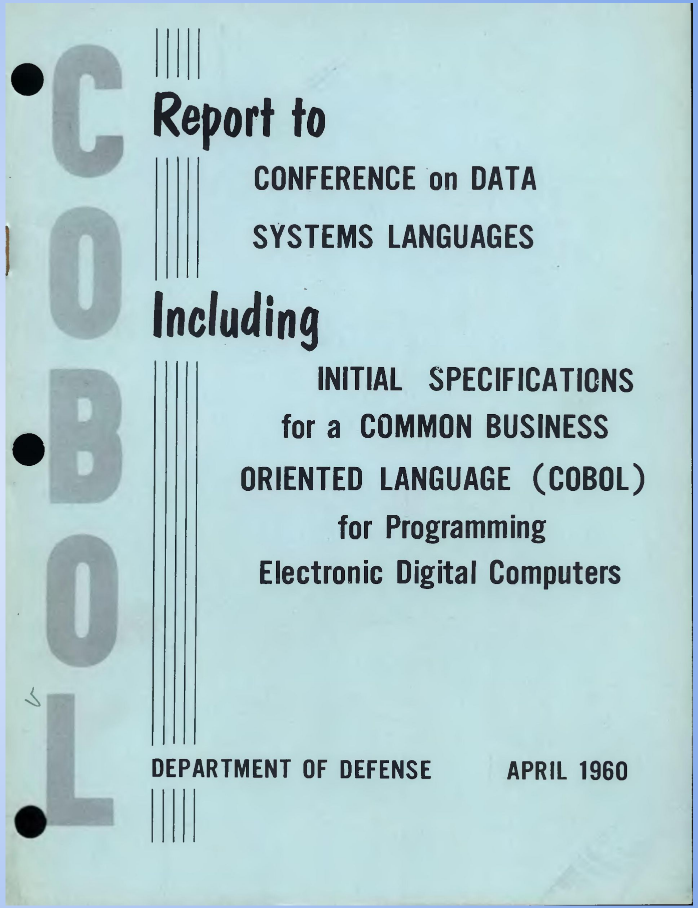
Compilers

CFGs and Parse Trees
2023 Fall


Report to
CONFERENCE on DATA
SYSTEMS LANGUAGES
Including
INITIAL SPECIFICATIONS
for a COMMON BUSINESS
ORIENTED LANGUAGE (COBOL)
for Programming
Electronic Digital Computers
DEPARTMENT OF DEFENSE
APRIL 1960

CFGs and Parse Trees

CFGs and Parse Trees

- You should be able to examine a CFG that describes "expression-like" strings and …
  - Identify all operators
  - State the relative precedence / associativity of all operators.
  - Add a new operator with a given precedence / associativity.
- You should be able to examine a CFG and a string and …
  - State whether the string is accepted by that CFG and, if so, …
  - Draw a parse tree for that string using that CFG.

Contrarily, you should also be able to examine a CFG and a parse tree and write the original string that led to that parse tree. This is absolutely trivial (why?) so we won't consider it again here.

CFG Reminder

- A CFG comprises four items,
  - A set of nonterminals, N
  - A set of terminals, T
  - A set of production rules, R
  - A start symbol, $S \in N$
- $N \cap T = \varnothing$, $V = N \cup T$
- Each production rule $r \in R$ is of the form $n \rightarrow \alpha$, $n \in N$, $\alpha \in V^*$

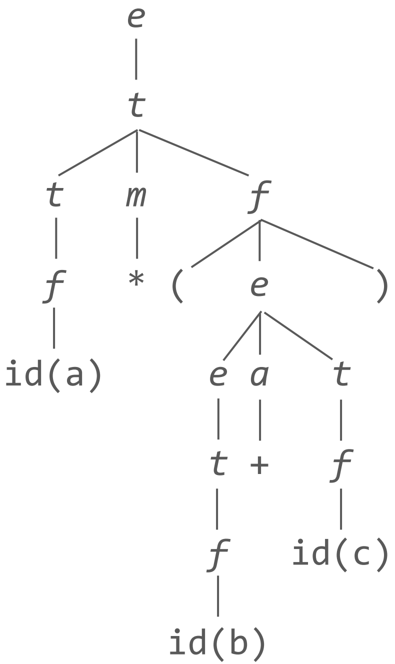
CFG Reminder

- Here's an example CFG for a certain kind of "expression-like" strings.
- A parse tree is shown for an example string "`a * ( b + c )`".
- NB The internal nodes of the tree are *nonterminals* whilst the leaves are *terminals*.

CFG

N = { a, e, f, m, t }
T = { +, -, *, /,
  (,), id, num }
R = {
  $e \to t$
  $e \to e\ a\ t$
  $t \to f$
  $t \to t\ m\ f$
  $f \to$ id
  $f \to$ num
  $f \to$ - f
  $f \to$ (e)
  $a \to$ +
  $a \to$ -
  $m \to$ *
  $m \to$ /
}
S = e

Parse tree



CFG Reminder

- Since N, T, and R are *sets*, it doesn't matter in which order their elements are listed.
- CFGs A and B are *identical* (and therefore accept the same language and get the same parse tree structures).

CFG A

$$
\begin{aligned}
&\mathrm{N} = \{\, a, e, f, m, t \,\} \\
&\mathrm{T} = \{\, +, -, *, /, \\
&\qquad (,), \mathtt{id}, \mathtt{num} \,\} \\
&\mathrm{R} = \{ \\
&\qquad e \longrightarrow t \\
&\qquad e \longrightarrow e\ a\ t \\
&\qquad t \longrightarrow f \\
&\qquad t \longrightarrow t\ m\ f \\
&\qquad f \longrightarrow \mathtt{id} \\
&\qquad f \longrightarrow \mathtt{num} \\
&\qquad f \longrightarrow -\ f \\
&\qquad f \longrightarrow (\ e\ ) \\
&\qquad a \longrightarrow + \\
&\qquad a \longrightarrow - \\
&\qquad m \longrightarrow * \\
&\qquad m \longrightarrow / \\
&\} \\
&\mathrm{S} = e
\end{aligned}
$$

CFG B

$$
\begin{aligned}
&\mathrm{N} = \{\, m, e, a, t, f \,\} \\
&\mathrm{T} = \{\, -, (, \mathtt{num}, +, *, \\
&\qquad \mathtt{id},), / \,\} \\
&\mathrm{R} = \{ \\
&\qquad f \longrightarrow (\ e\ ) \\
&\qquad f \longrightarrow \mathtt{id} \\
&\qquad m \longrightarrow * \\
&\qquad e \longrightarrow t \\
&\qquad a \longrightarrow + \\
&\qquad t \longrightarrow f \\
&\qquad f \longrightarrow \mathtt{num} \\
&\qquad m \longrightarrow / \\
&\qquad e \longrightarrow e\ a\ t \\
&\qquad a \longrightarrow - \\
&\qquad t \longrightarrow t\ m\ f \\
&\qquad f \longrightarrow -\ f \\
&\} \\
&\mathrm{S} = e
\end{aligned}
$$

CFG Reminder

- Similarly, it doesn't matter how the *nonterminals* are named.
- If they are used the same way in the rules, their exact names are immaterial.
- CFGs A and B have the same rule *structure* (and therefore accept the same language and get the same parse trees).

CFG A

N = { a, e, f, m, t }
T = { +, -, *, /,
(,), id, num }
R = {
$e \rightarrow t$
$e \rightarrow e\ a\ t$
$t \rightarrow f$
$t \rightarrow t\ m\ f$
$f \rightarrow$ id
$f \rightarrow$ num
$f \rightarrow$ - f
$f \rightarrow$ (e)
$a \rightarrow$ +
$a \rightarrow$ -
$m \rightarrow$ *
$m \rightarrow$ /
}
S = e

CFG B

N = { m, a, t, e, f }
T = { +, -, *, /,
(,), id, num }
R = {
$a \rightarrow f$
$a \rightarrow a\ m\ f$
$f \rightarrow t$
$f \rightarrow f\ e\ t$
$t \rightarrow$ id
$t \rightarrow$ num
$t \rightarrow$ - t
$t \rightarrow$ (a)
$m \rightarrow$ +
$m \rightarrow$ -
$e \rightarrow$ *
$e \rightarrow$ /
}
S = a

CFG Reminder

- Example: Given this CFG, what's the operator table?

(1 is the *highest* precedence level.)

Operator	*Precedence*	*Associativity*

CFG

N = { a, e, f, m, t }
T = { +, -, *, /,
(,), id, num }
R = {
$e \longrightarrow t$
$e \longrightarrow e\ a\ t$
$t \longrightarrow f$
$t \longrightarrow t\ m\ f$
$f \longrightarrow$ id
$f \longrightarrow$ num
$f \longrightarrow$ - f
$f \longrightarrow$ (e)
$a \longrightarrow$ +
$a \longrightarrow$ -
$m \longrightarrow$ *
$m \longrightarrow$ /
}
S = e

CFG Reminder

- (Well, I hope you at least *tried* before peeking.)

(1 is the *highest* precedence level.)

Operator	*Precedence*	*Associativity*
+	3	L → R
- (binary)	3	L → R
*	2	L → R
/	2	L → R
- (unary)	1	R → L

CFG

N = { *a*, *e*, *f*, *m*, *t* }
T = { +, -, *, /,
(,), id, num }
R = {
e⟶*t*
e⟶*e a t*
t⟶*f*
t⟶*t m f*
f⟶id
f⟶num
f⟶- *f*
f⟶(*e*)
a⟶+
a⟶-
m⟶*
m⟶/
}
S = *e*

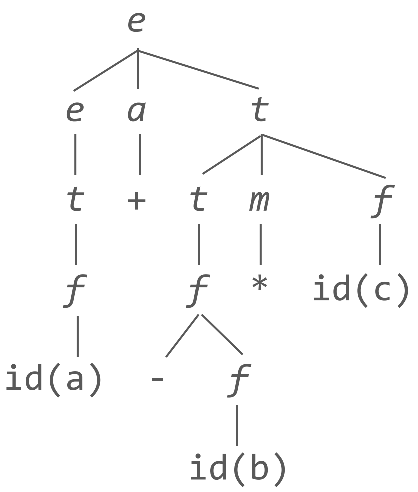
CFGs and Parse Trees

- Clear on that table?
- Here's the parse tree for the string "a + - b * c".
- NB The unary - is applied *before* the * which is applied *before* the +.
  - How would you write the expression string if you wanted the unary - applied *after* the *? Suppose you wanted the + applied *before* the *? What would the the two parse trees look like?

CFG

$N = \{ a, e, f, m, t \}$
$T = \{$ +, -, *, /,
(,), id, num $\}$
$R = \{$
$e \to t$
$e \to e\ a\ t$
$t \to f$
$t \to t\ m\ f$
$f \to$ id
$f \to$ num
$f \to$ - f
$f \to$ (e)
$a \to$ +
$a \to$ -
$m \to$ *
$m \to$ /
$\}$
$S = e$

Parse tree



CFG Reminder

- How should this grammar be changed to match this operator table?

(1 is the *highest* precedence level.)

Operator	*Precedence*	*Associativity*
+	4	L → R
- (binary)	4	L → R
*	3	L → R
/	3	L → R
- (unary)	2	R → L
^	1	R → L

CFG

N = { a, e, f, m, t }
T = { +, -, *, /,
(,), id, num }
R = {
$e \rightarrow t$
$e \rightarrow e\ a\ t$
$t \rightarrow f$
$t \rightarrow t\ m\ f$
$f \rightarrow$ id
$f \rightarrow$ num
$f \rightarrow$ - f
$f \rightarrow$ (e)
$a \rightarrow$ +
$a \rightarrow$ -
$m \rightarrow$ *
$m \rightarrow$ /
}
S = e

CFGs and Parse Trees

- Since we have a new precedence level, we need new rule(s).
  - We can't just add ^ as another operator in the + and - or * and / rules.
- This has to come *beyond* the rule(s) for (unary) - since ^ is of even *higher* precedence.
- Since the new operator is R→L, the rule has to recurse on the *right*.
- (Try to) do this yourself before peeking at the next slide.

Some details ...

`a^b^c^d` means `(a^(b^(c^d)))` since the associativity is R→L.

`a--b^c*d` means `a-((-(b^c))*d)` since the precedence order is
^ > - (unary) > * > - (binary).

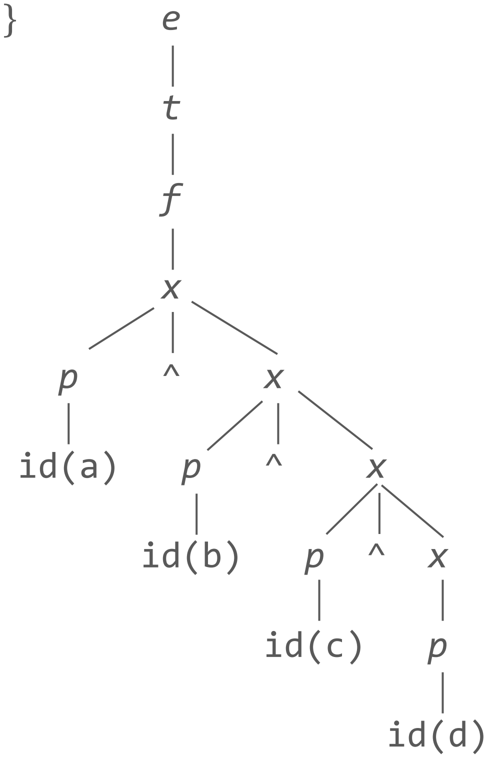
CFGs and Parse Trees

- How about this CFG?
  - Two new nonterminals *p* and *x*.
  - Two new rules for *x*.
  - Some restatement of *f* rules.
  - Some of *f* rules became *p* rules.
- Here's the parse tree for the string "a ^ b ^ c ^ d".
  - NB The associativity is R→L.
- Draw the parse tree for "a - - b ^ c * d" and "a + - b ^ c ^ d ^ e / f ^ g".

CFG

N = { *a*, *e*, *f*, *m*, *p*, *t*, *x* }
T = { +, -, *, /, ^,
  (,), id, num }
R = {
  e→*t*
  e→*e a t*
  t→*f*
  t→*t m f*
  f→- *f*
  f→*x*
  x→*p*
  x→*p* ^ *x*
  p→id
  p→num
  p→(*e*)
  a→+
  a→-
  m→*
  m→/
}
S = *e*

Parse tree



CFG Reminder

- Let's add some more operators, *unary* and *binary*, L→R and R→L.
- Take that grammar we just made and change it to match this updated operator table.

(1 is the *highest* precedence level.)

Operator	*Precedence*	*Associativity*
<-	9	R → L
or	8	L → R
and	7	L → R
== , <>	6	L → R
< , > , <= , >=	5	L → R
+ , - (binary)	4	L → R
* , /	3	L → R
+ , - (unary)	2	R → L
^	1	R → L

UNIVERSITY OF TEXAS ARLINGTON

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING