Typed Arithmetic Expressions

Principles of Programming Languages

CSE 526

Compiled at 18:30 on 2020/04/15

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

Terms

$$
\begin{array}{lcl}
t & ::= & \texttt{true} \\
 & | & \texttt{false} \\
 & | & \texttt{if}(t, t, t) \\
 & | & \texttt{0} \\
 & | & \texttt{succ}\ \ t \\
 & | & \texttt{pred}\ \ t \\
 & | & \texttt{iszero}\ \ t
\end{array}
$$

Types

$$
\begin{array}{lcl}
T & ::= & \\
 & & \texttt{Bool} \\
 & | & \texttt{Nat}
\end{array}
$$

Typing relation for arithmetic expressions

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

$$\texttt{true} : \texttt{Bool} \qquad \text{T-True}$$

$$\texttt{false} : \texttt{Bool} \qquad \text{T-False}$$

$$\frac{t_1 : \texttt{Bool} \quad t_2 : T \quad t_3 : T}{\texttt{if}(t_1, t_2, t_3) : T} \qquad \text{T-If}$$

$$\texttt{0} : \texttt{Nat} \qquad \text{T-Zero}$$

$$\frac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}} \qquad \text{T-Succ}$$

$$\frac{t_1 : \texttt{Nat}}{\texttt{pred}\ t_1 : \texttt{Nat}} \qquad \text{T-Pred}$$

$$\frac{t_1 : \texttt{Nat}}{\texttt{iszero}\ t_1 : \texttt{Bool}} \qquad \text{T-IsZero}$$

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 $t : T$.

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 $t \rightarrow t'$.

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 $t' : T$.

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 $t \rightarrow t'$.
- **Preservation:** If $t : T$ and $t \rightarrow t'$ then $t' : T$.
- Safety = Progress + 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
- $\lambda\mathbf{B}$ is an enriched calculus with boolean data types `true` and `false`, and operation `if`.
  $\lambda x.\ \lambda y.\ \texttt{if}(x, y, x)$ is a term in $\lambda\mathbf{B}$.
- $\lambda\mathbf{NB}$ is a similarly enriched calculus with numbers and booleans
  $\lambda x.\ \lambda y.\ \texttt{if}(\texttt{iszero}(x), \texttt{succ}(y), x)$ is a term in $\lambda\mathbf{NB}$

Simply-Typed λ-Calculus

Syntax:

t	::=		Terms
		x	Variable
	$\mid$	$\lambda x : T.\ t$	Abstraction
	$\mid$	$t\ t$	Application

Simply-Typed λ-Calculus

Syntax:

t	::=		Terms
		x	Variable
	$\mid$	$\lambda x : T.\ t$	Abstraction
	$\mid$	$t\ t$	Application
T	::=		Types
		A	Base types
	$\mid$	$T \to T$	type of functions

Simply-Typed λ-Calculus

Syntax:

t	$::=$		Terms
		x	Variable
	$\mid$	$\lambda x : T.\ t$	Abstraction
	$\mid$	$t\ t$	Application
T	$::=$		Types
		A	Base types
	$\mid$	$T \to T$	type of functions
Γ	$::=$		Contexts
		$\emptyset$	Empty Context
	$\mid$	$\Gamma, x : T$	Variable Binding

Evaluation (Call-By-Value)

Small-Step Evaluation Relation for simply-typed λ-calculus:

$$\frac{t_1 \rightarrow t_1'}{t_1\ t_2 \rightarrow t_1'\ t_2} \quad \text{E-APP1}$$

$$\frac{t_2 \rightarrow t_2'}{v_1\ t_2 \rightarrow v_1\ t_2'} \quad \text{E-ABS2}$$

$$(\lambda x : T.\ t_1)\ v_2 \rightarrow [x \mapsto v_2]t_1 \qquad \text{E-APPABS}$$

Typing Relation

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{T-VAR}$$

Typing Relation

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \textsc{T-Var}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.\ t_2 \ :\ T_1 \to T_2} \quad \textsc{T-Abs}$$

Typing Relation

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{T-VAR}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.\ t_2 \ :\ T_1 \rightarrow T_2} \quad \text{T-ABS}$$

$$\frac{\Gamma \vdash s : T_1 \rightarrow T_2 \qquad \Gamma \vdash t : T_1}{\Gamma \vdash (s\ t) : T_2} \quad \text{T-APP}$$

Properties of the typing relation

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

- **Uniqueness of types:** In a context Γ, 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

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

$$\begin{aligned} erase(x) &= x \\ erase(\lambda x : T.\ t) &= \lambda x.\ erase(t) \\ erase(t_1\ t_2) &= erase(t_1)\ erase(t_2) \end{aligned}$$

- If $t \rightarrow t'$ under typed evaluation relation, then $erase(t) \rightarrow erase(t')$

Erasure and Typability

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

$$
\begin{aligned}
erase(x) &= x \\
erase(\lambda x : T.\ t) &= \lambda x.\ erase(t) \\
erase(t_1\ t_2) &= erase(t_1)\ erase(t_2)
\end{aligned}
$$

- If $t \to t'$ under typed evaluation relation, then $erase(t) \to erase(t')$
- If $erase(t) \to m'$, then there is a simply-typed term t' such that $t \to t'$ (under typed evaluation relation) and $erase(t') = m'$

Erasure and Typability

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

$$
\begin{array}{rcl}
erase(x) & = & x \\
erase(\lambda x : T.\ t) & = & \lambda x.\ erase(t) \\
erase(t_1\ t_2) & = & erase(t_1)\ erase(t_2)
\end{array}
$$

- 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 Γ such that $erase(t) = m$ and $\Gamma \vdash t : T$.

Erasure and Typability

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

$$\begin{array}{rcl} erase(x) & = & x \\ erase(\lambda x : T.\ t) & = & \lambda x.\ erase(t) \\ erase(t_1\ t_2) & = & erase(t_1)\ erase(t_2) \end{array}$$

- 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 Γ such that $erase(t) = m$ and $\Gamma \vdash t : T$.
- Not every untyped lambda term is typable!
  Example: $(x\ x)$