

Comp 411 Notes - Wed, Mar 3 2005
REVIEW OF ALGORITHMIC TYPING AND SUBTYPING
Scribe: Mathias Ricken

Today we reviewed the relationship between declarative and algorithmic typing and subtyping.

1 Minutiae

The mail server was down yesterday. Does ePost provide an alternative? If ePost is running in several places, it does not provide a guarantee that you actually get a message: It might deliver it not at all or several times.

Will there be a midterm? Yes. When? Maybe during the 2nd week after Spring Break.

Diagrams should be embedded directly in \LaTeX , not as figures.

2 The Type Systems $<$

Our initial type system and our subtyping relation, denoted by $<$, were not suitable for implementation. The type system contained the T-SUB rule, which had a bare metavariable t_2 in its conclusion:

$$\frac{\Gamma \vdash e : t_1 \quad \Gamma \vdash t_1 < t_2}{\Gamma \vdash e : t_2} \text{T-SUB}$$

This rule is not based on the syntax of our language, i.e. it can be applied to any expression e . All the other rules only applied to specific forms, e.g. lambda abstractions or applications.

The subtyping relation also contained the problematic rules S-TRANS and S-REFL:

$$\frac{t_1 < t' \quad t' < t_2}{t_1 < t_2} \text{S-TRANS} \quad \frac{}{t < t} \text{S-REFL}$$

The conclusions of both S-TRANS and S-REFL overlap with other subtyping rules for the same reason as T-SUB: They are not tied to a specific form of term. The antecedents of S-TRANS furthermore contain a metavariable t' which does not appear in the conclusion, i.e. it must be guessed out of an infinite set of finite objects. Since there is an infinite number of types, this is not practical.

Using $<$, can you determine in a finite number of steps if $t_1 < t_2$, i.e. if one type is the subtype of another? It was not obvious in the initial type system that we can actually prove that one type is a subtype of another, and thus if some term is well-typed.

3 Introducing Algorithmic Typing and Subtyping

We thus reorganized our typing and subtyping relations by removing the rules mentioned above. The typing relation does not contain a subsumption rule anymore; instead, subtyping is pushed only into applications. This new typing relation is denoted \vdash_A . The subtyping relation lacks explicit reflexive and transitive rules; therefore, the rules for records and arrow types are changed to deal with reflexivity and transitivity implicitly. This new subtyping relation is called $<:$.

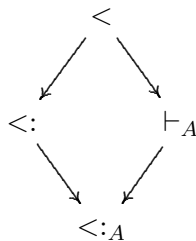


Figure 1: Relationships between the different type systems $<$, $<:$, \vdash_A , and $<:A$.

The type system that contains both these changes, denoted by $<:A$ thus can be reached in two ways, depending on what changes to $<$ are made first, as demonstrated by the diagram above.

4 The Type System $<:_A$

The type system $<:_A$ thus consists of the typing relation \vdash_A and the subtyping relation $<:_$.

The old and the new subtyping relations are completely equivalent:

$$t_1 < t_2 \iff t_1 <:_A t_2$$

The relationship between \vdash and \vdash_A is a bit more subtle: In \vdash , we did not have unicity of typing; \vdash_A does provide unicity of typing, since bound variables in lambda abstractions are explicitly annotated with their type and subtyping happens only in applications (Note: This might be a good question for the midterm.)

An equivalence between \vdash and \vdash_A can be formulated this way:

$$\Gamma \vdash e : t \iff \exists t'. \Gamma \vdash_A e : t' \wedge t' < t$$

5 A More Abstract View of Type Systems

Abstractly, a type system characterizes a set of closed expressions (or a set of pairs of typing contexts Γ and expressions).

We stated earlier that an expression will type in $<$ if and only if it will type in $<:_A$. This allows us to write our type checker based on $<:_A$. More abstractly, as Dan pointed out, a given expression is typable in one system if and only if it is typable in another system:

$$\forall e. \forall \Gamma. (\exists t'. \Gamma \vdash e : t') \iff (\exists t''. \Gamma \vdash_A e : t'') \text{ (Donkey Lemma :-)}$$

The two type systems accept the same pairs of typing contexts Γ s and expressions. Even more abstractly, we can use an empty context \emptyset and consider all closed terms:

$$\forall e. (\exists t'. \emptyset \vdash e : t') \iff (\exists t''. \emptyset \vdash_A e : t'')$$

This removes the restriction that the types for variables in Γ must be the same in both type systems. Alternatively, we could provide a function $[[\Gamma]]$ that maps one context to another.

6 Pushing Down Subtyping in an if-then-else Construct

Consider a calculus with an added **if-then-else** construct to the language:

$$\frac{\Gamma \vdash e_1 : \mathit{bool} \quad \Gamma \vdash e_2 : t \quad \Gamma \vdash e_3 : t}{\Gamma \vdash \mathbf{if} \ e_1 \ \mathbf{then} \ e_2 \ \mathbf{else} \ e_3 : t} \text{T-IF}$$

If we want to push occurrences of subtyping down just like in the other rules, how do we have to modify this rule? Below is a series of attempts:

We do not consider subtyping for e_1 since it has to be a *bool* (the only applicable subtyping would be $\mathit{bool} < \mathit{bool}$); however, subtyping can appear in e_2 and e_3 .

$$\frac{\Gamma \vdash e_1 : \mathit{bool} \quad \frac{\Gamma \vdash e_2 : t_1 < t}{\Gamma \vdash e_2 : t_1} \quad \frac{\Gamma \vdash e_3 : t_2 < t}{\Gamma \vdash e_3 : t_2}}{\Gamma \vdash \mathbf{if} \ e_1 \ \mathbf{then} \ e_2 \ \mathbf{else} \ e_3 : t}$$

But we do want to push the subtyping down, so we change the rule to:

$$\frac{\Gamma \vdash e_1 : \mathit{bool} \quad \Gamma \vdash e_2 : t_1 \quad \Gamma \vdash e_3 : t_2 \quad t_1 < t \quad t_2 < t}{\Gamma \vdash \mathbf{if} \ e_1 \ \mathbf{then} \ e_2 \ \mathbf{else} \ e_3 : t}$$

Just like with applications, we cannot push subtyping down any further. Additionally, the situation is complicated by the free metavariable t .

With this formulation, though, we have also lost unicity of typing again. For t , we actually want the “smallest common supertype” of t_1 and t_2 . This is called a “join” and is denoted by $t_1 \vee t_2 = J$.

Using joins, we can now formulate our typing rule correctly:

$$\frac{\Gamma \vdash e_1 : \mathit{bool} \quad \Gamma \vdash e_2 : t_1 \quad \Gamma \vdash e_3 : t_2 \quad t_1 \vee t_2 = t}{\Gamma \vdash \mathbf{if} \ e_1 \ \mathbf{then} \ e_2 \ \mathbf{else} \ e_3 : t} \text{TA-IF}$$

In the process of finding a join, we might be looking for supertypes of arrow types. Since the argument types for arrow types are contravariant, we also have to have an operation to find the “largest common subtype” of two types, a “meet” denoted by $t_1 \wedge t_2 = M$.