Grammar Flow Analysis

– Wilhelm/Maurer: Compiler Design, Chapter 8 –

Reinhard Wilhelm
Universität des Saarlandes
wilhelm@cs.uni-sb.de

2. November 2009
## Notation

<table>
<thead>
<tr>
<th>Generic names</th>
<th>for</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A, B, C, X, Y, Z$</td>
<td>Non-terminal symbols</td>
</tr>
<tr>
<td>$a, b, c, \ldots$</td>
<td>Terminal symbols</td>
</tr>
<tr>
<td>$u, v, w, x, y, z$</td>
<td>Terminal strings</td>
</tr>
<tr>
<td>$\alpha, \beta, \gamma, \varphi, \psi$</td>
<td>Strings over $V_N \cup V_T$</td>
</tr>
<tr>
<td>$p, p', p_1, p_2, \ldots$</td>
<td>Productions</td>
</tr>
</tbody>
</table>

- Standard notation for production
  \[ p = (X_0 \rightarrow u_0X_1u_1 \ldots X_{n_p}u_{n_p}) \]
  \[ n_p \text{ – Arity of } p \]
- $(p, i)$ – Position $i$ in production $p$ $(0 \leq i \leq n_p)$
- $p[i]$ stands for $X_i$, $(0 \leq i \leq n_p)$
- $X$ occurs at position $i$ – $p[i] = X$
Reachability and Productivity

Non-terminal $A$ is

**reachable:** iff there exist $\varphi_1, \varphi_2 \in V_T \cup V_N$ such that
$$S \Rightarrow^* \varphi_1 A \varphi_2$$

**productive:** iff there exists $w \in V_T^*$, $A \Rightarrow^* w$

These definitions are useless for tests; they involve quantifications over infinite sets.
A two level Definition

1. A non-terminal is **reachable through its occurrence** \((p, i)\) iff \(p[0]\) is reachable,
2. A non-terminal is **reachable** iff it is reachable through at least one of its occurrences,
3. \(S'\) is reachable.

1. A non-terminal \(A\) is **productive through production** \(p\) iff \(A = p[0]\) and all non-terminals \(p[i] (1 \leq i \leq n_p)\) are productive.
2. A non-terminal is **productive** iff it is productive through at least one of its alternatives.

- Reachability and productivity for a grammar given by a (recursive) system of equations.
- Least solution wanted to eliminate as many useless non-terminals as possible.
Typical Two Level Simultaneous Recursion

Productivity:
1. dependence of property of left side non-terminal on right side non-terminals,
2. combination of the information from the different alternatives for a non-terminal.

Reachability:
1. dependence of property of occurrences of non-terminals on the right side on the property of the left side non-terminal,
2. combination of the information from the different occurrences for a non-terminal,
3. the initial property.

In the specification
1. given by transfer functions
2. given by combination functions
Grammar Flow Analysis (GFA) computes a property function $I : V_N \rightarrow D$ where $D$ is some domain of information for non-terminals, mostly properties of sets of trees,

- Productivity computed by a bottom-up Grammar Flow Analysis (bottom-up GFA)
- Reachability computed by a top-down Grammar Flow Analysis (top-down GFA)
Trees, Subtrees, Tree Fragments

$S$

$X$

Parse tree

$X$

Subtree for $X$

$S$

$X$

upper tree fragment for $X$

$X$ reachable: Set of upper tree fragments for $X$ not empty,

$X$ productive: Set of subtrees for $X$ not empty.
Grammar Flow Analysis

Bottom-up GFA

Given a cfg $G$.

A **bottom-up GFA-problem** for $G$ and a property function $I$:

- **D**: a domain $D↑$,
- **T**: transfer functions $F_p↑: D↑^{nP} → D↑$ for each $p ∈ P$,
- **C**: a combination function $∇↑: 2D↑ → D↑$.

This defines a system of equations for $G$ and $I$:

$$I(X) = ∇↑\{F_p↑(I(p[1]), \ldots, I(p[n_p])) | p[0] = X\} \forall X ∈ V_N \quad (I↑)$$
Top-down GFA

Given a cfg $G$.

A top down – GFA-problem for $G$ and a property function $I$:

$D$: a domain $D\downarrow$;

$T$: $n_p$ transfer functions $F_{p,i}\downarrow: D\downarrow \rightarrow D\downarrow$, $1 \leq i \leq n_p$, for each production $p \in P$,

$C$: a combination function $\nabla\downarrow: 2^{D\downarrow} \rightarrow D\downarrow$,

$S$: a value $I_0$ for $S$ under the function $I$.

A top-down GFA-problem defines a system of equations for $G$ and $I$

\[
\begin{align*}
I(S) &= I_0 \\
I(p, i) &= F_{p,i}\downarrow (I(p[0])) \text{ for all } p \in P, \ 1 \leq i \leq n_p \\
I(X) &= \nabla\downarrow \{I(p, i) \mid p[i] = X\}, \text{ for all } X \in V_N - \{S\}
\end{align*}
\]

($I\downarrow$)
Recursive System of Equations

Systems like $(I\uparrow)$ and $(I\downarrow)$ are in general recursive. Questions: Do they have

- solutions?
- unique solutions?
They do have solutions if

- the domain
  - is partially ordered by some relation \( \sqsubseteq \),
  - has a uniquely defined smallest element, \( \bot \),
  - has a least upper bound, \( d_1 \sqcup d_2 \), for each two elements \( d_1, d_2 \)
  - and has only finitely ascending chains,

and

- the transfer and the combination functions are monotonic.

Our domains are finite, all functions are monotonic.
Fixpoint Iteration

- Solutions are fixpoints of a function $I : [V_N \rightarrow D] \rightarrow [V_N \rightarrow D]$.
- Computed iteratively starting with $\perp$, the function which maps all non-terminals to $\perp$.
- Apply transfer functions and combination functions until nothing changes.

We always compute least fixpoints.
Productivity Revisited

\[ D \uparrow \{ \text{false} \sqsubseteq \text{true} \} \quad \text{true for productive} \]
\[ F_p \uparrow \prod \quad (\text{true for } n_p = 0) \]
\[ \nabla \uparrow \bigvee \quad (\text{false for non-terminals without productions}) \]

Domain: \( D \uparrow \) satisfies the conditions,

transfer functions: conjunctions are monotonic,

combination function: disjunction is monotonic.

Resulting system of equations:
\[
Pr(X) = \bigvee \{ \prod_{i=1}^{n_p} Pr(p[i]) \mid p[0] = X \} \quad \text{for all } X \in V_N
\]
Example: Productivity

Given the following grammar:

\[ G = (\{S', S, X, Y, Z\}, \{a, b\}, \left\{ \begin{array}{l}
S' \rightarrow S \\
S \rightarrow aX \\
X \rightarrow bS \mid aYbY \\
Y \rightarrow ba \mid aZ \\
Z \rightarrow aZX 
\end{array} \right\}, S') \]

Resulting system of equations:

\[
\begin{align*}
Pr(S) &= Pr(X) \\
Pr(X) &= Pr(S) \lor Pr(Y) \\
Pr(Y) &= true \lor Pr(Z) = true \\
Pr(Z) &= Pr(Z) \land Pr(X)
\end{align*}
\]

Fixpoint iteration

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Reachability Revisited

\[ D \downarrow \quad false \sqsubseteq \{ true \} \quad true \text{ for reachable} \]
\[ F_{p,i} \downarrow \quad id \quad \text{identity mapping} \]
\[ \nabla \downarrow \quad \lor \quad \text{Boolean Or (false, if there is no occ. of the non-terminal)} \]

\[ l_0 \quad true \]

Domain: \( D \downarrow \) satisfies the conditions,

Transfer functions: identity is monotonic,

Combination function: disjunction is monotonic.

Resulting system of equations for reachability:

\[
\begin{align*}
Re(S) &= true \\
Re(X) &= \lor \{ Re(p[0]) \mid p[i] = X, \ 1 \leq i \leq n_p \} \ \forall X \neq S
\end{align*}
\]
### Example: Reachability

Given the grammar $G = (\{S, U, V, X, Y, Z\}, \{a, b, c, d\}$,

The equations:

\[
\begin{align*}
S & \rightarrow Y \\
Y & \rightarrow YZ \mid Ya \mid b \\
U & \rightarrow V \\
X & \rightarrow c \\
V & \rightarrow Vd \mid d \\
Z & \rightarrow ZX
\end{align*}
\]

Fixpoint iteration:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>U</th>
<th>V</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
First and Follow Sets

Parser generators need precomputed information about sets of

- prefixes of words for non-terminals (words that can begin words for non-terminals)
- followers of non-terminals (words which can follow a non-terminal).

Strategic use: Removing non-determinism from expand moves of the $P_G$
These sets can be computed by GFA.
Another Grammar for Arithmetic Expressions

Left-factored grammar $G_2$, i.e. left recursion removed.

$$
S \rightarrow E \\
E \rightarrow TE' \quad E \text{ generates } T \text{ with a continuation } E' \\
E' \rightarrow +E|\epsilon \quad E' \text{ generates possibly empty sequence of } +Ts \\
T \rightarrow FT' \quad T \text{ generates } F \text{ with a continuation } T' \\
T' \rightarrow *T|\epsilon \quad T' \text{ generates possibly empty sequence of } *Fs \\
F \rightarrow \text{id}|(E)
$$

$G_2$ defines the same language as $G_0$ and $G_1$. 
The $FIRST_1$ Sets

- A production $N \rightarrow \alpha$ is applicable for symbols that “begin” $\alpha$
- Example: Arithmetic Expressions, Grammar $G_2$
  - The production $F \rightarrow \text{id}$ is applied when the current symbol is $\text{id}$
  - The production $F \rightarrow (E)$ is applied when the current symbol is $($
  - The production $T \rightarrow F$ is applied when the current symbol is $\text{id}$ or $($
- Formal definition:

$$FIRST_1(\alpha) = \{ a \in V_T | \exists \gamma : \alpha \Rightarrow^* a\gamma \}$$
The $FOLLOW_1$ Sets

- A production $N \rightarrow \epsilon$ is applicable for symbols that “can follow” $N$ in some derivation
- Example: Arithmetic Expressions, Grammar $G_2$
  - The production $E' \rightarrow \epsilon$ is applied for symbols $\#$ and $)$
  - The production $T' \rightarrow \epsilon$ is applied for symbols $\#$, $)$ and $+$
- Formal definition:

$$FOLLOW_1(N) = \{ a \in V_T | \exists \alpha, \gamma : S \Rightarrow^{*} \alpha Na \gamma \}$$
Definitions

Let $k \geq 1$

$k$-prefix of a word $w = a_1 \ldots a_n$

$$k : w = \begin{cases} a_1 \ldots a_n & \text{if } n \leq k \\ a_1 \ldots a_k & \text{otherwise} \end{cases}$$

$k$-concatenation

$$\oplus_k : V^* \times V^* \rightarrow V^{\leq k},$$
defined by $u \oplus_k v = k : uv$

extended to languages

$$k : L = \{ k : w \mid w \in L \}$$

$$L_1 \oplus_k L_2 = \{ x \oplus_k y \mid x \in L_1, y \in L_2 \}.$$ 

$$V^{\leq k} = \bigcup_{i=1}^k V^i$$  

set of words of length at most $k$

$$V^{\leq k}_T\# = V^{\leq k}_T \cup V^{k-1}_T \{\#\}$$  

... possibly terminated by #.
\textbf{FIRST}_k \text{ and } \textbf{FOLLOW}_k

\[
\text{FIRST}_k : (V_N \cup V_T)^* \rightarrow 2^{V_T \leq k} \quad \text{where} \\
\text{FIRST}_k(\alpha) = \{ k : u \mid \alpha \rightarrow^* u \}
\]

set of \( k \)–prefixes of terminal words for \( \alpha \).

\[
\text{FOLLOW}_k : V_N \rightarrow 2^{V_T \#} \quad \text{where} \\
\text{FOLLOW}_k(X) = \{ w \mid S \rightarrow^* \beta X \gamma \text{ and } w \in \text{FIRST}_k(\gamma) \}
\]

set of \( k \)–prefixes of terminal words that may immediately follow \( X \).
GFA-Problem $\textit{FIRST}_k$

The recursive system of equations for $\textit{FIRST}_k$ is

$$
\textit{Fi}_k(X) = \bigcup_{\{p | p[0] = X\}} \textit{Fir}_p(\textit{Fi}_k(p[1]), \ldots, \textit{Fi}_k(p[n_p])) \quad \forall X \in V_N
$$

($\textit{Fi}_k$)
**FIRST** Example

The bottom up-GFA problem FIRST\(_1\) for grammar \(G_2\) with the productions:

\[
\begin{align*}
0: & \quad S \rightarrow E \quad 3: & \quad E' \rightarrow +E \quad 6: & \quad T' \rightarrow *T \\
1: & \quad E \rightarrow TE' \quad 4: & \quad T \rightarrow FT' \quad 7: & \quad F \rightarrow (E) \\
2: & \quad E' \rightarrow \varepsilon \quad 5: & \quad T' \rightarrow \varepsilon \quad 8: & \quad F \rightarrow \text{id}
\end{align*}
\]

\(G_2\) defines the same language as \(G_0\) and \(G_1\).

The transfer functions for productions 0 – 8 are:

\[
\begin{align*}
Fir_0(d) &= d \\
Fir_1(d_1, d_2) &= Fir_4(d_1, d_2) = d_1 \oplus_1 d_2 \\
Fir_2 &= Fir_5 = \{\varepsilon\} \\
Fir_3(d) &= \{+\} \\
Fir_6(d) &= \{\ast\} \\
Fir_7(d) &= \{()\} \\
Fir_8 &= \{\text{id}\}
\end{align*}
\]
Iteration

Iterative computation of the $FIRST_1$ sets:

<table>
<thead>
<tr>
<th></th>
<th>$S$</th>
<th>$E$</th>
<th>$E'$</th>
<th>$T$</th>
<th>$T'$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
GFA-Problem $FOLLOW_k$

The resulting system of equations for $FOLLOW_k$ is

$$FkO(X) = \bigcup \{p | p[i] = X, 1 \leq i \leq n_p \} \cup \forall X \in V_N - \{S\}$$

$$FkO(S) = \{\#\}$$

$(FkO)$
**FOLLOW\_k Example**

Regard grammar \( G_2 \). The transfer functions are:

\[
\begin{align*}
F_{0,1}(d) &= d \\
F_{1,1}(d) &= F_{i_1}(E') \oplus_1 d = \{+, \varepsilon\} \oplus_1 d, \\
F_{1,2}(d) &= d \\
F_{3,1}(d) &= d \\
F_{4,1}(d) &= F_{i_1}(T') \oplus_1 d = \{\ast, \varepsilon\} \oplus_1 d, \\
F_{4,2}(d) &= d \\
F_{6,1}(d) &= d \\
F_{7,1}(d) &= \{\}\}
\end{align*}
\]

Iterative computation of the \( \text{FOLLOW}_1 \) sets:

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>E'</th>
<th>T</th>
<th>T'</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>{#}</td>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>