Universiteit Utrecht

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

Stepwise Attribute Grammar Evaluation
Or: Tweaking AG Evaluation

Arie Middelkoop

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

LDTA, 27 March '11

Introduction

Contents of talk:

- Computations over tree structures with attribute grammars
- Crazy Idea: Control evaluation!
- Different setting: construct tree while evaluating attributes
- Deal with: BFS, side-effect, graphs, parallelism

- Type inference: proof search
- Breadth-first mini-max

- Implementation in UUAG (using Haskell)
- Proof of concept Java example
- Extended version: `www.cs.uu.nl/~ariem/thesis.pdf`

Relation to Yesterday's Talks

- Control stategies to direct evaluation of children; in an AG, such strategies are implicit
- Relation to Rinus' workflows.

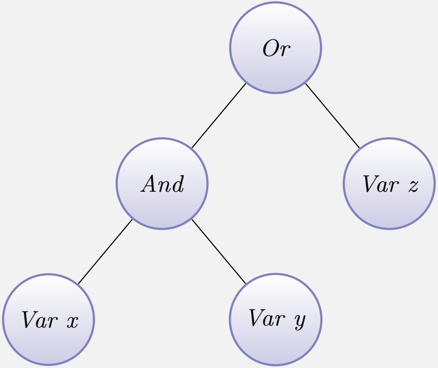
What is an Attribute Grammar? (notation)

$$
\begin{array}{l}
\textbf{gram}\ \mathit{Pred} \qquad \text{-- grammar} \\
\quad \textbf{prod}\ \mathit{Var}\ \textbf{term}\ \mathit{nm} :: \mathit{String} \\
\quad \textbf{prod}\ \mathit{Or}\ \mathit{And} \\
\qquad \textbf{nonterm}\ p : \mathit{Pred} \\
\qquad \textbf{nonterm}\ q : \mathit{Pred}
\end{array}
$$

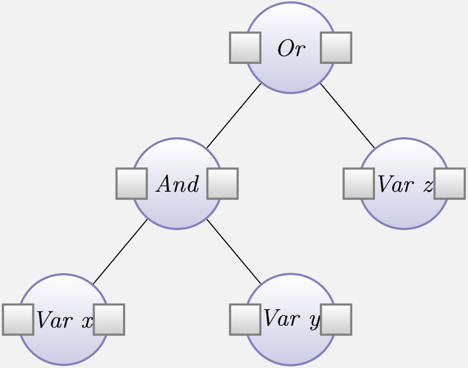
$$
\begin{array}{l}
\textbf{attr}\ \mathit{Pred} \qquad \text{-- attributes} \\
\quad \textbf{inh}\ \mathit{env} :: \mathit{Map}\ \mathit{String}\ \mathit{Bool} \\
\quad \textbf{syn}\ \mathit{val} :: \mathit{Bool}
\end{array}
$$

$$
\begin{array}{ll}
\textbf{sem}\ \mathit{Pred} & \text{-- rules} \\
\quad \textbf{prod}\ \mathit{Var} & \mathit{lhs.val} = \mathit{find}\ \mathit{nm}\ \mathit{lhs.env} \\
\quad \textbf{prod}\ \mathit{Or} & \mathit{lhs.val} = p.\mathit{val} \lor q.\mathit{val} \\
\quad \textbf{prod}\ \mathit{And} & \mathit{lhs.val} = p.\mathit{val} \land q.\mathit{val} \\
\quad \textbf{prod}\ \mathit{Or}\ \mathit{And} & \\
\qquad p.\mathit{env} \quad = \mathit{lhs.env} & \\
\qquad q.\mathit{env} \quad = \mathit{lhs.env} &
\end{array}
$$

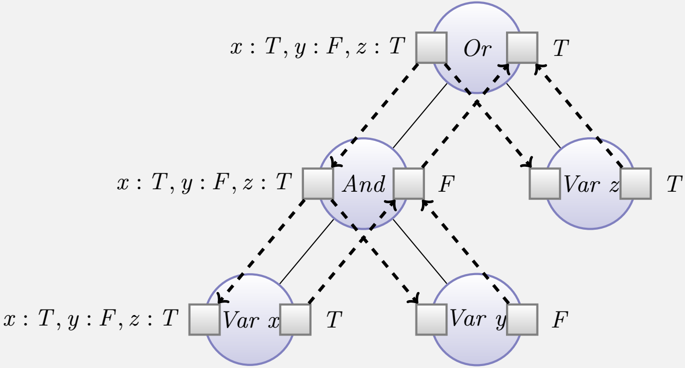
Visualization



Visualization



Visualization



What is an Attribute Grammar? (model)

- Rules: (Pure) functions between attributes
- Declarative!

What is an Attribute Grammar? (model)

- Rules: (Pure) functions between attributes
- Declarative! Evaluation algorithm?

Freedom: several algorithms with different properties.

What is an Attribute Grammar? (model)

- Rules: (Pure) functions between attributes
- Declarative! Evaluation algorithm?

Freedom: several algorithms with different properties.

- On-demand evaluation
  - Evaluator performs the least evaluation for an attribute
  - As supported by UUAG, JastAdd, Silver, ...

What is an Attribute Grammar? (model)

- Rules: (Pure) functions between attributes
- Declarative! Evaluation algorithm?

Freedom: several algorithms with different properties.

- On-demand evaluation
  - Evaluator performs the least evaluation for an attribute
  - As supported by UUAG, JastAdd, Silver, ...
- But also: eager evaluation
  - Evaluator dictates evaluation order
  - Kennedy-Warren '76
  - Kastens '80

While Working on my Ph.D...

Type inference seems a typical task for AGs. Nice example: UHC.

However, what about:

- Proof structure deviates from AST structure
- Multiple candidate solutions
- Sharing in proofs - graphs?
- Information about type variables discovered during evaluation. How to distribute this information? Is a single pass sufficient?

While Working on my Ph.D...

Type inference seems a typical task for AGs. Nice example: UHC.

However, what about:

- Proof structure deviates from AST structure
- Multiple candidate solutions
- Sharing in proofs - graphs?
- Information about type variables discovered during evaluation. How to distribute this information? Is a single pass sufficient?

Are these issues only related to type inference?

My Everyday Problems...

- Layout algorithms for hierarchical HTML menus
- Compute back edges of control flow graph
- In an AG for aspect-oriented programming, independent computations for each joint point.
- Operational semantics for a language with a nondeterministic choice

Remarkable similarities

My Everyday Problems...

- Layout algorithms for hierarchical HTML menus
  - Side Effect!
- Compute back edges of control flow graph
  - Graph node has multiple parents
  - However, depth-first traversal can be represented as a tree
- In an AG for aspect-oriented programming, independent computations for each joint point.
  - Parallelism!
- Operational semantics for a language with a nondeterministic choice
  - Breadth-first evaluation!

Remarkable similarities

Reflection

- A nice and essential aspect of AGs is that the evaluation order of rules is implicit.
- Consequently, there are algorithms that we would like to express as AGs, but cannot do so straightforwardly.

Reflection

- A nice and essential aspect of AGs is that the evaluation order of rules is implicit.
- Consequently, there are algorithms that we would like to express as AGs, but cannot do so straightforwardly.
- Can we control the evaluation order while keeping the advantages of AGs? (unordered rules, compositionality)

Visits to Children Explicit

Mix AGs with Visitors

- Be able to describe visits to children
- Be able to restrict their relative order
- GPCE'10 paper

$$\begin{array}{l} \textbf{attr}\ \mathit{Pred}\ \textbf{visit}\ \mathit{eval} \\ \quad \textbf{inh}\ \mathit{env} :: \mathit{Map}\ \mathit{String}\ \mathit{Bool} \\ \quad \textbf{syn}\ \mathit{val} :: \mathit{Bool} \end{array}$$

$$\begin{array}{l} \textbf{sem}\ \mathit{Pred} \mid \mathit{Or}\ \textbf{visit}\ \mathit{eval} \\ \quad \textbf{invoke}\ \mathit{eval}\ \textbf{of}\ q \\ \quad \textbf{invoke}\ \mathit{eval}\ \textbf{of}\ p \end{array}$$

Mix AGs with Visitors

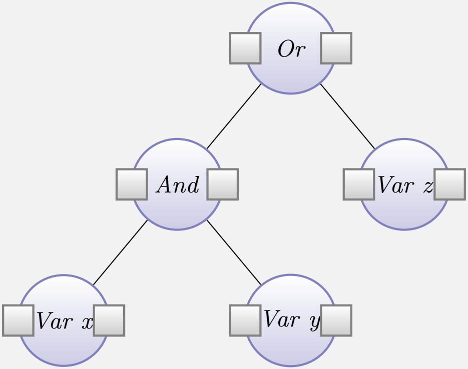
- Be able to describe visits to children
- Be able to restrict their relative order
- GPCE'10 paper

$$
\begin{array}{l}
\textbf{attr}\ \mathit{Pred}\ \textbf{visit}\ \mathit{eval} \\
\quad \textbf{inh}\ \mathit{env} :: \mathit{Map}\ \mathit{String}\ \mathit{Bool} \\
\quad \textbf{syn}\ \mathit{val} :: \mathit{Bool} \\
\textbf{sem}\ \mathit{Pred} \mid \mathit{Or}\ \textbf{visit}\ \mathit{eval} \\
\quad \textbf{invoke}\ \mathit{eval}\ \textbf{of}\ q \\
\quad \textbf{invoke}\ \mathit{eval}\ \textbf{of}\ p
\end{array}
$$

- Define external functions (possibly with side effect) as virtual children

Evaluation Algorithms Revisited

Typical Evaluation

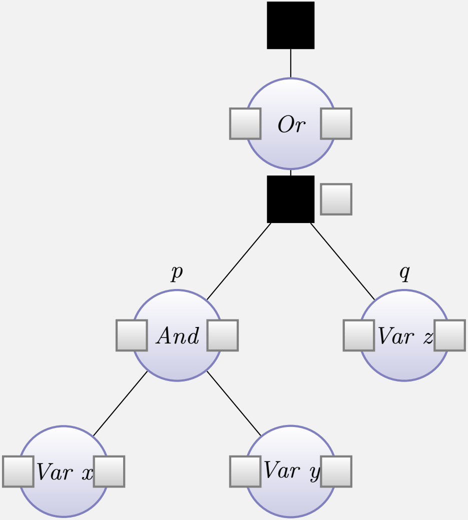


Kastens Style Evaluation

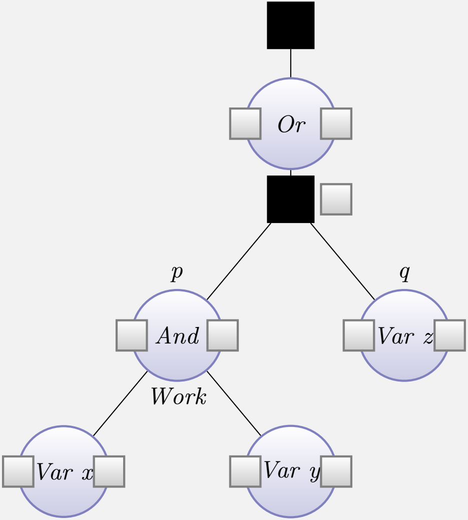
$$
\begin{array}{ll}
\textbf{plan } \mathit{Or} & p.env = lhs.env \\
& q.env = lhs.env \\
& \textbf{invoke } p \\
& \textbf{invoke } q \\
& lhs.val = p.val \vee q.val \\
& \textbf{yield } \mathit{Done} \\
\textbf{plan } \mathit{And} & p.env = lhs.env \\
& \textbf{invoke } p \\
& q.env = lhs.env \\
& \textbf{invoke } q \\
& lhs.val = p.val \wedge q.val \\
& \textbf{yield } \mathit{Done} \\
\textbf{plan } \mathit{Var} & lhs.val = \mathit{find}\ nm\ lhs.env \\
& \textbf{yield } \mathit{Done}
\end{array}
$$

Stepwise Evaluation

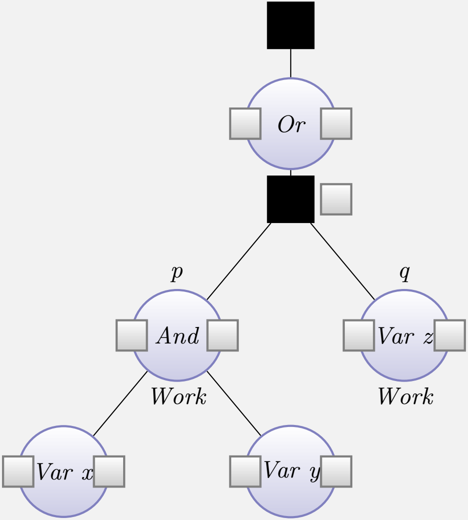
Example Instrumented with Events



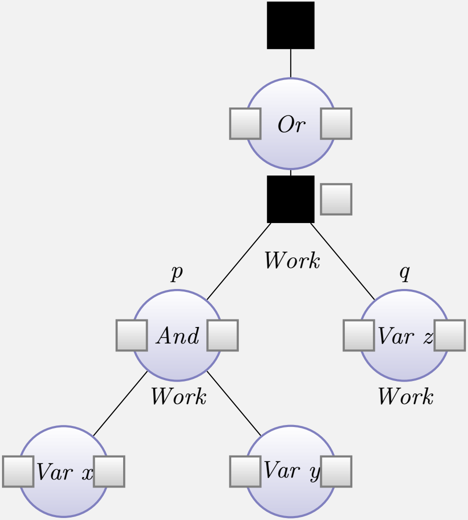
Example Instrumented with Events



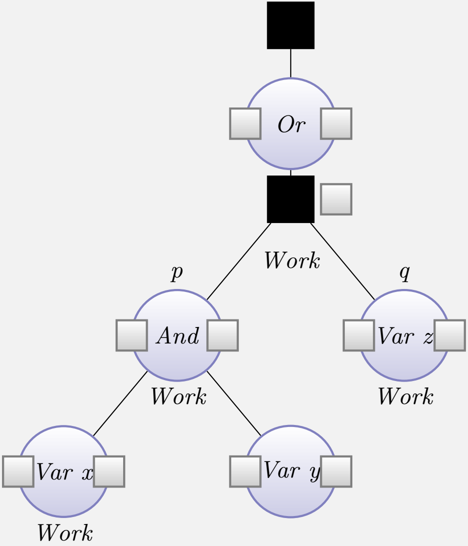
Example Instrumented with Events



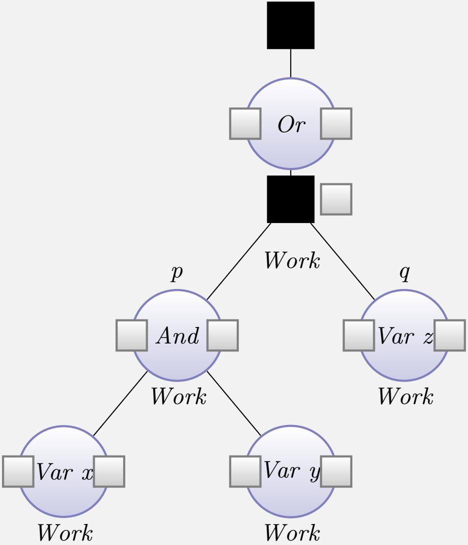
Example Instrumented with Events



Example Instrumented with Events



Example Instrumented with Events



Modified Evaluation Algorithm

- Eager algorithm - Kastens
- Coroutines

Modifications:

- Do not simply yield attribute values, but an execution trace
- Execution trace is composed from the traces of the children
- Man-in-the-middle mergers consume traces of children, and present themselves as replacement for these children with a transformed trace.
- At the root: repeatedly evaluate up to the next event
- Simplification: assume single-visit for each child

Execution trace and Inversion of Control

An execution trace of a child of (a single-visit) nonterminal N is a sequence of events:

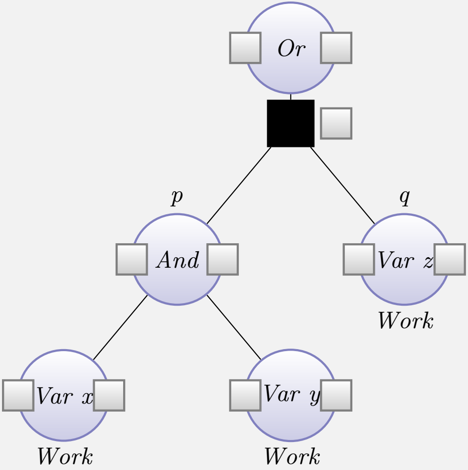
$$E_1, ..., E_n, Done_N$$

An event $E_i = X_I^O$ is user-defined and has:

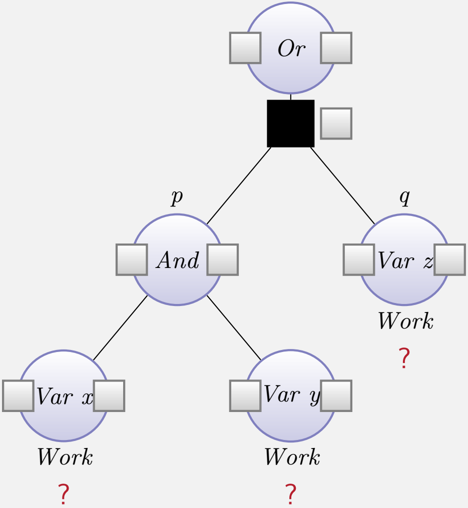
- A name X
- Values O provided by the child that yields the event, usable to the parent
- Values I usable by the continuation of the child, provided by the parent

The terminator $Done_N$ carries N's synthesized attributes.

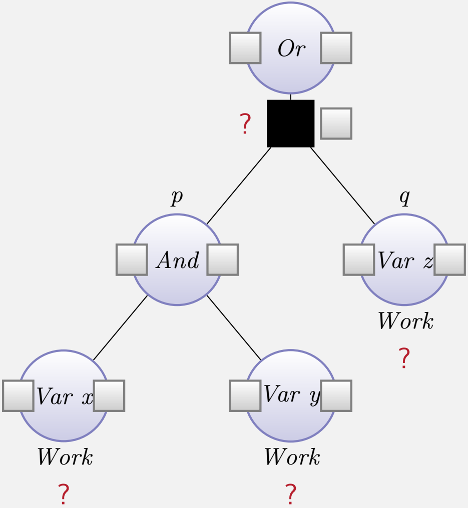
With Merging



With Merging



With Merging



Yielding Events

$$
\begin{array}{l}
\textbf{gram}\ \mathit{Yield} \mid \mathit{Yield} \\
\textbf{attr}\ \mathit{Yield}\ \textbf{inh}\ \emptyset\ \textbf{syn}\ \emptyset \\
\textbf{sem}\ \mathit{Pred} \mid \mathit{Var} \\
\quad \mathit{lhs.val} = \mathit{find}\ \mathit{nm}\ \mathit{lhs.env} \\
\quad \textbf{invoke}\ z \\
\quad \textbf{merge as}\ z : \mathit{Yield} = \textbf{do} \\
\quad\quad \textbf{raise}\ \mathit{Work}^{\emptyset} \\
\quad\quad \textbf{commit}\ z\ \$\ \mathit{wrap}\ \$\ \mathit{Syn\_Yield}\ \{\,\}
\end{array}
$$

Controlling Events

$$
\begin{array}{l}
\textbf{sem}\ \mathit{Pred} \mid \mathit{Or} \\
\quad p.\mathit{env} \ \ = \mathit{lhs}.\mathit{env} \\
\quad q.\mathit{env} \ \ = \mathit{lhs}.\mathit{env} \\
\quad \mathit{lhs}.\mathit{val} = z.\mathit{val} \\
\quad \textbf{merge}\ p, q\ \textbf{as}\ z : \mathit{Pred} = \textbf{catch} \\
\qquad p\ \textbf{raised}\ \mathit{Done} \mid p.\mathit{val} \rightarrow \textbf{commit}\ z\ p \\
\qquad q\ \textbf{raised}\ \mathit{Done} \mid q.\mathit{val} \rightarrow \textbf{commit}\ z\ q \\
\qquad p\ \textbf{raised}\ \mathit{Work}_{\emptyset} \quad q\ \textbf{raised}\ \mathit{Work}_{\emptyset} \rightarrow \textbf{do} \\
\qquad\quad r \leftarrow \textbf{raise}\ \mathit{Work}^{\emptyset} \\
\qquad\quad \textbf{return}\ (r, r)
\end{array}
$$

Static Semantics of Merge

$$\textbf{merge } c_1, ..., c_n \textbf{ as } k_1 : N_1, ..., k_m : N_m = e$$

- $n \geqslant 0, m \geqslant 1$
- $c_1, ..., c_n$: must be provided values for inhs, but may not refer to their syns
- $k_1, ..., k_m$: may refer to their syns, but not their inhs
- Monadic expression e that must ultimately commit semantics for each of the created children

Other Possibilities

Other Possibilities

Allow IO in monadic merge functions...

- Merge based on side-effect: encode graph traversal. Choose child depending on whether we visited the intended target already before.
- Run left and right child up till a couple of steps in parallel
- Create a nonterminal ApplySubst which takes a type variable as inherited attribute and its currently known expansion as synthesized attribute.
- Fixed-point computations: repeat evaluation of child, but with each iteration tweaked inherited attributes

Etc...

Conclusion

- The rules remained purely functional, and can still be automatically composed
- We pay a price:
  - Evaluation of children explicit
  - Explicit allocation of attributes to visits (to a certain degree)
- We gain: control over evaluation, traversals of more complex structures
- Overkill?

More information: `www.cs.uu.nl/~ariem/thesis.pdf`