Programming Language Metatheory
(Exercise 17)

COMP 105—Programming Languages
Tufts University

January 26, 2015
Inference rules compose to form proofs

Use rules to create **syntactic proofs**

Valid proof is a **derivation** $\mathcal{D}$

Compositionality again:

- Rule with no premises above the line?  
  A derivation by itself

- Rule with premises?  
  **Build derivations from smaller derivations**
A proof about evaluation is “theory”

Such a proof is a derivation:

\[
\begin{align*}
\text{Literal} & \quad \text{Apply} \quad \text{Add} \\
\langle \text{Literal}(2), \xi, \phi, \rho \rangle \Downarrow \langle 2, \xi, \phi, \rho \rangle & \quad \langle \text{Literal}(3), \xi, \phi, \rho \rangle \Downarrow \langle 3, \xi, \phi, \rho \rangle \\
\langle \text{Apply}(+, \text{Literal}(2), \text{Literal}(3)), \xi, \phi, \rho \rangle \Downarrow \langle 5, \xi, \phi, \rho \rangle
\end{align*}
\]

The proof is a syntactic object (data structure)
A proof about derivations is “metatheory”

Reason about entire classes of evaluations—or even all evaluations.

Examples:

- Evaluating an expression will never add a new global variable.
- OK to put environments on a stack
- Interactive browser does not leak space (POPL’12)
Proof by structural induction

How to prove a claim about any derivation?

- Big idea: structural induction

(Just like a recursive function.)

A derivation is a recursive data structure:

- Ends in application of a single rule
- Contains zero or more subderivations (one per premise)

The proof is a math-class proof (induction argument)
Proof by structural induction

How to prove a claim about any derivation?

- Big idea: structural induction

(Just like a recursive function.)

A derivation is a recursive data structure:

- Ends in application of a single rule
- Contains zero or more subderivations (one per premise)

The proof is a math-class proof (induction argument)
Proof by structural induction

How to prove a claim about any derivation?

- Big idea: structural induction

(Just like a recursive function.)

A derivation is a recursive data structure:

- Ends in application of a single rule
- Contains zero or more subderivations (one per premise)

The proof is a math-class proof (induction argument)
Two induction principles

Proof of induction hypothesis \( IH \) for all derivations \( \mathcal{D} \):

- **Case analysis**: one case per rule that can end a \( \mathcal{D} \)
- **For each** \( \text{Rule} \),
  - Consider all derivations \( \mathcal{D} \) ending in \( \text{Rule} \)
  - Each premise has subderivation \( \mathcal{D}_i \)
  - Assume \( IH(\mathcal{D}_i) \) for each \( i \)
  - Prove \( IH(\mathcal{D}) \)

Conclude \( \forall \mathcal{D} : \mathcal{D} \text{ valid} : IH(\mathcal{D}) \)

Proof of \( IH \) for all natural numbers:

- **Case analysis** over this proof system:
  
  - **ZERO**
    
    \[
    \begin{align*}
    0 & \in \mathbb{N} \\
    \end{align*}
    \]
  
  - **SUCC**
    
    \[
    \begin{align*}
    n & \in \mathbb{N} \\
    n + 1 & \in \mathbb{N} \\
    \end{align*}
    \]

  - For **ZERO**, prove \( IH(0) \)
  
  - For **SUCC**, assume \( IH(n) \), prove \( IH(n + 1) \)

Conclude \( \forall n \in \mathbb{N} : IH(n) \)
Two induction principles

Proof of induction hypothesis $IH$ for all derivations $\mathcal{D}$:

- **Case analysis**: one case per rule that can end a $\mathcal{D}$
- For each $\text{Rule}$,
  - Consider all derivations $\mathcal{D}$ ending in $\text{Rule}$
  - Each premise has subderivation $\mathcal{D}_i$
  - Assume $IH(\mathcal{D}_i)$ for each $i$
  - Prove $IH(\mathcal{D})$

Conclude $\forall \mathcal{D} : \mathcal{D}$ valid : $IH(\mathcal{D})$

Proof of $IH$ for all natural numbers:

- **Case analysis** over this proof system:
  - $\text{ZERO}$  
    \[
    0 \in \mathbb{N}
    \]
  - $\text{Succ}$  
    \[
    n \in \mathbb{N} \quad \Rightarrow \quad n + 1 \in \mathbb{N}
    \]

  - For $\text{ZERO}$, prove $IH(0)$
  - For $\text{Succ}$, assume $IH(n)$, prove $IH(n + 1)$

Conclude $\forall n \in \mathbb{N} : IH(n)$
### Impcore inference rules

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Rule Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literal</strong></td>
<td>$\langle \text{literal}(v), \xi, \phi, \rho \rangle \downarrow \langle v, \xi, \phi, \rho \rangle$</td>
</tr>
<tr>
<td><strong>FormalVar</strong></td>
<td>$x \in \text{dom} \rho$ $\therefore \langle \text{var}(x), \xi, \phi, \rho \rangle \downarrow \langle \rho(x), \xi, \phi, \rho \rangle$</td>
</tr>
<tr>
<td><strong>GlobalVar</strong></td>
<td>$x \notin \text{dom} \rho$ $x \in \text{dom} \xi$ $\therefore \langle \text{var}(x), \xi, \phi, \rho \rangle \downarrow \langle \xi(x), \xi, \phi, \rho \rangle$</td>
</tr>
<tr>
<td><strong>FormalAssign</strong></td>
<td>$x \in \text{dom} \rho$ $\therefore \langle \text{set}(x, e), \xi, \phi, \rho \rangle \downarrow \langle v, \xi', \phi', \rho' \rangle$ ${x \mapsto v}$</td>
</tr>
<tr>
<td><strong>GlobalAssign</strong></td>
<td>$x \notin \text{dom} \rho$ $x \in \text{dom} \xi$ $\therefore \langle \text{set}(x, e), \xi, \phi, \rho \rangle \downarrow \langle v, \xi', \phi, \rho' \rangle$ ${x \mapsto v}$</td>
</tr>
</tbody>
</table>

**ApplyPrint**

$\phi(f) = \text{PRIMITIVE(print)}$  
$\langle e, \xi, \phi, \rho \rangle \downarrow \langle v, \xi', \phi, \rho' \rangle$  
$\langle \text{apply}(f, e), \xi, \phi, \rho \rangle \downarrow \langle v, \xi', \phi, \rho' \rangle$ while printing $v$

**ApplyAdd**

$\phi(f) = \text{PRIMITIVE(+)}$  
$\langle e_1, \xi_0, \phi, \rho_0 \rangle \downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle$  
$\langle e_2, \xi_1, \phi, \rho_1 \rangle \downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle$  
$\langle \text{apply}(f, e_1, e_2), \xi_0, \phi, \rho_0 \rangle \downarrow \langle v_1 + v_2, \xi_2, \phi, \rho_2 \rangle$

**ApplyUser**

$\phi(f) = \text{USER(}\langle x_1, \ldots, x_n \rangle, e)$  
$x_1, \ldots, x_n$ all distinct  
$\langle e_1, \xi_0, \phi, \rho_0 \rangle \downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle$  
$\langle e_2, \xi_1, \phi, \rho_1 \rangle \downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle$  
$\vdots$  
$\langle e_n, \xi_{n-1}, \phi, \rho_{n-1} \rangle \downarrow \langle v_n, \xi_n, \phi, \rho_n \rangle$  
$\langle e, \xi_n, \phi, \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\} \rangle \downarrow \langle v, \xi', \phi, \rho' \rangle$  
$\langle \text{apply}(f, e_1, \ldots, e_n), \xi_0, \phi, \rho_0 \rangle \downarrow \langle v, \xi', \phi, \rho_n \rangle$
Zoom in on ApplyUser

\textbf{ApplyUser}

\[ \phi(f) = \text{USER}(\langle x_1, \ldots, x_n \rangle, e) \]

\[ x_1, \ldots, x_n \text{ all distinct} \]

\[ \langle e_1, \xi_0, \phi, \rho_0 \rangle \downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle \]

\[ \langle e_2, \xi_1, \phi, \rho_1 \rangle \downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle \]

\[ \vdots \]

\[ \langle e_n, \xi_{n-1}, \phi, \rho_{n-1} \rangle \downarrow \langle v_n, \xi_n, \phi, \rho_n \rangle \]

\[ \langle e, \xi_n, \phi, \{ x_1 \mapsto v_1, \ldots, x_n \mapsto v_n \} \rangle \downarrow \langle v, \xi', \phi, \rho' \rangle \]

\[ \langle \text{APPLY}(f, e_1, \ldots, e_n), \xi_0, \phi, \rho_0 \rangle \downarrow \langle v, \xi', \phi, \rho_n \rangle \]
Impcore inference rules cont’d

**IFTrue**
\[
\langle e_1, \xi, \phi, \rho \rangle \downarrow \langle v_1, \xi', \phi, \rho' \rangle \quad v_1 \neq 0 \quad \langle e_2, \xi', \phi, \rho' \rangle \downarrow \langle v_2, \xi'', \phi, \rho'' \rangle
\]
\[
\langle \text{IF}(e_1, e_2, e_3), \xi, \phi, \rho \rangle \downarrow \langle v_2, \xi'', \phi, \rho'' \rangle
\]

**IFFalse**
\[
\langle e_1, \xi, \phi, \rho \rangle \downarrow \langle v_1, \xi', \phi, \rho' \rangle \quad v_1 = 0 \quad \langle e_3, \xi', \phi, \rho' \rangle \downarrow \langle v_3, \xi'', \phi, \rho'' \rangle
\]
\[
\langle \text{IF}(e_1, e_2, e_3), \xi, \phi, \rho \rangle \downarrow \langle v_3, \xi'', \phi, \rho'' \rangle
\]

**WHILEIterate**
\[
\langle e_1, \xi, \phi, \rho \rangle \downarrow \langle v_1, \xi', \phi, \rho' \rangle \quad v_1 \neq 0
\]
\[
\langle e_2, \xi', \phi, \rho' \rangle \downarrow \langle v_2, \xi'', \phi, \rho'' \rangle \quad \langle \text{WHILE}(e_1, e_2), \xi'', \phi, \rho'' \rangle \downarrow \langle v_3, \xi''', \phi, \rho''' \rangle
\]
\[
\langle \text{WHILE}(e_1, e_2), \xi, \phi, \rho \rangle \downarrow \langle v_3, \xi''', \phi, \rho''' \rangle
\]

**WHILEEnd**
\[
\langle e_1, \xi, \phi, \rho \rangle \downarrow \langle v_1, \xi', \phi, \rho' \rangle \quad v_1 = 0
\]
\[
\langle \text{WHILE}(e_1, e_2), \xi, \phi, \rho \rangle \downarrow \langle 0, \xi', \phi, \rho' \rangle
\]
Two example problems

To prove: evaluation of expression $e$ does not change $\text{dom } \xi$

To prove: if $e$ has no $\text{SET}$, then evaluation of $e$ does not change $\xi$
Metatheory Summary

Theory involves proofs about individual derivations (the evaluation of single programs).

Metatheory involves proofs about collections of derivations (the evaluation of entire classes of programs or even all programs).

Theoretic proofs are derivations.

Metatheoretic proofs proceed by induction over derivations.