Preliminaries

We need a formal (meta-) language to describe and reason about properties of programming languages!

We need to specify things like the

- grammar
- scoping rules (static semantics)
- type system (static semantics)
- evaluation strategy (dynamic semantics)

of a language

Fortunately, it turns out that natural deduction (inference rules) can be used for all of these tasks!

Judgements and Inference Rules

Judgements: A judgement states that a certain property holds for an object, e.g.,

- $3 + 4 \times 5$ is a valid arithmetic expression
- the string "aba" is a palindrome
- 0.43423 is a floating point value

We write

$s \ A$

to express that a property $A$ holds for the object $s$ (or $s$ is in the set $A$)

*or a relationship between objects, but more on that later
For example:
- 6 is even, or 6 is an element of the set of even numbers
- $3 + 4 \times 5$ is a syntactically correct expression, or $3 + 4 \times 5$ is an element of the set containing all syntactically correct expressions
- $0.4323 \triangleq \text{float}$

Note:
- Similar to predicates in Predicate Logic
- The postfix notation will prove to be convenient later on, when the objects become fairly big

Inference Rules

are rules of the form:
If $J_1$, and $J_2$, and ... and $J_n$ are inferrable, then $J$ is inferrable

Examples:
- Axiom "0 is a natural number":
  
  Slide 7

  If $n$ is a natural number, then $n + 1$ is a natural number as well:
  
  $\frac{n \text{ nat}}{n + 1 \text{ nat}}$

Even and odd numbers:
- Axiom: "0 is even"

  Slide 8

  If $x$ is even, then $x + 1$ is odd
  
  $\frac{x \text{ even}}{x + 1 \text{ odd}}$

  If $x$ is even and $y$ is odd, then $x + y$ is odd
  
  $\frac{x \text{ even} \quad y \text{ odd}}{x + y \text{ odd}}$
NATURAL DEDUCTION PROOFS

Inference rules which define a set can be used to show that a certain object is in the set.

1. Find rule where conclusion matches statement which you want to prove
2. Show that preconditions can be inferred
3. Finished if rule is an axiom

HOW CAN GRAMMARS BE EXPRESSED USING INFERENCE RULES?

Example:

A language $M$ of properly matched parenthesis:

$M = \{ \epsilon, (), (())(), (())(), \ldots (())(), (())(), (())(), (())(), \ldots \}$

How can this language be defined using natural language?

1. The empty string (denoted by $\epsilon$) is in $M$.
2. If $s_1$ and $s_2$ are in $M$, then $s_1s_2$ is in $M$.
3. If $s$ is in $M$, then $(s)$ is in $M$.

How can this language be defined using EBNF?

$M \rightarrow \epsilon | MM | (M)$

HOW CAN GRAMMARS BE EXPRESSED USING INFERENCE RULES?

(1) The empty string (denoted by $\epsilon$) is in $M$:

$\epsilon \rightarrow M$

(2) If $s_1$ and $s_2$ are in $M$, then $s_1s_2$ is in $M$:

$\frac{s_1 \rightarrow M \quad s_2 \rightarrow M}{s_1s_2 \rightarrow M}$

(3) If $s$ is in $M$, then $(s)$ is in $M$:

$\frac{s \rightarrow M}{(s) \rightarrow M}$
Can we show that \( \texttt{()}() \) is in \( M \)?

\[
\begin{align*}
(1) & \quad \texttt{s} M \\
(2) & \quad \texttt{()} M \\
(3) & \quad \epsilon M
\end{align*}
\]

\[ \texttt{()() } M \]

What happens if we start with rule (3)?

Would anything change if we added the rule:

\[
\begin{align*}
(1) & \quad \texttt{s} M \\
(2) & \quad \texttt{()} M \\
(3) & \quad \epsilon M
\end{align*}
\]

\[ \texttt{()() } M \]

\[
\frac{s \ M}{((s)) \ M}
\]

is derivable from previous rules

\[
\frac{() s \ M}{s \ M}
\]

is admissible (since it does not change the language, but not derivable!)

\[
\frac{(s) \ M}{s \ M}
\]

is not admissible, as it introduces new strings into the set

\[
\frac{0 \ \text{nat}}{n \ \text{nat}} \quad \frac{n + 1 \ \text{nat}}{n \ \text{nat}}
\]

\( \text{nat} \) is the set of natural numbers \( \mathbb{N} \)

But why not the set of real numbers? \( 0 \in \mathbb{R}, n \in \mathbb{R} \implies n + 1 \in \mathbb{R} \)

\( \mathbb{N} \) is the smallest set that is consistent with the rules.

Why the smallest set?

- Objective: no junk. Only what must be in \( X \) shall be in \( X \).
- Gives rise to a nice proof principle (rule induction)
- Alternative (greatest set) occasionally also useful: coinduction
Formally:

Rules \( \frac{a_1 X \ldots a_n X}{a X} \) with \( a_1, \ldots, a_n, a \in A \)

define set \( X \subseteq A \)

Formally: set of rules \( R \subseteq A \times A \) \( (R, X \) possibly infinite) \n
Slide 17

Applying rules \( R \) to a set \( B \):

\[ \hat{R}(B) \equiv \{ x \mid \exists H. (H, x) \in R \land H \subseteq B \} \]

Example:

\[
\begin{align*}
R & \equiv \{ \{ \} \} \cup \{ \{ n \}, n+1 \} \\
\hat{R}(\{3, 6, 10\}) & = \{0, 4, 7, 11\}
\end{align*}
\]

The Set

Definition: \( B \) is \( R \)-closed iff \( \hat{R}(B) \subseteq B \)

Definition: \( X \) is the least \( R \)-closed subset of \( A \)

Slide 18

This does always exist:

Facts: \( B_1 \text{ R-closed } \land \ B_2 \text{ R-closed } \implies B_1 \cap B_2 \text{ R-closed} \)

\[ X = \bigcap \{ B \subseteq A. B \text{ R-closed} \} \]

Generation from Above

How to compute \( X \)?

\( X = \bigcap \{ B \subseteq A. B \text{ R-closed} \} \) hard to work with.

Instead: view \( X \) as least fixpoint, \( X \) least set with \( \hat{R}(X) = X \).

Fixpoints can be approximated by iteration:

Slide 20

\[
X_0 = \hat{R}^0(\{\}) = \{\} \\
X_1 = \hat{R}^1(\{\}) = \text{rules without hypotheses} \\
\vdots \\
X_n = \hat{R}^n(\{\}) \\
X_\omega = \bigcup_{n \in \mathbb{N}} R^n(\{\}) = X
\]
Generation from Below

$R^0(\{\}) \cup R^1(\{\}) \cup R^2(\{\}) \cup \ldots$