Universiteit Utrecht

Attribute Grammars: A short tutorial
Tree-Oriented Programming

S. Doaitse Swierstra (doaitse@cs.uu.nl), GPCE 2003
Arie Middelkoop (ariem@cs.uu.nl), LERNET 2008
Adapted for Brazil 2008/2009
Universiteit Utrecht

March 1, 2008

Implementing a Language

- ▶ Parsing is about syntax
- ▶ What about semantics?

Universiteit Utrecht

Describing Semantics With a Grammar?

- ▶ Context-free grammars have limited expressiveness, and thus fail to describe:

Describing Semantics With a Grammar?

- ▶ Context-free grammars have limited expressiveness, and thus fail to describe:
    - ▶ Scope rules
    - ▶ Typing rules
    - ▶ Pretty printing
    - ▶ Code generation

Universiteit Utrecht

Describing Semantics With a Grammar?

- ▶ Context-free grammars have limited expressiveness, and thus fail to describe:
  - ▶ Scope rules
  - ▶ Typing rules
  - ▶ Pretty printing
  - ▶ Code generation
- ▶ Are there extensions?

Parameterize Non-Terminal Symbols

Parameterize non-terminal symbols with values from some other domain: *attribute grammars* (Knuth)

$$E\langle read\ x\rangle\ \rightarrow x$$
$$E\langle n + m\rangle\ \rightarrow E\langle n\rangle\ \texttt{"+"}\ E\langle m\rangle$$
$$E\langle n * m\rangle\ \rightarrow E\langle n\rangle\ \texttt{"*"}\ E\langle m\rangle$$

Universiteit Utrecht

An Attribute Grammar Consists Of:

- An underlying context free grammar

An Attribute Grammar Consists Of:

- ▶ An underlying context free grammar
- ▶ A description of which non-terminals have which attributes:
  - ▶ *Inherited* attributes, that are used or passing information *downwards* in the tree
  - ▶ *Synthesized* attributes that are used to pass information *upwards*

An Attribute Grammar Consists Of:

- ► An underlying context free grammar
- ► A description of which non-terminals have which attributes:
  - ► *Inherited* attributes, that are used or passing information *downwards* in the tree
  - ► *Synthesized* attributes that are used to pass information *upwards*
- ► For *each production* a description how to compute the:
  - ► Inherited attributes of the non-terminals in the *right hand side*
  - ► The synthesized attributes of the non-terminal at the *left hand side*
- ► In this way we describe *global* data flow over a tree, by defining *local* data-flow building blocks, corresponding to the productions of the grammar

Universiteit Utrecht

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

Creating HTML From a Document

```
\section{Intro}                 <h1>Intro</h1>
  \section{Section 1}           <h2>Section 1</h2>
    \paragraph                  <p>
      paragraph 1                 Paragraph 1
    \end                        </p>
    \paragraph                  <p>
      paragraph 2                 Paragraph 2
    \end                        </p>
  \end
  \section{Section 2}           <h2>Section 2</h2>
    \paragraph                  <p>
      paragraph 1                 Paragraph 1
    \end                        </p>
    \paragraph                  <p>
      paragraph 2                 Paragraph 2
    \end                        </p>
  \end \end
```

Universiteit Utrecht

Introducing UUAG

- ▶ Special syntax for programming with attributes
- ▶ Domain specific language for specifying tree walks

Universiteit Utrecht

Introducing UUAG

- Special syntax for programming with attributes
- Domain specific language for specifying tree walks

UUAG generates *semantic functions* which define the semantics of an *abstract syntax tree*.

Universiteit Utrecht

Concrete and Abstract syntax

```
Docs  ::=  Doc*
Doc   ::=  "\section" "{" Text "}" Docs "\end"
        |  "\paragraph" Text "\end"
```

Universiteit Utrecht

Concrete and Abstract syntax

```
Docs  ::=  Doc*
Doc   ::=  "\section" "{" Text "}" Docs "\end"
        |  "\paragraph" Text "\end"
```

Using a parser for the above concrete syntax, we produce a tree with the following *abstract syntax*:

DATA *Doc* | *Section*   *title* : *String*   *body* : *Docs*
             | *Paragraph* *text* : *String*

DATA *Docs* | *Cons*   *hd* : *Doc*   *tl* : *Docs*
              | *Nil*

- ▸ *Docs* and *Doc* are non-terminals
- ▸ *Section* and *Paragraph* label different productions
- ▸ *title*, *body* and *string* are names for children

Universiteit Utrecht

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

Synthesized attributes

Semantics: Our First Attribute!

- We introduce an attribute $html$ of type $String$ to return the generated html code in a synthesized attribute:

ATTR $Doc\ Docs$ [| | $html : String$]

Semantics: Our First Attribute!

▶ We introduce an attribute $html$ of type $String$ to return the generated html code in a synthesized attribute:

$$\textbf{ATTR } Doc\ Docs\ [\quad |\quad |\quad html : String\quad]$$

▶ Nonterminal Doc has $html$ as attribute, so we now need to define for productions $Section$ and $Paragraph$ how to compute the value of this attribute. The same for productions $Cons$ and Nil of $Docs$.

Semantics: Our First Attribute!

▶ We introduce an attribute $html$ of type $String$ to return the generated html code in a synthesized attribute:

ATTR $Doc\ Docs\ [\ \ |\ \ |\ \ html : String\ \ ]$

▶ Nonterminal Doc has $html$ as attribute, so we now need to define for productions $Section$ and $Paragraph$ how to compute the value of this attribute. The same for productions $Cons$ and Nil of $Docs$.

▶ Definitions for attributes are given in Haskell, with embedded references to attributes, in the form of `@<ntname>.<attrname>`:

▶ Assume for now that you can refer to:
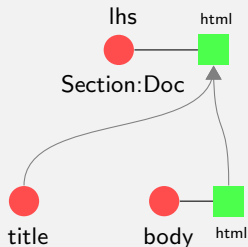 ▶ the synthesized attributes defined on the children
  ▶ values of child-terminals

Universiteit Utrecht

A Picture for Section

DATA *Doc* | *Section*   *title* : { *String* }   *body* : *Docs*

ATTR *Doc* [|| *html* : { *String* }]

SEM *Doc*
| *Section*   **lhs**.*html* = "<bf>" ++ @*title* ++ "</bf>\n"
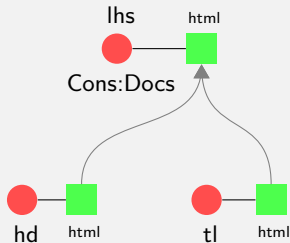   ++ @*body*.*html*

Universiteit Utrecht

A Picture for Cons

DATA *Docs* | *Cons*   *hd* : *Doc*   *tl* : *Docs*

ATTR *Docs* [|| *html* : { *String* }]

SEM *Docs*
  | *Cons*   **lhs**.*html* = @*hd*.*html* ++ @*tl*.*html*

Universiteit Utrecht

To Summarize: Our First Attribute!

- ▶ We introduce an attribute *html* of type *String* to return the generated html code in a synthesized attribute:

ATTR *Doc Docs* [| | *html* : *String*]

To Summarize: Our First Attribute!

- ▶ We introduce an attribute $html$ of type $String$ to return the generated html code in a synthesized attribute:

ATTR Doc $Docs$ [| | $html : String$]

- ▶ Definitions for attributes are given in Haskell, with embedded references to attributes, in the form of `@<ntname>.<attrname>`:

```
SEM Doc
  | Section    lhs.html = "<bf>" ++ @title ++ "</bf>\n"
                          ++ @body.html
  | Paragraph  lhs.html = "<P>" ++ @text ++ "</P>"
SEM Docs
  | Cons       lhs.html = @hd.html ++ @tl.html
  | Nil        lhs.html = ""
```

Universiteit Utrecht

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

Inherited attributes

Summary: Adding The Level Aspect

- ▶ Introduce an **inherited** attribute with name $level$, indicating the nesting level of the headings:

ATTR *Doc Docs* [$level : Int$ | |]

- ▶ You can refer to the inherited attributes defined on the left-hand side
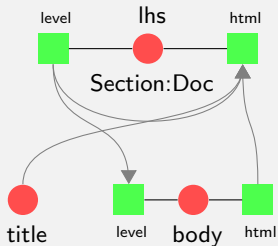- ▶ You need to define the inherited attributes of the children

A Picture For Section

$$\mathbf{SEM}\ Doc\ |\ Section$$
$$body.level = @\mathbf{lhs}.level + 1$$
$$\mathbf{lhs}.html = mk\_tag\ (\texttt{"H"} \mathbin{+\mkern-10mu+} show\ @\mathbf{lhs}.level)$$
$$\texttt{""}\ @title$$
$$\mathbin{+\mkern-10mu+} @body.html$$

Universiteit Utrecht

A Picture For Cons

SEM *Docs* | *Cons*
      $hd.level = @\mathbf{lhs}.level$
      $tl.level = @\mathbf{lhs}.level$
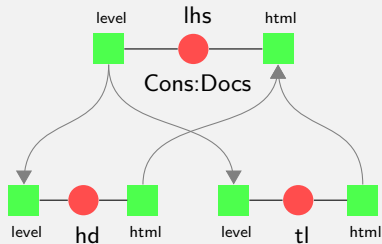
A Picture For Cons

> **SEM** *Docs* | *Cons*
> $hd.level = @\mathbf{lhs}.level$
> $tl.level = @\mathbf{lhs}.level$
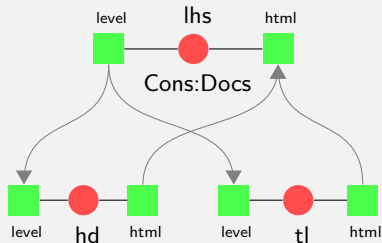
Actually, these two rules are not needed...

Summary: Adding The Level Aspect

- Introduce an **inherited** attribute with name *level*, indicating the nesting level of the headings:

ATTR *Doc Docs* [*level* : *Int* | |]

Summary: Adding The Level Aspect

- Introduce an **inherited** attribute with name $level$, indicating the nesting level of the headings:

ATTR $Doc\ Docs$ [	$level : Int$	|	|	]

- With the semantic rules:

SEM $Doc \mid Section$
$body.level = @\mathbf{lhs}.level + 1$
$\mathbf{lhs}.html = mk\_tag\ (\text{"H"} + show\ @\mathbf{lhs}.level)$
$\text{""}\ @title$
$+ @body.html$

Summary: Adding The Level Aspect

- Introduce an **inherited** attribute with name $level$, indicating the nesting level of the headings:

ATTR $Doc\ Docs$ [$\quad level : Int \quad$ | |]

- With the semantic rules:

SEM $Doc\ |\ Section$
$$body.level = @\textbf{lhs}.level + 1$$
$$\textbf{lhs}.html = mk\_tag\ (\text{"H"} + show\ @\textbf{lhs}.level)$$
$$\text{""}\ @title$$
$$+\ @body.html$$

- Where the function mk_tag is defined by:

$$mk\_tag\ tag\ attrs\ elem = \quad \text{"<"} + tag + attrs + \text{">"} + elem$$
$$+\ \text{"</"} + tag + \text{">"}$$

Universiteit Utrecht

Copy rules

Copy rules

- We do not need to give rules for $level$ for Docs?

Universiteit Utrecht

Copy rules

- We do not need to give rules for *level* for Docs?
- We generate copy rules in case attributes are passed on unmodified
- The *copy rules* that were automatically generated are:

> **SEM** *Docs*
> | *Cons* $hd.level = @\textbf{lhs}.level$
> $tl.level\ = @\textbf{lhs}.level$

- The same process holds for the synthesized attributes, except that if there is more than one child with this synthesized attribute, then the right most child with this attribute is chosen.

Universiteit Utrecht

Formally: Copy Rules

If a rule for an attribute $k.a$ is missing:

- Use $@\mathbf{loc}.a$
- Use $@c.a$ for the rightmost child c to the left of k, which has a synthesized attribute named a
- Use $@\mathbf{lhs}.a$

Special copy rule: the USE rule

Remember:

SEM *Docs*
| *Cons*   **lhs**.*html* = @*hd.html* ++ @*tl.html*
| *Nil*   **lhs**.*html* = ""

These rules cannot be produced by the copy rules. Why not?

Special copy rule: the USE rule

Remember:

SEM *Docs*
| *Cons* **lhs**.*html* = @*hd.html* ++ @*tl.html*
| *Nil*   **lhs**.*html* = ""

These rules cannot be produced by the copy rules. Why not?
But this code can be produced by a special copy rule, for which
we need to provide extra information:

ATTR *Docs* [|| *html USE* { ++ } { "" } : { *String* }]

Universiteit Utrecht

Chained attributes

Adding The Section Counter Aspect

- Introduce two inherited attributes:
  - The *context*, representing the outer blocks
  - A *count*er for keeping track of the number of encountered siblings.

> **ATTR** *Doc Docs* [*context* : *String*
> | *count* : *Int*
> |]

- Since we do not now whether a *Doc* will update the counter we will have to pass it from *Docs* to *Doc*, and back up again. So *count* becomes a *threaded attribute*

- **loc** represents a local attribute, which is just a local definition

Universiteit Utrecht

A picture With The count Added
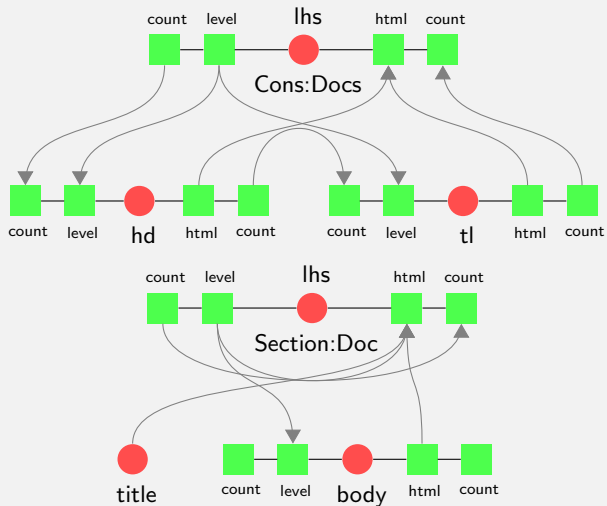
The Semantic Functions

```
SEM Doc
  |  Section body.count   = 1
           body.context = @loc.prefix
           lhs.count    = @lhs.count + 1
           lhs.html     = @loc.html
           loc.prefix   = if   null @lhs.context
                          then show @lhs.count
                          else @lhs.context
                               ++ "."
                               ++ show @lhs.count
           loc.html     = mk_tag ("H" ++ show @lhs.level)
                                  ""
                                  (@loc.prefix ++ " "
                                   ++ @title)
                          ++ @body.html
```
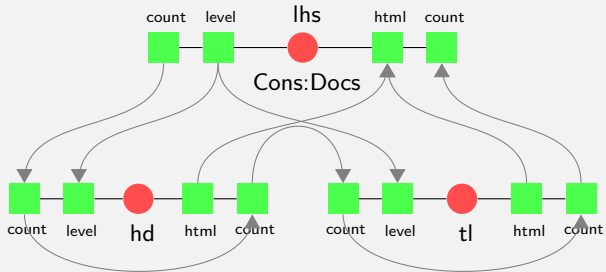
A Pictorial Representation

- ▶ We show some different aspects
- ▶ We show the aspects *count* and *level* and html

Universiteit Utrecht

Adding Extra Productions

- ▶ We may also add extra productions, and as an example we will insert a table of contents

Universiteit Utrecht

Adding Extra Productions

- We may also add extra productions, and as an example we will insert a table of contents
- An extra synthesized attribute *toclines* in which the table of contents is constructed
- An extra inherited attribute *toc*, containing the table of contents

$$
\begin{aligned}
&\textbf{DATA } Root \ \mid Root \ doc : Doc \\
&\textbf{ATTR } Root \ [\mid\mid html : String] \\
&\textbf{DATA } Doc \ \ \mid Toc \\
&\textbf{ATTR } Doc \ Docs \ [\ toc : String \\
&\qquad\qquad\qquad\qquad \mid \\
&\qquad\qquad\qquad\quad \mid toclines \ USE \ \{+\!\!+\} \ \{""\} : String]
\end{aligned}
$$

- The *USE* clause defines default semantic computation

Universiteit Utrecht

A picture with the toc and toclines added

Universiteit Utrecht
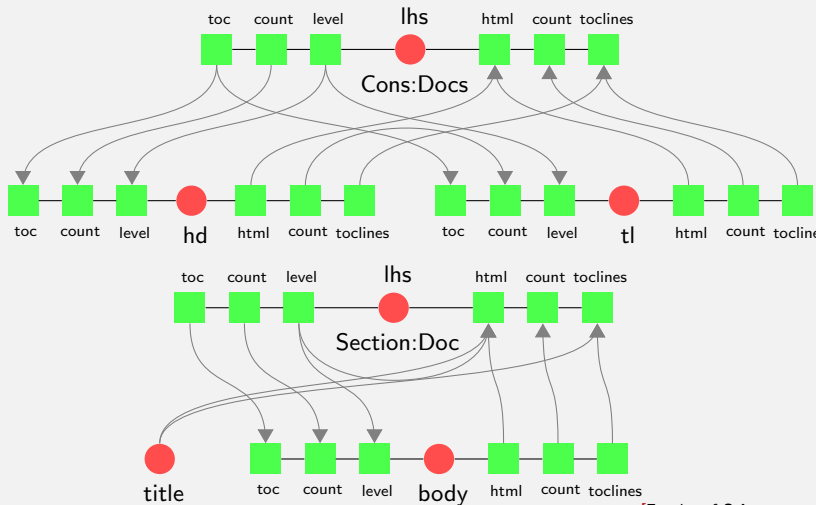
SEM *Doc*

   | *Section*

      lhs *.toclines* = mk_tag `"LI"` `""`

                     (mk_tag (`"A"`)

                     (`" HREF=#"` $+\!\!\!+$ @**loc**.*prefix*)

                     (@**loc**.*prefix* $+\!\!\!+$ `" "`

                      $+\!\!\!+$ @*title*))

                     $+\!\!\!+$ mk_tag `"UL"` `""`@*body.toclines*

      lhs *.html* := mk_tag `"A"` (`" NAME="`

                     $+\!\!\!+$ @**loc**.*prefix*) `""`

                     $+\!\!\!+$ @**loc**.*html*

   | *Toc* **lhs** *.html* = @**lhs**.*toc*
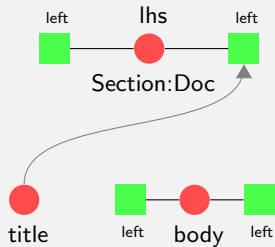
SEM *Root*

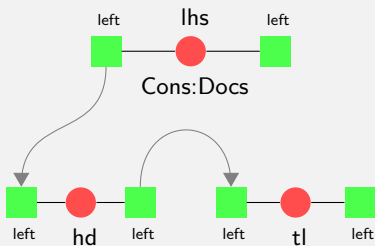 | *Root doc.toc* = @*doc.toclines*

         doc.level = 1
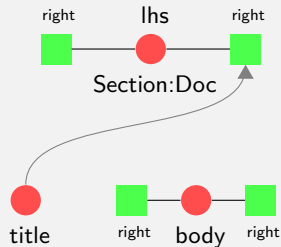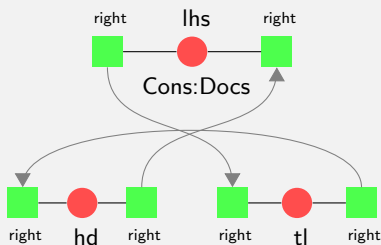
         doc.context = `""`

         doc.count = 1

Backward Flow Of Data

- We want to be able to jump to the section to the *left* and the *right* of the current section
- We introduce two new attributes for passing this information around

Universiteit Utrecht

Right

$$\textbf{SEM } Docs$$
$$| \; Cons$$
$$hd.right = @tl.right$$
$$tl.right = @\textbf{lhs}.right$$
$$\textbf{lhs}.right = @hd.right$$

$$\textbf{SEM } Doc$$
$$| \; Section$$
$$\textbf{lhs}.right = @title$$
$$body.right = \texttt{""}$$

What Is Generated?

▶ Data types

```
data Doc = Doc_Paragraph String
         | Doc_Section String Docs
         | Doc_Toc
```

Universiteit Utrecht

What Is Generated?

▶ Data types

data *Doc = Doc_Paragraph String*
          | *Doc_Section String Docs*
          | *Doc_Toc*

▶ Types

type *T_Doc = String* →
              Int   →
              Int   →
              String →
              (*Int, String, String*)

Universiteit Utrecht

Semantic functions:

$$sem\_Docs\_Cons\ (\_hd)\ (\_tl) =$$
$$\lambda\_lhs\_context$$
$$\quad \_lhs\_count$$
$$\quad \_lhs\_level$$
$$\quad \_lhs\_toc \rightarrow$$
$$\quad\quad \textbf{let}\ (\_hd\_count,\ \_hd\_html,\ \_hd\_toclines) =$$
$$\quad\quad (\_hd\ (\_lhs\_context)\ (\_lhs\_count)\ (\_lhs\_level)\ (\_lhs\_toc))$$
$$\quad\quad\quad (\_tl\_count,\ \_tl\_html,\ \_tl\_toclines) =$$
$$\quad\quad (\_tl\ (\_lhs\_context)\ (\_hd\_count)\ (\_lhs\_level)\ (\_lhs\_toc))$$
$$\quad\quad \textbf{in}\ (\_tl\_count,\ \_hd\_html + \_tl\_html$$
$$\quad\quad\quad ,\ \_hd\_toclines + \_tl\_toclines)$$

Optimizations

- Perform an abstract interpretation of the grammar
- Computing dependencies between attributes

Universiteit Utrecht

Optimizations

- Perform an abstract interpretation of the grammar
- Computing dependencies between attributes
- Schedule the attributes for computation per non-terminal (multiple visits)

Universiteit Utrecht

Optimizations

- Perform an abstract interpretation of the grammar
- Computing dependencies between attributes
- Schedule the attributes for computation per non-terminal (multiple visits)
- And is this way achieve a data-driven evaluation

Universiteit Utrecht

Optimizations

- Perform an abstract interpretation of the grammar
- Computing dependencies between attributes
- Schedule the attributes for computation per non-terminal (multiple visits)
- And is this way achieve a data-driven evaluation
- That may be somewhat cheaper
- And takes far less space

Conclusions

- ▶ Attribute grammars are your friend if you want to implement a language
- ▶ Attributes may even depend on themselves if you are building on a lazy language
- ▶ Even thinking in terms of attribute grammars may help you

- ▶ http://www.cs.uu.nl/wiki/HUT/WebHome

Universiteit Utrecht