Plan for Today

REMINDERS

- NO Recitation tomorrow
- HW5 is due Monday.
- PA3 is due on Monday October 17th
- HW4 feedback will be provided by Saturday night

Plan

- Dangling Else Problem
- Forcing Type Checking in Haskell
- Mixed Byte and Int
- Control-flow Code Gen
- Building AST

Left Factoring

Left recursion does not work for predictive parsing. Neither does a grammar that has a non-terminal with two productions that start with a common phrase, so we left factor the grammar:

$$\begin{array}{l} S \rightarrow \alpha\beta_1 \\ S \rightarrow \alpha\beta_2 \end{array} \xrightarrow{\textit{Left refactor}} \begin{array}{l} S \rightarrow \alpha S' \\ S' \rightarrow \beta_1 \mid \beta_2 \end{array}$$

E.g.: if statement:

S → IF t THEN S ELSE S | IF t THEN S | o

becomes

S → IF t THEN S X | o

X→ ELSE S | ε

When building the predictive parse table, there will be a multiple entries.

WHY?

Dangling else problem: ambiguity

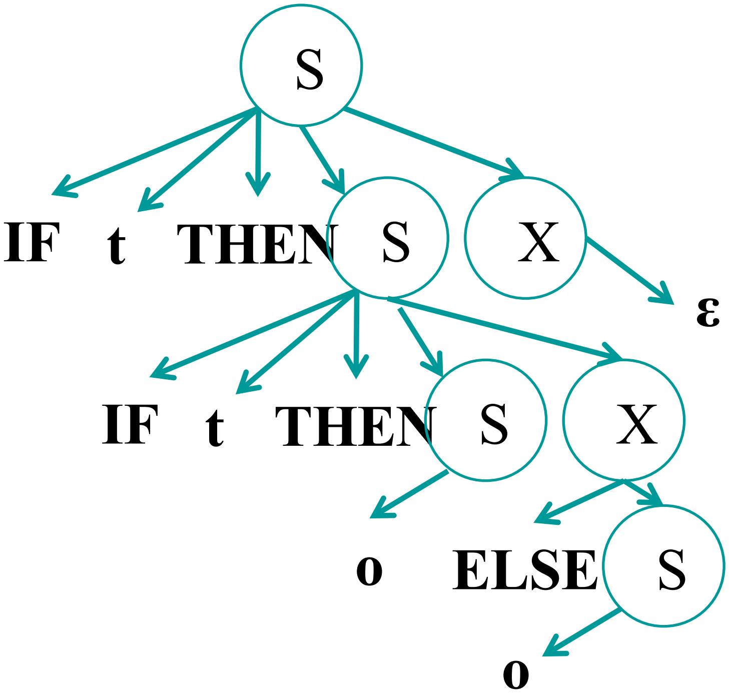
Given

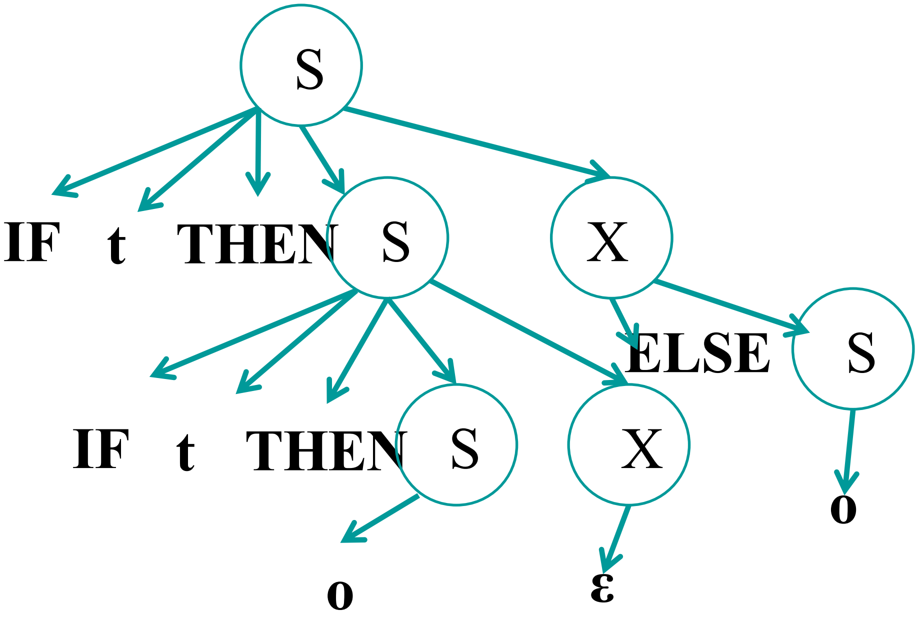
S → IF t THEN S X | o

X→ ELSE S | ε

construct two parse trees for

IF t THEN IF t THEN o ELSE o

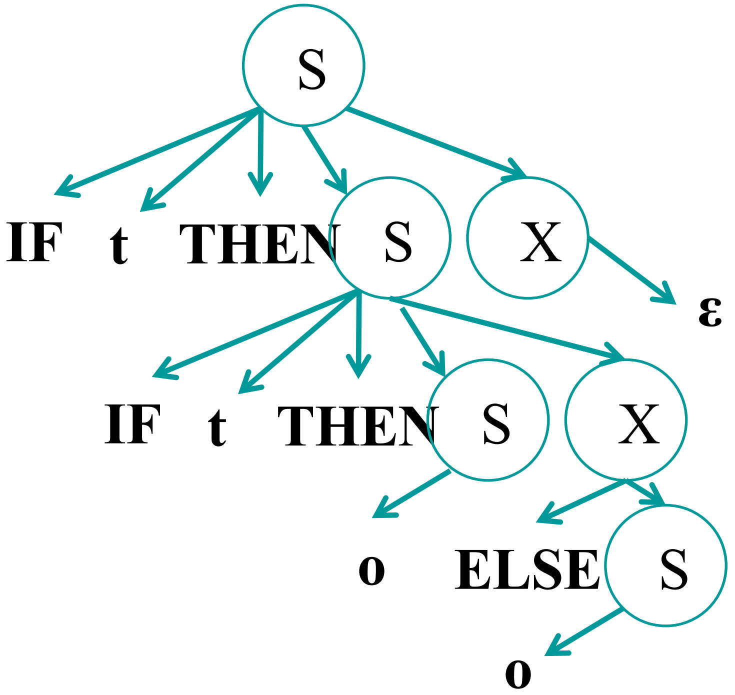




Which is the correct parse tree? (C, Java rules)

Dangling else disambiguation

The correct parse tree is:



We can get this parse tree by removing the X→ε rule in the multiple entry slot in the parse tree.

Forcing Type Checking in Haskell

Haskell performs lazy evaluation

- Java, C, and most other languages perform eager evaluation.
- Lazy evaluation means that if the result of a function call is not used when evaluating main, then that function call will not be performed.

Why this could be a problem

```
main = do
    ...
    let
    ast = parser $ lexer file_as_str
    typeresults = astTypeCheck ast
    output = astCodeGen ast
    hPutStrLn outfile output
```

Possible Solutions

- `output = seq typeresults (astCodeGen ast)`
- Print `typeresults` out to a file.

Type Checking and Code Generation of Mixed Int and Bytes

Java allows mixing numeric types. For MeggyJava this means that many operators allow mixing byte and int.

```
     (byte)3 + 4
    7 - (byte) 7
    - - - (byte)(2+1)
```

CodeGen: How should a byte value be promoted to an integer?

If Statement code generation

When the visitor encounters ifStmt, simple pre or post order code generation does not suffice. WHY?

We need more complex control:

```
        if
     /   |   \
    B  S1   S2
```

We need to control the order that code is generated for its children, using branches, jumps and labels.

First, code needs to be generated for the condition (the result of the condition evaluation has been pushed on the RTS) followed by branching instructions, the then block, control to jump over else block, then the else block, and then the end label.

Branches and jumps

An AVR detail: conditional branches can only go so far in the code, and code generated, e.g for then or else block is not bounded and thus can exceed that limit. Therefore we have to use jmp sometimes.

Notice: breq is replaced with with a brne followed by a jmp to handle this

```
cp     r24, r25
#WANT breq MJ_L6
brne   MJ_L7
jmp    MJ_L6
MJ_L7:
...  unbounded stretch of code …
MJ_L6:
```

Not: there is no not in AVR, but there is xor

truth table for not and xor

x	y	!x	x xor y
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	0

We can implement NOT x with x XOR 1 :

outNotExp

```
        pop     r24
        ldi     r22, 1
        eor     r24,r22
        push    r24
```

While statement

```
   while
  /     \
 B       S
```

What is the wiring logic?

```
SLbl:
     eval B on stack
     if false jump to endLbL
     gen Code for S
     jump to Slbl
  endLbL:
```

Short circuited (wired) AND, equals

Similar to the If Statement and While Statement, code generation will need to be implemented in the visitAndExp()

```
        &&
       /   \
    B1      B2
```

can be implemented as: if (B1) return B2 else return false

equalExp, the equality operator ==

Just like in plus and minus, we need to take the mixed type semantics of Java into account, by promoting a byte (1 register) to an int (register pair), making sure the int value correctly preserves the sign

Building the AST while parsing

- → Parse tree to AST example for (byte)(3-(byte)2).

- → Example modifications to PA2 code to create AST.