SWEN430 - Compiler Engineering

Lecture 8 - Typing II

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Syntax for $\lambda_{\mathbb{W}}$

p	::=	$f_1 \ldots f_n\ e$	*programs*
f	::=	$T_1\ m(T_2\ n)\{\ \overline{s}\ \}$	*functions*
e	::=		*expressions*
	|	v	*constants*
	|	n, m	*names*
	|	$e_1\ op\ e_2$	*binary*
	|	$e_1(e_2)$	*application*
	|	$\overline{s}$	*block*
s	::=	$n = e \mid \texttt{return}\ e$	*statements*
v	::=	null | true | false | ... | -1 | 0 | 1 | ... | $\lambda x.s$	*values*
T	::=	void | bool | int | null | $T_1 \vee T_2$ | $T_1 \rightarrow T_2$	*types*
op	::=	'!=' | '==' | '<' | '<=' | '>=' | '>'	*operators*
		| '+' | '−' | '*' |	

Typing Statements

$$\frac{\Gamma \vdash \mathtt{x} : \mathtt{T}_1 \quad \Gamma \vdash \mathtt{e} : \mathtt{T}_2 \quad \mathtt{T}_2 \leq \mathtt{T}_1}{\Gamma \vdash \mathtt{x} = \mathtt{e} : \mathtt{void}} \text{ (T-Assign)}$$

$$\frac{\Gamma \vdash \mathtt{e} : \mathtt{T}}{\Gamma \vdash \mathtt{return}\ \mathtt{e} : \mathtt{T}} \text{ (T-Return)}$$

$$\frac{\Gamma \vdash \mathtt{s}_1 : \mathtt{T}_1 \quad \Gamma \vdash \mathtt{s}_2 : \mathtt{T}_2}{\Gamma \vdash \mathtt{s}_1 ; \mathtt{s}_2 : \mathtt{T}_2} \text{ (T-Sequence)}$$

- Example:

$$\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{x} = 1; \mathtt{return}\ \mathtt{x} : \mathtt{int}$$

- **NOTE:** Subtyping operator "$\leq$" discussed later!

Typing Programs

$$\frac{\Gamma \cup \{\mathtt{n} : \mathtt{T}_1\} \vdash \overline{\mathtt{s}} : \mathtt{T}_3 \quad \mathtt{T}_3 \leq \mathtt{T}_2}{\Gamma \vdash \mathtt{T}_2\ \mathtt{m}(\mathtt{T}_1\ \mathtt{n})\ \{\overline{\mathtt{s}}\} : \mathtt{T}_1 \rightarrow \mathtt{T}_2} \text{ (T-Fun)}$$

$$\frac{\begin{array}{c}\mathtt{f} = \mathtt{T}_2\ \mathtt{m}(\mathtt{T}_1\ \mathtt{n})\ \{\overline{\mathtt{s}}\} \\ \Gamma \vdash \mathtt{f} : \mathtt{T}_1 \rightarrow \mathtt{T}_2 \quad \Gamma \cup \{\mathtt{m} : \mathtt{T}_1 \rightarrow \mathtt{T}_2\} \vdash \overline{\mathtt{f}}\ \mathtt{e} : \mathtt{T}\end{array}}{\Gamma \vdash \mathtt{f}\ \overline{\mathtt{f}}\ \mathtt{e} : \mathtt{T}} \text{ (T-Prog)}$$

- Example:

$$\emptyset \vdash \mathtt{int\ f(int\ x)\{\ return\ x\ \}\ f(x)\ :\ int}$$

Typing Expressions (Examples)

$$\frac{}{\Gamma \vdash c : \texttt{int}} \text{ (T-Int)} \qquad \frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-Var)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \; \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}} \text{ (T-Add)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \; \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 < e_2 : \texttt{bool}} \text{ (T-LessThan)}$$

$$\frac{\Gamma \vdash e_1 : T \; \Gamma \vdash e_2 : T}{\Gamma \vdash e_1 == e_2 : \texttt{bool}} \text{ (T-Eq)}$$

$$\frac{m : T_2 \rightarrow T_1 \in \Gamma \quad \Gamma \vdash e : T_3 \quad T_3 \leq T_2}{\Gamma \vdash m(e) : T_1} \text{ (T-App)}$$

Inside the WHILE Compiler!

```
public Type check(Expr.Variable expr, Map<String, Type> environment) {¶
    Type type = environment.get(expr.getName());¶
    if (type == null) {¶
        syntaxError("unknown variable encountered: " + expr.getName(), file.source, expr);¶
    }¶
    return type;¶
}¶
```

```
public void check(Stmt.Assign stmt, Map<String,Type> environment) {¶
    Type lhs = check(stmt.getLhs(),environment);¶
    Type rhs = check(stmt.getRhs(),environment);¶
    // Make sure the type being assigned is a subtype of the destination¶
    checkSubtype(lhs,rhs,stmt.getRhs());¶
}¶
```

```
public void check(Stmt.Return stmt, Map<String, Type> environment) {¶
    if(stmt.getExpr() != null) {¶
        Type ret = check(stmt.getExpr(),environment);¶
        // Make sure return value is subtype of enclosing method's return type¶
        checkSubtype(method.getRet(),ret,stmt.getExpr());¶
    } else {¶
        // Make sure return type is instance of Void¶
        checkInstanceOf(new Type.Void(),stmt,method.getRet().getClass());¶
    }¶
}¶
```

Subtyping WHILE

- Subtyping in WHILE exists because of **union types!**
- Examples:
  - » $\texttt{int} \leq \texttt{int} \vee \texttt{null}$
  - » $\texttt{int} \vee \texttt{int} \leq \texttt{int}$
  - » $\{(\texttt{int} \vee \texttt{bool})\ \texttt{f}\} \leq \{\texttt{int f}\} \vee \{\texttt{bool f}\}$
- So, how do we **implement** the subtyping algorithm?

Subtyping Relation (First Attempt)

$$\frac{}{\mathtt{T} \leq \mathtt{T}} \quad \text{(S-Reflex)} \qquad \frac{\forall \mathtt{i} \,.\, \mathtt{T_i} \leq \mathtt{T'_i}}{\{\overline{\mathtt{T\ n}}\} \leq \{\overline{\mathtt{T'\ n}}\}} \quad \text{(S-Depth)}$$

$$\frac{\mathtt{T_1} \leq \mathtt{T_3} \quad \mathtt{T_2} \leq \mathtt{T_3}}{\mathtt{T_1} \vee \mathtt{T_2} \leq \mathtt{T_3}} \quad \text{(S-Union1)} \qquad \frac{\mathtt{T_1} \leq \mathtt{T_2}}{\mathtt{T_1} \leq \mathtt{T_2} \vee \mathtt{T_3}} \quad \text{(S-Union2)}$$

- **Examples**:
  - » $\mathtt{int} \leq \mathtt{int} \vee \mathtt{null}$ (by S-Union2, S-Reflex)
  - » $\mathtt{int} \vee \mathtt{int} \leq \mathtt{int}$ (by S-Union1, S-Reflex)
  - » $\{\,\mathtt{int\ f}\,\} \vee \{\,\mathtt{bool\ f}\,\} \leq \{\,(\mathtt{int} \vee \mathtt{bool})\ \mathtt{f}\,\}$ (by S-Union1)

Soundness & Completeness of Subtyping

Type Semantics

$$\begin{aligned}
[\![\texttt{int}]\!] &= \mathbb{Z} \\
[\![\{\overline{\texttt{T f}}\}]\!] &= \left\{ \{\overline{\texttt{f} : \texttt{v}}\} \;\middle|\; \texttt{v}_1 \in [\![\texttt{T}_1]\!], \ldots, \texttt{v}_n \in [\![\texttt{T}_n]\!] \right\} \\
[\![\texttt{T}_1 \vee \ldots \vee \texttt{T}_n]\!] &= [\![\texttt{T}_1]\!] \cup \ldots \cup [\![\texttt{T}_n]\!]
\end{aligned}$$

Subtyping Soundness

If $\texttt{T}_1 \leq \texttt{T}_2$ then $[\![\texttt{T}_1]\!] \subseteq [\![\texttt{T}_2]\!]$

Subtyping Completeness

If $[\![\texttt{T}_1]\!] \subseteq [\![\texttt{T}_2]\!]$ then $\texttt{T}_1 \leq \texttt{T}_2$

- Here, $\texttt{T}_1 \leq \texttt{T}_2$ represents outcome of **subtype algorithm** whilst $[\![\texttt{T}_1]\!] \subseteq [\![\texttt{T}_2]\!]$ represents **idealised solution**

Soundness & Completeness of Subtyping (in English)

- Again, in English:
  - » **Soundness**: if subtype algorithm says $\texttt{T}_1$ subtype of $\texttt{T}_2$, then every value which could be in $\texttt{T}_1$ must be permitted in $\texttt{T}_2$
  - » **Completeness**: if every value which could be in $\texttt{T}_1$ is permitted in $\texttt{T}_2$, then subtype algorithm should report that $\texttt{T}_1$ is subtype of $\texttt{T}_2$
- **Question:** *is our subtype relation sound and complete?*

  (how do we even go about trying to answer this?)
- **Answer:** *it's sound, but not complete*

Complete Subtyping for WHILE

- This **holds** under given rules:

$$\{\,\texttt{int f}\,\} \vee \{\,\texttt{bool f}\,\} \leq \{\,(\texttt{int} \vee \texttt{bool})\ \texttt{f}\,\}$$

- This **does not hold** under given rules:

$$\{\,(\texttt{int} \vee \texttt{bool})\ \texttt{f}\,\} \leq \{\,\texttt{int f}\,\} \vee \{\,\texttt{bool f}\,\}$$

- *So what?*
- *How do we fix this?*

References

- **See:** Sound and Complete Flow Typing with Unions, Intersections and Negations. In *Proc. VMCAI*, David J. Pearce, 2013.