Attribute Grammars

Nuno Oliveira

Outline

- Formal Definition
- Attributes
  - Inherited Attributes
  - Synthesized Attributes
  - Properties
  - Production Attributes
- Computation Rules
- Contextual Conditions
- Transformation Rules
- The Locality Concept
- Contexts in a Production
- The Game of “Passing the Word”
- Objective Oriented Programming
  - Example
- Conclusion

Formal Definition

- Attribute Grammar is a tuple

$$AG = (G, A, CR, CC, TR)$$

  - Where
    - G – is a context independent grammar;
    - A – is the set of attributes associated to Symbols (N and T) of G;
    - CR – is the set of rules for computing the value of attributes in all productions of G;
    - CC – is the set of contextual conditions in all productions of G;
    - RT – is the set of translation rules in all production of G;

Attributes

- Attributes
  - Associated to any symbol X (terminal or non-terminal)
    - Terminal attributes are
      - named “intrinsic” and
      - pre-established (ex. text, pos, line, etc.)
  - Semantically characterise symbol X
  - A(X) – is the set of attributes associated to symbol X
  - A(X) is divided into 2 disjoint subsets
    - **IA(X)** – inherited attributes of symbol X
    - **SA(X)** – synthesized attributes of symbol X

Attributes

Inherited Attributes

- Transport contextual information down the derivation tree
- Terminal symbols and the axiom do not have inherited attributes

Attributes
Synthesized Attributes

- Synthesize information from the leafs in the derivation tree and transport it up the tree

- Terminal symbols attributes (intrinsic) are seen as synthesized attributes
  - Actually, the synthesis is made during lexical analysis

Attributes

Properties

- Set of **attribute-value** associated to each instance of a symbol X in the grammar, when parsing a concrete sentence.

- Completely define the meaning of each symbol X

- Example
  - Nonterminal “Person”
  - A(Person) = {name, age, …}
  - Properties of Person
    - {<name, ”João”>, <age, 23>, …}

Attributes

Production Attributes

- Let p be a Production
  - In(p) – are the attributes that bring value to the context of the production
    - Incoming Attributes
      - IA of the LHS symbol
      - SA of the RHS symbols
  - Out(p) – are the attributes that take value to other productions
    - Outgoing Attributes
      - IA of the RHS symbols
      - SA of the LHS symbol

Computation Rules

- Mathematical expressions to compute the concrete value of attributes of a symbol X, taking into account the context of X.
  - Context is given by the production and the involved symbols

- **For each outgoing attribute (in a production) there is only one way to compute its value.**

- Computation rules are expressed using (mathematical combinations of)
  - Synthesised attributes
  - Inherited Attributes
  - Constants

  } Incoming Attributes of the production

Contextual Conditions

- Mathematical expressions to check the semantic validity of the concrete sentences

- Restrictions imposed to the value of attributes (in each production)
  - Example:
    - Suppose nonterminal Date has attributes “year”, “month” and “day”
    - Contextual condition shall restrict “month” values to numbers from 1 to 12 and “day” values to numbers from 1 to 31.
    - Suppose a date properties are {<year,2012>, <month,20>, <day,24>}
    - One semantic error occurs!

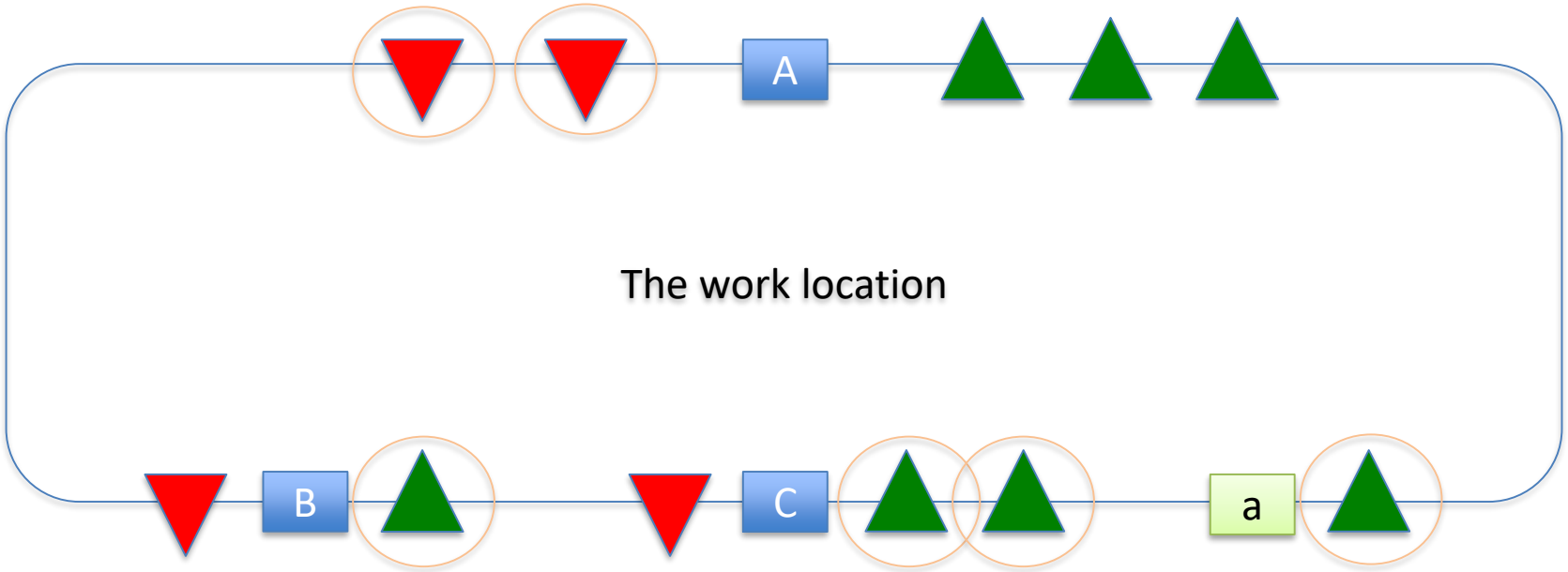
- If-condition expressions are usually used to check values
  - Example:
    - If(date.month < 1 && date.month > 31) then {error} else {no-error}

Translation Rules

- Mathematical expressions to transform the sentence in a desired result
  - Translation rules can only be “executed” if the semantics is valid!
- Translation rules are expressed used (mainly)
  - Synthesized attributes (semantically checked)

The Locality Concept

In each production, work only with the local resources!



Need more resources?

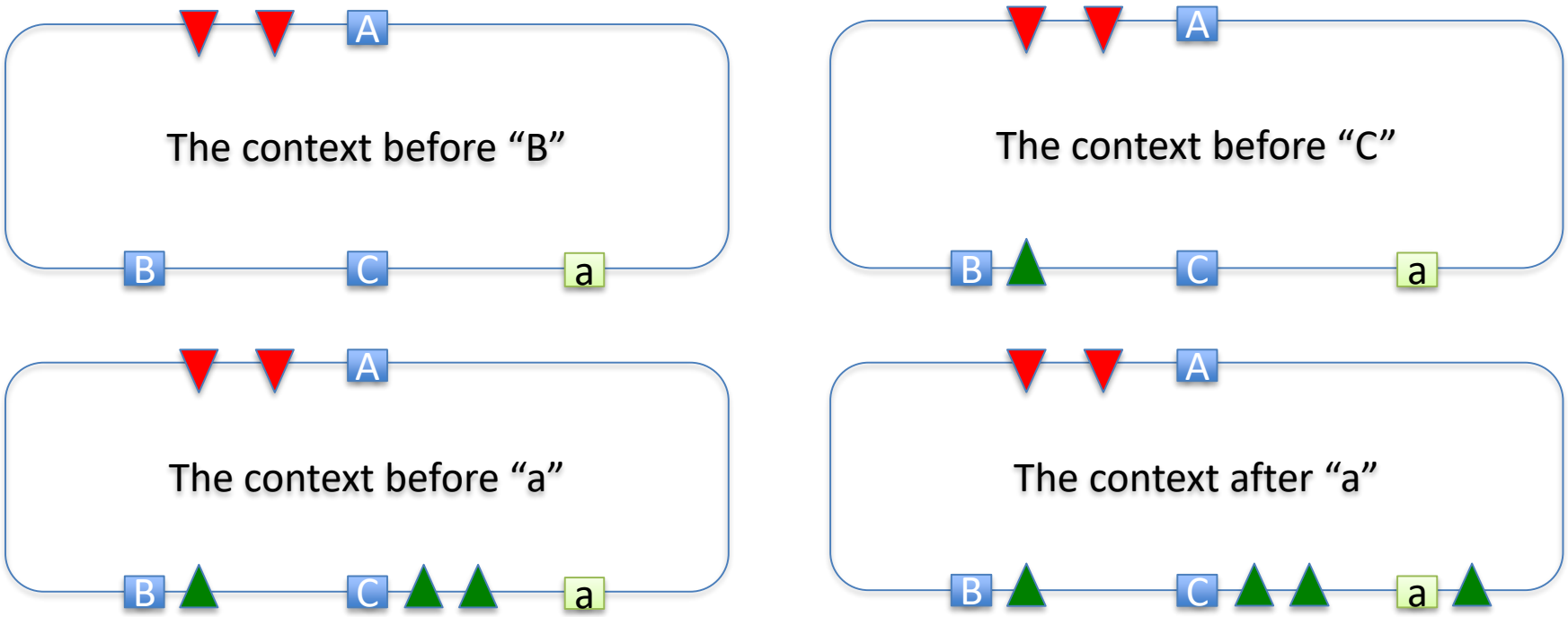
Import them… how?

The Locality Concept

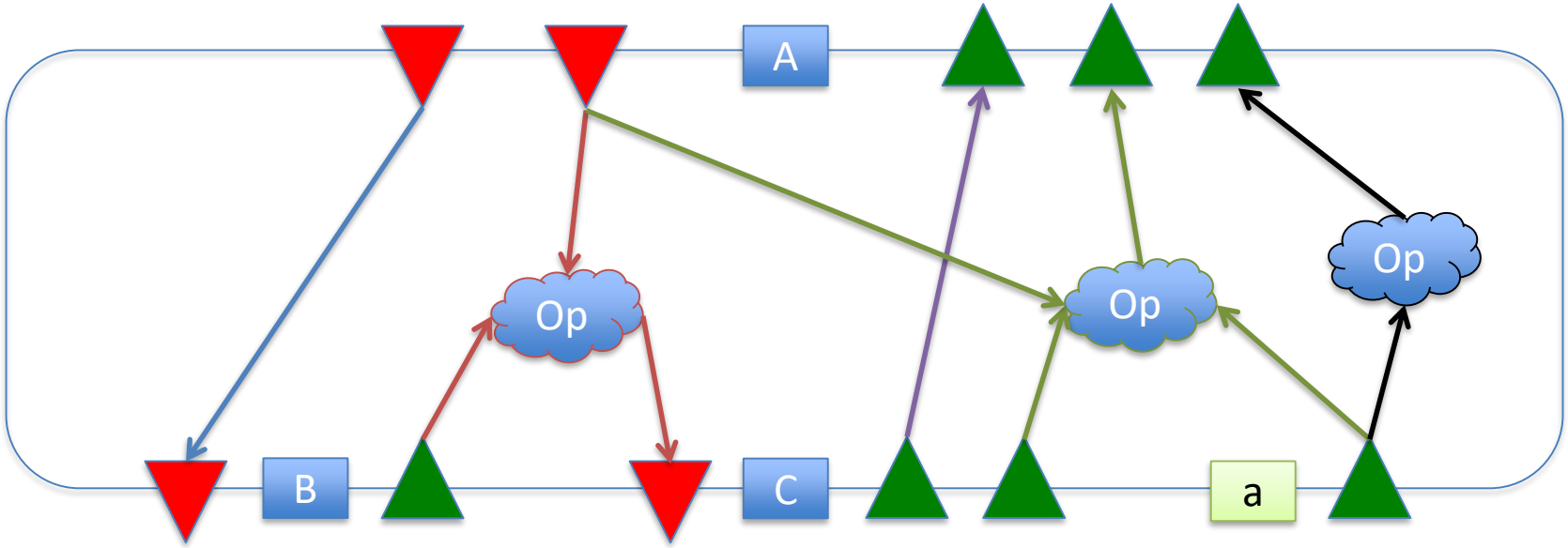
- Incoming attributes are the resources!
- Outgoing attributes shall be computed using the production resources!
- The production is “the work location”!

The Contexts in a Production

A context is characterised by the values of "incoming attributes" at each processing phase



The Game of “Passing the Word”



Objective Oriented Programming

- Building an AG is a complex task
  - A lot of information is needed
  - Several attributes come into play to solve a problem

- Hint:
  - Divide the problem into small objectives
  - Select productions needed to reach each objective
  - Define suitable and clear attributes for each symbol
    - Needed attributes
    - Auxiliary attributes (for computation of the needed ones)
    - Use prefixes in_ and out_ to distinguish IA from SA
  - Know the involved production trees
  - Understand the “passing the word” game for the objective
  - Compute attribute values using the locality concept
    - Clearly encode the computation rules within the production

Objective Oriented Programming

Example

- Sum the numeric elements in a list (a sentence from the NEList language)

```
P1: List    -> '[' Content ']'
P2: Content -> Item
P3:         | Item (, Content)
P4: Item    -> num
P5:         | wrd
```

Objective Oriented Programming

Example

- Productions needed

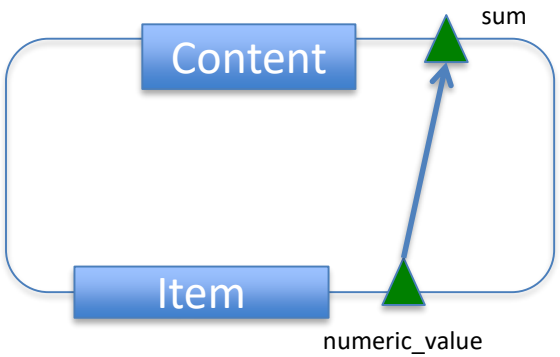
```
P2: Content -> Item
P3:          | Item (, Content)
P4: Item    -> num
P5:          | wrd
```

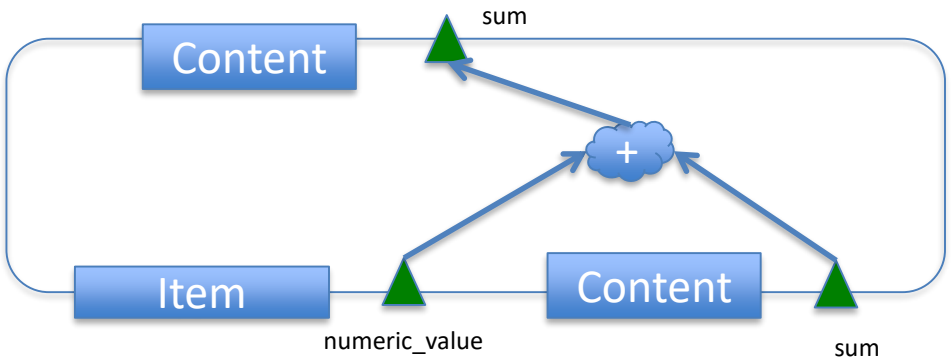
- Needed Attributes
  - Content.sum :: int :: SA
  - Item.numeric_value :: int :: SA

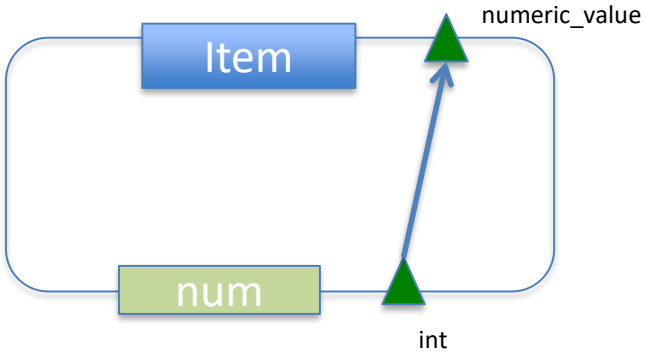
Objective Oriented Programming

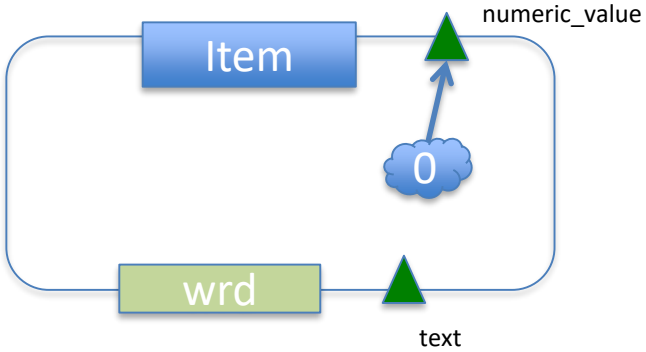
Example

- Production trees and “word passing” game







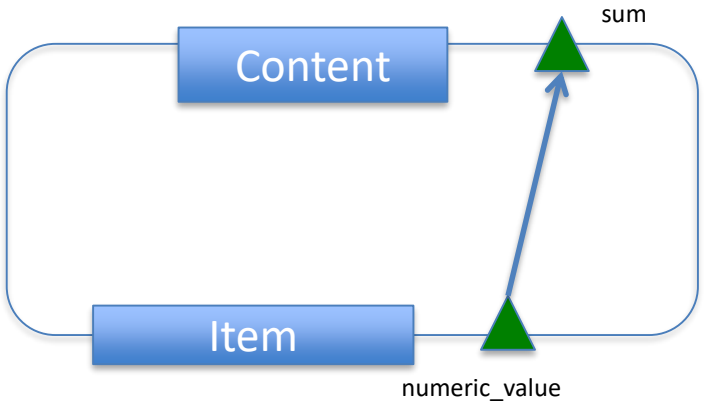


Objective Oriented Programming

Example

- Compute the attribute values

```
P2: Content -> Item
{
   Content.sum = Item.numeric_value
}
```

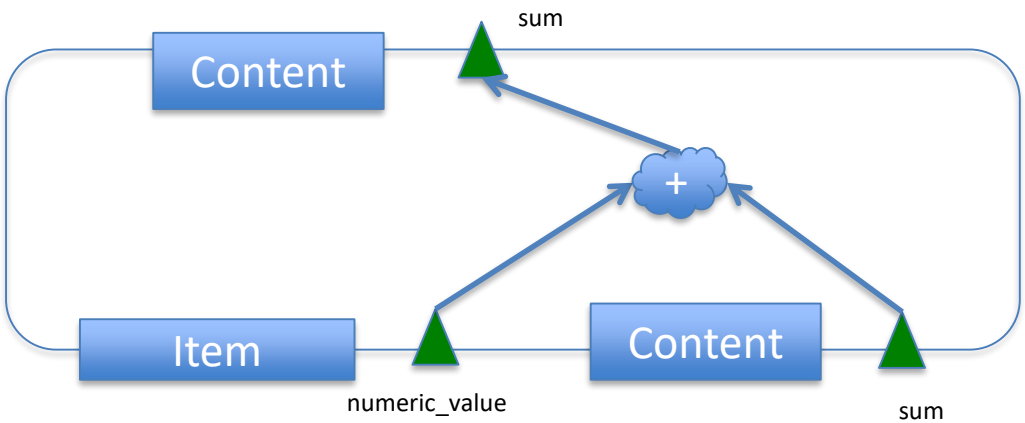


Objective Oriented Programming

Example

- Compute the attribute values

```
P3: Content -> Item (, Content)
{
   Content[0].sum =
      Item.numeric_value + Content[1].sum
}
```

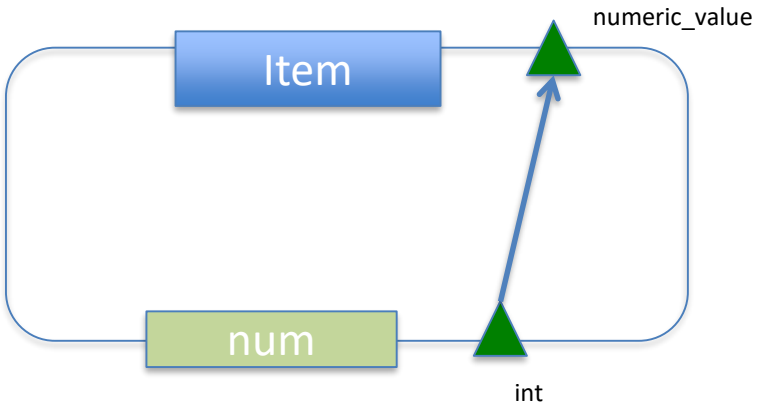


Objective Oriented Programming

Example

- Compute the attribute values

```
P4: Item     -> num
{
   Item.numeric_value = num.int
}
```

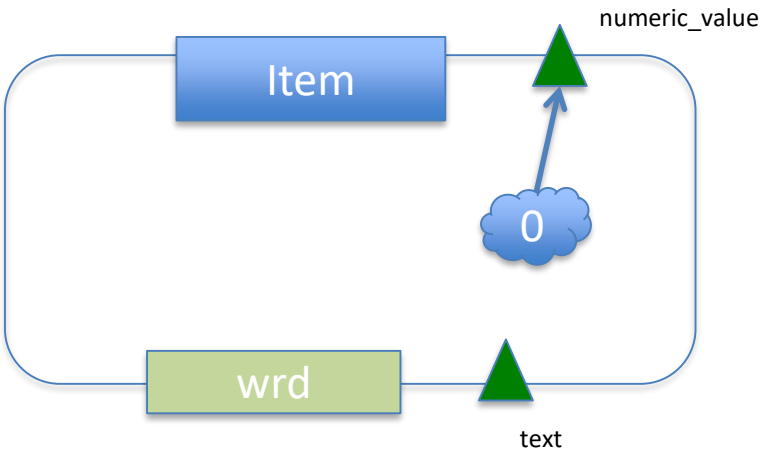


Objective Oriented Programming

Example

- Compute the attribute values

```
P5: Item     -> wrd
{
   Item.numeric_value = 0
}
```



Conclusion

- Computation code is formal, rigorous, clean and correct;

- Concrete values are
  - computed in productions
    - With local attributes
  - **Semantically valid**
    - **Context-based validation**