductive Cases

• Progress: if $e : \tau$ for any $\tau$, then either $e$ is a value, or there is an $e'$ such that $e \rightarrow e'$

\[
\begin{align*}
\frac{e_1 : \text{int} \quad e_2 : \text{int}}{e_1 + e_2 : \text{int}}
\end{align*}
\]

• Assume:
  — Either $e_1$ is a value, or $e_1 \rightarrow e_1'$
  — Either $e_2$ is a value, or $e_2 \rightarrow e_2'$

\[
\begin{align*}
\frac{e_1 \rightarrow e_1'}{e_1 + e_2 \rightarrow e_1' + e_2} \\
&\frac{e_2 \rightarrow e_2'}{e_1 + e_2 \rightarrow e_1 + e_2'} \\
&\frac{(v_1 + v_2 = v)}{v_1 + v_2 \rightarrow v}
\end{align*}
\]
Proving Progress: Inductive Cases

• Progress: if $e : \tau$ for any $\tau$, then either $e$ is a value, or there is an $e'$ such that $e \rightarrow e'$

$$
\begin{align*}
\text{if } e \text{ then } e_1 \text{ else } e_2 : \tau & \quad \text{if } e \rightarrow e' \\
\text{if true then } e_1 \text{ else } e_2 & \rightarrow e_1 \\
\text{if false then } e_1 \text{ else } e_2 & \rightarrow e_2
\end{align*}
$$

• Assume:
  — Either $e$ is a value, or $e \rightarrow e'$
  — Either $e_1$ is a value, or $e_1 \rightarrow e'_1$
  — Either $e_2$ is a value, or $e_2 \rightarrow e'_2$
Proving Progress: Rule Induction

• Progress: if $e : \tau$ for any $\tau$, then either $e$ is a value, or there is an $e'$ such that $e \rightarrow e'$

• Proof: by induction on the proof that $e : \tau$. 
Proving Preservation

• Preservation: if $e : \tau$ and $e \rightarrow e'$, then $e' : \tau$
• Proof: by induction on the proof that $e \rightarrow e'$. 
Proving Preservation: Base Cases

• Preservation: if \( e : \tau \) and \( e \rightarrow e' \), then \( e' : \tau \)

\[
\begin{align*}
(v_1 + v_2 = v) \\
v_1 + v_2 \rightarrow v
\end{align*}
\]

\( v_1 + v_2 : \tau \), so \( \tau \) must be \( \text{int} \)

\( v \) is a number, so \( v : \text{int} \)

if true then \( e_1 \) else \( e_2 \) \( \rightarrow e_1 \)

if true then \( e_1 \) else \( e_2 : \tau \),
so \( e_1 : \tau \) and \( e_2 : \tau \)

if false then \( e_1 \) else \( e_2 \) \( \rightarrow e_2 \)
Proving Preservation: Inductive Cases

• Preservation: if $e : \tau$ and $e \rightarrow e'$, then $e' : \tau$

\[
\frac{e_1 \rightarrow e_1'}{e_1 + e_2 \rightarrow e_1' + e_2}
\]

• $e_1 + e_2 : \tau$, so $\tau$ is int and $e_1 : \text{int}$, $e_2 : \text{int}$
• From inductive hypothesis, $e_1' : \text{int}$
• So by the typing rule for $+$, $e_1' + e_2 : \text{int}$
Expressions: Type Safety

• Progress: if $e : \tau$ for any $\tau$, then either $e$ is a value, or there is an $e'$ such that $e \rightarrow e'$

• Proof: by induction on the proof that $e : \tau$.

• Preservation: if $e : \tau$ and $e \rightarrow e'$, then $e' : \tau$

• Proof: by induction on the proof that $e \rightarrow e'$.

• So the expression language is type-safe!
Small-Step vs. Big-Step

Small-Step

• Includes intermediate states
• Allows control of evaluation order
• Can count the number of steps

Why not both?

Big-Step

• Doesn’t need structural rules
• Turns into a more direct interpreter
• Sometimes internal steps don’t matter
Big- and Small-Step Equivalence

• Equivalence: \( e \Downarrow v \) if and only if \( e \rightarrow \cdots \rightarrow v \)

• Theorem 1: If \( e \Downarrow v \), then \( e \rightarrow \cdots \rightarrow v \)
• Proof: By induction on the proof that \( e \Downarrow v \).

• Theorem 2: If \( e \rightarrow \cdots \rightarrow v \), then \( e \Downarrow v \)
Big to Small: Base Cases

• Theorem 1: If $e \downarrow v$, then $e \rightarrow \cdots \rightarrow v$

\[
\frac{(i \text{ is a number})}{i \downarrow i} \quad i \rightarrow ?
\]

\[
\frac{(b \text{ is a boolean})}{b \downarrow b} \quad b \rightarrow ?
\]

• $i$ and $b$ are values, so they reach a value in 0 steps.
Big to Small: Inductive Cases

\[
e_1 \downarrow i_1 \quad e_2 \downarrow i_2 \quad (i = i_1 + i_2) \\
\hline
\]
\[
e_1 + e_2 \downarrow i
\]

• Assume \(e_1 \to \cdots \to i_1, e_2 \to \cdots \to i_2\)
• Either \(e_1\) is already \(i_1\), or
• We can use the rule \(e_1 \to e'_1\)

\[
e_1 + e_2 \to e'_1 + e_2
\]

• So \(e_1 + e_2 \to \cdots \to i_1 + e_2\)
Big to Small: Inductive Cases

\[
\begin{align*}
  e_1 \downarrow i_1 & \quad e_2 \downarrow i_2 \quad (i = i_1 + i_2) \\
  e_1 + e_2 \downarrow i
\end{align*}
\]

• Assume \( e_1 \to \cdots \to i_1, \ e_2 \to \cdots \to i_2 \)
• Either \( e_2 \) is already \( i_2 \), or
• We can use the rule \( e_2 \to e'_2 \)
  \[
  \frac{e_2 \to e'_2}{e_1 + e_2 \to e_1 + e'_2}
  \]
• So \( i_1 + e_2 \to \cdots \to i_1 + i_2 \)
Big to Small: Inductive Cases

\[
e_1 \Downarrow i_1 \quad e_2 \Downarrow i_2 \quad (i = i_1 + i_2) \\
\hline
\text{e}_1 + \text{e}_2 \Downarrow i
\]

• Assume \( e_1 \rightarrow \cdots \rightarrow i_1, e_2 \rightarrow \cdots \rightarrow i_2 \)
• We showed that \( e_1 + e_2 \rightarrow \cdots \rightarrow i_1 + e_2 \rightarrow \cdots \rightarrow i_1 + i_2 \)
• We can use the rule \( (v_1 + v_2 = v) \)
  \[
  \frac{v_1 + v_2 \rightarrow v}{v_1 + v_2 \rightarrow v}
  \]
• So \( i_1 + i_2 \rightarrow i \)
Big to Small: Inductive Cases

\[ e \downarrow \text{true} \quad e_1 \downarrow v \]

\[
\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v
\]

• Assume \( e \rightarrow \cdots \rightarrow \text{true} \), \( e_1 \rightarrow \cdots \rightarrow v \)

• Either \( e \) is already \( \text{true} \), or

• We can use the rule

\[
e \rightarrow e'
\]

\[
\text{if } e \text{ then } e_1 \text{ else } e_2 \rightarrow \text{if } e' \text{ then } e_1 \text{ else } e_2
\]

• So if \( e \) then \( e_1 \) else \( e_2 \) \( \rightarrow \cdots \rightarrow \) if true then \( e_1 \) else \( e_2 \)
Big to Small: Inductive Cases

\[ e \downarrow \text{true} \quad e_1 \downarrow v \]
\[ \text{if } e \text{ then } e_1 \text{ else } e_2 \downarrow v \]

• Assume \( e \rightarrow \cdots \rightarrow \text{true}, \ e_1 \rightarrow \cdots \rightarrow v \)

• We showed that
  \[ \text{if } e \text{ then } e_1 \text{ else } e_2 \rightarrow \cdots \rightarrow \text{if true then } e_1 \text{ else } e_2 \]

• We can use the rule \( \text{if true then } e_1 \text{ else } e_2 \rightarrow e_1 \)

• So if \( e \text{ then } e_1 \text{ else } e_2 \rightarrow \cdots \rightarrow e_1 \rightarrow \cdots \rightarrow v, \ QED! \]
Big- and Small-Step Equivalence

• Equivalence: $e \Downarrow v$ if and only if $e \rightarrow \cdots \rightarrow v$

• Theorem 1: If $e \Downarrow v$, then $e \rightarrow \cdots \rightarrow v$
• Proof: By induction on the proof that $e \Downarrow v$.

• Theorem 2: If $e \rightarrow \cdots \rightarrow v$, then $e \Downarrow v$
• Proof: By induction on the number of steps in $e \rightarrow \cdots \rightarrow v$. 
Small to Big

• Theorem 2: If \( e \rightarrow \cdots \rightarrow v \), then \( e \Downarrow v \).

• Proof: By induction on the number of steps in \( e \rightarrow \cdots \rightarrow v \).
  — Zero steps: \( e \) is already a value, so it’s either \( i \Downarrow i \) or \( b \Downarrow b \).
  — \( N+1 \) steps: \( e \rightarrow e' \rightarrow \cdots \rightarrow v \) and \( e' \Downarrow v \), so we look at the cases of the proof structure of \( e \rightarrow e' \) and find corresponding big-step rules.

• Conclusion: big- and small-step semantics are equivalent!

• So by type safety, if \( e : \tau \) and \( e \Downarrow v \), then \( v : \tau \).
Class Project (for 4-credit students only)

• Work alone or in group of 2-3

• Project ideas:
  — Read a paper about an interesting language/feature and write semantics or interpreter for it
  — Define and implement a stronger/more interesting type system for an existing language
  — Prove equivalence of two semantics for a language, by hand or mechanically
  — Design and implement a small domain-specific language

• Project proposal due 10/12, but email me anytime!