

$D \rightarrow T$ $\{ \text{type: id.ref} \}$
 $T \rightarrow \text{id}$ $\{ \text{type: id.ref} \}$
 $T \rightarrow \text{ref}$ $\{ \text{type: id.ref} \}$
 $L \rightarrow \text{id}$ $\{ \text{type: id.ref} \}$
 $L \rightarrow L_1, \text{id}$ $\{ \text{type: id.ref} \}$

System-directed translation scheme
 Just like an SDD, but instead of rules we have semantic actions. Code which may or may not compute attributes in the production, and may have side-effects (printing object code) written in special blocks.

This is what bison does for you to do.
 Ex: Prefix to postfix
 $C \rightarrow (A + B) \Rightarrow C \text{ is } A + B$
 $E \rightarrow E + T \quad \{ \text{print}(" + ") \}$
 $E \rightarrow T \quad \{ \}$
 $T \rightarrow T * F \quad \{ \text{print}(" * ") \}$
 $T \rightarrow F \quad \{ \}$
 $F \rightarrow (E) \quad \{ \}$
 $F \rightarrow \text{const} \quad \{ \text{print}(\text{val}, \text{const}); \}$

SDDs & SDDs don't have any number for the same symbol.

Mar 23-11:02 AM

Intermediate address
 In an SDD actions can occur anywhere (with carets) in the production (not just at the end).
 bison allows this
 $A \rightarrow X Y$
 $A \rightarrow \{ \text{action} \} X \{ \text{action} \} Y \{ \text{action} \}$
 $A \rightarrow _A1 _ X _ A2 _ Y _ \{ \text{action} \}$
 $A1 \rightarrow _ \text{empty} _ / \{ \text{action} \}$
 $A2 \rightarrow _ \text{empty} _ / \{ \text{action} \}$
 $\{ \text{action} \} _ A1 _ A2 _ \{ \}$
 $A : \{ \text{action} \} X \{ \text{action} \} Y \{ \}$
 In production part
 immediately after action takes place
 $\$n$ means n-th after from last

Mar 23-11:35 AM

One exception to bison's implicit translation of intermediate address.
 $A : X Y \{ \text{action} \} Z$
 $A : X Y _ A1 _ Z$
 $A1 : \{ \text{action} \} _ A2 _ Z$
 bison does this automatically.
 Doing this 'by hand':
 $A : X Y \{ \text{action} \} Z$
 $\text{get_sum} : _ \text{empty} _ / \{ \text{action} \} _ A1 _ A2 _ Z$
 $\$1$ $\$2$ $\$3$ $\$4$ $\$5$
 $\$1$ $\$2$ $\$3$ $\$4$ $\$5$
 $\$n$ if $n \leq 0$, bison assumes the default type unless you tell it otherwise.
 $\%union \{ \}$
 $\%type \{ \}$
 $\%type \{ \}$
 $\%type \{ \}$
 $\text{get_sum} : \{ \text{action} \} _ A1 _ A2 _ Z$
 $\$1$ $\$2$ $\$3$ $\$4$ $\$5$

Mar 23-11:50 AM

Project 1:
 Implement expressions
 assignment statements
 production definitions
 proc/for calls
 program for
 type
 ...
 var ...
 { produce fact (param);
 type ...
 begin ...
 end;
 function ...
 type ...
 begin ...
 end;
 begin;
 end;
 A begin-end block in Pascal is a sequence of 0 or more statements (between begin and end) separated by semicolons.
 A block is itself a statement.
 Assignment statement
 var $x = \text{expression}$
 $+$, $-$, $*$ same as in C
 $/$ real division only
 div and mod (integer division)

Mar 23-12:06 PM

An expression can also be a function call

e.g. $\text{fact}(5)$

A procedure call is considered a statement.

Mar 23-12:15 PM