Typed Arithmetic Expressions

Principles of Programming Languages

CSE 526

1. Typed Arithmetic Expressions
2. Simply-Typed λ-Calculus
Types

- Types are way to classify terms (programs)
- Meaningful terms (e.g. those that do not get stuck) should have a type
- A typing relation relates terms to types.
- Two ways to define semantics:
  - *Curry-style*: Define terms and their semantics, then define types to reject those terms whose semantics are problematic.
  - *Church-style*: Define terms and a typing relation, then define semantics only for well-typed terms.
Typed arithmetic expressions

$t ::= \text{true} \quad \text{Terms}$
$\quad | \quad \text{false}$
$\quad | \quad \text{if}(t,t,t)$
$\quad | \quad 0$
$\quad | \quad \text{succ } t$
$\quad | \quad \text{pred } t$
$\quad | \quad \text{iszero } t$

$T ::= \quad \text{Types}$
$\quad \text{Bool}$
$\quad | \quad \text{Nat}$
Typing relation for arithmetic expressions

The smallest binary relation “:" between types and terms satisfying all instances of the following inference rules:

\[
\begin{align*}
\text{true} : \text{Bool} & \quad \text{T-TRUE} \\
\text{false} : \text{Bool} & \quad \text{T-FALSE} \\
\text{if}(t_1, t_2, t_3) : T & \quad \text{T-IF} \\
\end{align*}
\]
A term $t$ is said to be \textit{well-typed} if there is a type $T$ such that $t : T$.

- \textbf{Uniqueness of types:} Each term $t$ has at most one type $T$ such that $t : T$. 
Properties of the typing relation

A term \( t \) is said to be \textit{well-typed} if there is a type \( T \) such that \( t : T \).

- **Uniqueness of types:** Each term \( t \) has at most one type \( T \) such that \( t : T \).

- **Progress:** For every well-typed term \( t \), either \( t \) is a value or there is a \( t' \) such that \( t \rightarrow t' \).
Properties of the typing relation

A term $t$ is said to be **well-typed** if there is a type $T$ such that $t : T$.

- **Uniqueness of types:** Each term $t$ has at most one type $T$ such that $t : T$.
- **Progress:** For every well-typed term $t$, either $t$ is a value or there is a $t'$ such that $t \to t'$.
- **Preservation:** If $t : T$ and $t \to t'$ then $t' : T$. 
A term $t$ is said to be *well-typed* if there is a type $T$ such that $t : T$.

- **Uniqueness of types:** Each term $t$ has at most one type $T$ such that $t : T$.
- **Progress:** For every well-typed term $t$, either $t$ is a value or there is a $t'$ such that $t \rightarrow t'$.
- **Preservation:** If $t : T$ and $t \rightarrow t'$ then $t' : T$.
- **Safety** = Progress + Preservation
Inversion of the typing relation

- If `true : R` then `R = Bool`
  (similarly, if `false : R` then `R = Bool`)
- If `if(t_1, t_2, t_3) : R` then `t_1 : Bool`, `t_2 : R` and `t_3 : R`.
- If `0 : R` then `R = Nat`
- If `succ t_1 : R` then `R = Nat` and `t_1 : Nat`
  (similarly for `pred t_1 : R`)
- If `iszero t_1 : R` then `R = Bool` and `t_1 : Nat`

These results follow from the definition of the typing relation, and are used to prove progress and preservation.
Enriched λ-Calculus

- Recall booleans, numbers and operations on them can be encoded in the pure λ-calculus.
- Nevertheless, it is convenient to include primitive data types in the calculus as well.
- \( \lambda \text{B} \) is an enriched calculus with boolean data types \text{true} and \text{false}, and operation \text{if}.
  \[ \lambda x. \lambda y. \text{if}(x, y, x) \] 
  is a term in \( \lambda \text{B} \).
- \( \lambda \text{NB} \) is a similarly enriched calculus with numbers and booleans
  \[ \lambda x. \lambda y. \text{if}(\text{iszero}(x), \text{succ}(y), x) \] 
  is a term in \( \lambda \text{NB} \).
Simply-Typed λ-Calculus

Syntax:

\[ t ::= \]

- \( x \) Variable
- \( \lambda x : T. t \) Abstraction
- \( t_1 t_2 \) Application

Terms

\( T ::= \)

- Base types
- \( T \to T \) type of functions

\( \Gamma ::= \)

- Empty Context
- Variable Binding
Simply-Typed λ-Calculus

Syntax:

\[
\begin{align*}
t & ::= & x & \quad \text{Variable} \\
   & & \| & \lambda x : T . t & \quad \text{Abstraction} \\
   & & \| & t \ t & \quad \text{Application} \\
\end{align*}
\]

\[
\begin{align*}
T & ::= & A & \quad \text{Base types} \\
   & & \| & T \rightarrow T & \quad \text{type of functions}
\end{align*}
\]
Simply-Typed $\lambda$-Calculus

Syntax:

$t ::= \begin{align*}
\text{Terms} \\
& \quad x \quad \text{Variable} \\
| \quad \lambda x : T. \ t \quad \text{Abstraction} \\
| \quad t \ t \quad \text{Application}
\end{align*}$

$T ::= \begin{align*}
\text{Types} \\
& \quad A \quad \text{Base types} \\
| \quad T \rightarrow T \quad \text{type of functions}
\end{align*}$

$\Gamma ::= \begin{align*}
\text{Contexts} \\
& \quad \emptyset \quad \text{Empty Context} \\
| \quad \Gamma, x : T \quad \text{Variable Binding}
\end{align*}$
Evaluation (Call-By-Value)

Small-Step Evaluation Relation for simply-typed $\lambda$-calculus:

- **E-App1**
  \[
  t_1 \rightarrow t'_1 \\
  t_1 t_2 \rightarrow t'_1 \ t_2
  \]

- **E-Abs2**
  \[
  t_2 \rightarrow t'_2 \\
  v_1 t_2 \rightarrow v_1 t'_2
  \]

- **E-AppAbs**
  \[
  (\lambda x : T. \ t_1) v_2 \rightarrow [x \mapsto v_2] t_1
  \]
Typing Relation

\[
\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad T\text{-VAR}
\]
Typing Relation

**T-VAR**

\[
\frac{x : T \in \Gamma}{\Gamma \vdash x : T}
\]

**T-ABS**

\[
\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.t_2 : T_1 \to T_2}
\]

\[
\frac{\Gamma \vdash s : T_1 \to T_2}{\Gamma \vdash t : T_1 \vdash (s \, t) : T_2}
\]

**T-App**
Typing Relation

\[
\begin{align*}
\frac{x : T \in \Gamma}{\Gamma \vdash x : T} & \quad \text{T-VAR} \\
\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.t_2 : T_1 \rightarrow T_2} & \quad \text{T-ABS} \\
\frac{\Gamma \vdash s : T_1 \rightarrow T_2 \quad \Gamma \vdash t : T_1}{\Gamma \vdash (s \ t) : T_2} & \quad \text{T-APP}
\end{align*}
\]
Properties of the typing relation

A term $t$ is said to be well-typed in context $\Gamma$ if there is a type $T$ such that $t : T$.

- **Uniqueness of types:** In a context $\Gamma$, each term $t$ has at most one type $T$ such that $t : T$.

- **Progress:** For every closed, well-typed term $t$, either $t$ is a value or there is a $t'$ such that $t \to t'$.

- **Preservation under substitution:** If $\Gamma, x : S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.

- **Preservation:** If $\Gamma \vdash t : T$ and $t \to t'$ then $\Gamma \vdash t' : T$.

- **Safety = Progress + Preservation**
Erasure and Typability

Erasure is a function that maps simply-typed \( \lambda \)-terms to untyped \( \lambda \)-terms.

\[
\begin{align*}
\text{erase}(x) & = x \\
\text{erase}(\lambda x : T. \ t) & = \lambda x. \ \text{erase}(t) \\
\text{erase}(t_1 \ t_2) & = \text{erase}(t_1) \ \text{erase}(t_2)
\end{align*}
\]

- If \( t \rightarrow t' \) under typed evaluation relation, then \( \text{erase}(t) \rightarrow \text{erase}(t') \)
Erasure and Typability

`erase` is a function that maps simply-typed \( \lambda \)-terms to untyped \( \lambda \)-terms.

\[
\begin{align*}
\text{erase}(x) &= x \\
\text{erase}(\lambda x : T. t) &= \lambda x. \text{erase}(t) \\
\text{erase}(t_1 \ t_2) &= \text{erase}(t_1) \ \text{erase}(t_2)
\end{align*}
\]

- If \( t \rightarrow t' \) under typed evaluation relation, then \( \text{erase}(t) \rightarrow \text{erase}(t') \)
- If \( \text{erase}(t) \rightarrow m' \), then there is a simply-typed term \( t' \) such that \( t \rightarrow t' \) (under typed evaluation relation) and \( \text{erase}(t') = m' \)
Erasure and Typability

erase is a function that maps simply-typed λ-terms to untyped λ-terms.

\[
\begin{align*}
erase(x) &= x \\
erase(\lambda x : T. t) &= \lambda x. \erase(t) \\
erase(t_1 \ t_2) &= erase(t_1) \ erase(t_2)
\end{align*}
\]

- If \( t \rightarrow t' \) under typed evaluation relation, then \( \erase(t) \rightarrow \erase(t') \)
- If \( \erase(t) \rightarrow m' \), then there is a simply-typed term \( t' \) such that \( t \rightarrow t' \) (under typed evaluation relation) and \( \erase(t') = m' \)
- An untyped term \( m \) is **typable** if there is some simply-typed term \( t \) and type \( T \) and context \( \Gamma \) such that \( \erase(t) = m \) and \( \Gamma \vdash t : T \).
Erasure and Typability

\textit{erase} is a function that maps simply-typed \( \lambda \)-terms to untyped \( \lambda \)-terms.

\[
\begin{align*}
\text{erase}(x) &= x \\
\text{erase}(\lambda x : T. \ t) &= \lambda x. \ \text{erase}(t) \\
\text{erase}(t_1 \ t_2) &= \text{erase}(t_1) \ \text{erase}(t_2)
\end{align*}
\]

- If \( t \rightarrow t' \) under typed evaluation relation, then \( \text{erase}(t) \rightarrow \text{erase}(t') \)
- If \( \text{erase}(t) \rightarrow m' \), then there is a simply-typed term \( t' \) such that \( t \rightarrow t' \) (under typed evaluation relation) and \( \text{erase}(t') = m' \)
- An untyped term \( m \) is \textbf{typable} if there is some simply-typed term \( t \) and type \( T \) and context \( \Gamma \) such that \( \text{erase}(t) = m \) and \( \Gamma \vdash t : T \).
- Not every untyped lambda term is typable!

Example: \((x \ x)\)