Simplification and Normalization of Context-Free Grammars

5DV037 — Fundamentals of Computer Science
Umeå University
Department of Computing Science

Stephen J. Hegner
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

Motivation

- The material in this presentation is motivated by two needs in the processing of CFGs.
  - Some of the productions of a CFG may be "useless" in terms of generating terminal strings; such parts may be safely eliminated.
  - By converting a CFG to an equivalent one which is of a certain form, or has certain properties, it may become easier to establish certain results or carry out certain tasks (such as parsing).
- This material is necessarily of a technical nature, sometimes without immediate motivation.

Useless Symbols

Example: $G = (V, \Sigma, E, P)$, $V = \{E, F, T, R\}$, $\Sigma = \{a, +, *, -, (,)\}$

$$P = \begin{cases} E & \rightarrow & E + E \mid T \mid F \\ F & \rightarrow & F * E \mid (T) \mid a \\ T & \rightarrow & E - T \mid E + R \\ R & \rightarrow & T + E \mid T - E \\ A & \rightarrow & (E) \mid a \end{cases}$$

- Neither T nor R can derive a terminal string.
- A can never be used in a derivation starting from E.
- Such symbols are called *useless* because they can never be used in a derivation, from the start symbol, of a string of terminal symbols.
- It is useful to have a means of eliminating useless symbols from a grammar in a systematic fashion.

Formal Definition of Useful and Useless Symbols

Context: A CFG $G = (V, \Sigma, S, P)$.

- Let $A \in V$.
  - A is *observable* (in G) if $A \overset{*}{\Rightarrow} \alpha$ (equivalently $A \overset{+}{\Rightarrow} \alpha$) for some $\alpha \in \Sigma^*$.
  - G is *observable* if each $A \in V$ has that property.
  - A is *reachable* (in G) if $S \overset{*}{\Rightarrow} \alpha_1 A \alpha_2$ for some $\alpha_1, \alpha_2 \in (V \cup \Sigma)^*$.
  - G is *reachable* if each $A \in V$ has that property.
  - $A \in V$ is *useful* if it is both reachable and observable.
  - Otherwise, it is *useless*.
- Define $\mathcal{O}\langle G \rangle = \{A \in V \mid A \text{ is observable in } G\}$.
- Define $\mathcal{R}\langle G \rangle = \{A \in V \mid A \text{ is reachable in } G\}$.

Construction of the Observable Set of a CFG

Context: A CFG $G = (V, \Sigma, S, P)$.

Algorithm: Construct $\mathcal{O}\langle G\rangle$:

- $\mathcal{O}_1\langle G\rangle = \{A \in V \mid A \to \alpha \text{ for some } \alpha \in \Sigma^*\}$.
- $\mathcal{O}_{k+1}\langle G\rangle = \{A \in V \mid A \to \alpha \text{ for some } \alpha \in (\mathcal{O}_k\langle G\rangle \cup \Sigma)^*\}$.
- $\mathcal{O}\langle G\rangle = \mathcal{O}_k\langle G\rangle$ for the first $k \in \mathbb{N}$ with $\mathcal{O}_k\langle G\rangle = \mathcal{O}_{k+1}\langle G\rangle$.

Example: (Start symbol is E):

$$
\begin{array}{lcl}
E & \to & E + E \mid T \mid F \\
F & \to & F * E \mid (T) \mid a \\
T & \to & E - T \mid E + R \\
R & \to & T + E \mid T - E \\
A & \to & (E) \mid a
\end{array}
$$

- $\mathcal{O}_1\langle G\rangle = \{F, A\}$, $\mathcal{O}_2\langle G\rangle = \mathcal{O}_3\langle G\rangle = \{F, A, E\}$,

Construction of an Equivalent Observable CFG

Context: A CFG $G = (V, \Sigma, S, P)$.

Algorithm: Construct an CFG $G' = (V', \Sigma, S', P')$ with $\mathcal{L}(G') = \mathcal{L}(G)$ which is observable provided that $\mathcal{L}(G) \neq \emptyset$.

- $V' = \mathcal{O}\langle G\rangle \cup \{S\}$
- $P' = \{A \underset{P}{\rightarrow} \alpha \mid \alpha \in (\mathcal{O}\langle G\rangle \cup \Sigma)^*\}$.

Observation: $\mathcal{L}(G) = \emptyset$ iff $S \notin \mathcal{O}\langle G\rangle$. $\square$

Example: (Start symbol is E):

$$\begin{array}{lcl} E & \rightarrow & E + E \mid T \mid F \\ F & \rightarrow & F * E \mid (T) \mid a \\ T & \rightarrow & E - T \mid E + R \\ R & \rightarrow & T + E \mid T - E \\ A & \rightarrow & (E) \mid a \end{array} \quad \rightsquigarrow \quad \begin{array}{lcl} E & \rightarrow & E + E \mid F \\ F & \rightarrow & F * E \mid a \\ & & \\ & & \\ A & \rightarrow & (E) \mid a \end{array}$$

- $\mathcal{O}_1\langle G\rangle = \{F, A\}$, $\mathcal{O}_2\langle G\rangle = \mathcal{O}_3\langle G\rangle = \{F, A, E\}$,

An Equivalent Observable CFG when $\mathcal{L}(G) = \emptyset$

Context: A CFG $G = (V, \Sigma, S, P)$.

Recall: $\mathcal{L}(G) = \emptyset$ iff $S \notin \mathcal{O}\langle G \rangle$. $\square$

Algorithm: Construct an observable G' with $\mathcal{L}(G') = \mathcal{L}(G)$.

- $V' = \mathcal{O}\langle G \rangle \cup \{S\}$
- If $S \in \mathcal{O}\langle G \rangle$ then $P' = \{A \underset{G}{\rightarrow} \alpha \mid \alpha \in (\mathcal{O}\langle G \rangle \cup \Sigma)^*\}$.
- If $S \notin \mathcal{O}\langle G \rangle$ then $P' = \emptyset$.

- Thus, if $\mathcal{L}(G) = \emptyset$, the start symbol S is useless (but must be retained as part of the grammar nevertheless).

Example: Remove $E \rightarrow F$ from the previous example. (Start symbol still E):

$$\begin{array}{lcl} E & \rightarrow & E + E \mid T \mid \cancel{F} \\ F & \rightarrow & F * E \mid (T) \mid a \\ T & \rightarrow & E - T \mid E + R \\ R & \rightarrow & T + E \mid T - E \\ A & \rightarrow & (E) \mid a \end{array} \quad \leadsto \quad \begin{array}{c} \mathcal{O}_1\langle G \rangle = \mathcal{O}_2\langle G \rangle = \{A, F\} \\ \\ \mathcal{L}(G) = \emptyset \\ \\ G' = (\{S\}, \Sigma, S, \emptyset) \end{array}$$

Construction of the Reachable Set of a CFG

Context: A CFG $G = (V, \Sigma, S, P)$.

Algorithm: Construct $\mathcal{R}\langle G\rangle$:

- $\mathcal{R}_0\langle G\rangle = \{S\}$.
- $\mathcal{R}_{k+1}\langle G\rangle = \mathcal{R}_k\langle G\rangle \cup \{A \in V \mid B \underset{G}{\rightarrow} \alpha_1 A \alpha_2$ for some $B \in \mathcal{R}_k\langle G\rangle$ and $\alpha_1, \alpha_2 \in (V \cup \Sigma)^*\}$.
- $\mathcal{R}\langle G\rangle = \mathcal{R}_k\langle G\rangle$ for the first $k \in \mathbb{N}$ with $\mathcal{R}_k\langle G\rangle = \mathcal{R}_{k+1}\langle G\rangle$.

Example: (Start symbol is E):

$$\begin{array}{lcl} E & \rightarrow & E + E \mid F \\ F & \rightarrow & F * E \mid a \\ A & \rightarrow & (E) \mid a \end{array}$$

- $\mathcal{R}_0\langle G\rangle = \{E\}$, $\mathcal{R}_1\langle G\rangle = \mathcal{R}_2\langle G\rangle = \{E, F\}$,

Construction of an Equivalent Reachable CFG

Context: A CFG $G = (V, \Sigma, S, P)$.

Algorithm: Construct a reachable CFG $G' = (V', \Sigma, S', P')$ with $\mathcal{L}(G') = \mathcal{L}(G)$.

- $V' = \mathcal{R}\langle G \rangle$
- $P' = \{A \underset{G}{\rightarrow} \alpha \mid A \in V'\}$.

Example: (Start symbol is E):

$$\begin{array}{lcl} E & \rightarrow & E + E \mid F \\ F & \rightarrow & F * E \mid a \\ A & \rightarrow & (E) \mid a \end{array} \quad \rightsquigarrow \quad \begin{array}{lcl} E & \rightarrow & E + E \mid F \\ F & \rightarrow & F * E \mid a \end{array}$$

- $\mathcal{R}_0\langle G \rangle = \{E\}$, $\mathcal{R}_1\langle G \rangle = \mathcal{R}_2\langle G \rangle = \{E, F\}$,

Reduced Grammars

Context: A CFG $G = (V, \Sigma, S, P)$.

- Need to exercise a little care in defining a grammar with no useless symbols.
- If $\mathcal{L}(G) = \emptyset$, then the start symbol must be useless, yet every grammar must have a start symbol.
- Call G *reduced* if it has one of the following two properties:
  - $P = \emptyset$ and $V = \{S\}$; or
  - G is both observable and reachable.

Algorithm: Construct a grammar $G' = (V', \Sigma, S', P')$ which is reduced and which satisfies $\mathcal{L}(G') = \mathcal{L}(G)$.

- Apply the previous two algorithms, which already take these cases into account.
- Must remove unobservable variables first, then unreachable.

Order Matters in Reduction

Example: (Start symbol is E):
$$\begin{array}{rcl} E & \rightarrow & E + E \mid T \mid F \\ F & \rightarrow & F * E \mid (T) \mid a \\ T & \rightarrow & E - T \mid E + R \\ R & \rightarrow & T + E \mid T - E \mid RA \\ A & \rightarrow & (E) \mid a \end{array}$$

- All variables are reachable: $\mathcal{R}\langle G \rangle = \{E, F, T, R, A\}$.
- Only $\{E, F, A\}$ are observable.
- If unreachable variables are removed first, and then the unobservable ones, the resulting grammar will not be reachable:
$$\begin{array}{rcl} E & \rightarrow & E + E \mid F \\ F & \rightarrow & F * E \mid a \\ A & \rightarrow & (E) \mid a \end{array}$$
- Thus, the unobservable symbols must be removed first.

Null Rules

Context: A CFG $G = (V, \Sigma, S, P)$.

- A *null rule* is a production of the form
$$A \to \lambda$$
- Why null rules are anomalous:
  - They are the only productions $A \to \alpha$ in which Length(A) $>$ Length(α).
  - Thus, if G has no null rules, Length(A) $\leq$ Length(α) for every production $A \to \alpha$.
- It would be nice to be able to eliminate null rules entirely.
- However, this is clearly not possible if $\lambda \in \mathcal{L}(G)$.
- There is, however, a solution which is almost as good:
  - If $\lambda \in \mathcal{L}(G)$, then $S \to \lambda$
  - No other null rules are allowed.
- The means to transform G to achieve this will now be addressed.

Nonerasing Grammars

Context: A CFG $G = (V, \Sigma, S, P)$.

- A variable $A \in V$ is *recursive* if $A \overset{+}{\Rightarrow} \alpha_1 A \alpha_2$ for some $\alpha_1, \alpha_2 \in (V \cup \Sigma)^*$.
- Here $\overset{+}{\Rightarrow}$ means "derives in one or more steps".
- The trivial derivation $A \overset{*}{\Rightarrow} A$ in zero steps, (always present), is excluded.
- The variable $A \in V$ is *nullable* if $A \overset{*}{\Rightarrow} \lambda$.
- Define $\mathcal{N}\langle G \rangle$ to be the set of all nullable variables of G.
- Call G *nonerasing* if
  - S is not recursive, and
  - $\mathcal{N}\langle G \rangle \subseteq \{S\}$.
- This means:
  - $S \to \lambda$ is the only possible null rule; and
  - it is the only way to derive λ.

Construction of $\mathcal{N}\langle G\rangle$

Context: A CFG $G = (V, \Sigma, S, P)$.

Algorithm: Construct $\mathcal{N}\langle G\rangle$ inductively:

- $\mathcal{N}_0\langle G\rangle = \emptyset$
- $\mathcal{N}_{k+1}\langle G\rangle = \mathcal{N}_k\langle G\rangle \cup \{A \in V \mid A \to \alpha \text{ for some } \alpha \in \mathcal{N}_k\langle G\rangle^*\}$.
- Stop when $\mathcal{N}_k\langle G\rangle = \mathcal{N}_{k+1}\langle G\rangle$ with $\mathcal{N}\langle G\rangle = \mathcal{N}_k\langle G\rangle$.

- Example: $V = \{S, O, Q, E\}$, $\Sigma = \{a, b, c\}$;

$$P = \begin{cases} S & \to & aOb \\ O & \to & QEQ \mid aOb \mid OOO \mid OEcEO \\ Q & \to & c \mid EE \\ E & \to & a \mid \lambda \end{cases}$$

- $\mathcal{N}_0\langle G\rangle = \emptyset$; $\mathcal{N}_1\langle G\rangle = \{E\}$; $\mathcal{N}_2\langle G\rangle = \{E, Q\}$; $\mathcal{N}_3\langle G\rangle = \{E, Q, O\} = \mathcal{N}_4\langle G\rangle = \mathcal{N}\langle G\rangle$.

Construction of an Equivalent Nonerasing CFG

Context: A CFG $G = (V, \Sigma, S, P)$.

Algorithm: Construct an equivalent nonerasing CFG $G' = (V', \Sigma, S', P')$.

- $V' = V \cup S'$.
- The productions in P' are of the following three forms:
  - $S' \to S$
  - $S' \to \lambda$ if $S \in \mathcal{N}\langle G \rangle$
  - $A \to \alpha_1 \ldots \alpha_k$ iff
    - $\alpha_1 \ldots \alpha_k \neq \lambda$, and
    - There are (not necessarily distinct) $A_1, \ldots A_n \in \mathcal{N}\langle G \rangle$ with $A \to \alpha_1 A_1 \alpha_2 A_2 \ldots A_n \alpha_n \in P$.
  - The last form must be done for *all combinations* of variables which produce λ.

Remark: This algorithm has exponential complexity. It is possible to do much better (linear).

Example of Nonerasing Construction

- Example: $V = \{S, O, Q, E\}$, $\Sigma = \{a, b, c\}$;

$$P = \begin{cases} S & \rightarrow & aOb \\ O & \rightarrow & QEQ \mid aOb \mid OOO \mid OEcEO \\ Q & \rightarrow & c \mid EE \\ E & \rightarrow & a \mid \lambda \end{cases}$$

- $\mathcal{N}\langle G \rangle = \{E, Q, O\}$.
- New productions:
  - $S' \rightarrow S$
  - $S \rightarrow aOb \mid ab$
  - $O \rightarrow QEQ \mid QE \mid QQ \mid EQ \mid Q \mid E \mid aOb \mid ab$
    $\mid OOO \mid OO \mid O \mid OEcEO \mid OEcE \mid OEcO \mid OcEO \mid EcEO$
    $\mid OEc \mid OcE \mid OcO \mid cEO \mid EcE \mid EcO \mid Oc \mid Ec \mid cE \mid cO \mid c$
  - $Q \rightarrow c \mid E \mid EE$
  - $E \rightarrow a$

Chain Rules

Context: A CFG $G = (V, \Sigma, S, P)$.

- A *unit production* or *chain rule* is a production of the form

$$A \to B$$

for some $A, B \in V$.

- Unit productions rules are not necessarily bad.
- Examples from programming language specification:
  - $\langle stmt \rangle \to \langle if\_stmt \rangle$
  - $\langle number \rangle \to \langle digit \rangle$
- It is *recursive* chain rules which are can lead to problems.
- In any case, from a theoretical point of view, it is often useful to eliminate such rules from a grammar.

The Chain Set of a Grammar

- For $A \in V$, define
  - $\mathcal{C}_1\langle G, A\rangle = \{B \in V \mid A \to B\}$.
  - $\mathcal{C}_{k+1}\langle G, A\rangle = \mathcal{C}_k\langle G, A\rangle \cup \{B \in V \mid C \to B \text{ for some } C \in \mathcal{C}_k\langle G, A\rangle\}$.

Observation: The addition of new elements to $\mathcal{C}\langle G, A\rangle$ stops as soon as $\mathcal{C}_k\langle G, A\rangle = \mathcal{C}_{k+1}\langle G, A\rangle$, so this set may be computed in a finite number of steps. □

- For $A \in V$, define
  - $\mathcal{C}\langle G, A\rangle = \mathcal{C}_k\langle G, A\rangle$, where k is the first index for which $\mathcal{C}_k\langle G, A\rangle = \mathcal{C}_{k+1}\langle G, A\rangle$.
- The variable $A \in V$ is called *chain recursive* if $A \in \mathcal{C}\langle G, A\rangle$.
- Thus, A is chain recursive if it can be derived from itself using unit productions.
  - A "chain loop"

Example of a Chain Set

Nonterminals: $\{\langle \mathit{Expr} \rangle, \langle \mathit{Ident} \rangle\}$
Terminals: $\{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}, \mathtt{(}, \mathtt{)}, \mathtt{+}, \mathtt{*}\}$
Start symbol: $\langle \mathrm{Expr} \rangle$
Productions:

$$
\begin{aligned}
\langle \mathit{Ident} \rangle &\rightarrow \mathtt{A} \mid \mathtt{B} \mid \ldots \mid \mathtt{Y} \mid \mathtt{Z} \\
\langle \mathit{Expr} \rangle &\rightarrow \langle \mathit{Expr} \rangle + \langle \mathit{Term} \rangle \mid \langle \mathit{Term} \rangle \\
\langle \mathit{Term} \rangle &\rightarrow \langle \mathit{Term} \rangle * \langle \mathit{Factor} \rangle \mid \langle \mathit{Factor} \rangle \\
\langle \mathit{Factor} \rangle &\rightarrow (\langle \mathit{Expr} \rangle) \mid \langle \mathit{Ident} \rangle
\end{aligned}
$$

- $\mathcal{C}_1\langle G, \langle \mathit{Ident} \rangle\rangle = \mathcal{C}_2\langle G, \langle \mathit{Ident} \rangle\rangle = \emptyset$,
- $\mathcal{C}_1\langle G, \langle \mathit{Expr} \rangle\rangle = \{\langle \mathit{Term} \rangle\}$, $\mathcal{C}_2\langle G, \langle \mathit{Expr} \rangle\rangle = \{\langle \mathit{Term} \rangle, \langle \mathit{Factor} \rangle\}$,
  $\mathcal{C}_3\langle G, \langle \mathit{Expr} \rangle\rangle = \mathcal{C}_4\langle G, \langle \mathit{Expr} \rangle\rangle = \{\langle \mathit{Term} \rangle, \langle \mathit{Factor} \rangle, \langle \mathit{Ident} \rangle\}$,
- $\mathcal{C}_1\langle G, \langle \mathit{Term} \rangle\rangle = \{\langle \mathit{Factor} \rangle\}$,
  $\mathcal{C}_2\langle G, \langle \mathit{Term} \rangle\rangle = \mathcal{C}_3\langle G, \langle \mathit{Term} \rangle\rangle = \{\langle \mathit{Factor} \rangle, \langle \mathit{Ident} \rangle\}$,
- $\mathcal{C}_1\langle G, \langle \mathit{Factor} \rangle\rangle = \mathcal{C}_2\langle G, \langle \mathit{Factor} \rangle\rangle = \{\langle \mathit{Ident} \rangle\}$,

Eliminating Chain Rules

Context: A CFG $G = (V, \Sigma, S, P)$.

Algorithm: Construct an equivalent CFG $G' = (V', \Sigma, S', P')$ without unit productions.
$P' = \{A \to \alpha \mid \alpha \notin V$ and there is a $B \underset{G}{\to} \alpha$ with $B \in \{A\} \cup \mathcal{C}\langle G, A\rangle\}$.

Example:

$$\begin{array}{rcl}
\langle \mathit{Ident} \rangle & \to & \mathtt{A} \mid \mathtt{B} \mid \ldots \mid \mathtt{Y} \mid \mathtt{Z} \\
\langle \mathit{Expr} \rangle & \to & \langle \mathit{Expr} \rangle + \langle \mathit{Term} \rangle \mid \langle \mathit{Term} \rangle \\
\langle \mathit{Term} \rangle & \to & \langle \mathit{Term} \rangle * \langle \mathit{Factor} \rangle \mid \langle \mathit{Factor} \rangle \\
\langle \mathit{Factor} \rangle & \to & (\langle \mathit{Expr} \rangle) \mid \langle \mathit{Ident} \rangle
\end{array}$$

Repaired:

$$\begin{array}{rcl}
\langle \mathit{Ident} \rangle & \to & \mathtt{A} \mid \mathtt{B} \mid \ldots \mid \mathtt{Y} \mid \mathtt{Z} \\
\langle \mathit{Expr} \rangle & \to & \langle \mathit{Expr} \rangle + \langle \mathit{Term} \rangle \mid \langle \mathit{Term} \rangle * \langle \mathit{Factor} \rangle \mid (\langle \mathit{Expr} \rangle) \mid A \mid \ldots \mid Z \\
\langle \mathit{Term} \rangle & \to & \langle \mathit{Term} \rangle * \langle \mathit{Factor} \rangle \mid (\langle \mathit{Expr} \rangle) \mid A \mid \ldots \mid Z \\
\langle \mathit{Factor} \rangle & \to & (\langle \mathit{Expr} \rangle) \mid A \mid \ldots \mid Z
\end{array}$$

Nonerasing and No Chain Rules

Context: A CFG $G = (V, \Sigma, S, P)$.

- The algorithm which makes a grammar nonerasing can easily introduce new chain rules.
- On the other hand, the algorithm which removes chain rules does not introduce any new null rules.
- Therefore, to construct a grammar which is both nonerasing and without chain rules, remove the null rules first, and then remove the chain rules.

Left Recursion and Greibach Normal Form

Context: A CFG $G = (V, \Sigma, S, P)$.

- G is *left recursive* if there is a derivation of the form $A \stackrel{+}{\Rightarrow} A\alpha$ for some $A \in V$ and $\alpha \in (V \cup \Sigma)^*$.
- Left recursion makes the design of parsers more difficult, because of the possibility of an infinite loop for so-called "recursive descent" parsers which always try to replace the leftmost symbol first.
- G is in *Greibach normal form* if every production is of one of the following two forms:
  - $A \to a\alpha$ for some $A \in V$, $a \in \Sigma$, and $\alpha \in (V \setminus \{S\})^*$; or
  - $S \to \lambda$.

Theorem: There is an algorithm to convert any CFG G into an equivalent one which is in Greibach normal form.

Proof: Consult an advanced textbook. (The proof is tedious but not particularly deep.) $\square$

Chomsky Normal Form

Context: A CFG $G = (V, \Sigma, S, P)$.

- Chomsky normal form guarantees that the productions are very short.
- G is in *Chomsky normal form* if every productions is of one of the following three forms:
  - $A \to BC$ for some $A \in V$, and $B, C \in V \setminus \{S\}$.
  - $A \to a$ for some $A \in V$ and $a \in \Sigma$.
  - $S \to \lambda$.

Theorem: There is an algorithm which converts any CFG G into an equivalent one in Chomsky normal form.

Proof: There is a sketch in the textbook. Consult a more advanced book for a complete proof. $\square$

Note: The proof uses ideas similar to that used in converting a right-linear grammar to a simple right-linear grammar.