# Attribute Grammars

- **Definitions: synthesized, inherited, dependence graph**
- **Example: syntax-directed translation**
- **S-attributed grammars**
- **L-attributed grammars**
- **Bottom Up evaluation of inherited attributes**
- **Top Down translation**



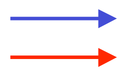
# Attribute Grammars

- **Attributes: properties associated with nonterminal symbols of a context free grammar**
- **E.G., Binary numbers**

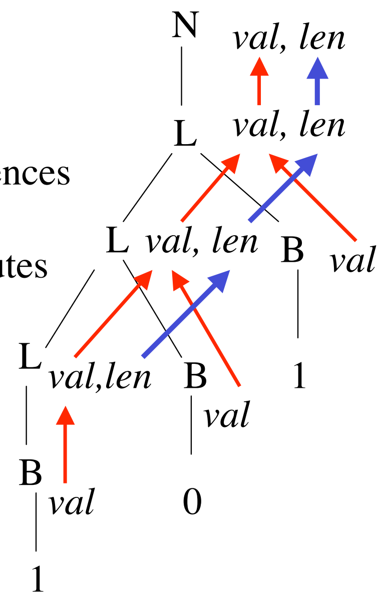
Semantic rules defining attributes are side-effect free

1. $B \rightarrow 0$    $val(B) = 0$
2. $B \rightarrow 1$    $val(B) = 1$
3. $L \rightarrow B$    $val(L) = val(B);\ len(L) = 1$
4. $L \rightarrow L_1\ B$    $val(L) = 2*val(L_1) + val(B)$
   $len(L) = len(L_1) + 1$
5. $N \rightarrow L$    $val(N) = val(L);\ len(N) = len(L)$

# Parse Tree of $101_2$



Represent dependences between semantic functions of attributes

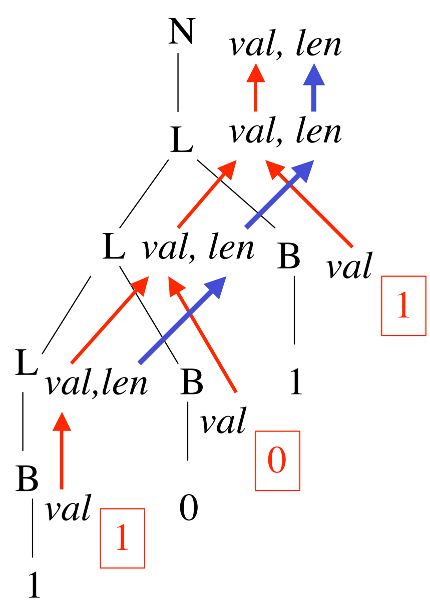



Dependence Graph: acyclic, evaluated in topsort order, contains attribute instances



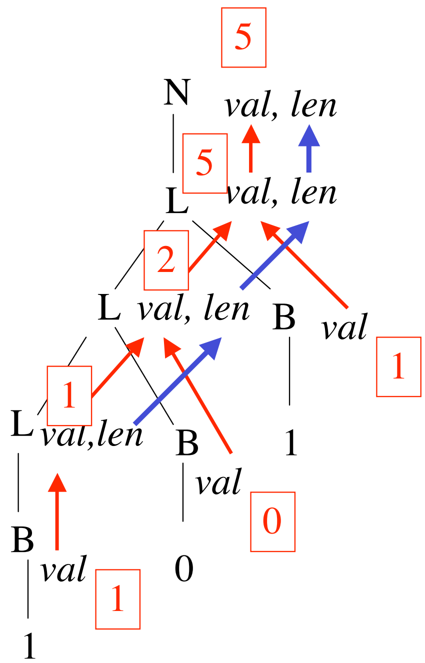
# Evaluate(Decorate) Parse Tree

Initial values



# Evaluate Parse Tree

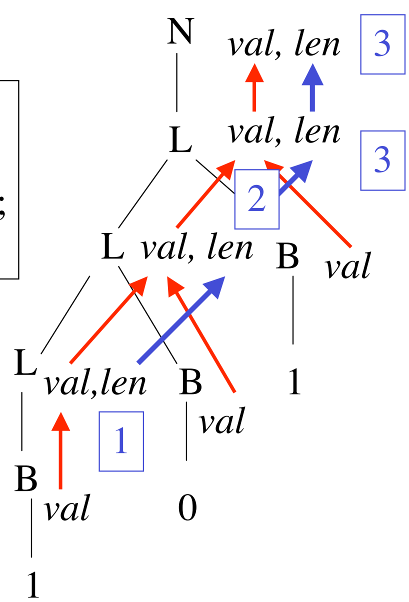
Evaluate *val*






# Evaluate Parse Tree

Evaluate *len;*
Full evaluation yields *val*(N) = 5; *len*(N) = 3.



# Classifications

- **Inherited attributes:**
  - Values based on attributes of parent (LHS nonterminal) or siblings (nonterminals on RHS of same production
- **Synthesized attributes:**
  - Values based on attributes of descendents (child nonterminals in same production)



# Classifications

- **Local context: always within focus of a single production**
  - Dependence edges go only one level in parse tree
- **Terminals can be associated with values returned by the scanner**
- **Distinguished nonterminal cannot have inherited attributes**

# Example - Identifiers

**Identifiers with no letters repeated (e.g., moon - illegal, money - legal)**

**D → I** *str*(I) = {}; *val*(D) = *val*(I);
accept, if *val*(D) != *error*

**I → L $I_1$** *str*(L) = *str*(I); *str*($I_1$) = *val*(L);
*val*(I) = *val*($I_1$)

**I → L** *str*(L) = *str*(I); *val*(I) = *val*(L)

**L → a | b | … | z** *val*(L) = concatenation of *val* returned by scanner to *str*(L), if this character is not a repeated letter, else *error*.

(note: any comparison to *error* returns *error*.)



# Inherited Attributes

**D → I** *str*(I) = {}; *val*(D) = *val*(I);
accept, if *val*(D) != *error*

**I → L $I_1$** $\boxed{str(L) = str(I);\ str(I_1) = val(L);}$
*val*(I) = *val*($I_1$)

**I → L** $\boxed{str(L) = str(I);}$ *val*(I) = *val*(L)

**L → a | b | … | z** *val*(L) = concatenation of *val* returned by scanner to *str*(L), if this character is not a repeated letter, else *error*.

(note: any comparison to *error* returns *error*.)

# Synthesized Attributes

**D → I**   ***str*(I) = {}; $\boxed{\mathit{val}(D) = \mathit{val}(I);}$**

   **accept, if *val*(D) != *error***

**I → L $I_1$**   ***str*(L) = *str*(I); *str*($I_1$) = *val*(L);**

   **$\boxed{\mathit{val}(I) = \mathit{val}(I_1)}$**

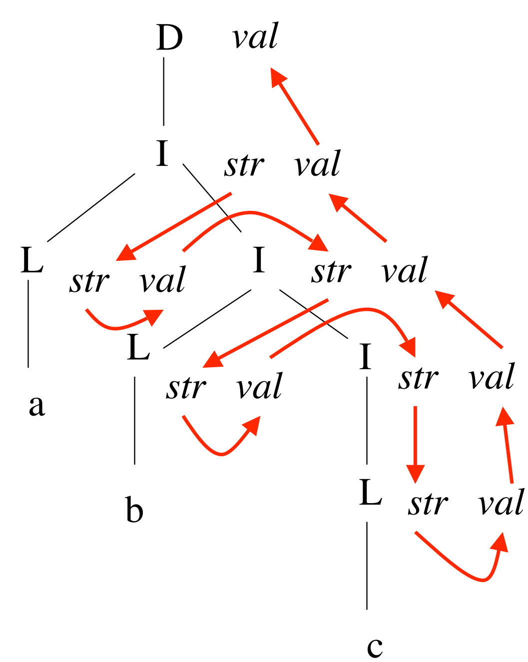
**I → L**   ***str*(L) = *str*(I); $\boxed{\mathit{val}(I) = \mathit{val}(L)}$**

**L → a | b | … | z**   **$\boxed{\mathit{val}(L)}$ = concatenation of *val* returned by scanner to *str*(L), if this character is not a repeated letter, else *error*.**

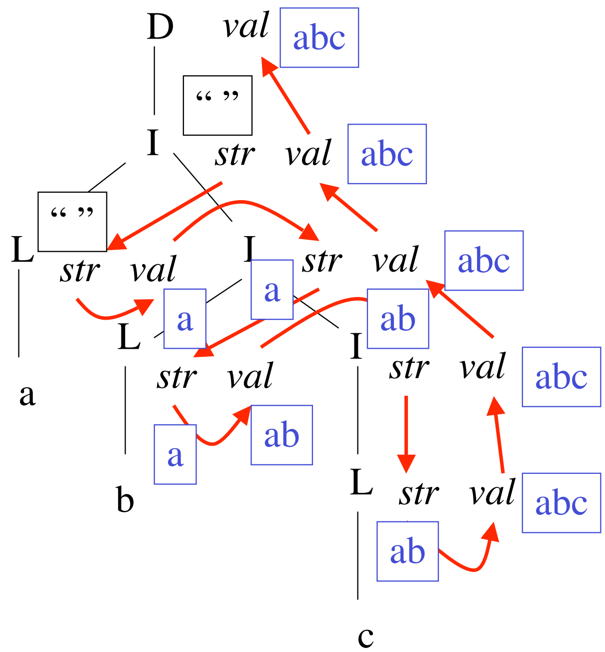
**(note: any comparison to *error* returns *error*.)**



# Parse Tree of abc



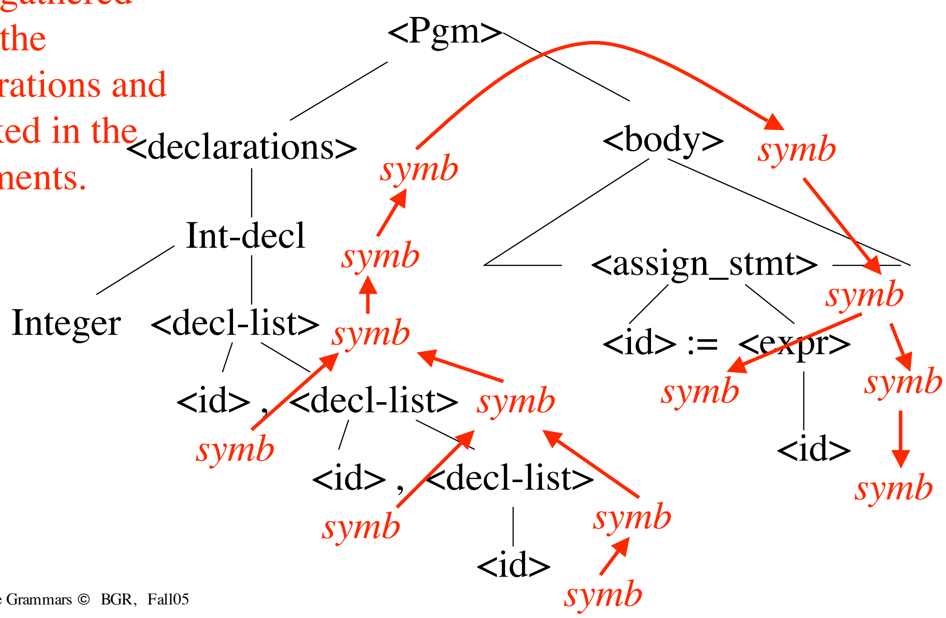
# Decorated Parse Tree






# Compiler Example

*symb* is the symbol table gathered from the declarations and checked in the statements.



# Syntax-directed Translation

- Idea: to use attribute grammars to cover some of the context-sensitive issues in translation
- Syntax-directed definition: an attributed grammar such that <u>every</u> grammar symbol has an attribute.
- Conceptually, attribute evaluation is
  – Build parse tree
  – Find attribute dependences
  – Decorate parse tree



# Evaluation Methods

- Want to interleave attribute evaluation with parsing
- Use dependence graph (but cannot handle circular dependences)
- Predetermine evaluation order at compiler construction time, using knowledge of grammar
- Ad-hoc: chosen parsing method imposes evalution order when interleaved with parsing; restricts grammars that can be handled

# S-attributed Grammars

- **S-attributed grammars:** **all attributes are synthesized**
- **Easy to interleave with BU parsing by using a parallel stack for attribute values**
  - **Evaluate as do a reduction**
- **Important: can code semantic functions *a priori,* because know all the handles from the grammar, so *know where the associated attributes will be in the stack when a reduction is about to take place*.**



# Attribute Grammars

- **Attributes:** **properties associated with nonterminal symbols of a context free grammar**
- **E.G., Binary numbers**

Semantic rules defining attributes are side-effect free

| | |
|---|---|
| **1.** $B \to 0$ | $val(B) = 0$ |
| **2.** $B \to 1$ | $val(B) = 1$ |
| **3.** $L \to B$ | $val(L) = val(B)$; $len(L) = 1$ |
| **4.** $L \to L_1\ B$ | $val(L) = 2*val(L_1) + val(B)$ |
| | $len(L) = len(L_1) + 1$ |
| **5.** $N \to L$ | $val(N) = val(L)$; $len(N) = len(L)$ |

# Example - Binary Nos

| | |
|---|---|
| 1. B → 0 | *val*(B) = 0 |
| 2. B → 1 | *val*(B) = 1 |
| 3. L → B | *val*(L) = *val*(B); *len*(L) = 1 |
| 4. L → $L_1$ B | *val*(L) = 2**val*($L_1$ ) + *val*(B) |
| | *len*(L) = *len*($L_1$ ) + 1 |
| 5. N → L | *val*(N) = *val*(L); *len*(N) = *len*(L) |

| Stack | Input | |
|---|---|---|
| $ | 1 1 $ | shift |
| $ (B 1 _) | 1 $ | red(2), *find B* |
| $ (L 1 1) | 1 $ | red(3), *find L* |
| $ (L 1 1) (1 1 _) | $ | shift |
| $ (L 1 1) (B 1 _) | $ | red(2), *find B* |
| $ (L 3 2) | $ | red (5), *find L* |
| $ (N 3 2) | $ | accept |

(<symbol> val() len())



# L-attributed Grammars

- Every attribute in the grammar is synthesized, or for production $A \rightarrow X_1 \ldots X_k$ an inherited attribute $X_k$ only depends on attributes of $X_1 \ldots X_{k-1}$ or inherited attributes of A.
- Can use *depth-first evaluation scheme* on parse tree
- Includes all syntax-directed definitions from LL(1) grammars

# L-attributed Grammars

*Translation scheme*: embeds semantic actions to evaluate attributes in RHS of productions (use {…} to delimit actions) to accomplish depth-first evaluation order

1. An inherited attributed for a nonterminal on RHS of production, must be computed in an action BEFORE that symbol



# L-attributed Grammars

2. An action cannot refer to a synthesized attribute of a symbol to the right of the action
3. A synthesized attribute of the LHS nonterminal can only be computed after all attributes it refers to are computed; place this action at the end of the RHS of the production

# Example - Identifiers as a Translation Scheme

D → {*str*(I) = ε } I {*val*(D) = *val*(I)}
{accept, if *val*(D) != *error*}

I → { *str*(L) = *str*(I)} L {*str*($I_1$ ) = *val*(L)} $I_1$
{*val*(I) = *val*($I_1$ )}

I → { *str*(L) = *str*(I) } L {*val*(I) = *val*(L)}

L → a | b | … | z {*val*(L) = concatenation of *val* returned by scanner to *str*(L), if this character is not a repeated letter, else *error*}

Try to evaluate earlier example abc with depth-first walk and these rules.



# Intuition

- Can see TD parsing relates well to L-attributed grammars
- Can see BU parsing relates well to S-attributed grammars

# BU Eval of Inherited Attribs

- ***Idea*: transform grammar so all embedded actions of translation scheme occur at end of RHS of some production (at a reduction) without changing LR(k) nature of the grammar**
- **Can handle all L-attributed defns corresponding to LL(1) grammars plus some LR(1)**



# Marker Nonterminals

***Used to move all actions to end of RHS of productions***

**Always $X \rightarrow \varepsilon$ for X, a marker nonterminal.**

**Replace an embedded action by a unique marker nonterminal that generates ε**

**Make the action for that nonterminal the same as the embedded action removed**

**But: grammar must stay LR(k) after these changes (this needs to be checked.)**

**Language accepted is same.**

**Actions occur in same order during parse.**

# Example, ASU p 309

S → E
E → E + T | E - T | T
T → *num*

becomes after recursion removal with actions:

S → E
E → T R
R → + T { print "+"} R
R → - T { print "-"} R
R → ε
T → *num* {print *num*}



# Marker Nonterminals

LR(1) grammar

S → E
E → T R
R → + T { print "+"} R
R → - T { print "-"} R
R → ε
T → *num* {print *num*}

After transformation

S → E
E → T R
R → + T M R
R → - T N R
R → ε
M → ε { print "+"}
N → ε {print "-"}
T → *num* {print

*num*}

# Marker Nonterminals (Copies)

- **Handling copy rules with marker nonterminals**

  **A $\rightarrow$ X Y where i(Y) = s(X)**

  **Translation scheme would be:**

  **A $\rightarrow$ X {i(Y) = s(X)} Y**

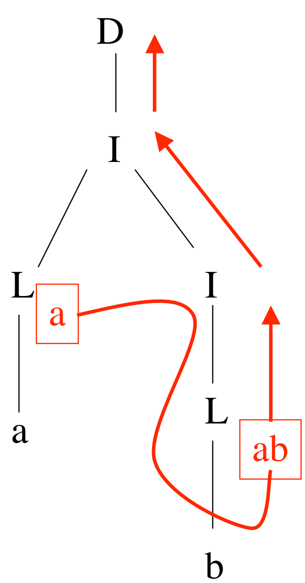
- **Example of this in our identifier grammar**

  ***str*($I_1$) = *val*(L) in I $\rightarrow$ L $I_1$**

  **would become I $\rightarrow$ L {*str*($I_1$) = *val*(L)} $I_1$**



# Example



# Example

| Stack | Input | Attribute Stack |
|---|---|---|
| \$ | a b \$ | \$ |
| \$ a | b \$ | \$ |
| \$ L | b \$ | a (val(L)) |
| \$ L b | \$ | a |
| \$ L $L_1$ | \$ | ab (val($L_1$)) |
| \$ L I | \$ | ab (val(I)) |
| \$ I | \$ | ab (val(I)) |
| \$ D | \$ | ab (val(D)) |



# Marker Nonterminals,(Copies ii)

- In previous example, copies never need to be performed as value is at top of attribute stack due to shape of grammar rules
- Not always this lucky

  S → a A C    i(C) = s(A) [1.]

  S → b A B C    i(C) = s(A) [2.]

  C → c    s(C) = g (i(C))

  Problem: in 1., s(A) is in stack(top) when find C but in 2., s(A) is in stack(top-1). Must rewrite grammar to try to make attribute value end up in same place in both rules.

# Grammar Transformation

| | |
|---|---|
| $S \rightarrow a\ A\ C$ | $i(C) = s(A)$ [1.] |
| $S \rightarrow b\ A\ B\ M\ C$ | $i(M) = s(A)$; $i(C) = s(M)$ [2.'] |
| $C \rightarrow c$ | $s(C) = g\ (i(C))$ |
| $M \rightarrow \varepsilon$ | $s(M) = i(M)$ |

M saves the value of s(A) so it goes on the value stack at the same place in both rules 1., 2.'; when encounter C, makes i(C) in same stack position.



# Marker Nonterminals, (Non-copies)

Previous transformation works even for non-copy actions:

if $S \rightarrow b\ A\ C$ has action $i(C) = f(s(A))$ then s(A) is on stack, not f(s(A)).

Fix:

| | |
|---|---|
| $S \rightarrow a\ A\ N\ C$, | $i(N) = s(A)$, $i(C) = s(N)$ |
| $N \rightarrow \varepsilon$ , | $s(N) = f(i(N))$ |

Problem: can destroy the LR(1) property of grammar with added markers; LL(1) grammars remain okay.

# Top Down Translation

- L-attributed grammars work well with TD translation, but when remove left recursion must also transform attributes
- Involves changing all synthesized attributes to a mixture of inherited and synthesized



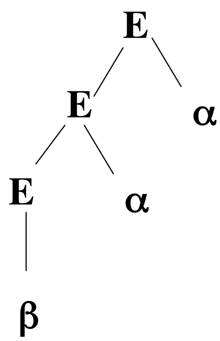
# Example

| | |
|---|---|
| $S \rightarrow E$ | $\mathit{val}(S) = \mathit{val}(E)$ |
| $E \rightarrow E_1 + T$ | $\mathit{val}(E) = \mathit{val}(E_1) + \mathit{val}(T)$ |
| $E \rightarrow E_1 - T$ | $\mathit{val}(E) = \mathit{val}(E_1) - \mathit{val}(T)$ |
| $E \rightarrow T$ | $\mathit{val}(E) = \mathit{val}(T)$ |
| $T \rightarrow \mathit{int}$ | $\mathit{val}(T) = \mathit{int\_value}$ |

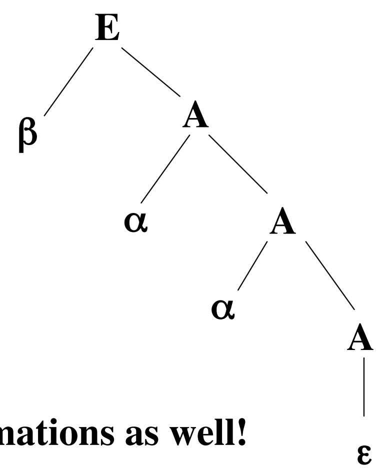
All synthesized attributes (L-attributed).

# Removing Left Recursion

$E \rightarrow E\,\alpha \mid \beta \qquad E \rightarrow \beta\, A;\ \ A \rightarrow \alpha\, A \mid \varepsilon$




becomes




**Want to specify attribute transformations as well!**



# Example

$S \rightarrow E\ \{val(S) = val(E)\}$

$E \rightarrow T\ \{i(R) = val(T)\}\ R\ \{val(E) = s(R)\}$

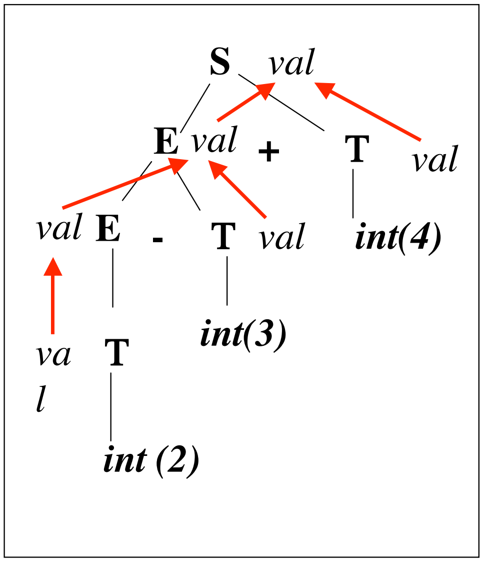
$R \rightarrow +\ T\ \{i(R_1) = i(R) + val(T)\}\ R_1$
$\qquad \{s(R) = s(R_1)\}$

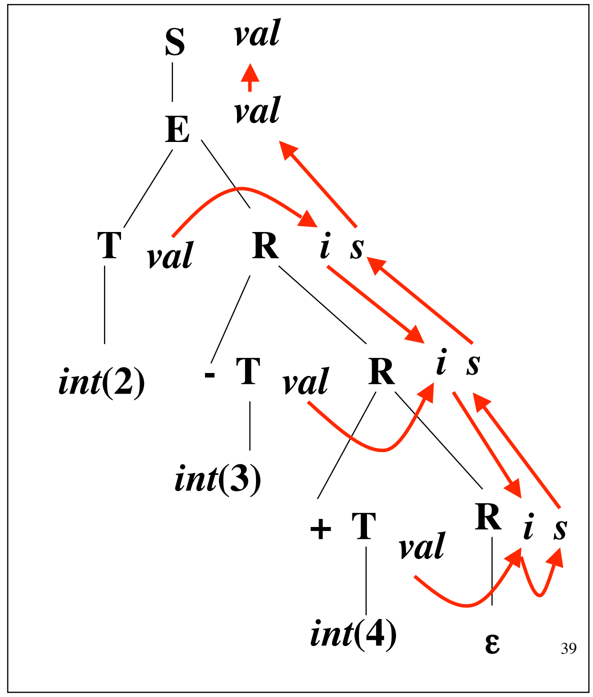
$R \rightarrow -\ T\ \{i(R_1) = i(R) - val(T)\}\ R_1$
$\qquad \{s(R) = s(R_1)\}$

$R \rightarrow \varepsilon\ \{s(R) = i(R)\}$

$T \rightarrow int\ \{val(T) = int\_const\}$

# Corresponding Parse Trees




$\boxed{2 - 3 + 4}$



# Transformation, ASUp304ff

$$A \rightarrow A_1 \ Y \quad \{a(A) = g(a(A_1), y(Y))\}$$

$$A \rightarrow X \qquad \{a(A) = f(x(X))\}$$

**Becomes**

$$A \rightarrow X \ \{i(R) = f(x(X))\} \ R \ \{ a(A) = s(R)\}$$

$$R \rightarrow Y \ \{i(R_1) = g(i(R), y(Y))\} \ R_1 \{s(R) = s(R_1)\}$$

$$R \rightarrow \varepsilon \quad \{s(R) = i(R)\}$$

# Transformation