Normal forms

10/8/18

Idea: Put some restrictions on what ~~our grammar~~ productions in our grammar can look like.

Why do we want to do this?

If we are designing a parsing algorithm (or some other algorithm that does something w/ a grammar) it might be easier if ~~it~~ the alg. doesn't have to handle all possible productions

We want the normal form to be as restrictive as possible so the alg. we're designing is simpler.

On the other hand, we want to be able to (easily?) convert every context-free grammar into an equivalent one in our normal form.

Why CNF?

Reasonably simple membership algorithm:

find out if a string can be generated

"Bottom-up" algorithm

w = string

Notation: n = length of string

We will compute $V_{ij} = \{$ variables that can generate the part of the string starting at letter i and ending at letter $j\}$.

At the end, we compute $V_{1n} = \{$ vars that can gen. the whole string$\}$

If $S \in V_{1n}$, the answer is "Yes"

otherwise, the answer is "No"

new vars: $A, B,$

new prods $A \to a,$

$B \to b$

---

changed prods: $S \to ASB \mid AB$.

Step 3: Fix productions of > 2 vars by introducing more new vars.

$A \to BCDE$, change to

$A \to$ ~~BCDE~~ B [CDE]

[CDE] $\to C$ [DE]

[DE] $\to DE$

For our example:

$S \to AB$

$S \to AC$

$C \to SB$

$A \to a$

$B \to b$.

Chomsky Normal Form

Allowed productions:

$A \rightarrow x$ (var $\rightarrow$ single letter)

or

$A \rightarrow BC$ (var $\rightarrow$ 2 vars) ~~B=A~~

(B=A or C=A or both okay)

Converting grammars into CNF

E.g. $S \rightarrow aSb \mid \lambda$

Step 1: Do all the things we talked about Friday. (Get rid of λ-prods, unit prods, useless vars)

$S \rightarrow aSb \mid ab$

Step 2: For every ~~var~~ letter that occurs in ~~a~~ ~~production w/ more than one sym~~ the RHS of a production where RHS has length > 1, introduce a new variable that stands for this letter, and a production producing the letter from the new var.

Step 0: Compute all the V_{ii}'s

i.e. look for all the prods that gen. the i-th letter of w (for each i).

i.e. look for all productions

$V \to w_i$, w_i = i-th letter of w.

All such V's are in V_{ii}

Step 1: Compute all the $V_{i,i+1}$'s.

i.e. look for prods

$A \to BC$ where $B \in V_{i,i}$

$C \in V_{i+1,i+1}$

If there is such a prod,

$A \in V_{i,i+1}$.

Step 2: Compute all the $V_{i,i+2}$

Find all vars A with prods

$A \to BC$ where $B \in V_{i,i}$

$C \in V_{i+1,i+2}$

or $B \in V_{i,i+1}$

$C \in V_{i+2,i+2}$.

Step 3: Compute all the $V_{i,i+3}$

Find all vars A with prods

$A \to BC$ where $B \in V_{i,i}$
$C \in V_{i+1,i+3}$

or $B \in V_{i,i+1}$ or $B \in V_{i,i+2}$
$C \in V_{i+2,i+3}$ $C \in V_{i+3,i+3}$

etc. until we do Step n-1,
which computes $V_{1,n}$.

Extra credit: worth up to 5 HW problems

Write code (your language of choice, as long as it's not some purposely obfuscated language) implementing this algorithm.

with documentation that explains what you are doing.

E-mail me ~~sou~~ documented source code. Due ~~right before~~ Friday, Nov 16.

Greibach Normal Form

Productions are only allowed to look like

$A \to x BCD \ldots$ (one letter, followed by any number of vars)

0 is a number

Converting arbitrary grammars to GNF is complicated, but

$S \to aSb \mid ab$

$S \to aSB \mid aB$

$B \to b$

$S \to SS \mid () \mid (S)$

$S \to SS \mid (R \mid (SR$

$R \to)$

$S' \to SSS \mid (RS \mid (SRS$