

University of Kansas | Drew Davidson

ECS 665
COMPILER
CONSTRUCTION

The title is set against a dark blue and purple background with a starry space theme. The words 'COMPILER' and 'CONSTRUCTION' are in large, bold, metallic-style fonts. 'COMPILER' has a blue and white gradient with a black outline. 'CONSTRUCTION' is in a smaller, italicized, blue font with a white outline. Above 'COMPILER' is the text 'ECS 665' in a pink, cursive, italicized font. The background features a grid of purple lines and several bright stars.

Lecture 5
Syntax-Directed Definition

I *Think* We're Done with Review

Lecture 5 – Syntax-Directed Definition

COMPILER

LAND

Code
Generation

Execution

Runtime
Environment

Intermediate
Representation

Optimization

SDD

Semantics

Parsing

Lexical
Analysis

Syntactic
Definition

Lecture Outline

Syntax-Directed Definition

Recall Syntactic Ambiguity

Assigning Meaning to (Parse) Trees

- Tree translation intuition
- Introduce Syntax-Directed Definition

Tools for SDD

- Bison



**Syntax-Directed
Definition**

Last Time

Review Lecture 4 –Syntactic Ambiguity

Recognizing Context-Free Grammars

- The parser wants a parse tree

Some Challenges in Syntactic Analysis

- Ambiguous Syntax
 - Precedence
 - Associativity



**Syntactic
Definition**

Last Time

Review Lecture 4 –Syntactic Ambiguity

Force precedence
constraints

Force associativity
constraints

$E := E \text{ minus } E$
 $E := E \text{ times } E$
 $E := E \text{ pow } E$

$E := E \text{ minus } E$
| T
 $T := T \text{ times } T$
| F
 $F := F \text{ pow } F$
| G
 $G := \text{intlit}$

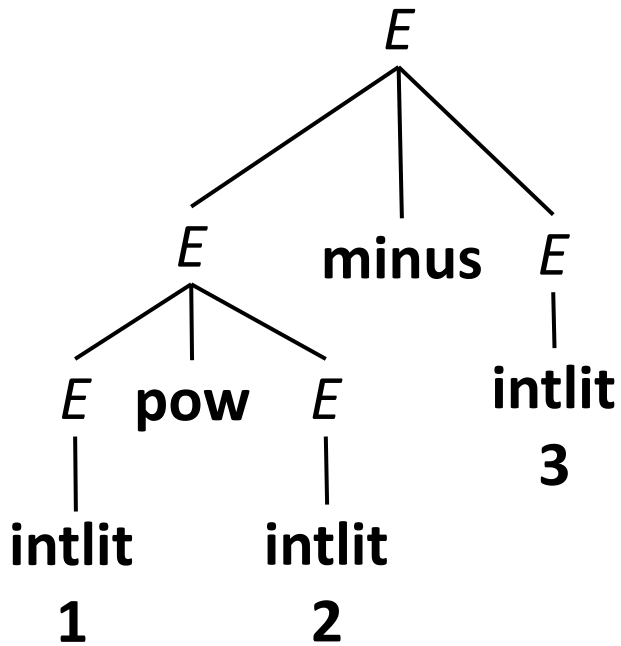
$E := E \text{ minus } T$
| T
 $T := T \text{ times } F$
| F
 $F := G \text{ pow } F$
| G
 $G := \text{intlit}$



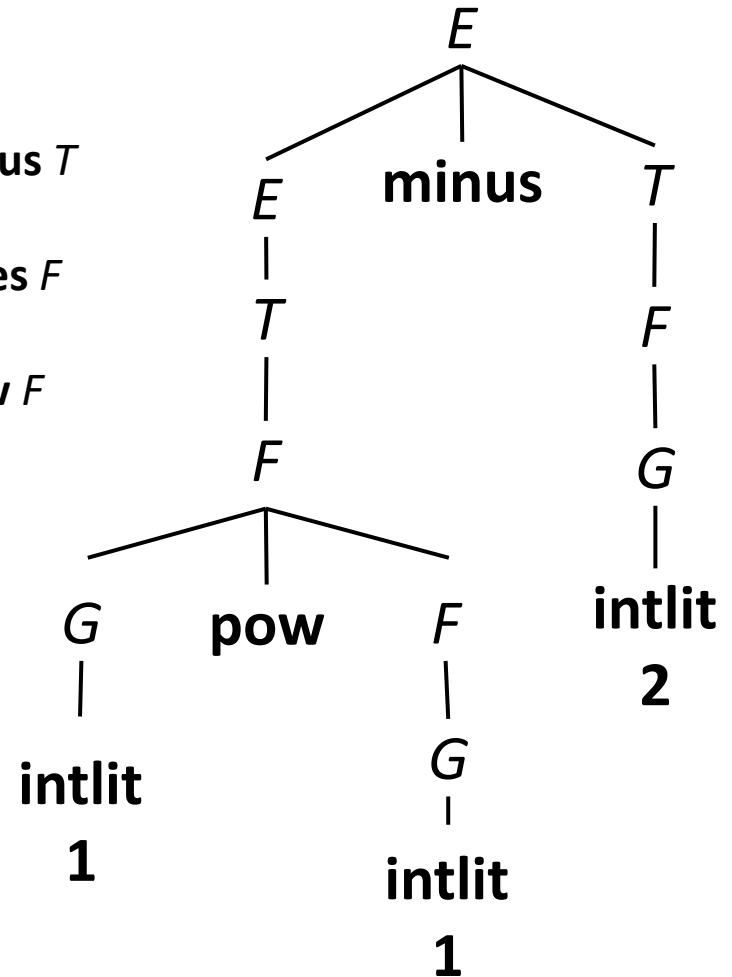
Syntactic
Definition

Ensure Bad Exprs are Invalid Syntax

Working with Parse Trees



$E := E \text{ minus } T$
| T
 $T := T \text{ times } F$
| F
 $F := G \text{ pow } F$
| G
 $G := \text{intlit}$



Lecture Outline

Preview Lecture 5 – Syntax-Directed Translation

Recall Syntactic Ambiguity

Assigning Meaning to (Parse) Trees

- Tree translation intuition
- Introduce Syntax-Directed Definition

Tools for SDD

- Bison



**Syntax-Directed
Definition**

Benefits of Parse Trees

Assigning Meaning to Parse Trees

All of the known methods for defining the meaning of computer programs were based on rather intricate algorithms having roughly the same degree of complexity as compilers, or worse. This was in stark contrast to Chomsky's simple and elegant method of syntax definition via context-free grammars. As Dr. Caracciolo said, "How simple to realize [semantic correctness] if you write a procedure. The problem is, however, to find a metalanguage for doing that in a declarative way, not in an operational way" [3].

Benefits of Parse Trees

Assigning Meaning to Parse Trees

- Impose structure on tokens
- Easy to specify
- **Easy** to process

good algorithms
are known



More generally:

Trees are great data structures
for nesting relationships

Two Ways of Thinking About Trees

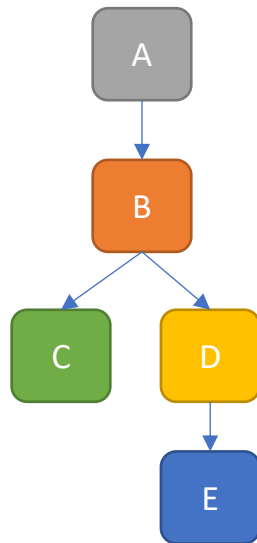
Working with Trees

As a type of graph

Compilers
use insights
from both
views

Root is a node, children are successors

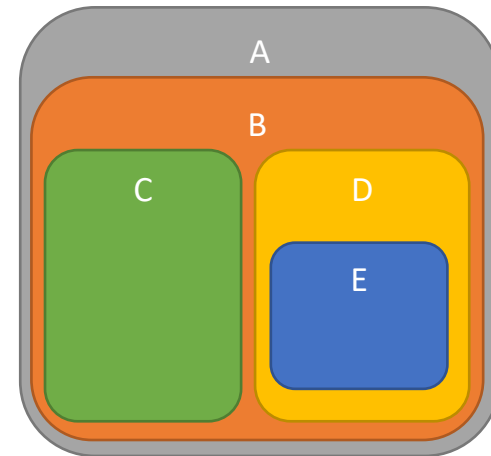
Work “root-down” vs “leaves-up”



As a type of nesting

Root is a tree, children are inside

Work “outside in” vs “inside out”

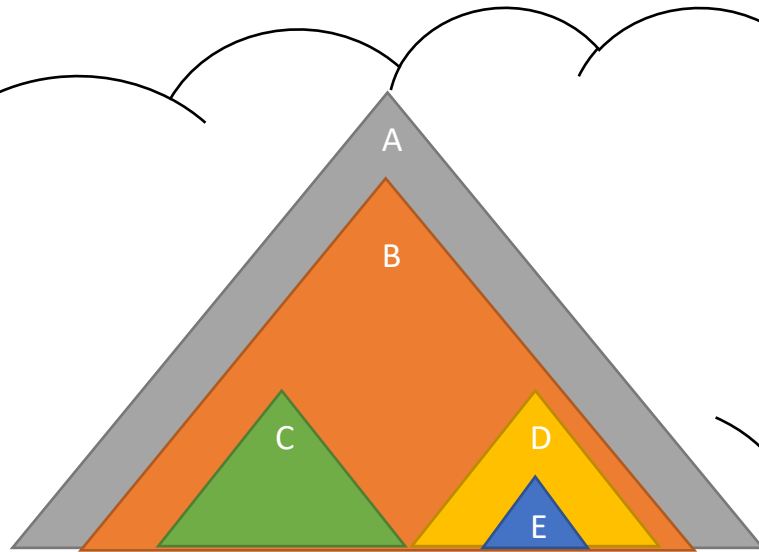


Two Ways of Thinking About Trees

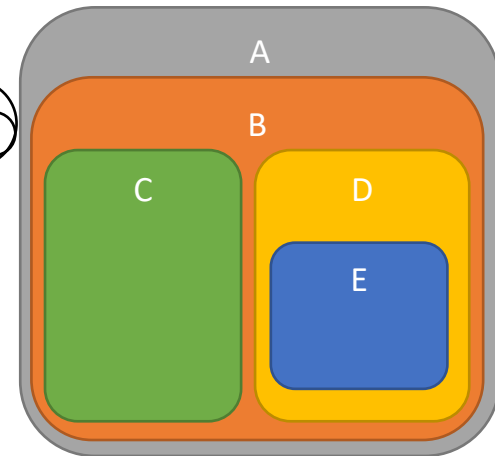
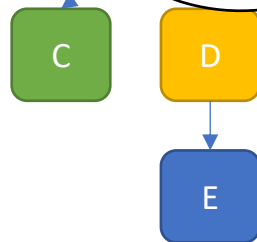
Working with Trees

As a type of nesting

Root is a tree, children are inside
Work “outside in” vs “inside out”

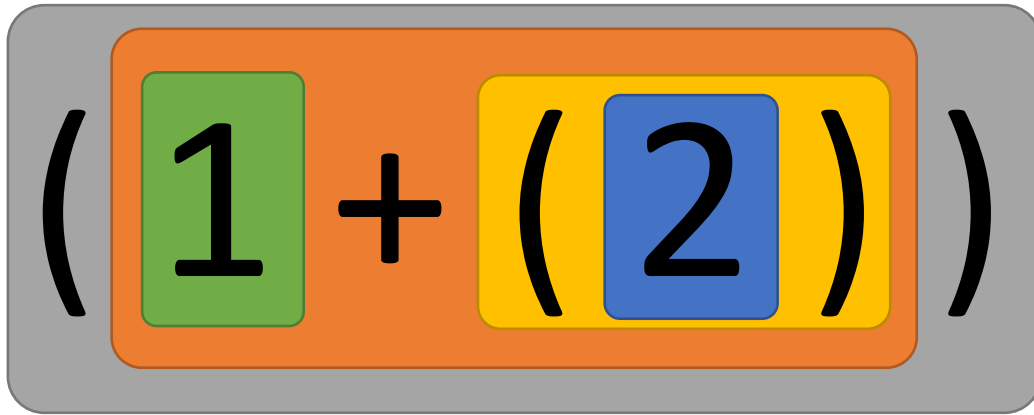


Same but subtrees depicted as triangles

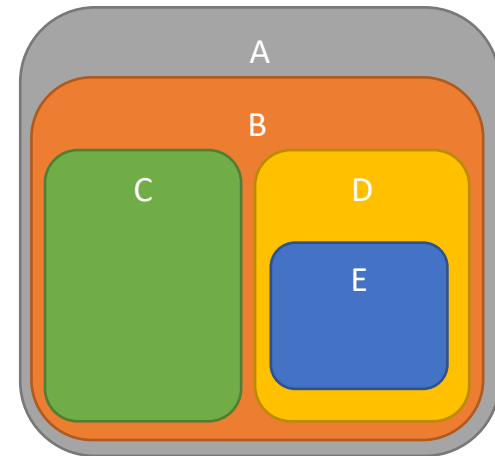


Trees as a Nesting

Working with Trees



$A = (B)$
 $B = C + D$
 $C = 1$
 $D = (E)$
 $E = 2$



Trees as a Nesting

Working with Trees

also 10, 11]. If we know the meaning of α and the meaning of β , then we know the meanings of such things as $\alpha + \beta$, $\alpha - \beta$, $\alpha \times \beta$, α/β , and (α) . Thus the meaning of an arbitrarily large expression can be synthesized in a straightforward way by recursively building up the meanings of its subexpressions.

Assigning Meaning to Subtrees

Processing Parse Trees

Assign a **translation** for each node / subtree

Goal:

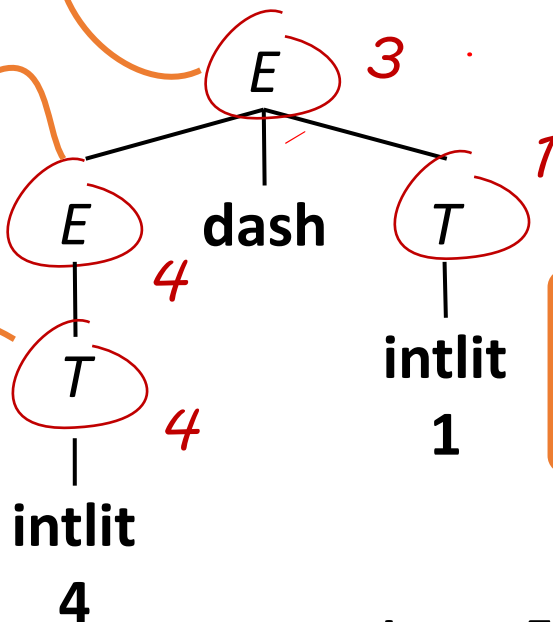
Translation is the value of the expression

$E \text{ trans} =$
subtree 1 - subtree 3
translation translation

$E \text{ trans} =$
subtree
translation

$T \text{ trans} =$
subtree
literal value

$T \text{ trans} =$
subtree
literal value



Grammar:

$E ::= E \text{ dash } T$
 $\quad \mid T$
 $T ::= \text{intlit}$

Input Expression: 4 - 1

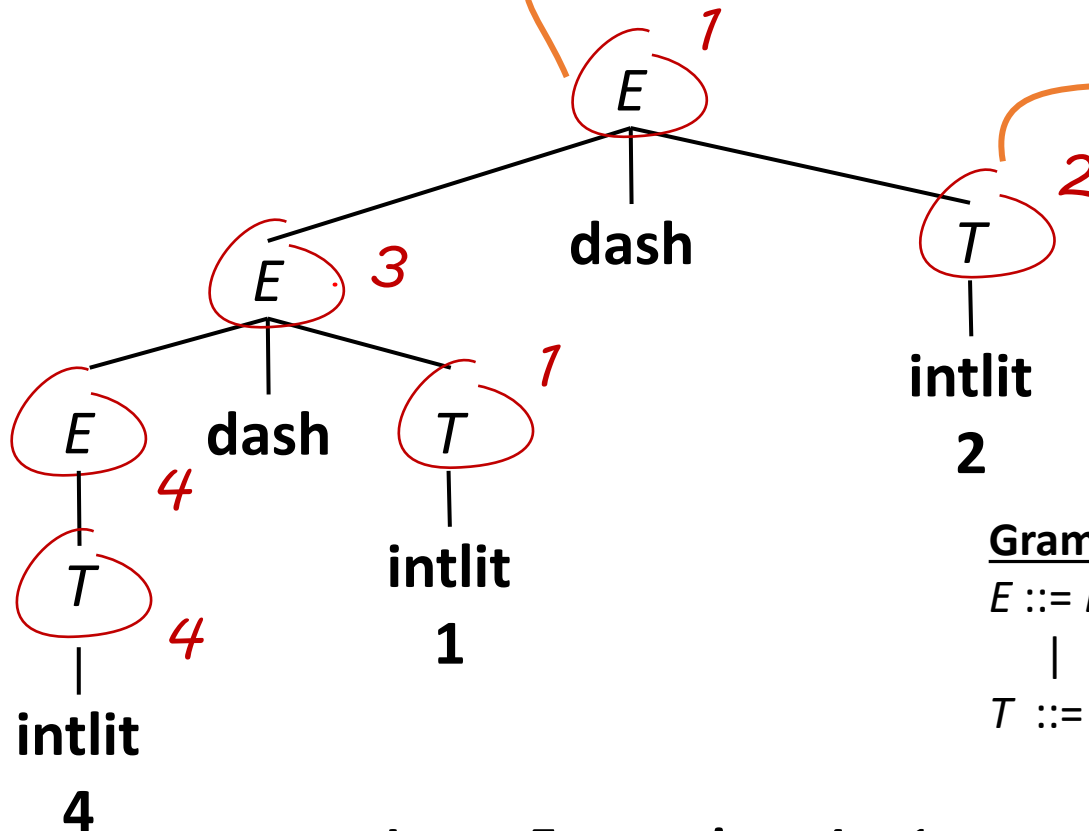
Assigning Meaning to Subtrees

Processing Parse Trees

Assign a **translation** for each node / subtree

$E \text{ trans} =$
 $\text{subtree 1 translation} - \text{subtree 3 translation}$

Goal:
Translation is
the value of
the expression



$T \text{ trans} =$
 subtree
 literal value

Grammar:
 $E ::= E \text{ dash } T$
 $\quad | \quad T$
 $T ::= \text{intlit}$

Input Expression: 4 - 1 - 2

Assigning Meaning to Subtrees

Processing Parse Trees

Assign a **translation** for each node / subtree

$E \text{ trans} = \text{subtree 1 translation} - \text{subtree 3 translation}$

Goal:
Translation is the value of the expression

$E \text{ trans} = \text{subtree 1 translation} - \text{subtree 3 translation}$

$T \text{ trans} = \text{subtree literal value}$

$E \text{ trans} = \text{subtree translation}$

dash

intlrit
2

dash

intlrit
1

$T \text{ trans} = \text{subtree literal value}$

Grammar:
 $E ::= E \text{ dash } T$
 $T ::= \text{intlrit}$

$T \text{ trans} = \text{subtree literal value}$

intlrit
4

Input Expression: 4 - 1 - 2

In Summary

Processing Parse Trees

- Translation depends on the goal



Assigning Meaning to Subtrees

Processing Parse Trees

Goal:
Translation is
the number of
operators

$$E \text{ trans} = \text{subtree 1 translation} + \text{subtree 3 translation} + 1$$

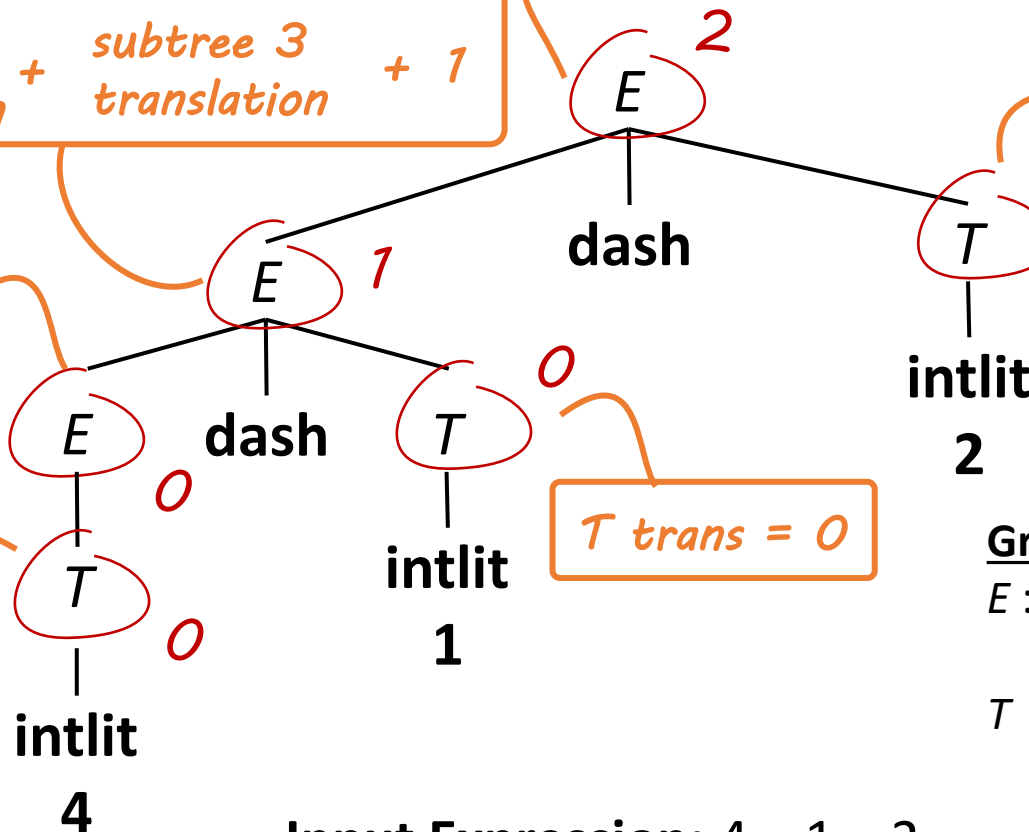
$$E \text{ trans} = \text{subtree 1 translation} + \text{subtree 3 translation} + 1$$

$$T \text{ trans} = 0$$

$$E \text{ trans} = \text{subtree translation}$$

$$T \text{ trans} = 0$$

$$T \text{ trans} = 0$$



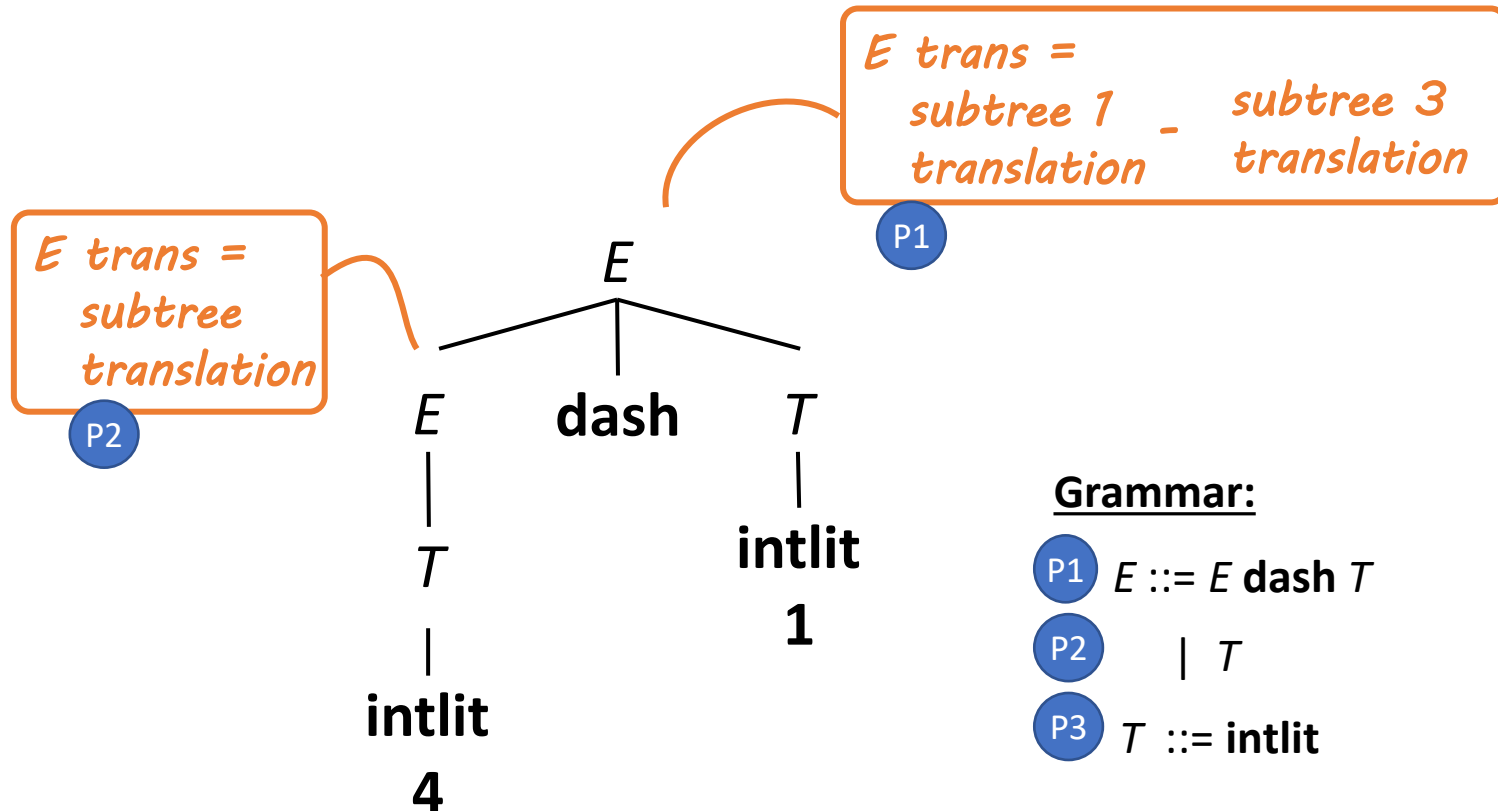
Grammar:
 $E ::= E \text{ dash } T$
 $\quad | \quad T$
 $T ::= \text{intlit}$

Input Expression: 4 - 1 - 2

Tree Translation Intuition: Summary

Processing Parse Trees

- Translation depends on the goal
- Translation is selected based on the production
- Conceptually, a post-pass over the complete parse tree



Lecture Outline

Preview Lecture 5 – Syntax-Directed Translation

Recall Syntactic Ambiguity

Assigning Meaning to (Parse) Trees

- Tree translation intuition
- Introduce Syntax-Directed Definition

Tools for SDD

- Bison



**Syntax-Directed
Definition**

Syntax-Directed Definitions

Processing Parse Trees

Semantics of Context-Free Languages

by

DONALD E. KNUTH

California Institute of Technology

ABSTRACT

“Meaning” may be assigned to a string in a context-free language by defining “attributes” of the symbols in a derivation tree for that string. The attributes can be defined by functions associated with each production in the grammar. This paper

Syntax-Directed Definitions

Processing Parse Trees

- Attach translation rules per-production

$X ::= \alpha_1 \dots \alpha_n \{ LHS.trans = \langle \text{translation of } X \text{ that can use translations of } \alpha_1 \dots \alpha_n \rangle \}$
 $\quad | \beta_1 \dots \beta_n \{ LHS.trans = \langle \text{translation of } X \text{ that can use translations of } \beta_1 \dots \beta_n \rangle \}$

$Y ::= \gamma_1 \dots \gamma_n \{ LHS.trans = \langle \text{translation of } Y \text{ that can use translations of } \gamma_1 \dots \gamma_n \rangle \}$



Goal:
Translation is
the value of
the expression

Grammar:

P1 $E ::= E \text{ dash } T \{ LHS.trans = E.trans - T.trans \}$

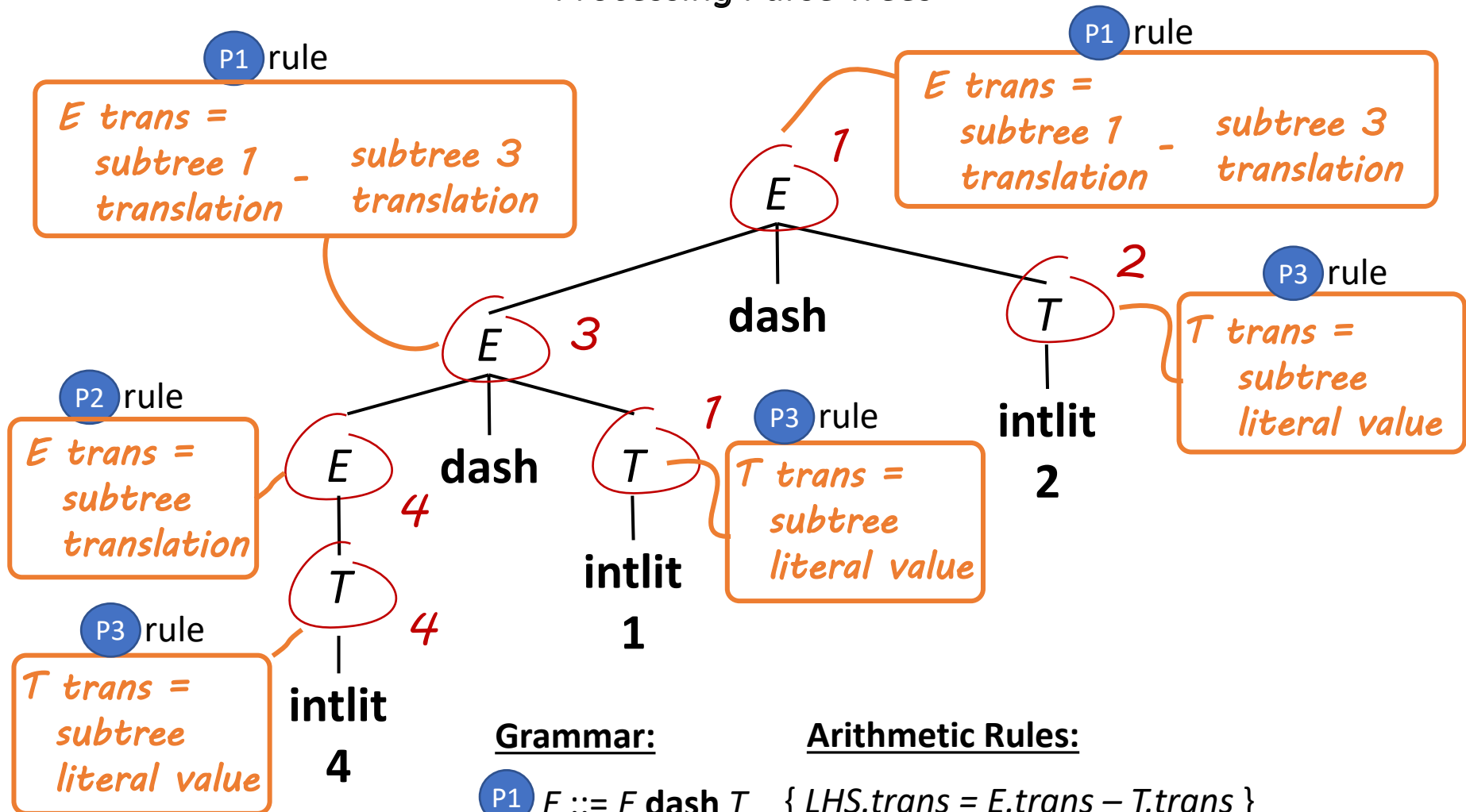
P2 $\quad | T \{ LHS.trans = T.trans \}$

P3 $T ::= \text{intlit} \{ LHS.trans = \text{intlit.value} \}$

Arithmetic Rules:

SDD: Example

Processing Parse Trees



Grammar:

- P1** $E ::= E \text{ dash } T$ { $LHS.trans = E.trans - T.trans$ }
- P2** $| T$ { $LHS.trans = T.trans$ }
- P3** $T ::= \text{intlit}$ { $LHS.trans = \text{intlit.value}$ }

Arithmetic Rules:

Lecture Outline

Syntax-Directed Definition

Working with Parse Trees

- Benefits of Parse Trees / Trees in general
- Tree translation intuition
- Syntax-Directed Definition
 - Finer points

Tools for SDD

- Bison



**Syntax-Directed
Definition**

SDD: Parse Tree Processing Multitool

Processing Parse Trees

SDD is a flexible tool for assigning meaning to parse trees

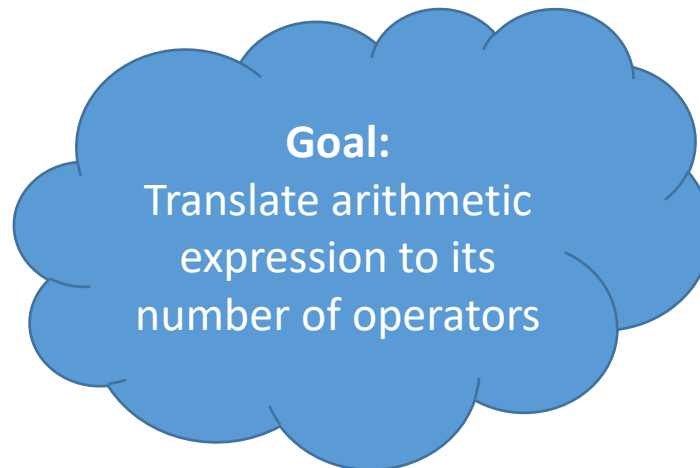
- Useful beyond compilers
- We won't even use the full power of the technique



SDD: A Different Translation Scheme

Processing Parse Trees

- SDD can do more than evaluate expressions



Grammar:

P1 $E ::= E \text{ dash } T$

P2 $\quad | T$

P3 $T ::= \text{intlit}$

P4 $T ::= T \text{ mult intlit}$

Operator Count Rules:

$\{ LHS.trans = E.trans + T.trans + 1 \}$

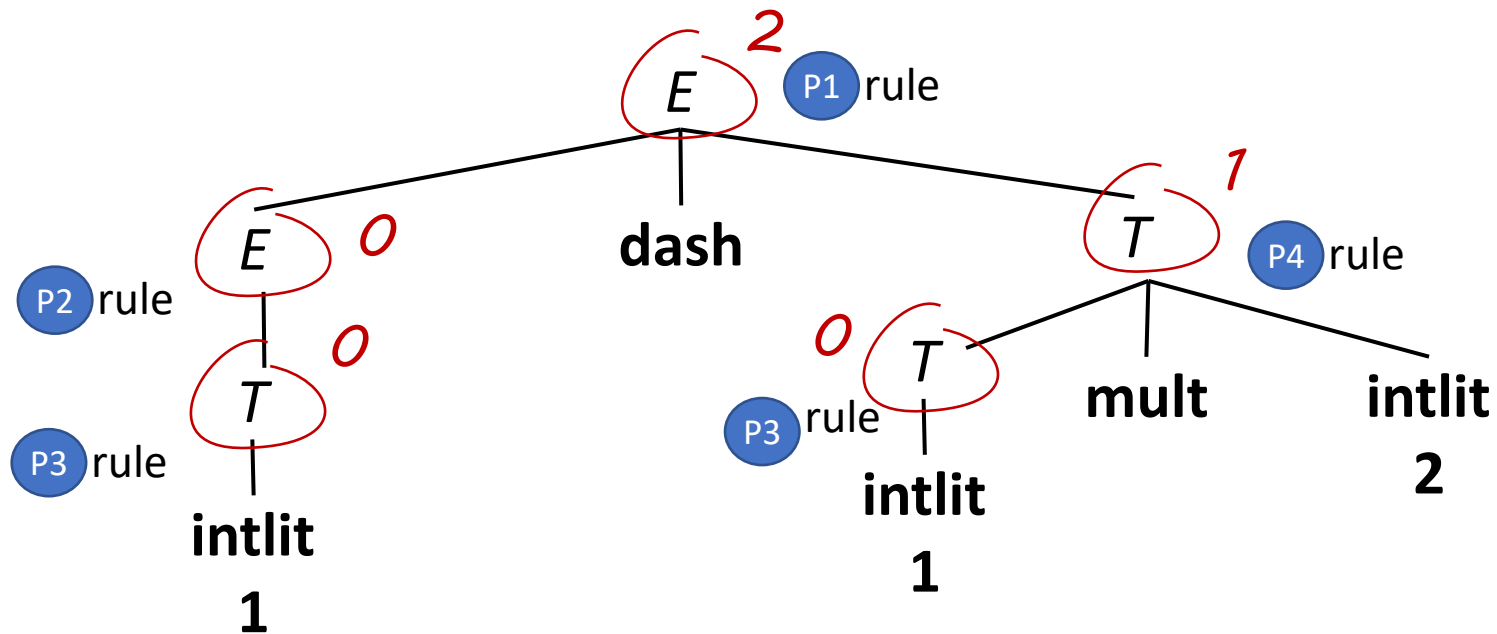
$\{ LHS.trans = T.trans \}$

$\{ LHS.trans = 0 \}$

$\{ LHS.trans = T.trans + 1 \}$

SDD: A Different Translation Scheme

Processing Parse Trees



Grammar:

P1 $E ::= E \text{ dash } T$

P2 $\quad \mid T$

P3 $T ::= \text{intlit}$

P4 $T ::= T \text{ mult intlit}$

Operator Count Rules:

$\{ LHS.trans = E.trans + T.trans + 1 \}$

$\{ LHS.trans = T.trans \}$

$\{ LHS.trans = 0 \}$

$\{ LHS.trans = T.trans + 1 \}$

SDD: Yet *Another* Translation

Processing Parse Trees

- Myth: SDD translations are always int values



Grammar:

P1 $E ::= E \text{ dash } T$

P2 $\quad \mid T$

P3 $T ::= \text{intlit}$

P4 $T ::= T \text{ mult intlit}$

List of integers Rules:

$\{ LHS.trans = E.trans.extend(T.trans) \}$

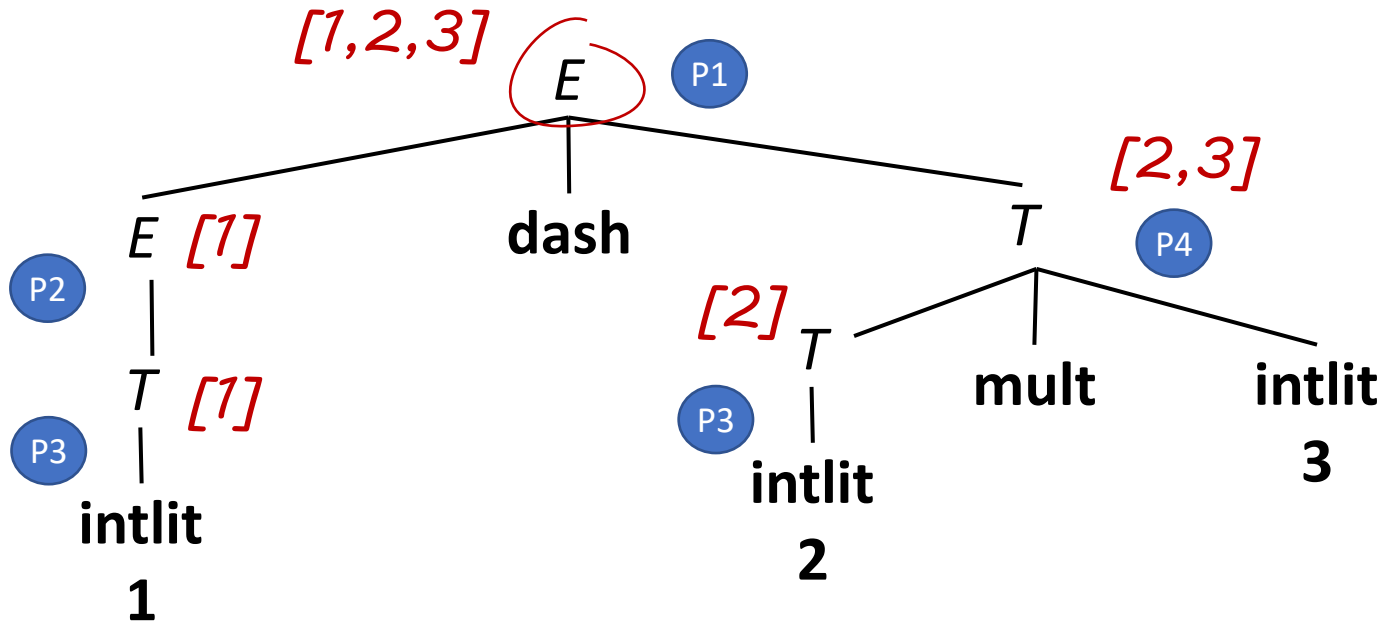
$\{ LHS.trans = T.trans \}$

$\{ LHS.trans = [\text{intlit.value}] \}$

$\{ LHS.trans = T.trans.extend([\text{intlit.value}]) \}$

SDD: Yet *Another* Translation

Processing Parse Trees



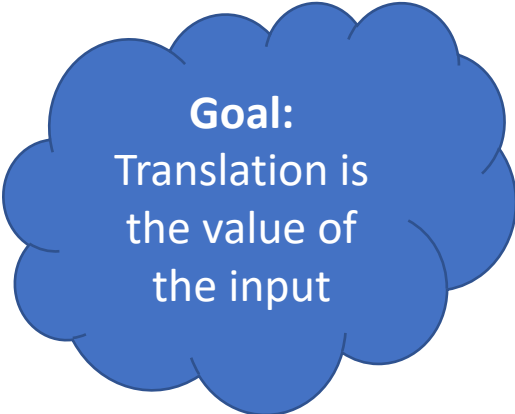
Grammar:

- | | |
|--|---|
| <div style="display: flex; align-items: center;"> <div style="background-color: #4a7ebb; color: white; border-radius: 50%; width: 30px; height: 30px; display: flex; align-items: center; justify-content: center; margin-right: 5px;">P1</div> <div> $E ::= E \text{ dash } T$ </div> </div> | $\{ LHS.trans = E.trans.extend(T.trans) \}$ |
| <div style="display: flex; align-items: center;"> <div style="background-color: #4a7ebb; color: white; border-radius: 50%; width: 30px; height: 30px; display: flex; align-items: center; justify-content: center; margin-right: 5px;">P2</div> <div> T </div> </div> | $\{ LHS.trans = T.trans \}$ |
| <div style="display: flex; align-items: center;"> <div style="background-color: #4a7ebb; color: white; border-radius: 50%; width: 30px; height: 30px; display: flex; align-items: center; justify-content: center; margin-right: 5px;">P3</div> <div> $T ::= \text{intlit}$ </div> </div> | $\{ LHS.trans = [\text{intlit.value}] \}$ |
| <div style="display: flex; align-items: center;"> <div style="background-color: #4a7ebb; color: white; border-radius: 50%; width: 30px; height: 30px; display: flex; align-items: center; justify-content: center; margin-right: 5px;">P4</div> <div> $T ::= T \text{ mult intlit}$ </div> </div> | $\{ LHS.trans = T.trans.extend([\text{intlit.value}]) \}$ |

SDD: Binary Numbers

Processing Parse Trees

<u>CFG</u>	<u>Rules</u>
P1: $B ::= 0$	$LHS.trans = ???$
P2: $ 1$	$LHS.trans = ???$
P3: $ B 0$	$LHS.trans = ???$
P4: $ B 1$	$LHS.trans = ???$



Goal:
Translation is
the value of
the input

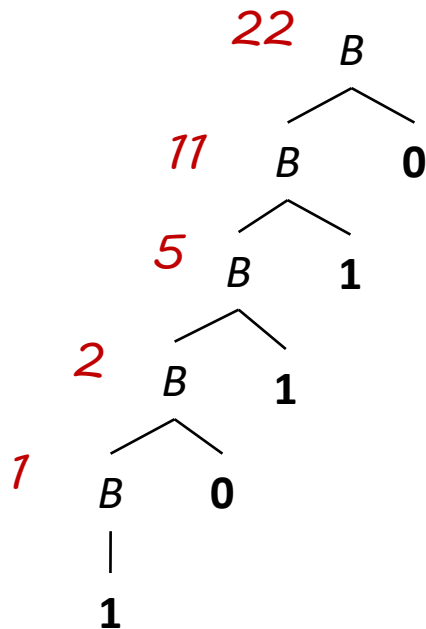
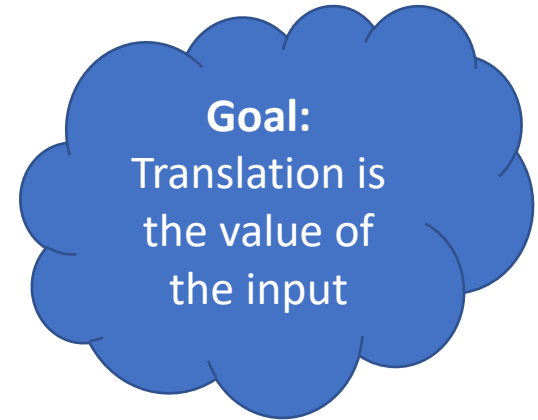
Example:

Input string 10110
should be 22

SDD: Binary Numbers

Processing Parse Trees

<u>CFG</u>	<u>Rules</u>
P1: $B ::= 0$	$LHS.trans = 0$
P2: $ 1$	$LHS.trans = 1$
P3: $ B 0$	$LHS.trans = B_1.trans * 2$
P4: $ B 1$	$LHS.trans = B_1.trans * 2 + 1$



Example:

Input string 10110
should be 22

SDD: Int Declaration List

Processing Parse Trees

CFG

$DList ::= \epsilon$
 $\quad \mid DList Decl$
 $Decl ::= Type \textbf{id} ;$
 $Type ::= \textbf{int}$
 $\quad \mid \textbf{bool}$

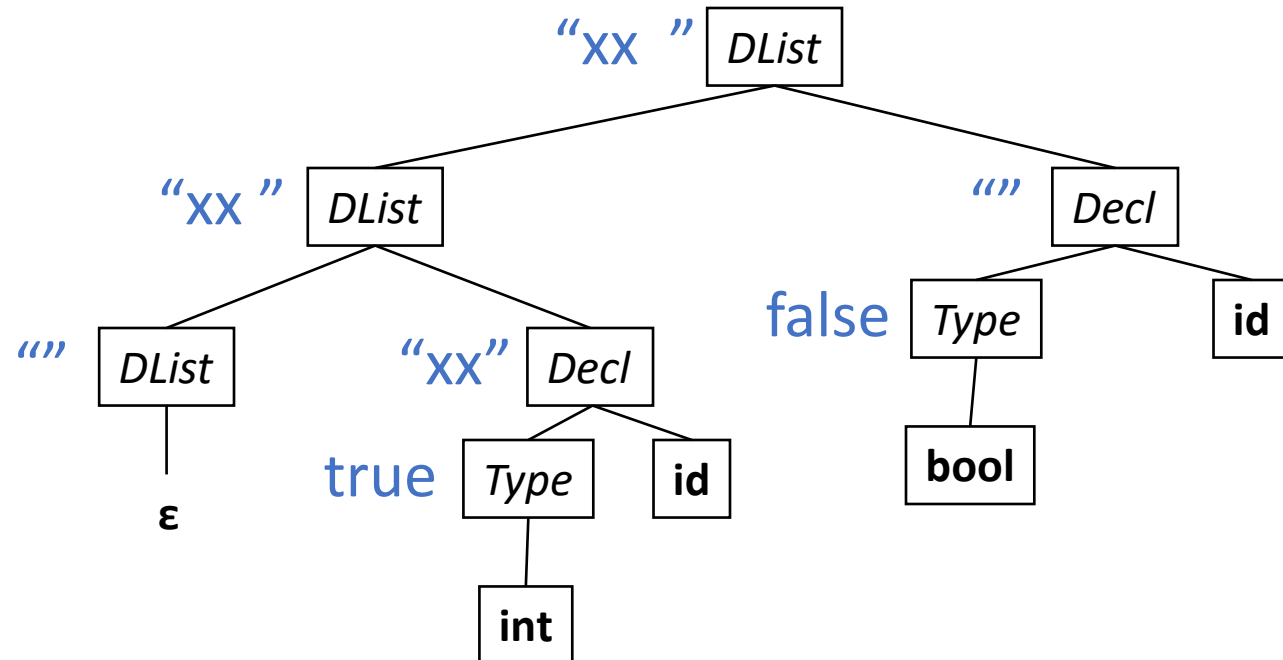
Rules

$LHS.trans = ""$
 $LHS.trans = Decl.trans + " " + DList_2.trans$
 if ($Type.trans$) { $LHS.trans = \textbf{id.value}$ } else { $Decl.trans = ""$ }
 $LHS.trans = \text{true}$
 $LHS.trans = \text{false}$

Input string

int xx;
 bool yy;

Translation is a
String of **int** ids
only



BISON: A tool for SDD

Assigning Meaning to Parse Trees

```
15 %union {
16     int intval;
17 }
18
19 %token zero
20 %token one
21
22
23 %type <intval> B
24
25 %%
26
27 S : B      { printf("Value is %d\n", $1); }
28 B : zero   { $$ = 0; }
29   | one    { $$ = 1; }
30   | B zero { $$ = $1 * 2 + 0; }
31   | B one  { $$ = $1 * 2 + 1; }
32 ;
33
34 %%
```

Next Time

Lecture Preview

Discuss a use of SDD of particular use to Compilers:

- Translating the parse tree into an *Abstract Syntax Tree*