Chomsky Normal Form

Chomsky Normal Form

Guarantied to resolve a Push Down Automata in (2n-1) steps

For some ambiguous grammar we can find unambiguous grammar that generates the same language (ie trompsky normal)

Some Context Free Languages can only be generated using ambiguous grammar. Such languages are called *Inherently ambiguous* and cannot be put in trompsky form

CNF:

A grammar is in CNF if every rule is of the form:
A->BC
A->a

Where a is any terminal and A B & C are any variable
also, s->e is permitted where s is the start variable

Four Steps to Converting:

1. Add a new start variable S_0
   and a new rule $S_0 \rightarrow S$ where S is the original start variable
   This guarantees that the start variable is not on RHS
2. Eliminate e rules
   Remove an e rule $A \rightarrow$e where A is not the start variable
   If $R \rightarrow UAV$ where U and V are strings of variables and terminals
   we add the rule $R \rightarrow UV$
   We do this for every occurance of A

   Eg: R -> uAvAw so we add...
   R -> uvAw

R -> uAvw
R -> uvw

3. Remove unit rules $A \to B$
   Remove unit rule then whenever rule $B \to u$ appears, add $A \to u$
   where u is a string of variables and terminals
   Unit rules is a variable going to a single variable (not a terminal)

4. Convert all remaining rules to proper form
   Replace $A \to u_1, u_2, ...u_k$ where k ≥ 3
   Until each u_i is a variable or terminal symbol with rules:
   $A \to u_1 A_1$
   $A \to u_2 A_2$
   ...
   $A \to u_k A_k$

   We replace any terminal u_i in the preceding rules with the new variable U_i and add rule:
   $U_i \to u_i$

Example: converting to CNF

```
S   -> A S A | a B
A   -> B | S
B   -> b | e
```

what's wrong?
B going to epsilon
S goes to ASA (more than 2)
S goes to aB (variable and terminal in same rule)
A goes to B | S (both unit rules)

.
.

1. New start symbol S_0

```
S0  -> S
S   -> ASA | aB
A   -> B | S
B   -> b | e
```

.
.

2. Remove e rules B -> e

```
S0  -> S
S   -> ASA | aB | a
A   -> B | S | e
B   -> b
```

(add e transitions)
Now we gotta finish, adding the possibilities for ASA (where A can be e)

```
S0  -> S
S   -> ASA | AS | SA | S | aB | a
A   -> B | S
B   -> b
```

.
.
3. Unit Rules (remove s->s)

```
S0  -> S
S   -> ASA | AS | SA | aB | a
A   -> B | S
B   -> b
```

(remove s0->S)

```
S0  -> ASA | AS | SA | aB | a
S   -> ASA | AS | SA | aB | a
A   -> B | S
B   -> b
```

(remove A->B)

```
S0  -> ASA | AS | SA | aB | a
S   -> ASA | AS | SA | aB | a
A   -> b | S
B   -> b
```

(remove A->S)

```
S0  -> ASA | AS | SA | aB | a
S   -> ASA | AS | SA | aB | a
A   -> b | ASA | AS | SA | aB | a
B   -> b
```

At this point:
* marks non compliant bits

```
S0  -> *ASA | AS | SA | *aB | a
S   -> *ASA | AS | SA | *aB | a
A   -> b | *ASA | AS | SA | *aB | a
B   -> b
```

Let's add some rules:

```
A1  -> AS
U   -> a
```

This becomes:

```
S0  -> A1 A | AS | SA | UB | a
S   -> A1 A | AS | SA | UB | a
A   -> b | A1 A | AS | SA | UB | a
B   -> b
```

Don't convert the single a's, because that would violate unit rules

CNF
$\therefore$ (2n-1) steps for string of length n