Extending Hindley-Milner Type Inference with Coercive Structural Subtyping

Dmitriy Traytel Stefan Berghofer Tobias Nipkow

APLAS 2011

Technische Universität München

Outline

Why coercions?

A naive algorithm

Constraint-based algorithm

Conclusion

Real-world examples

- 2004: Avigad verifies in Isabelle:

  ```
  (λx.  pi x * ln (real x) / (real x)) ----> 1
  ```

Real-world examples

- 2004: Avigad verifies in Isabelle:

```
(λx. pi x * ln (real x) / (real x)) ----> 1
```

i.e. the prime number theorem

$$\lim_{x \to \infty} \frac{\pi(x) \ln x}{x} = 1$$

Real-world examples

- 2004: Avigad verifies in Isabelle:

  ```
  (λx. pi x * ln (real x) / (real x)) ----> 1
  ```

  i.e. the prime number theorem

$$\lim_{x \to \infty} \frac{\pi(x) \ln x}{x} = 1$$

- 2009: Hölzl uses 1061 explicit conversions in a single theory

Real-world examples

- 2004: Avigad verifies in Isabelle:

  ```
  (λx. pi x * ln (real x) / (real x)) ----> 1
  ```

  i.e. the prime number theorem

  $$\lim_{x \to \infty} \frac{\pi(x) \ln x}{x} = 1$$

- 2009: Hölzl uses 1061 explicit conversions in a single theory
- Both report “headaches”

Solution: coercive structural subtyping

Related work

- Subtyping part of the type system:

  Mitchell, Fuh & Mishra, Wand & O'Keefe, Pottier, Simonet

  Cardelli, Pratt & Tiuryn, Luo, Kießling, Frey, Benke, Barthe, Chen

  Reynolds, Swamy, Hicks & Bierman, Nordlander, Shields & Peyton Jones

  ...

Solution: coercive structural subtyping

Related work

- Subtyping part of the type system:

  Mitchell, Fuh & Mishra, Wand & O'Keefe, Pottier, Simonet
  Cardelli, Pratt & Tiuryn, Luo, Kießling, Frey, Benke, Barthe, Chen
  Reynolds, Swamy, Hicks & Bierman, Nordlander, Shields & Peyton Jones

  ...

- Incomplete coercion inference system:

  Saïbi, Luo

Solution: coercive structural subtyping

Related work

- Subtyping part of the type system:

  Mitchell, Fuh & Mishra, Wand & O'Keefe, Pottier, Simonet
  Cardelli, Pratt & Tiuryn, Luo, Kießling, Frey, Benke, Barthe, Chen
  Reynolds, Swamy, Hicks & Bierman, Nordlander, Shields & Peyton Jones

  ...

- Incomplete coercion inference system:

  Saïbi, Luo

- Complete coercion inference system:

  this publication

The Hindley-Milner typing rules remain unchanged:
No subtypes here

Type inference is extended with coercion inference
and coercion insertion

Our coercion inference system

- Coercions: $\mathbb{N} <:_{\texttt{real}} \mathbb{R}$
- Lifted by map functions: $\mathbb{N}$ *list* $<:_{\texttt{map real}} \mathbb{R}$ *list*
- Programmer inputs terms omitting coercions
- The system infers and inserts coercions
- Result is well-typed according to Hindley-Milner

Our coercion inference system

- Coercions: $\mathbb{N} <:_{\texttt{real}} \mathbb{R}$
- Lifted by map functions: $\mathbb{N}$ *list* $<:_{\texttt{map real}} \mathbb{R}$ *list*
- Programmer inputs terms omitting coercions
- The system infers and inserts coercions
- Result is well-typed according to Hindley-Milner
- The coercion inference system:
  - is sound and complete
  - does not change the underlying type system

Local coercion insertion

- Use judgement $\Gamma \vdash \mathtt{t} \leadsto \mathtt{u} : \tau$
- Idea: insert coercions whenever the function's domain does not match the argument type:

$$\frac{\vdash \mathtt{t}_1 \leadsto \mathtt{u}_1 : \tau_{11} \rightarrow \tau_{12} \qquad \vdash \mathtt{t}_2 \leadsto \mathtt{u}_2 : \tau_2 \qquad \tau_2 <:_{\mathtt{c}} \tau_{11}}{\vdash \mathtt{t}_1\ \mathtt{t}_2 \leadsto \mathtt{u}_1\ (\mathtt{c}\ \mathtt{u}_2) : \tau_{12}}$$

Local coercion insertion

- Use judgement $\Gamma \vdash \mathtt{t} \rightsquigarrow \mathtt{u} : \tau$
- Idea: insert coercions whenever the function's domain does not match the argument type:

$$\frac{\vdash \mathtt{t}_1 \rightsquigarrow \mathtt{u}_1 : \tau_{11} \rightarrow \tau_{12} \qquad \vdash \mathtt{t}_2 \rightsquigarrow \mathtt{u}_2 : \tau_2 \qquad \tau_2 <:_{\mathtt{c}} \tau_{11}}{\vdash \mathtt{t}_1\ \mathtt{t}_2 \rightsquigarrow \mathtt{u}_1\ (\mathtt{c}\ \mathtt{u}_2) : \tau_{12}}$$

- Used in Coq

Problematic example

Example: `leq i n` vs. `leq n i`

- Signatures: $\texttt{leq} :: \alpha \to \alpha \to \mathbb{B}$, $\texttt{n} :: \mathbb{N}$ and $\texttt{i} :: \mathbb{Z}$
- Declared coercion: $\mathbb{N} <:_{\texttt{int}} \mathbb{Z}$

Problematic example

Example: `leq i n` **vs.** `leq n i`

- Signatures: $\texttt{leq} :: \alpha \to \alpha \to \mathbb{B}$, $\texttt{n} :: \mathbb{N}$ and $\texttt{i} :: \mathbb{Z}$
- Declared coercion: $\mathbb{N} <:_{\texttt{int}} \mathbb{Z}$
- Correctly, `leq i n` becomes `leq i (int n)`, as
  - $\texttt{leq i} :: \mathbb{Z} \to \mathbb{B}$
  - $\texttt{n} :: \mathbb{N}$

Problematic example

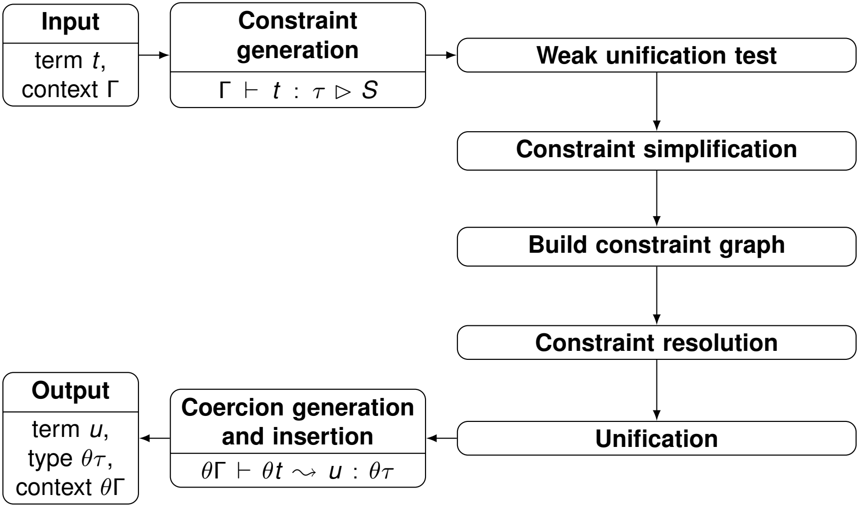
Example: `leq i n` vs. `leq n i`

- Signatures: $\texttt{leq} :: \alpha \to \alpha \to \mathbb{B}$, $\texttt{n} :: \mathbb{N}$ and $\texttt{i} :: \mathbb{Z}$
- Declared coercion: $\mathbb{N} <:_{\texttt{int}} \mathbb{Z}$
- Correctly, `leq i n` becomes `leq i (int n)`, as
  - $\texttt{leq i} :: \mathbb{Z} \to \mathbb{B}$
  - $\texttt{n} :: \mathbb{N}$
- Unfortunately, the coercion inference of `leq n i` fails, as
  - $\texttt{leq n} :: \mathbb{N} \to \mathbb{B}$
  - $\texttt{i} :: \mathbb{Z}$
  - no coercion from $\mathbb{Z}$ to $\mathbb{N}$

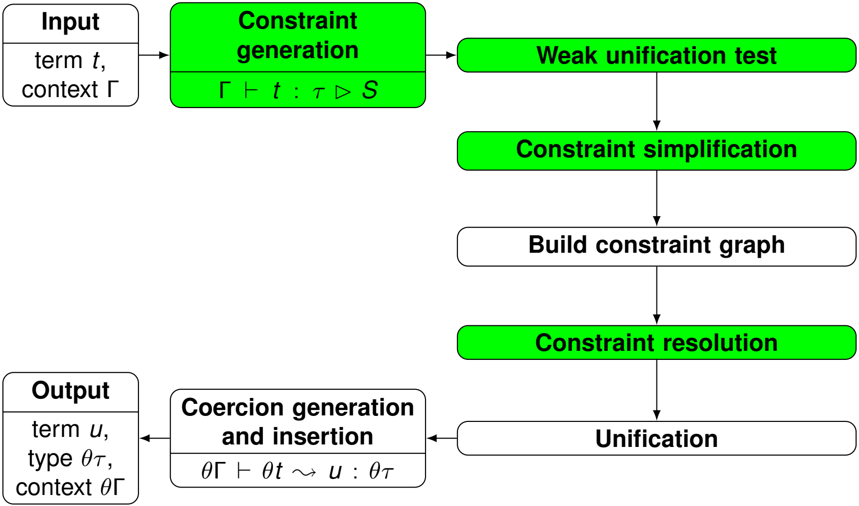
This is “normal” behaviour of coercions.

Coq Reference Manual

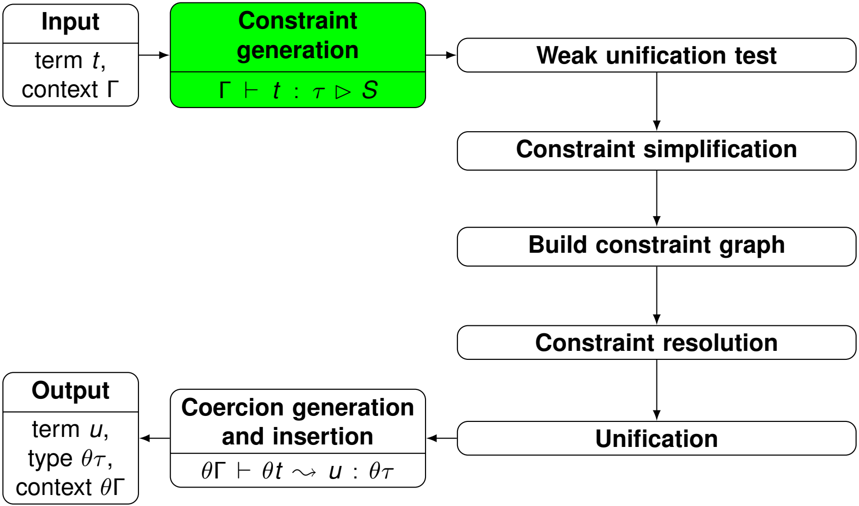
The subtyping pipeline



The subtyping pipeline



The subtyping pipeline



Constraint generation

$$\frac{\vdash \texttt{t}_1 : \tau \rhd S_1 \qquad \vdash \texttt{t}_2 : \sigma \rhd S_2 \qquad \alpha, \beta \text{ fresh}}{\vdash \texttt{t}_1\ \ \texttt{t}_2 : \beta \rhd S_1 \cup S_2 \cup \{\tau \doteq \alpha \to \beta, \sigma <: \alpha\}}$$

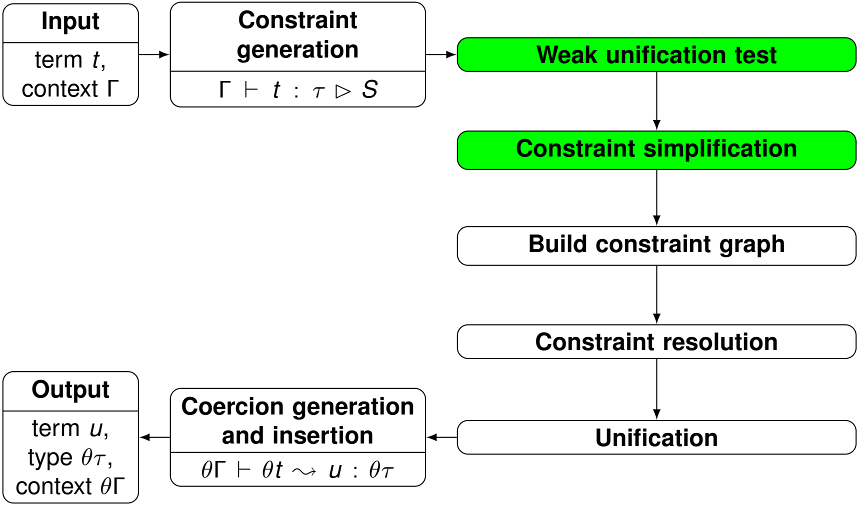
Constraint generation

$$\frac{\vdash \texttt{t}_1 : \tau \rhd S_1 \qquad \vdash \texttt{t}_2 : \sigma \rhd S_2 \qquad \alpha, \beta \text{ fresh}}{\vdash \texttt{t}_1\ \ \texttt{t}_2 : \beta \rhd S_1 \cup S_2 \cup \{\tau \doteq \alpha \to \beta, \sigma <: \alpha\}}$$

Example: `leq n i`

$$\dfrac{\dfrac{\dfrac{\texttt{leq} :: \alpha \to \alpha \to \mathbb{B}}{\vdash \texttt{leq} : \alpha \to \alpha \to \mathbb{B} \rhd \emptyset} \qquad \dfrac{\texttt{n} :: \mathbb{N}}{\vdash \texttt{n} : \mathbb{N} \rhd \emptyset}}{\vdash \texttt{leq}\ \ \texttt{n} : \beta_2 \rhd \{\alpha \to \alpha \to \mathbb{B} \doteq \alpha_2 \to \beta_2,\ \mathbb{N} <: \alpha_2\}} \qquad \dfrac{\texttt{i} :: \mathbb{Z}}{\vdash \texttt{i} : \mathbb{Z} \rhd \emptyset}}{\vdash \texttt{leq}\ \ \texttt{n}\ \ \texttt{i} : \beta_1 \rhd \left\{ \begin{array}{rcl} \alpha \to \alpha \to \mathbb{B} & \doteq & \alpha_2 \to \beta_2, \\ \beta_2 & \doteq & \alpha_1 \to \beta_1, \\ \mathbb{N} & <: & \alpha_2, \\ \mathbb{Z} & <: & \alpha_1 \end{array} \right\}}$$

The subtyping pipeline



Constraint simplification

- Goal: only atomic constraints $\alpha <: \beta$, $\alpha <: T$, $T <: \alpha$

Constraint simplification

- Goal: only atomic constraints $\alpha <: \beta$, $\alpha <: T$, $T <: \alpha$

$$\sigma \textit{ list} <: \tau \textit{ list} \quad \Leftrightarrow \quad \sigma <: \tau$$

Constraint simplification

- Goal: only atomic constraints $\alpha <: \beta$, $\alpha <: T$, $T <: \alpha$

$$\sigma \textit{ list} <: \tau \textit{ list} \quad \Leftrightarrow \quad \sigma <: \tau$$

$$\sigma_1 \to \sigma_2 <: \tau_1 \to \tau_2 \quad \Leftrightarrow \quad \tau_1 <: \sigma_1 \text{ and } \sigma_2 <: \tau_2$$

Constraint simplification

- Goal: only atomic constraints $\alpha <: \beta$, $\alpha <: T$, $T <: \alpha$

$$
\begin{aligned}
\sigma\ \textit{list} <: \tau\ \textit{list} &\quad\Leftrightarrow\quad \sigma <: \tau \\
\sigma_1 \to \sigma_2 <: \tau_1 \to \tau_2 &\quad\Leftrightarrow\quad \tau_1 <: \sigma_1 \text{ and } \sigma_2 <: \tau_2 \\
\alpha <: \tau\ \textit{list} &\quad\Leftrightarrow\quad \exists \alpha'.\ \alpha \doteq \alpha'\ \textit{list} \wedge \alpha'\ \textit{list} <: \tau\ \textit{list}
\end{aligned}
$$

Constraint simplification

- Goal: only atomic constraints $\alpha <: \beta$, $\alpha <: T$, $T <: \alpha$

$$\begin{array}{rcl} \sigma \ \mathit{list} <: \tau \ \mathit{list} & \Leftrightarrow & \sigma <: \tau \\ \sigma_1 \rightarrow \sigma_2 <: \tau_1 \rightarrow \tau_2 & \Leftrightarrow & \tau_1 <: \sigma_1 \text{ and } \sigma_2 <: \tau_2 \\ \alpha <: \tau \ \mathit{list} & \Leftrightarrow & \exists \alpha'.\ \alpha \doteq \alpha' \ \mathit{list} \wedge \alpha' \ \mathit{list} <: \tau \ \mathit{list} \end{array}$$

- $\Rightarrow$ corresponds to simplification
- $\Leftarrow$ corresponds to coercion generation
- variances are derived from map functions
  - `map` $:: (\alpha \rightarrow \beta) \rightarrow \alpha \ \mathit{list} \rightarrow \beta \ \mathit{list}$
  - λ`f g h. g ∘ h ∘ f` $::$ $(\beta_1 \rightarrow \alpha_1) \rightarrow (\alpha_2 \rightarrow \beta_2) \rightarrow (\alpha_1 \rightarrow \alpha_2) \rightarrow (\beta_1 \rightarrow \beta_2)$

Weak unification

$$\alpha <: \alpha \textit{ list} \quad \Leftrightarrow \quad \exists \alpha'.\, \alpha \doteq \alpha' \textit{ list} \text{ and } \alpha' \textit{ list} <: \alpha \textit{ list}$$

Weak unification

$$
\begin{aligned}
\alpha <: \alpha\ \textit{list} &\Leftrightarrow \exists\alpha'.\ \alpha \doteq \alpha'\ \textit{list}\ \text{and}\ \alpha'\ \textit{list} <: \alpha\ \textit{list} \\
&\Leftrightarrow \exists\alpha'.\ \alpha \doteq \alpha'\ \textit{list}\ \text{and}\ \alpha' <: \alpha \\
&\Leftrightarrow \alpha' <: \alpha'\ \textit{list}
\end{aligned}
$$

Weak unification

$$\begin{aligned}\alpha <: \alpha\ \textit{list} &\Leftrightarrow \exists\alpha'.\ \alpha \doteq \alpha'\ \textit{list} \text{ and } \alpha'\ \textit{list} <: \alpha\ \textit{list}\\ &\Leftrightarrow \exists\alpha'.\ \alpha \doteq \alpha'\ \textit{list} \text{ and } \alpha' <: \alpha\\ &\Leftrightarrow \alpha' <: \alpha'\ \textit{list}\end{aligned}$$

- Simplification process does not terminate
- Not solvable with structural coercions

Weak unification

$$\begin{aligned} \alpha <: \alpha\ \textit{list} &\Leftrightarrow \exists \alpha'.\ \alpha \doteq \alpha'\ \textit{list} \text{ and } \alpha'\ \textit{list} <: \alpha\ \textit{list} \\ &\Leftrightarrow \exists \alpha'.\ \alpha \doteq \alpha'\ \textit{list} \text{ and } \alpha' <: \alpha \\ &\Leftrightarrow \alpha' <: \alpha'\ \textit{list} \end{aligned}$$

- Simplification process does not terminate
- Not solvable with structural coercions
- Weak unification := unification after identifying all base types
- Initial constraint set weakly unifiable $\Rightarrow$ termination proof

Constraint simplification (example)

Example: `leq n i`

$$\{\alpha \to \alpha \to \mathbb{B} \doteq \alpha_2 \to \beta_2,\ \beta_2 \doteq \alpha_1 \to \beta_1,\ \mathbb{N} <: \alpha_2,\ \mathbb{Z} <: \alpha_1\}$$

Constraint simplification (example)

Example: `leq n i`

$$\{\alpha \to \alpha \to \mathbb{B} \doteq \alpha_2 \to \beta_2,\ \beta_2 \doteq \alpha_1 \to \beta_1,\ \mathbb{N} <: \alpha_2,\ \mathbb{Z} <: \alpha_1\}$$

$$\Downarrow$$

$$\{\mathbb{N} <: \alpha,\ \mathbb{Z} <: \alpha\}$$

Constraint simplification (example)

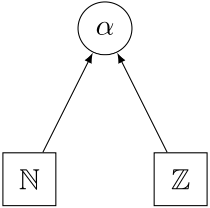
Example: `leq n i`

$$\{\alpha \to \alpha \to \mathbb{B} \doteq \alpha_2 \to \beta_2,\ \beta_2 \doteq \alpha_1 \to \beta_1,\ \mathbb{N} <: \alpha_2,\ \mathbb{Z} <: \alpha_1\}$$

$$\Downarrow$$

$$\{\mathbb{N} <: \alpha,\ \mathbb{Z} <: \alpha\}$$

$$\Downarrow$$



Constraint graph

Constraint simplification (example)

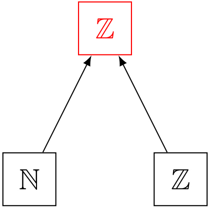
Example: `leq n i`

$$\{\alpha \to \alpha \to \mathbb{B} \doteq \alpha_2 \to \beta_2,\ \beta_2 \doteq \alpha_1 \to \beta_1,\ \mathbb{N} <: \alpha_2,\ \mathbb{Z} <: \alpha_1\}$$

$$\Downarrow$$

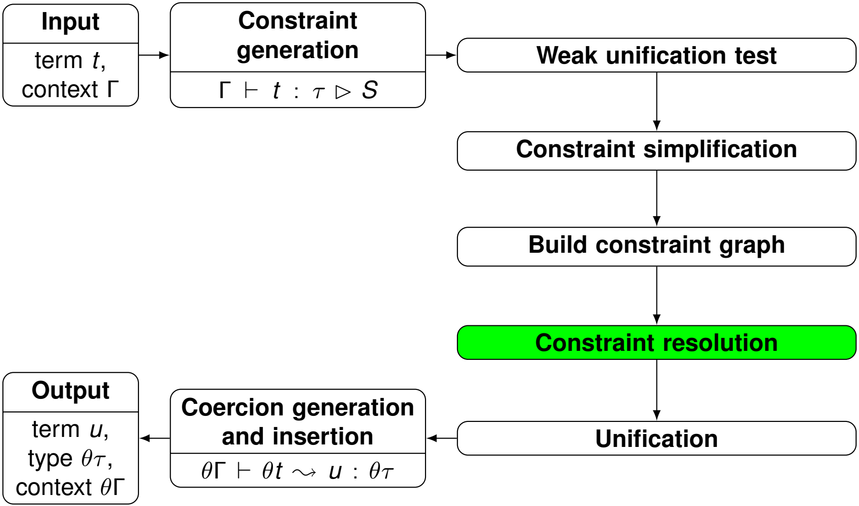
$$\{\mathbb{N} <: \alpha,\ \mathbb{Z} <: \alpha\}$$

$$\Downarrow$$

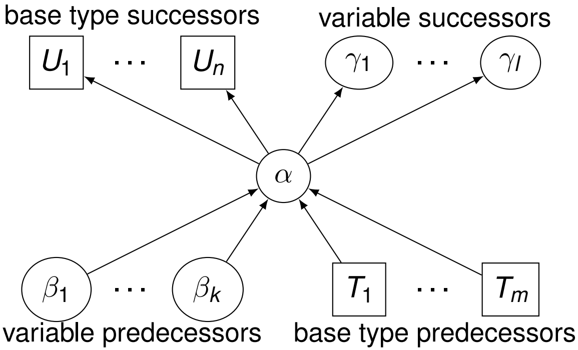


Constraint graph

The subtyping pipeline

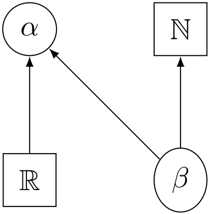


Constraint resolution

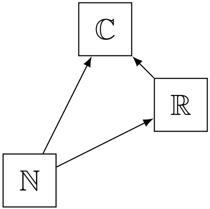


- Compute the intersection of sets of all supertypes of base type predecessors of α
- Assign α the "smallest" type from the intersection
- Check that the assignment is subtype of all base type successors

Constraint resolution (example)

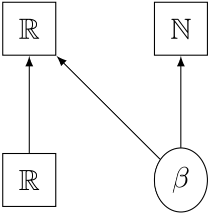


Constraint graph

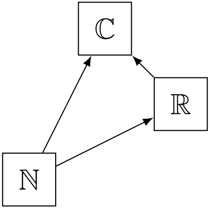


Partial order on base types

Constraint resolution (example)



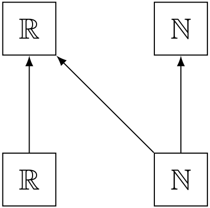
Constraint graph



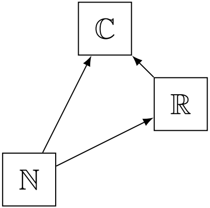
Partial order on base types

- Possibly, the algorithm assigns α the type $\mathbb{R}$ first

Constraint resolution (example)



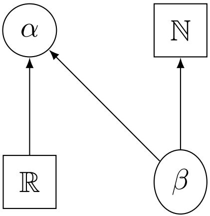
Constraint graph



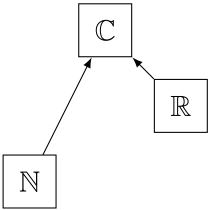
Partial order on base types

- Possibly, the algorithm assigns α the type $\mathbb{R}$ first
- Then β is assigned the infimum of $\{\mathbb{N}, \mathbb{R}\}$

Constraint resolution (example)



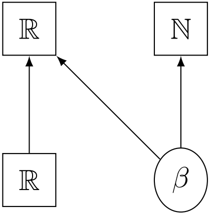
Constraint graph



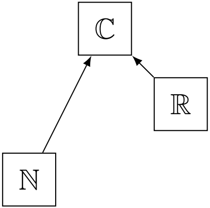
Partial order on base types

- Same constraints, different coercion declarations

Constraint resolution (example)



Constraint graph

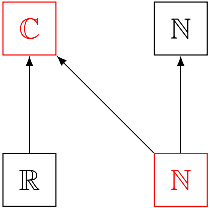


Partial order on base types

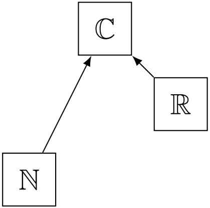
- Same constraints, different coercion declarations
- Then, there is no allowed assignment for β

$\Rightarrow$ Coercion inference fails

Constraint resolution (example)



Constraint graph



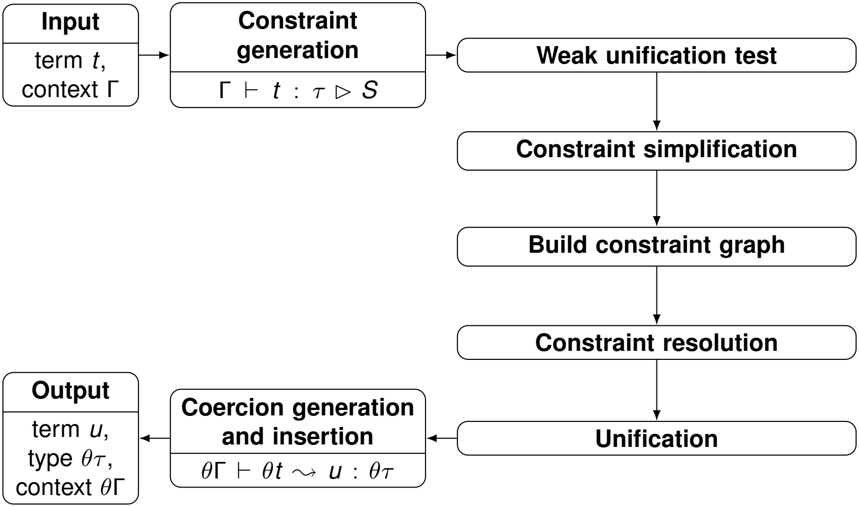
Partial order on base types

- Same constraints, different coercion declarations
- Then, there is no allowed assignment for β

$\Rightarrow$ Coercion inference fails

- But: $\{\alpha \mapsto \mathbb{C},\ \beta \mapsto \mathbb{N}\}$ is a solution

The subtyping pipeline



Correctness & Completeness

- Total correctness
  - The algorithm terminates for any input `t` and Γ
  - The output term `u` has type $\theta\tau$ in context $\theta\Gamma$

Correctness & Completeness

- Total correctness
  - The algorithm terminates for any input `t` and Γ
  - The output term `u` has type $\theta\tau$ in context $\theta\Gamma$
- Completeness
  - Assumption: subtyping relation is a disjoint union of lattices
  - If `t` can be coerced to a well-typed term `u` in the context Γ, then the algorithm will output a term `u'`

Correctness & Completeness

- Total correctness
  - The algorithm terminates for any input $\mathtt{t}$ and Γ
  - The output term $\mathtt{u}$ has type $\theta\tau$ in context $\theta\Gamma$
- Completeness
  - Assumption: subtyping relation is a disjoint union of lattices
  - If $\mathtt{t}$ can be coerced to a well-typed term $\mathtt{u}$ in the context Γ, then the algorithm will output a term $\mathtt{u'}$
  - Can't guarantee $\mathtt{u} = \mathtt{u'}$
  - $\Rightarrow$ refined notion of completeness

Ambiguity example

Example: `sin (- n)`

- Signatures: $\texttt{sin} :: \mathbb{R} \to \mathbb{R}$, $\texttt{-} :: \alpha \to \alpha$ and $\texttt{n} :: \mathbb{N}$
- Declared coercion: $\mathbb{N} <:_{\texttt{real}} \mathbb{R}$

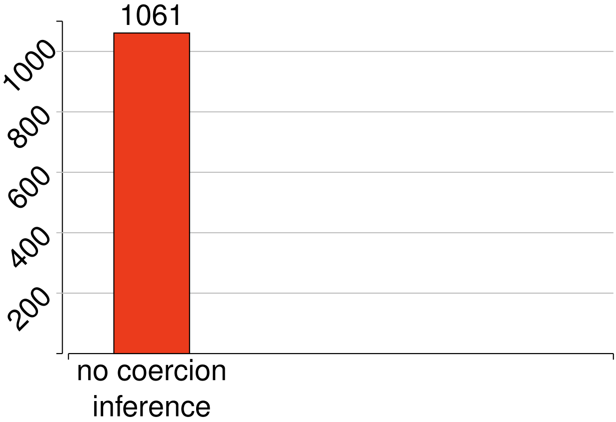
Ambiguity example

Example: `sin (- n)`

- Signatures: `sin` $:: \mathbb{R} \to \mathbb{R}$, `-` $:: \alpha \to \alpha$ and `n` $:: \mathbb{N}$
- Declared coercion: $\mathbb{N} <:_{\texttt{real}} \mathbb{R}$
- Two possible output terms:
  - `sin (real (- n))`
  - `sin (- (real n))`

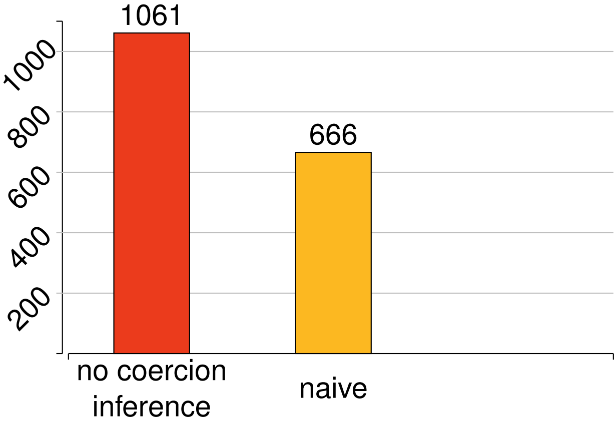
Headache reduction factor

- Necessary coercions in Hölzl's theory



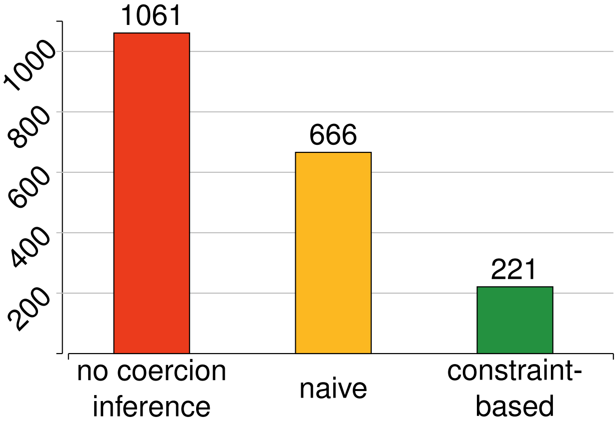
Headache reduction factor

- Necessary coercions in Hölzl's theory



Headache reduction factor

- Necessary coercions in Hölzl's theory



Thank you for your attention!

Questions?

Extending Hindley-Milner Type Inference with Coercive Structural Subtyping

Dmitriy Traytel Stefan Berghofer Tobias Nipkow

APLAS 2011

Technische Universität München

Another ambiguity example

Example: `sin (- n)`

- Signatures: $\texttt{sin} :: \mathbb{R} \to \mathbb{R}$, $\texttt{-} :: \alpha \to \alpha$ and $\texttt{n} :: \mathbb{N}$
- Declared coercions: $\mathbb{N} <:_{\texttt{int}} \mathbb{Z}$, $\mathbb{Z} <:_{\texttt{real}} \mathbb{R}$
- Derived coercion: $\mathbb{N} <:_{\texttt{real} \circ \texttt{int}} \mathbb{R}$

Another ambiguity example

Example: `sin (- n)`

- Signatures: $\texttt{sin} :: \mathbb{R} \to \mathbb{R}$, $- :: \alpha \to \alpha$ and $\texttt{n} :: \mathbb{N}$
- Declared coercions: $\mathbb{N} <:_{\texttt{int}} \mathbb{Z}$, $\mathbb{Z} <:_{\texttt{real}} \mathbb{R}$
- Derived coercion: $\mathbb{N} <:_{\texttt{real} \circ \texttt{int}} \mathbb{R}$
- Two possible output terms:
  - `sin ((real ∘ int) (- n)))`
  - `sin (- ((real ∘ int) n)))`
- Impossible output term:
  - `sin (real (- (int n)))`

Coercive subtyping and *let*-polymorphism

Example: `let f = s in u`
where $\mathtt{s} \equiv \lambda\mathtt{x.\ if\ x > n} \wedge \mathtt{sin\ x > r\ then\ x\ else\ x}$
and $\mathtt{u} \equiv \mathtt{(Suc\ (f\ n),\ f\ r)}$

- Signatures: $\Sigma(\mathtt{sin}) = \mathbb{R} \to \mathbb{R}$, $\Sigma(\mathtt{Suc}) = \mathbb{N} \to \mathbb{N}$, $\Sigma(>) = \alpha \to \alpha \to \mathbb{B}$, $\Sigma(\mathtt{n}) = \mathbb{N}$ and $\Sigma(\mathtt{r}) = \mathbb{R}$
- Declared coercion: $\mathbb{N} <:_{\mathtt{real}} \mathbb{R}$

Coercive subtyping and *let*-polymorphism

Example: `let f = s in u`
where $\texttt{s} \equiv \lambda\texttt{x. if x > n} \wedge \texttt{sin x > r then x else x}$
and $\texttt{u} \equiv \texttt{(Suc (f n), f r)}$

- Signatures: $\Sigma(\texttt{sin}) = \mathbb{R} \to \mathbb{R}$, $\Sigma(\texttt{Suc}) = \mathbb{N} \to \mathbb{N}$, $\Sigma(>) = \alpha \to \alpha \to \mathbb{B}$, $\Sigma(\texttt{n}) = \mathbb{N}$ and $\Sigma(\texttt{r}) = \mathbb{R}$
- Declared coercion: $\mathbb{N} <:_{\texttt{real}} \mathbb{R}$
- Possible types for `s`: $\mathbb{N} \to \mathbb{N}$ and $\mathbb{R} \to \mathbb{R}$
- Any algorithm that only inserts coercions has to choose one type

Coercive subtyping and *let*-polymorphism

Example: `let f = s in u`
where $\texttt{s} \equiv \lambda\texttt{x.\ if\ x\ >\ n} \wedge \texttt{sin\ x\ >\ r\ then\ x\ else\ x}$
and $\texttt{u} \equiv \texttt{(Suc (f n), f r)}$

- Signatures: $\Sigma(\texttt{sin}) = \mathbb{R} \to \mathbb{R}$, $\Sigma(\texttt{Suc}) = \mathbb{N} \to \mathbb{N}$, $\Sigma(>) = \alpha \to \alpha \to \mathbb{B}$, $\Sigma(\texttt{n}) = \mathbb{N}$ and $\Sigma(\texttt{r}) = \mathbb{R}$
- Declared coercion: $\mathbb{N} <:_{\texttt{real}} \mathbb{R}$
- Possible types for `s`: $\mathbb{N} \to \mathbb{N}$ and $\mathbb{R} \to \mathbb{R}$
- Any algorithm that only inserts coercions has to choose one type
- `let f = s in u` is not coercible either way
- On the other hand `u[s/f]` can be coerced