Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Iterative Type Inferencing with Attribute Grammars

Arie Middelkoop

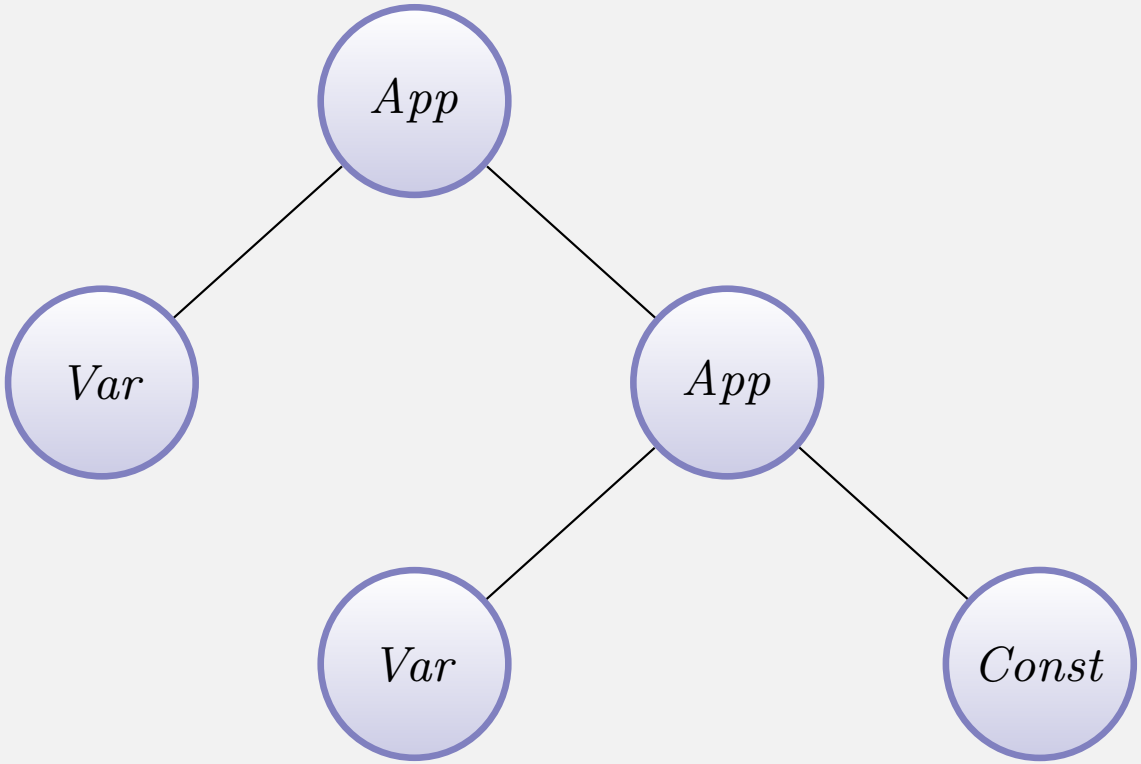
Dept. of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
Web pages: `http://www.cs.uu.nl/wiki/Center`

GPCE '10

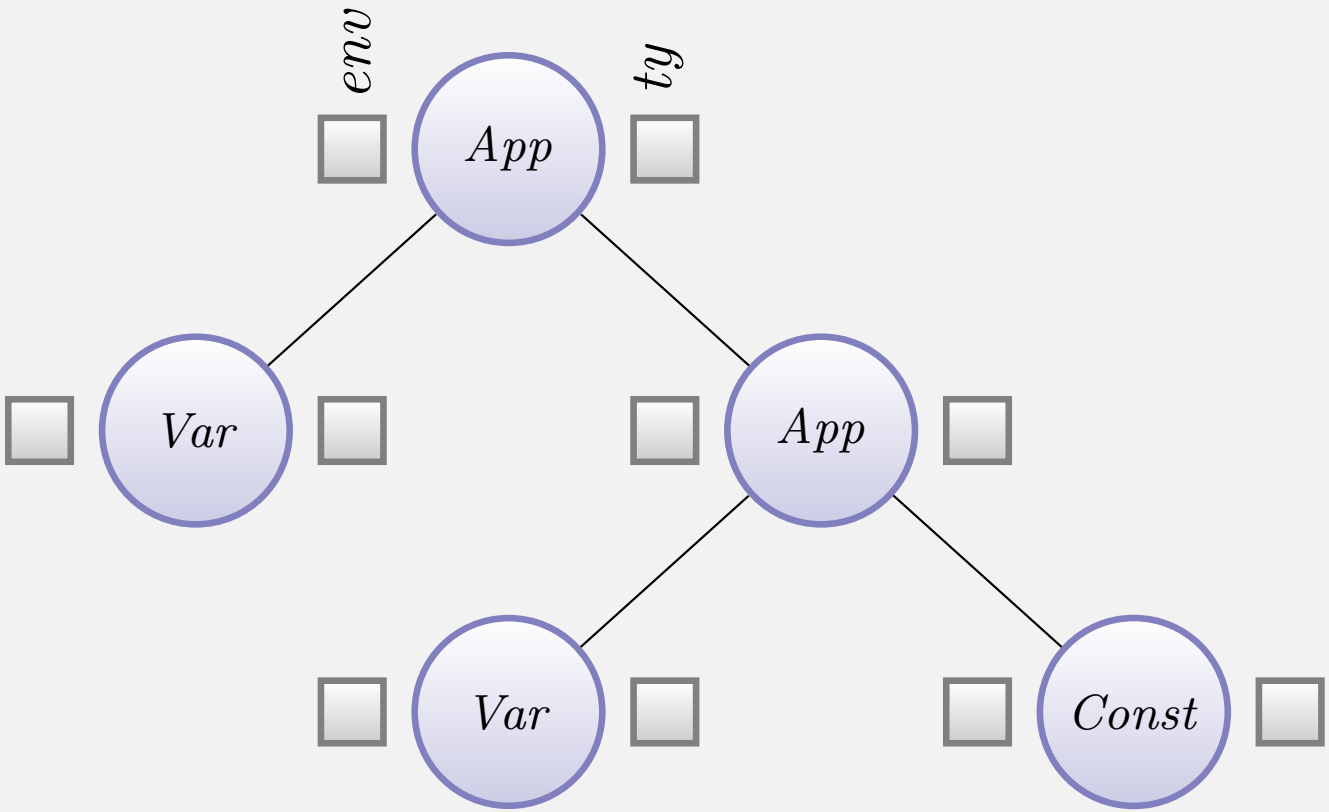
Software Engineering

- Research: Abstract over patterns in Attribute Grammar (AG) descriptions of type inference algorithms
- Patterns related to Choice
- Iteration in a multi-visit AG

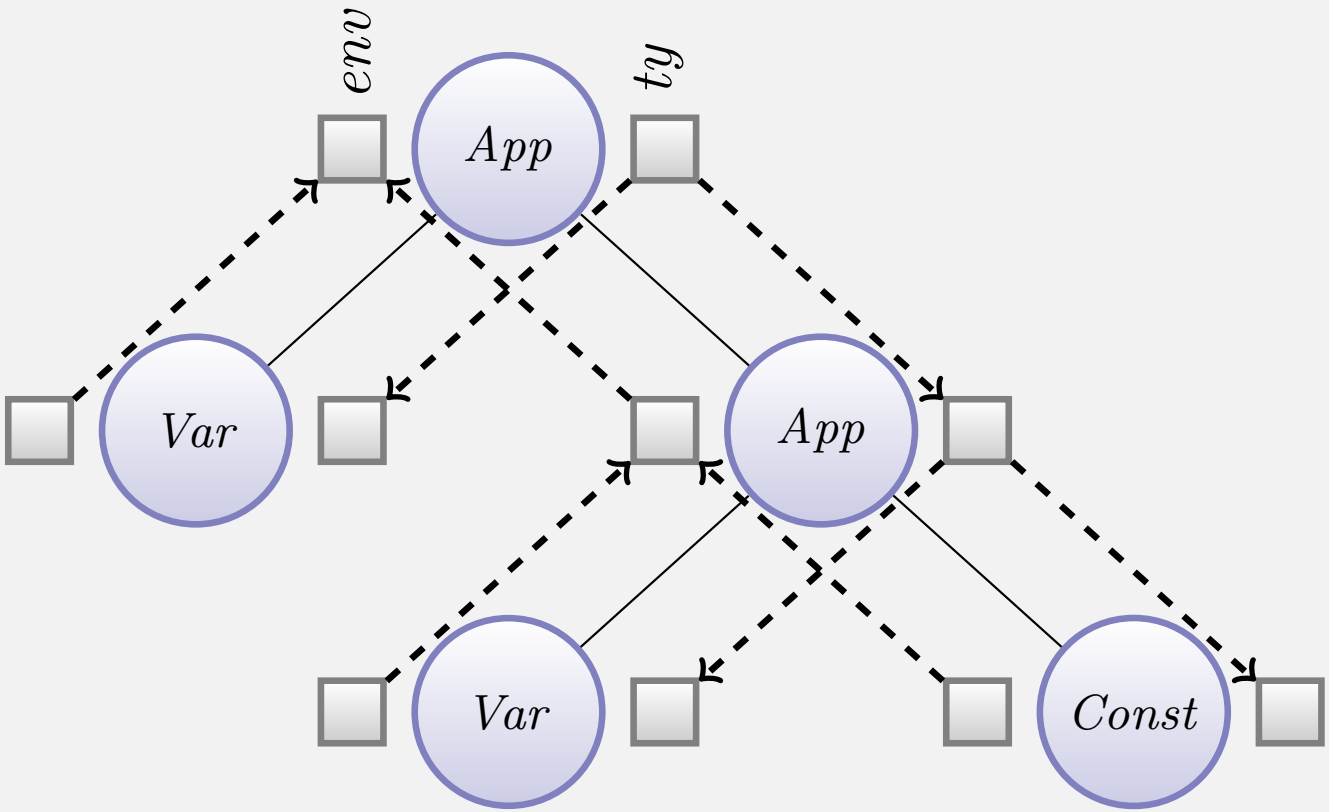
Attribute Grammar: Functions on Attributes



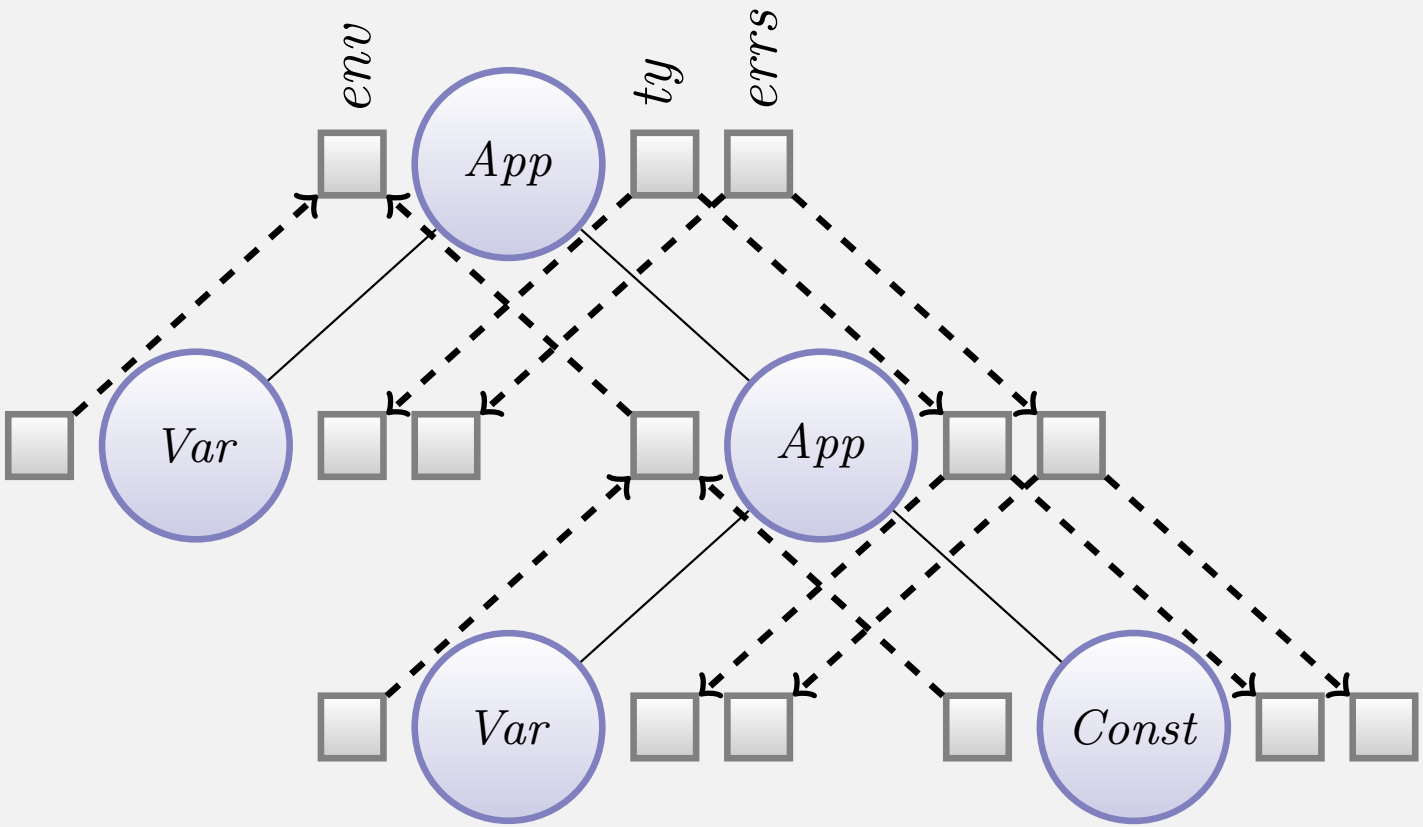
Attribute Grammar: Functions on Attributes



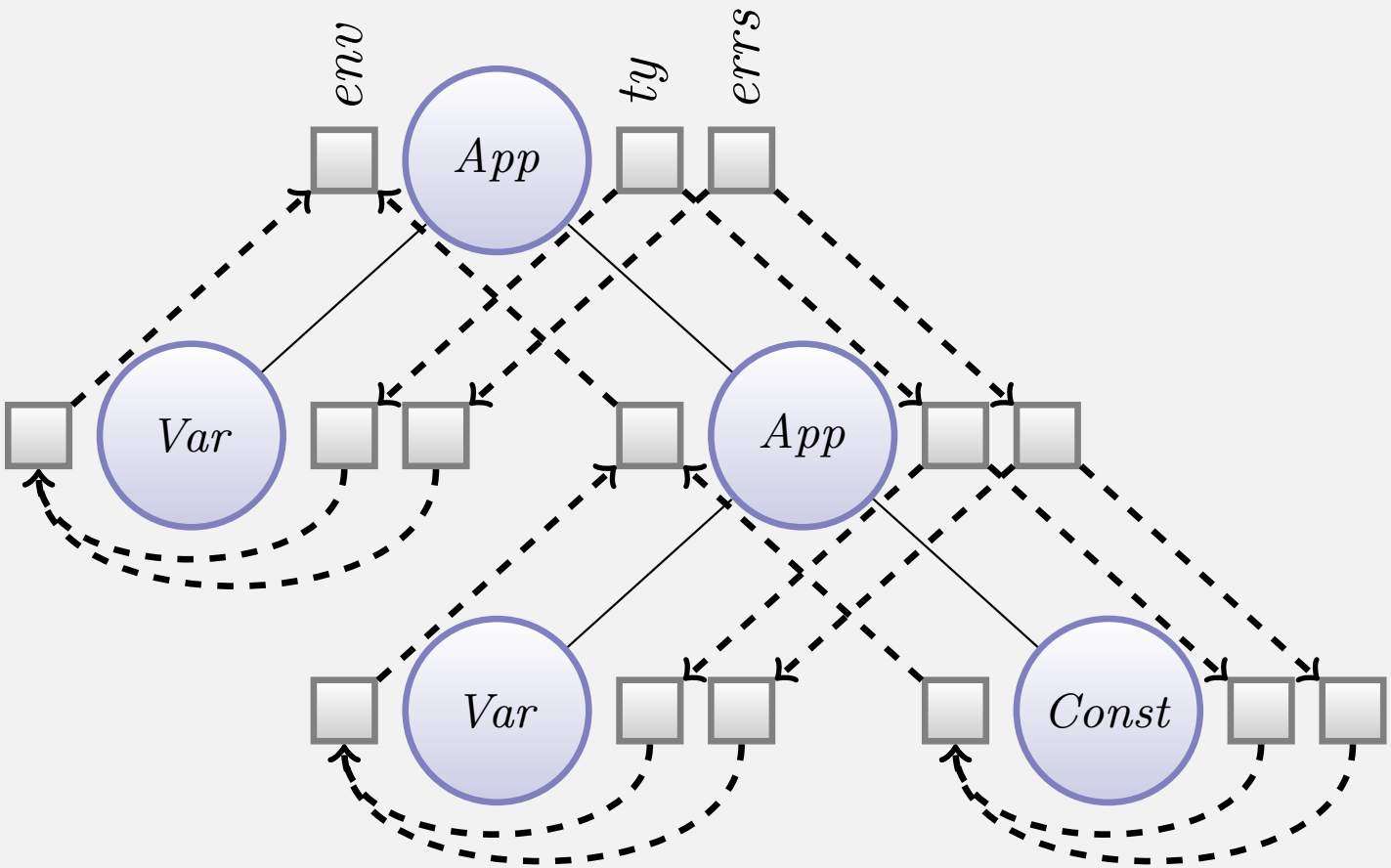
Attribute Grammar: Functions on Attributes



Attribute Grammar: Functions on Attributes



Attribute Grammar: Functions on Attributes

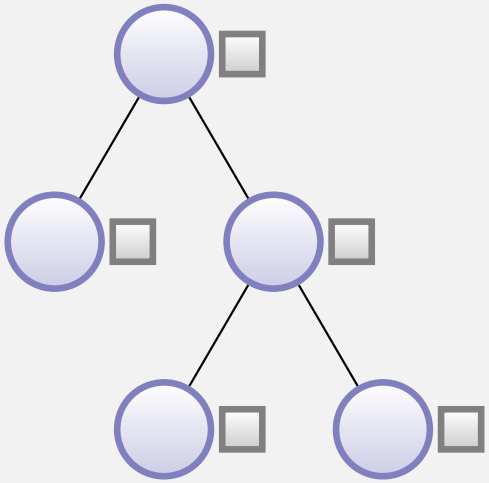


AG Properties

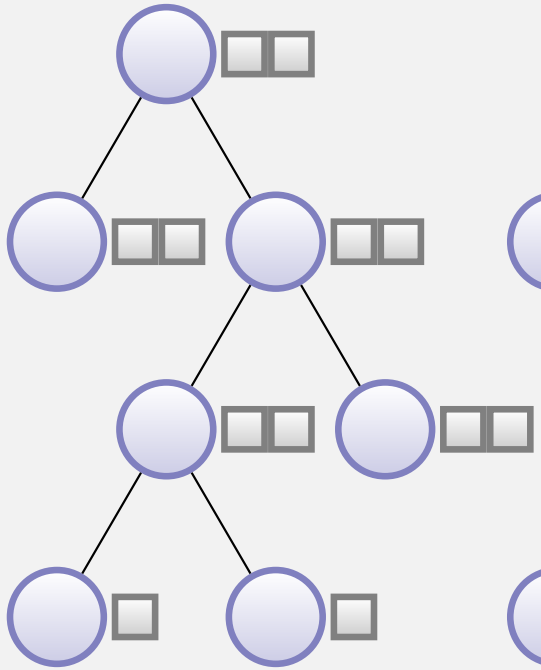
- Evaluation order unspecified: derived automatically
  - Attribute functions may be written in arbitrary order
  - Composition via concatenation
- Visits implicit
  - Sometimes advantagous to have these explicit
- Code completion patterns
  - Insert trivial missing functions for attributes
  - Offers abstractions similar to reader/write/state monads, without affecting evaluation order
- Static guarantees

Rationale for Multi-visit AGs

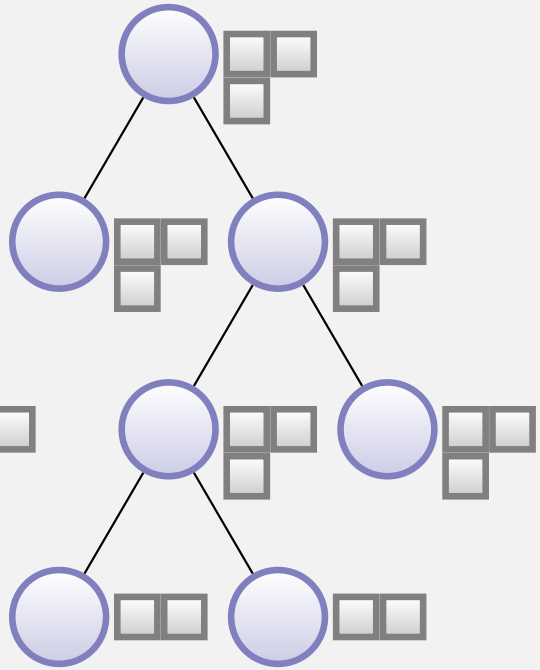
Name Analysis



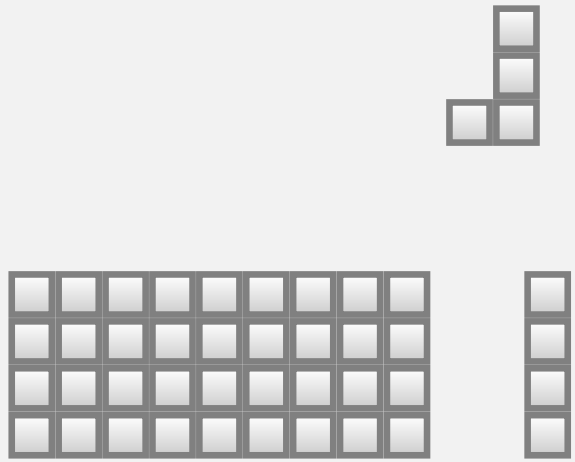
Type Inference



Code Generation



Fit The Pieces Together



Type Rules to AG

$g \vdash e : t$ -- expression type rules

$t_1 \leqslant t_2$ -- subtyping rules

$$\begin{aligned}&\textbf{itf } E \textbf{ inh } g : Env, e : Expr\\ &\qquad\textbf{syn } t : Type\\ &\textbf{itf } U \textbf{ inh } l : Type, r : Type\end{aligned}$$

Type Rules to AG

$g \vdash e : t$ -- expression type rules

$t_1 \leqslant t_2$ -- subtyping rules

$$\begin{aligned}&\textbf{itf } E \textbf{ inh } g : Env, e : Expr\\ &\qquad\textbf{syn } t : Type\\ &\textbf{itf } U \textbf{ inh } l : Type, r : Type\end{aligned}$$

$$\begin{aligned}&\textbf{data } Expr = VarE\ String \mid AppE\ Expr\ Expr\\ &\textbf{data } Type = Arr\ Type\ Type \mid Con\ String\\ &\textbf{type } Env = Map\ String\ Type\end{aligned}$$

Choice Issue

$$\frac{(x,t) \in g}{g \vdash x : t}\ \textsc{Var}$$

$$\begin{array}{ll} Var : E \to \epsilon & (VarE\ x) = lhs.e \\ \quad lhs & lhs.t = lookup\ x\ lhs.g \end{array}$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Choice Issue

$$\frac{(x, t) \in g}{g \vdash x : t}\ \textsc{Var}$$

$$\begin{array}{ll} Var : E \to \epsilon & (VarE\ x) = lhs.e \\ \quad lhs & lhs.t = lookup\ x\ lhs.g \end{array}$$

$$\frac{g \vdash f : tf \qquad g \vdash a : ta \qquad g \vdash tf \leqslant (ta \to s)}{g \vdash f\ a : s}\ \textsc{App}$$

$$\begin{array}{llll} App : E & \to E & E & U \\ \quad lhs & f & a & u \end{array} \qquad \begin{array}{ll} (AppE\ f.e\ a.e) = lhs.e & \\ f.g = lhs.g, & a.g = lhs.g \\ u.l = f.t, & u.r = a.t\ `Arr`\ s \\ lhs.t = s & \end{array}$$

Potential Consequences of Choice

$$t \leqslant t \ \textsc{Refl}$$

$$Refl : \underset{lhs}{U \to \epsilon} \qquad lhs.l \equiv lhs.r$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Potential Consequences of Choice

$$t \leqslant t \ \textsc{Refl}$$

$$\mathit{Refl} : \underset{\mathit{lhs}}{U \to \epsilon} \qquad \mathit{lhs}.l \equiv \mathit{lhs}.r$$

$$\frac{t_3 \leqslant t_1 \qquad t_2 \leqslant t_4}{(t_1 \to t_2) \leqslant (t_3 \to t_4)} \ \textsc{Arr}$$

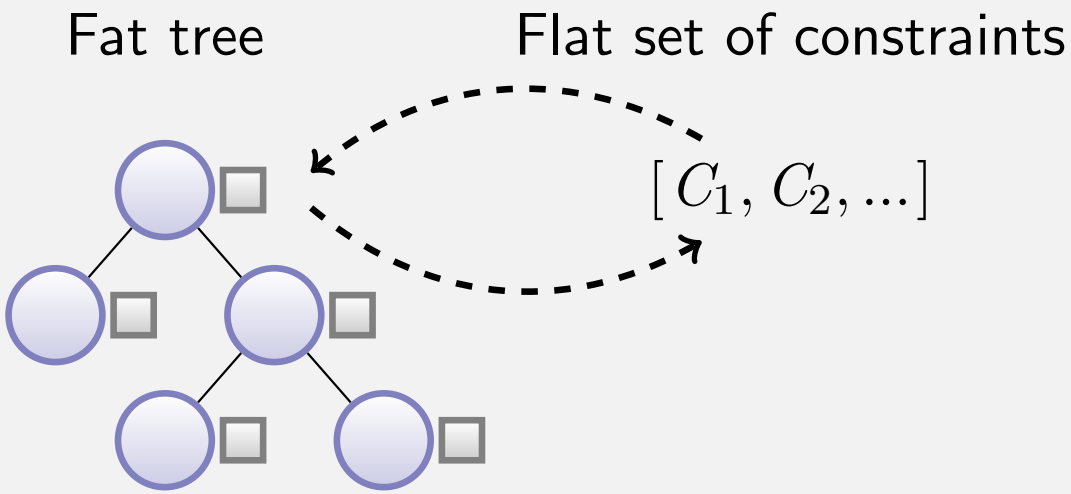
$$\mathit{Arr} : \underset{\mathit{lhs}}{U} \to \underset{p}{U} \ \underset{q}{U} \qquad \begin{aligned} \mathit{Arr}\ q.l\ p.l &= \mathit{lhs}.l \\ \mathit{Arr}\ p.r\ q.r &= \mathit{lhs}.r \end{aligned}$$

Choice: What Type To Choose?

Two strategies to guess a type:

- Active: try all possibilties, backtrack if this does not work out later
  - Infrequently employed in inference algorithms
  - Reasons: efficiency, error messages, principal typing
- Passive: fill in a place holder, delay the choice until the value is fixed later by unifications
  - Placeholders for inhs attrs used as syns, unification for syn attrs treated as inhs
  - Employed by all unification-based algorithms
  - Delayed judgments: encoded as constraints
  - Iteratively: fixpoint iteration
  - Defaulting: By absence of any demands on the place holder, actively assign a default value

The Curse of Constraints



- Advantage: constraints are first-class values, and can for example occur in types.
- Decoupling of judgments requires bidirectional transport of information:
  - context-information for error messages
  - constructed proofs e.g. needed for code gen

AG-only solution

Goal:

- Delayable judgements
- Fixpoint iteration

Non-solution:

- Encode substitution as cyclic attributes?
- Defaulting is a side-effectful operation
- lack of control where to iterate

Proposed solution:

- Explicitly iterate visits
- Detachable visits
- Visit-local attributes

AG With Explicit Visits

itf *Infer*
  visit *Analyze*
    inh e : *Expr*
    inh g : *Env*
    syn *errs1* : *Errs*
  visit *Process*
    inh *subst* : *Subst*
    syn *subst* : *Subst*

Notation For Rules

-- rules

$r ::= p = e$ -- conventional AG-rule

$\mid$ **match** $p = e$ -- backtracking variation

$\mid$ **child** $c : I = e$ -- (higher-order) child

e -- right-hand side

c -- child name

I -- interface name

-- patterns

$p ::= c.x$ -- attribute reference

$\mid$ $C\ \overline{p}$ -- constructor match

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Notation For Productions

$$
\begin{aligned}
&nonterm\_Expr :: Sem\ Infer \\
&nonterm\_Expr = \\
&\quad \mathbf{sem} : Infer \\
&\qquad \mathbf{clause}\ e.var \\
&\qquad\quad \mathbf{match}\ (Expr\_Var\ loc.x) = lhs.e \\
&\qquad\quad \ldots \\
&\qquad \mathbf{clause}\ e.app \\
&\qquad\quad \mathbf{match}\ (Expr\_App\ loc.fe\ loc.ae) = lhs.e \\
&\qquad\quad \mathbf{child}\ f : Infer = nonterm\_Expr \\
&\qquad\quad \mathbf{child}\ a : Infer = nonterm\_Expr \\
&\qquad\quad f.e = loc.fe \\
&\qquad\quad a.e = loc.ae \\
&\qquad\quad \ldots
\end{aligned}
$$

Explicit Invocation

invoke x **of** c -- execute visit x of child c

clause $e.app$

...

$f.g = lhs.g$

$a.g = lhs.g$

invoke $infer$ **of** f

invoke $infer$ **of** a

Iterated Invoctation

$$\textbf{invoke}\ \mathit{infer}\ \textbf{of}\ a = \mathit{repeat}$$

$$\begin{aligned}
&\mathit{repeat} :: \mathit{CoSem}\ \mathit{Expr}\ \mathit{Process}\\
&\mathit{repeat} =\\
&\quad \mathit{cosem} : \mathit{Expr}\ \mathit{Process}\\
&\qquad \textbf{clause}\ e.\mathit{repeat}\\
&\qquad\quad \textbf{match}\ \mathit{True} = \mathit{isChanged}\ \mathit{lhs.subst}\\
&\qquad\quad \mathit{lhs.subst} = \mathit{lhs.subst}
\end{aligned}$$

Advantages

- Keeps the dependency-graph cycle-free
- Pointpoint where to iterate, and under what conditions
- Construct parts of the tree while iterating
- Store visit-local attributes to save information about an iteration

Conclusion

- AGs are a useful language to describe type inferencing algorithms
- Type inferencing is not syntax-directed
- Type inferencing requires choice/guessing/unknowns
- Iterative inferencing algorithms can be described with multi-visit AGs

More material: `http://www.cs.uu.nl/~ariem/thesis.pdf` (under construction...)