CS 510: Programming Languages

Subtyping and Coercion

David Walker

Type Conversion

Arithmetic subtyping requires **type conversion**.

- Arguments to floating point operations check for integers.
- Integers are silently converted to floating point.

Costly to implement directly!

Type Conversion

Observation: conversions can be made at compile time.

- Mixed-mode expressions rely on subsumption during type checking.

- We can detect statically where subsumption is used.

Type Conversion

Another strategy is to use **coercions**.

- For each subtyping σ <: τ, then there is a (unique) coercion function $\sigma \rightarrow \tau$.

- Uses of subsumption insert coercion from sub- to super-type.

Coherence

Issue: there might be many (different?) ways to insert coercions!

- Typing rules are not syntax-directed.
- Each typing derivation determines a different program.

Coherence of subtyping means that all ways are "equivalent".

Coherence

Consider the expression `sin`(3).

We may “read” this in two ways:

1. Convert 3 to floating point, then apply `sin` to the result.

2. Convert `sin` to the function that, when applied, converts its integer argument to floating point, then applies floating point `sin` to the result.

These are **equivalent**, but they are not **identical**.

Coercion Interpretation

Associate with each subtype relation a coercion mapping the sub- to the super-type:

$$\sigma \texttt{<:} \tau \leadsto v$$

The expression v is a value of type $\sigma \rightarrow \tau$.

Associate with each well-typed expression its translation into pure MinML.

$$\Gamma \vdash e : \tau \leadsto e'$$

Coercion Interpretation of Subtyping

$$\frac{}{\texttt{int} \mathrel{\texttt{<:}} \texttt{float} \leadsto \texttt{to\_float}}$$

$$\frac{}{\tau \mathrel{\texttt{<:}} \tau \leadsto \texttt{id}_\tau} \qquad \frac{\rho \mathrel{\texttt{<:}} \sigma \leadsto v \quad \sigma \mathrel{\texttt{<:}} \tau \leadsto v'}{\rho \mathrel{\texttt{<:}} \tau \leadsto v \texttt{;} v'}$$

$$\frac{\tau_1 \mathrel{\texttt{<:}} \sigma_1 \leadsto v_1 \quad \sigma_2 \mathrel{\texttt{<:}} \tau_2 \leadsto v_2}{\sigma_1 \to \sigma_2 \mathrel{\texttt{<:}} \tau_1 \to \tau_2 \leadsto v_1 \to v_2}$$

Coercion Interpretation of Subtyping

Auxiliary functions:

1. Primitive conversion: `to_float`.

2. Identity: $\mathtt{id}_\tau = \mathtt{fn}\, x\mathtt{:}\tau \,\mathtt{in}\, x \,\mathtt{end}$.

3. Composition: $v\,\mathtt{;}\,v' = \mathtt{fn}\, x\mathtt{:}\tau \,\mathtt{in}\, v'\mathtt{(}v\mathtt{(}x\mathtt{))} \,\mathtt{end}$.

4. Functions: $v_1 \to v_2 =$
   $\mathtt{fn}\, f\mathtt{:}\sigma_1\to\sigma_2 \,\mathtt{in}\, \mathtt{fn}\, x\mathtt{:}\tau_1 \,\mathtt{in}\, v_2\mathtt{(}f\mathtt{(}v_1\mathtt{(}x\mathtt{)))} \,\mathtt{end}\,\mathtt{end}$.

Coercion Interpretation of Subtyping

Theorem

1. If σ <: $\tau \leadsto v$, then $\vdash^- v : \sigma \to \tau$.

2. If σ <: $\tau \leadsto v_1$ and σ <: $\tau \leadsto v_2$, then $\vdash^- v_1 = v_2 : \sigma \to \tau$.

That is, any two coercions from τ to τ' are equivalent.

Subsumption Elimination

Subsumption inserts coercion function:

$$\frac{\Gamma \vdash e : \sigma \leadsto e' \qquad \sigma \mathrel{<:} \tau \leadsto v}{\Gamma \vdash e : \tau \leadsto v(e')}$$

The remaining rules simply compose the translations. For example,

$$\frac{\Gamma \vdash e_1 : \tau_2 \to \tau \leadsto e_1' \qquad \Gamma \vdash e_2 : \tau_2 \leadsto e_2'}{\Gamma \vdash e_1(e_2) : \tau \leadsto e_1'(e_2')}$$

Subsumption Elimination

Theorem

1. If $\Gamma \vdash e : \tau \leadsto e'$, then $\Gamma \vdash^{-} e' : \tau$.

2. If $\Gamma \vdash e : \tau \leadsto e_1$ and $\Gamma \vdash e : \tau \leadsto e_2$, then $\Gamma \vdash^{-} e_1 = e_2 : \tau$.

The proof of the first part is a straightforward induction on derivations. The second and third are more difficult and are omitted here.

Coercion Interpretation

Where else might this method apply?

- Tuple subtyping: truncate to supertype.
- Record subtyping: drop "extra" fields.

Coercion Interpretation

Not necessary for tuples, but useful for records.

- Tuples: projection operations typically are not affected by extra components at the end.

- Records: efficient field selection relies on pre-computing position of each field.

Record Subtyping

The **width** subtyping axiom states that "fatter" records are a subtype of "skinnier" ones:

$$\frac{m \geq n}{\{l_1\!:\!\tau_1\,,\,\ldots\,,l_m\!:\!\tau_m\} \texttt{<:} \{l_1\!:\!\tau_1\,,\,\ldots\,,l_n\!:\!\tau_n\}}$$

In other words, a wider record may be provided where a narrower record with the same fields is required.

Dynamic Semantics of Records

The primitive instruction for record field selection is given by

$$\{l{:}v\,,\ldots\}\,.\,l \mapsto v.$$

In the **absence** of subtyping we can implement this efficiently.

- Order fields canonically (*e.g.*, alphabetically).

- Type determines position of field l in the record.

Dynamic Semantics of Records

In the **presence** of subtyping the type does not reveal the position of the fields!

- Fields might have been dropped anywhere.
- Can always "weaken" the type to hide additional fields.

Dynamic Semantics of Records

This suggests that we must **search** for the lth field on each access.

- Logarithmic in the number of fields using a balanced tree.

- Non-trivial constant factors.

Can we avoid run-time search?

Avoiding Search

Coercions to the rescue!

- If records are immutable, we can coerce records by copying.

- Type now reveals the position of each field.

But only if record fields are **immutable**!

- Otherwise sharing is lost by copying.

Coercion Interpretation

$$\frac{m > n}{\{l_1 \mathbin{:} \tau_1\,,\,\ldots\,,l_m \mathbin{:} \tau_m\} \mathrel{\texttt{<:}} \{l_1 \mathbin{:} \tau_1\,,\,\ldots\,,l_n \mathbin{:} \tau_n\} \leadsto \texttt{drop}_{m,n,l,\tau}}$$

$$\frac{\sigma_1 \mathrel{\texttt{<:}} \tau_1 \leadsto v_1 \quad \ldots \quad \sigma_n \mathrel{\texttt{<:}} \tau_n \leadsto v_n}{\{l_1 \mathbin{:} \sigma_1\,,\,\ldots\,,l_n \mathbin{:} \sigma_n\} \mathrel{\texttt{<:}} \{l_1 \mathbin{:} \tau_1\,,\,\ldots\,,l_n \mathbin{:} \tau_n\} \leadsto \texttt{copy}_{n,l,\sigma,v}}$$

Coercion Interpretation

The coercions are defined as follows:

$$\begin{array}{l} \texttt{drop}_{m,n,l,\sigma} \quad = \\ \quad \texttt{fn}\, x : \{l_1{:}\sigma_1, \ldots, l_m{:}\sigma_m\} \qquad = \\ \quad \{l_1{:}x.l_1, \ldots, l_n{:}x.l_n\} \end{array}$$

$$\begin{array}{l} \texttt{copy}_{n,l,\sigma,v} \quad = \\ \quad \texttt{fn}\, x : \{l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\} \qquad = \\ \quad \{l_1{:}v_1(x.l_1), \ldots, l_n{:}v_n(x.l_n)\} \end{array}$$

Coercion Interpretation

Theorem

1. If σ <: $\tau \leadsto v$, then $\vdash^{-} v : \sigma \to \tau$.

2. If σ <: $\tau \leadsto v_1$ and σ <: $\tau \leadsto v_2$, then $\vdash^{-} v_1 = v_2 : \sigma \to \tau$.

Mutable Records

What if records are **mutable**?

- Coercion by copying is precluded.

- Must retain “hidden” fields.

All are issues of **aliasing**, multiple active views of the same mutable data structure.

Mutable Records

A compromise solution:

- Represent records as a pair:
  - A **dictionary** (**view**, **dope vector**, or **access vector**) that determines the position of the lth field.
  - A **data array** that records the contents of the fields.
- Coercion truncates the dictionary, but not the data array so that sharing is preserved.

Dictionaries

Represent $\{l_1 : v_1, \ldots, l_n : v_n\}$ as a pair

$$(\{l_1 : j_1, \ldots, l_n : j_n\}, v_1 \texttt{*} \cdots \texttt{*} v_n)$$

where, initially, $j_i = i$ for $1 \leq i \leq n$.

Translate $v.l$ into

$$\texttt{snd}(v).(\texttt{fst}(v).l)$$

which fetches the index, and then the value, of field l.

(This has to be a primitive!)

Dictionaries

Coercion copies dictionary, but leaves data array intact.

- Maintains constant-time access to record components.

- Preserves sharing of (mutable) data.

Dictionaries

Specifically, if $\sigma = \{l_1\texttt{:}\sigma_1\texttt{,}\ldots\texttt{,}l_n\texttt{:}\sigma_n\}$ and $\tau = \{l_1\texttt{:}\mathtt{int}\texttt{,}\ldots\texttt{,}l_n\texttt{:}\mathtt{int}\}$, then

$$\mathtt{drop}_{m,n,l,\sigma} = \mathtt{fn}\,x\,\mathtt{in}\,\mathtt{(drop}_{m,n,l,\tau}\mathtt{(fst(}x\mathtt{)),snd(}x\mathtt{))}\,\mathtt{end}.$$

The dictionary is coerced, but the data is left intact.

Dictionaries

Similar issues arise in OOP:

- The dictionary is called the **vtable**.
- The record type is determined by the class (defines methods, fields).
- Coercion is often limited to implicit truncation (position matters).

Summary

Coercions support subtyping.

- Avoids run-time type checks.
- Applicable only indirectly to the mutable case.
- Copying records takes substantial extra space/time