Universiteit Utrecht

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

Dependently-Typed Attribute Grammars

Arie Middelkoop

in cooperation with:
S. Doaitse Swierstra
Atze Dijkstra

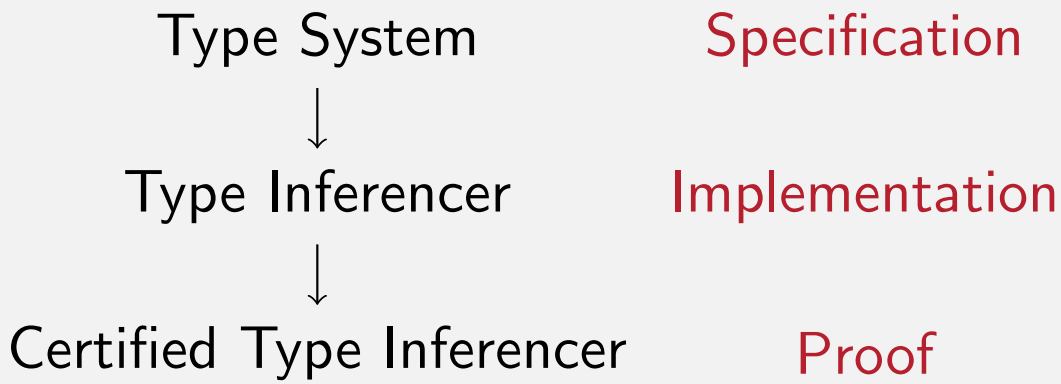
Dept. of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
Web pages: `http://www.cs.uu.nl/wiki/Center`

IFL 2010
01 Sep 2010

Introduction

- Attribute Grammar
- Strongly-typed attributes
- Ad-hoc: attributes with polymorphic types
- Now: attributes with dependent types

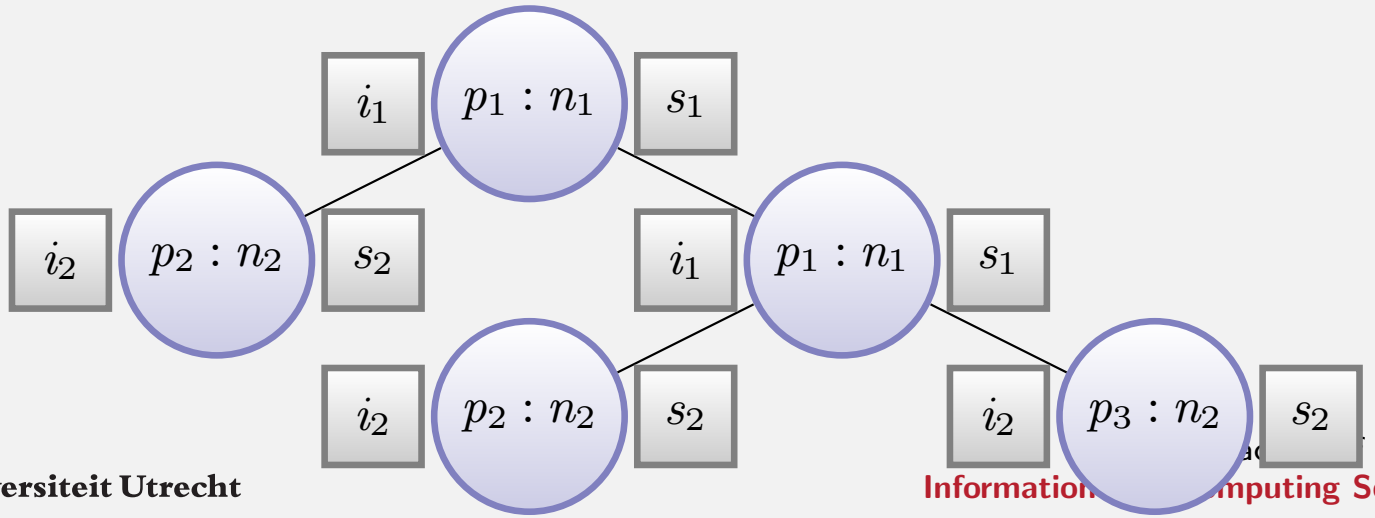
Research



With Attribute Grammars?

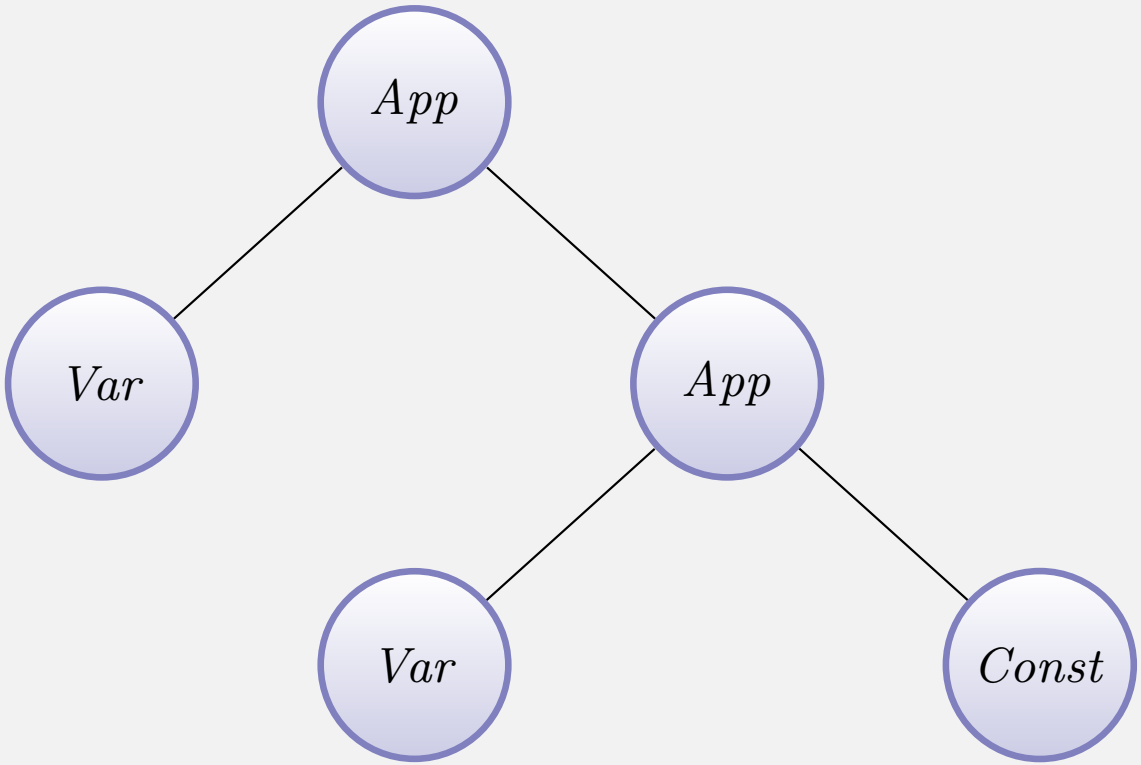
Attribute Grammars

- Context-free grammar
  - (Non)terminals
  - Productions
  - Parse tree, derivation tree
- Attributes
  - Parameters of nonterminals
  - Inherited attributes
  - Synthesized attributes
- Rules
  - Functions between attributes per production



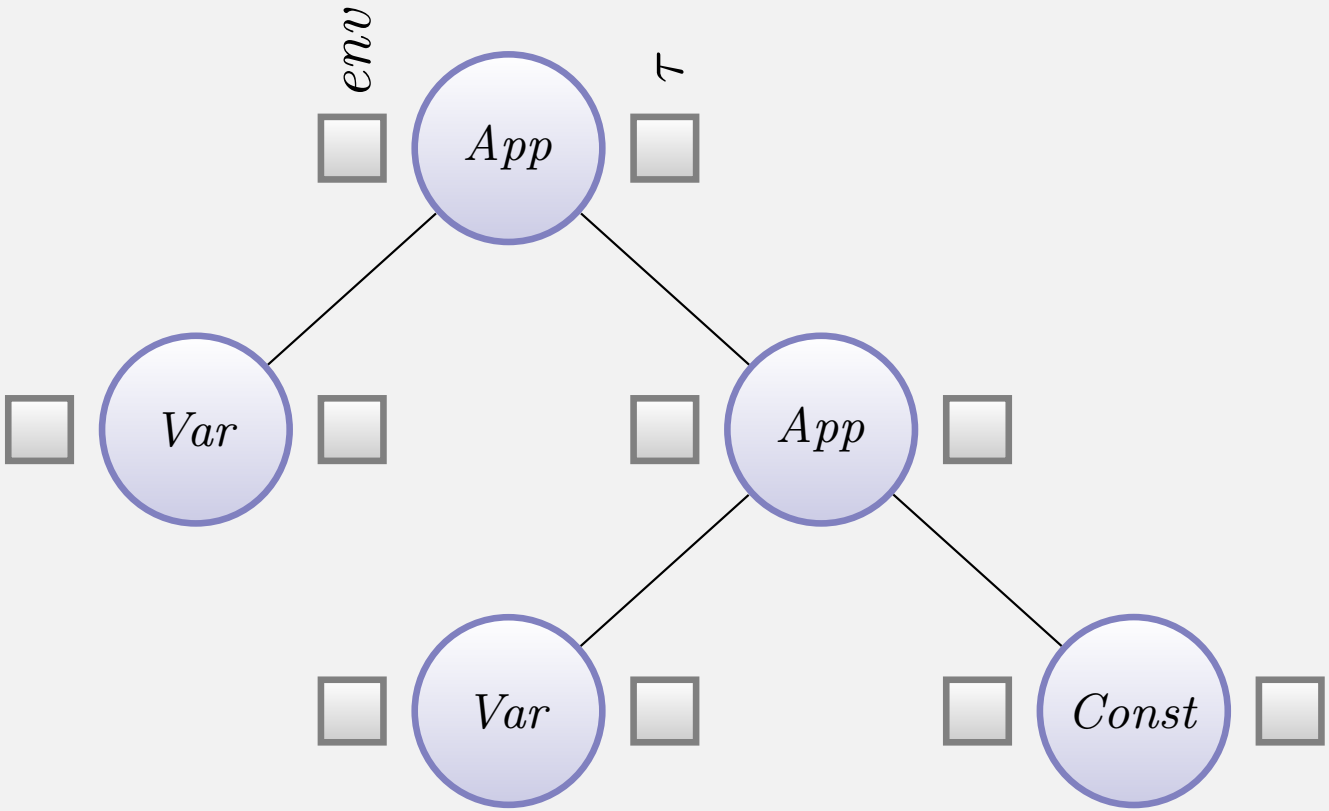
Functional Model

- AGs: attributes as a function of other attributes



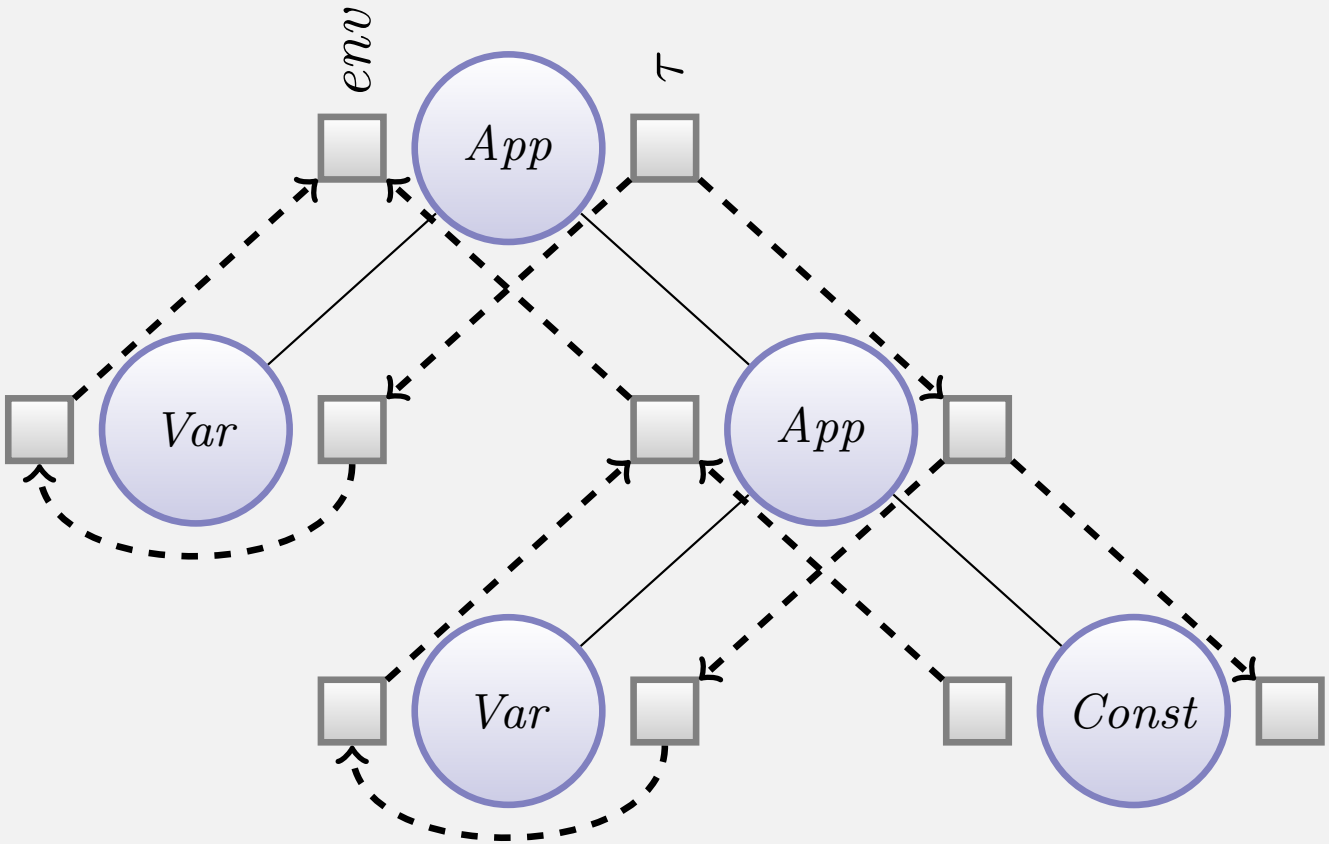
Functional Model

- AGs: attributes as a function of other attributes



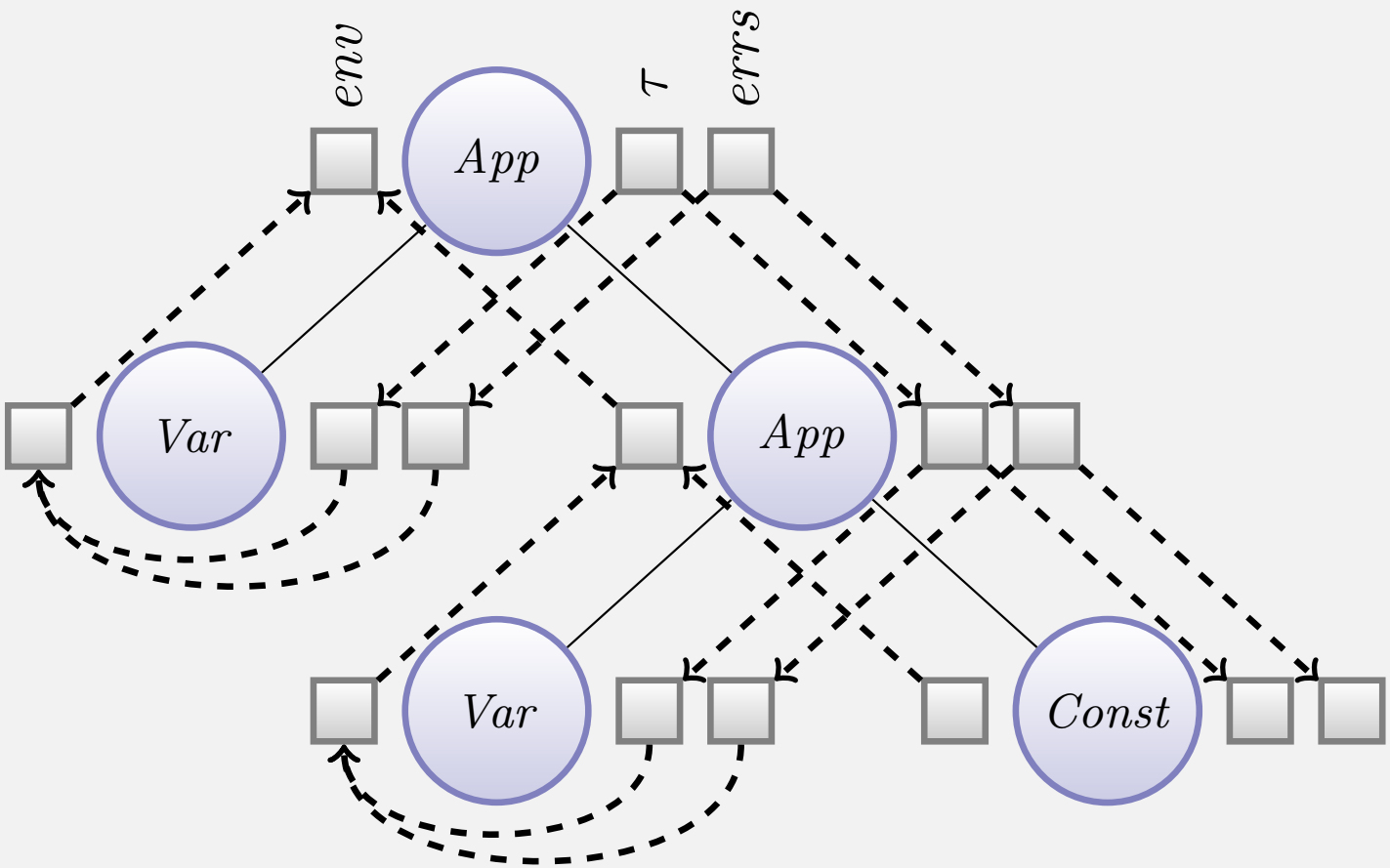
Functional Model

- AGs: attributes as a function of other attributes



Functional Model

- AGs: attributes as a function of other attributes



Just Add Some Extra Attributes (tm)

- Separate Specification
- Implicit Rules

$$\text{Algorithm} = \text{Evaluation Strategy} + \text{Extra Attributes}$$

$$\text{Algorithm} = \text{Evaluation Strategy} + \text{Extra Attributes}$$

With a Higher-Order AG,

$$\text{Evaluation Strategy} = \text{Basic Strategy} + \text{Extra Attributes}$$

With dependently-typed programming:

$$\text{Proof} = \text{Program} + \text{quite a bit More Program}$$

Challenges

In theory:

- Termination
  - Ordered Attribute Grammars
  - Total attribute functions
- Attributes may take types as values
- The type of an attribute may refer to an attribute

In practice (see paper):

- Attribute functions not total
- Reason: synthesized attributes not independent
- Example: suppose attribute tp and attribute $errs$, such that former is defined iff the latter is empty

Example

- Compiler written in Agda, using AGs
- Translates some source AST to a target AST
- Source and target language: sequence of def/use sites

Source Language

$$
\begin{array}{lllll}
\textbf{grammar}\ \mathit{Source} : \mathit{Set} & & & & \\
\quad \textbf{prod}\ \mathit{use} & \textbf{term} & & \mathit{nm} & : \mathit{Ident} \\
\quad \textbf{prod}\ \mathit{def} & \textbf{term} & & \mathit{nm} & : \mathit{Ident} \\
\quad \textbf{prod}\ \_ \diamond \_ & \textbf{nonterm} & \mathit{left} & & : \mathit{Source} \\
 & \textbf{nonterm} & \mathit{right} & & : \mathit{Source}
\end{array}
$$

$$\mathit{def}\ x \diamond \mathit{def}\ y \diamond \mathit{use}\ x \diamond \mathit{use}\ y \diamond \mathit{use}\ x$$

Target Language

$$
\begin{aligned}
&\underline{\text{data}}\ \mathit{Target} : (\Gamma : \mathit{Env}) \to \mathit{Set}\ \underline{\text{where}} \\
&\quad \mathit{def}\ \ : (\mathit{nm} : \mathit{Ident}) \to (\mathit{nm} \in \Gamma) \to \mathit{Target}\ \Gamma \\
&\quad \mathit{use}\ \ : (\mathit{nm} : \mathit{Ident}) \to (\mathit{nm} \in \Gamma) \to \mathit{Target}\ \Gamma \\
&\quad \_ \diamond \_ : (\mathit{left} : \mathit{Target}\ \Gamma) \to (\mathit{right} : \mathit{Target}\ \Gamma) \\
&\qquad\qquad\qquad\qquad \to \mathit{Target}\ \Gamma
\end{aligned}
$$

$$
\begin{aligned}
&\underline{\text{data}}\ \mathit{Err} : \mathit{Env} \to \mathit{Set}\ \mathbf{where} \\
&\quad \mathit{scope} : (\mathit{nm} : \mathit{Ident}) \to \neg(\mathit{nm} \in \Gamma) \to \mathit{Err}\ \Gamma
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{Errs} : \mathit{Env} \to \mathit{Set} \\
&\mathit{Errs}\ \mathit{env} = \mathit{List}\ (\mathit{Err}\ \mathit{env})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{Env} : \mathit{Set} \\
&\mathit{Env} = \mathit{List}\ \mathit{Ident}
\end{aligned}
$$

Interface of *Source* nonterminal

$$
\begin{array}{l}
\textbf{itf}\ \mathit{Source} \\
\quad \textbf{visit}\ \mathit{analyze} \\
\qquad \textbf{syn}\ \mathit{gathEnv} \quad : \mathit{Env} \\
\quad \textbf{visit}\ \mathit{translate} \\
\qquad \textbf{inh}\ \mathit{finEnv} \quad : \mathit{Env} \\
\qquad \textbf{inh}\ \mathit{gathInFin} : \mathit{syn.gathEnv} \sqsubseteq \mathit{inh.finEnv} \\
\qquad \textbf{syn}\ \mathit{outcome} \quad : (\mathit{Errs}\ \mathit{inh.finEnv}) \uplus \\
\qquad\qquad\qquad\qquad\ \ (\mathit{Target}\ \mathit{inh.finEnv})
\end{array}
$$

Support Code

$$\underline{\mathsf{data}}\ \_ \sqsubseteq \_ : (\Gamma_1 : Env) \to (\Gamma_2 : Env) \to Set\ \underline{\mathsf{where}}$$
$$\begin{aligned} \quad trans &: \Gamma_1 \sqsubseteq \Gamma_2 \to \Gamma_2 \sqsubseteq \Gamma_3 \to \Gamma_1 \sqsubseteq \Gamma_3 \\ \quad subLeft &: \Gamma_1 \sqsubseteq (\Gamma_1 +\!\!+ \Gamma_2) \\ \quad subRight &: \Gamma_2 \sqsubseteq (\Gamma_1 +\!\!+ \Gamma_2) \end{aligned}$$

$$\underline{\mathsf{data}}\ \_ \in \_ : (nm : Ident) \to (\Gamma : Env) \to Set\ \underline{\mathsf{where}}$$
$$\begin{aligned} \quad here &: nm \in (nm :: \Gamma') \\ \quad next &: nm \in \Gamma \to nm \in (nm' :: \Gamma) \end{aligned}$$

$$inSubset : (\Gamma' \sqsubseteq \Gamma) \to nm \in \Gamma' \to nm \in \Gamma$$

Semantics - 1

$$
\begin{array}{l}
\textbf{datasem } \mathit{Source} \\
\quad \textbf{prod } \mathit{use} \\
\qquad \mathit{lhs.gathEnv} = [\,] \\
\qquad \mathit{lhs.outcome} \ \underline{\textsf{with}} \ \mathit{loc.nm} \in_? \mathit{lhs.finEnv} \\
\qquad\qquad\qquad\qquad \mid \mathit{inj}_1 \ \mathit{notIn} = \mathit{inj}_1 \ [\mathit{scope} \ \mathit{loc.nm} \ \mathit{notIn}] \\
\qquad\qquad\qquad\qquad \mid \mathit{inj}_2 \ \mathit{isIn} \ \ = \mathit{inj}_2 \ (\mathit{use} \ \mathit{loc.nm} \ \mathit{isIn}) \\
\quad \textbf{prod } \mathit{def} \\
\qquad \mathit{lhs.gathEnv} = [\mathit{loc.nm}] \\
\qquad \mathit{loc.inFin} \ \ = \mathit{inSubset} \ \mathit{lhs.gathInFin} \ \mathit{here} \\
\qquad \mathit{lhs.outcome} = \mathit{inj}_2 \ (\mathit{def} \ \mathit{loc.nm} \ \mathit{loc.inFin})
\end{array}
$$

Semantics - 2

$$
\begin{array}{l}
\textbf{datasem } \mathit{Source} \\
\quad \textbf{prod } \_ \diamond \_ \\
\qquad \mathit{left.finEnv} = \mathit{lhs.finEnv} \\
\qquad \mathit{right.finEnv} = \mathit{lhs.finEnv} \\
\qquad \mathit{lhs.gathEnv} = \mathit{left.gathEnv} +\!\!+ \mathit{right.gathEnv} \\
\qquad \mathit{left.gathInFin} = \mathit{trans}\ \mathit{subLeft}\ \mathit{lhs.gathInFin} \\
\qquad \mathit{right.gathInFin} = \mathit{trans}\ (\mathit{subRight}\ \mathit{lhs.finEnv})\ \mathit{lhs.gathInFin} \\
\qquad \mathit{lhs.outcome} \quad \underline{\mathsf{with}}\ \mathit{left.outcome} \\
\qquad\quad \mid \mathit{inj}_1\ \mathit{es}\ \underline{\mathsf{with}}\ \mathit{right.outcome} \\
\qquad\quad \mid \mathit{inj}_1\ \mathit{es}_1 \mid \mathit{inj}_1\ \mathit{es}_2 = \mathit{inj}_1\ (\mathit{es}_1 +\!\!+ \mathit{es}_2) \\
\qquad\quad \mid \mathit{inj}_1\ \mathit{es}_1 \mid \mathit{inj}_2\ \_ \quad = \mathit{inj}_1\ \mathit{es}_1 \\
\qquad\quad \mid \mathit{inj}_2\ t_1 \quad \underline{\mathsf{with}}\ \mathit{left.outcome} \\
\qquad\quad \mid \mathit{inj}_2\ t_1 \quad \mid \mathit{inj}_1\ \mathit{es}_1 = \mathit{inj}_1\ \mathit{es}_1 \\
\qquad\quad \mid \mathit{inj}_2\ t_1 \quad \mid \mathit{inj}_2\ t_2 \quad = \mathit{inj}_2\ (t_1 \diamond t_2)
\end{array}
$$

Cool Stuff

- Represent (Dependent) Data Types as Attribute Grammar
  - Parameters of data types are attributes
  - E.g. $Target$ can be specified with AGs
- Model for representing type variables for AGs in a polymorphically typed language
  - Inherited type-attr: universally quantified type var
  - Synthesized type-attr: existential type var
  - The type of such an attribute: a constraint (e.g. type class)

Towards a Certified Compiler

- Ensure termination
- Gradually prove more properties by adding additional attributes
- Negative side of the coin: manual labour, still infeasible for large compilers
- Unless we have reusable building blocks...
  - Ongoing work on Nested Attribute Grammars

Conclusion

- Dependently-typed AGs: proofs are extra attributes
- Just Add Extra Attributes (tm)
- Advantage: specify total attr functions between attributes, then evaluation code and termination proof for free.