SDT Actions and Values

Semantic Actions are the procedures we execute during a parse when using **syntax direct translation**. Associated with a particular production rule.

Semantic Values are the *grammar symbols in the production rule*, in some way made available to the semantic action (we've been using $node$, $node.children, \ldots$).
Depending on the software architecture, the semantic action may have access to other values as well (parent node, siblings, global variables, . . .)

$$
\begin{array}{lcl}
NUCLEUS & \rightarrow & open\ ALT\ close \\
 & | & char\ CHARRNG \\
 & | & dot \\
CHARRNG & \rightarrow & dash\ char \\
 & | & \lambda
\end{array}
$$

```
procedure NUCLEUS( node, parse tree T )

  if ( node.children[0] is open ) then (
    replace node with node.children[1] in parse tree T
    return
  )

  if ( node.CHARRNG.char exists ) then (
    let rangeNode ← new range node
    rangeNode.addChild(node.char)
    rangeNode.addChild(node.CHARRNG.char)
    replace node.children with rangeNode in parse tree T
    return
  )

  if ( node.CHARRNG.child is λ ) then (
    remove node.CHARRNG child
    return
  )

  if ( node.child is dot ) then (
    replace node with node.child in parse tree T
    return
  )
```

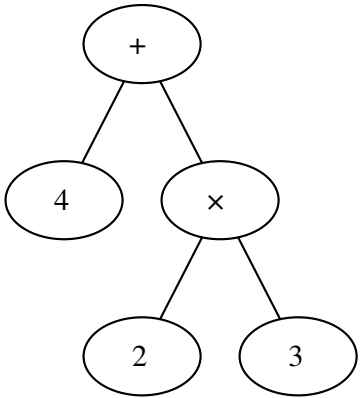
Syntax Directed Translation Vocabulary

An AST is a mix of nodes influenced by the grammar and the values of terminals read from the source.

Inherited Attributes "Flow" from parent ("grammar") to its descendents in the parse tree — typically *semantic* in nature.
For this expression tree, the order of operations of $4 + 2 \times 3$

Synthetic Attributes "Flow" up from the values of source terminals toward the root of the tree.
For source `4+2*3` generating this expression tree, the values 6 then 10

In this case, "attributes" are an abstract idea, we're not talking about class or object attributes of a language definition or compiler implementation.



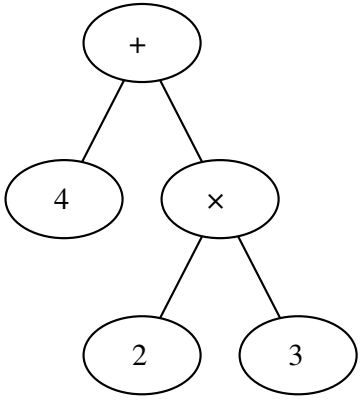
Syntax Directed Translation Vocabulary

An AST is a mix of nodes influenced by the grammar and the values of terminals read from the source.

Inherited Attributes "Flow" from parent ("grammar") to its descendents in the parse tree — typically *semantic* in nature.
For this expression tree, the order of operations of $4 + 2 \times 3$

Synthetic Attributes "Flow" up from the values of source terminals toward the root of the tree.
For source `4+2*3` generating this expression tree, the values 6 then 10

The question seems murkier for `4*(2+3)`: is the altered order of operations dictated from the parenthetical grouping in the source? OTOH: it is the **grammar** that pushes $+$ operation lower in the tree.



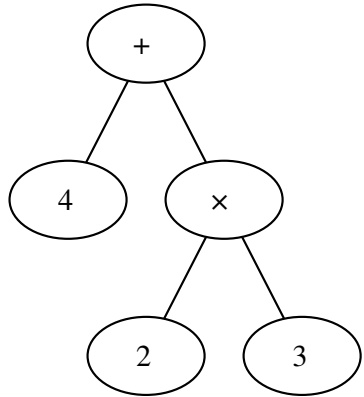
Syntax Directed Translation Vocabulary

An AST is a mix of nodes influenced by the grammar and the values of terminals read from the source.

Inherited Attributes "Flow" from parent ("grammar") to its descendents in the parse tree — typically *semantic* in nature.
For this expression tree, the order of operations of $4 + 2 \times 3$

Synthetic Attributes "Flow" up from the values of source terminals toward the root of the tree.
For source `4+2*3` generating this expression tree, the values 6 then 10

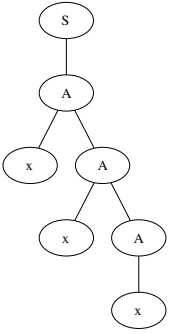
My thought process: if it is an attribute solely attributed to data or values from the source, a **synthetic attribute**, otherwise probably **inherited**.



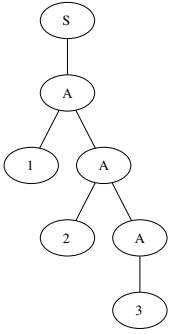
Syntax Directed Translation Vocabulary

A rather simplisitic example of **Inherited Attributes** would be counting the length of a string of `x`s in a top-down parse:

Syntax Tree



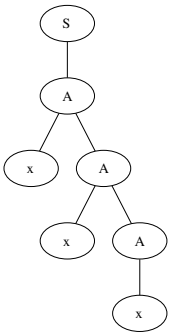
Inherited Values Tree



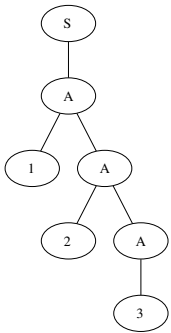
Syntax Directed Translation Vocabulary

A rather simplisitic example of **Inherited Attributes** would be counting the length of a string of `x`s in a top-down parse:

Syntax Tree



Inherited Values Tree



Would these still be inherited attributes in a **shift-reduce (LR)** parse?
Why?

Semantic Actions for Integer Values

$$
\begin{array}{rcl}
Num & \rightarrow & Digits\ \$ \\
Digits & \rightarrow & d\ Digits \\
 & | & \lambda
\end{array}
$$

- Note that each procedure is **rule oriented**.
- A simplistic example, in order to focus on the SDT details.

Semantic Actions

```
Num_Digits( node, tree )
  return global integer value

Digits_d_Digits( node, tree )
  global integer value, factor
  v ← node.d as its decimal value
  value ← v × factor + value
  factor ← factor × 10

Digits_lambda( node, tree )
  global integer value ← 0
  global integer factor ← 1
```

Semantic Actions for Integer Values

$$
\begin{array}{rcl}
Num & \rightarrow & Digits\ \$ \\
Digits & \rightarrow & d\ Digits \\
 & | & \lambda
\end{array}
$$

- Note that each procedure is **rule oriented**.
- A simplistic example, in order to focus on the SDT details.
- Let's trace an example for an LL(1) parse of `473`.

Semantic Actions

```
Num_Digits( node, tree )
  return global integer value

Digits_d_Digits( node, tree )
  global integer value, factor
  v ← node.d as its decimal value
  value ← v × factor + value
  factor ← factor × 10

Digits_lambda( node, tree )
  global integer value ← 0
  global integer factor ← 1
```

Rule Cloning for Decimal, Octal Integer Values

Rule Cloning: modifying an existing production rule(s) so that a more specialized SDT can be achieved.

$$
\begin{array}{rcl}
Num & \rightarrow & Base\ Digits\ \$ \\
Digits & \rightarrow & d\ Digits \\
 & | & \lambda \\
Base & \rightarrow & oh \\
 & | & \lambda
\end{array}
$$

```
Num_Base_Digits( node, tree )
  return global integer value
```

```
Digits_d_Digits( node, tree )
  global integer value, factor, base
  v ← node.d as its decimal value
  value ← v × factor + value
  factor ← factor × base

Digits_lambda( node, tree )
  global integer value ← 0
  global integer factor ← 1

Base_oh( node, tree)
  global integer base ← 8
Base_lambda( node, tree)
  global integer base ← 10
```

Forcing Semantic Actions

By the careful construction of a grammar and judicious use of λ-rules, we can invoke specialized logic at virtually any point in a parse. When the grammar has been modified for this purpose, it is called **forcing** a semantic action.

$$
\begin{array}{rcl}
\mathit{ATOM} & \rightarrow & \mathit{NUCLEUS\ ATOMMOD} \\
\mathit{ATOMMOD} & \rightarrow & \mathit{kleene} \\
 & | & \mathit{plus} \\
 & | & \lambda \\
\mathit{NUCLEUS} & \rightarrow & \mathit{open\ ALT\ close} \\
 & | & \mathit{char\ CHARRNG} \\
 & | & \mathit{dot} \\
\mathit{CHARRNG} & \rightarrow & \mathit{dash\ char\ CHECKRANGE} \\
 & | & \lambda \\
\mathit{CHECKRANGE} & \rightarrow & \lambda
\end{array}
$$

```
CHECKRANGE_lambda( node, tree )
  if ( ASCII(node.parent.parent.char) >
       ASCII(node.parent.char) ) then (
    ERROR: BAD RANGE VALUES
  ) else (
    trim last element from parent.children
  )
```

Common Notation for Semantic Actions in Academia

show_author_semantic-actions.pdf

- Heavily dependent on typesetting, not syntax
- Semantic actions organized by **rule**, not by non-terminal.
  Sometimes this is helpful, sometimes not.
- Subscripts in production rule's symbols (**Semantic Values**) are the **semantic action**'s variable names
- Since the logic and syntax of **implemented** semantic actions are heavily dependent on your language of choice, semantic actions for **teaching** usually use pseudo code.