Grammars, Languages & Fragments

Maximilian Meffert

DISCLAIMER
Proper Citation Missing!

Abstract. elation between These notes summarize ideas on the notion of fragments of elements, i.e. words, of formal languages and their generating grammars. The main thought is, that fragments are not necessarily words of a formal language in the strict sense, that they can be derived in a finite number of steps starting with the start variable of its grammar. In fact, there must be a superset of the language containing all possible fragments.

Motivation

Consider the following code describing some sort of interface:

```
interface Foo {
        method Bar;
}
```

we want to formally define all *fragments* (well-formed partial code pieces) of that code, i.e.:

```
interface Foo { method Bar; }  // the code itself
method Bar;                    // the first piece
Foo                            // a terminal
Bar                            // another terminal
;                              // yet another terminal
```

A *Formal Grammar* is a 4-tuple: $G := (V, \Sigma, P, S)$ with:

- V a finite set of variables
- Σ a finite set of terminal symbols (the alphabet)
- $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ a finite set of production rules $x \to x'$
- $S \in V$ a start variable
- $V \cap \Sigma = \emptyset$ holds

and the *Formal Language* generated by a grammar is the set:

$$\mathcal{L}(G) := \{w \in \Sigma^* | S \Rightarrow_G^* w\}$$

where $\Rightarrow_G^*$ is the reflexive, transitive closure or the derivation relation $\Rightarrow_G$. This means, a word w is an element of $\mathcal{L}(G)$ if and only if there exists a finite derivation:

$$S \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G ... \Rightarrow_G w$$

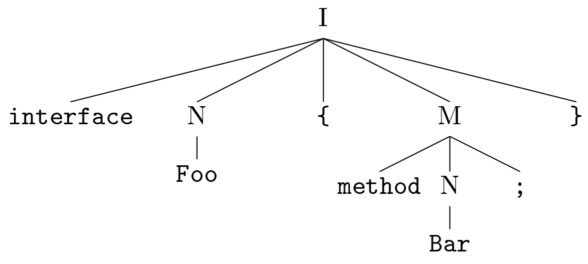
The derivation must start with the starting symbol of the grammar and ends with the word containing no variables. However, this produces a problem. A possible grammar for the code above may be:

$$
\begin{aligned}
&G = (V, \Sigma, P, I)\\
&V = \{N, M, I\}\\
&\Sigma = \{\texttt{interface}, \texttt{method}, \texttt{Foo}, \texttt{Bar}, \texttt{\{}, \texttt{\}}, \texttt{;}\}\\
&P = \{\\
&\quad N \rightarrow \texttt{Foo},\\
&\quad N \rightarrow \texttt{Bar},\\
&\quad M \rightarrow \texttt{method}N\texttt{;},\\
&\quad I \rightarrow \texttt{interface}N\texttt{\{}M\texttt{\}}\\
&\quad \}
\end{aligned}
$$

It is easy to observe, that there is no derivation starting with I and ending with the fragment $f =$ `method Bar;`. Thus f is not an element of $\mathcal{L}(G)$. (Note, we omitting, that another grammar might exist which produces the exact same words and allows derivation of all fragments. However, the fact remains, that the given grammar still creates fragments while not allowing a derivation.)

Fragment Grrammars & Languages

The introductory code example produces the following syntax tree:



One can observe, that each fragment is produced by an arbitrary derivation, however, not necessarily starting with the defined start variable of the grammar. From this, we assume that each fragment is governed by its own grammar. For example, fragments can be generated by slightly altered grammars just switching the start variable. For instance the grammar $G' = (V_G, \Sigma_G, P_G, M)$ can produce the fragment:

```
method Bar;
```

So for any given grammar there is a set of grammars of which it is also an element:

$$G \in \bigcup_{v \in V_G} \{(V_G, \Sigma_G, P_G, v)\}$$

Note, that these fragment grammars are restricted to context-free grammars. Our method of switching start variables assumes $P \subseteq V \times (V \cup \Sigma)^*$.

In order, to address terminal symbols as fragments, we construct a set of minimal context-free grammars, each only capable of deriving one terminals

$$\bigcup_{t \in \Sigma_G} \{(\{V_t\}, \{t\}, \{V_t \to t\}, V_t)\}$$

For example: $G''' = (\{V_{\texttt{Foo}}\}, \{\texttt{Foo}\}, \{V_{\texttt{Foo}} \to \texttt{Foo}\}, V_{\texttt{Foo}})$

Since context-free grammars are closed under union, meaning the union is also context-free, we can now create *Fragment Grammar* for a given grammar:

$$F(G) = \bigcup_{v \in V_G} (V_G, \Sigma_G, P_G, v) \cup \bigcup_{t \in \Sigma_G} (\{V_t\}, \{t\}, \{V_t \to t\}, V_t)$$

The example grammar above has the fragment grammar:

$$
\begin{aligned}
&F(G) = (V, \Sigma, P, S)\\
&\quad V = \{E, F, B, V_{\texttt{interface}}, V_{\texttt{method}}, V_{\texttt{Foo}}, V_{\texttt{Bar}}, V_{\texttt{\{}}, V_{\texttt{\}}}, V_{;}\}\\
&\quad \Sigma = \{\texttt{interface}, \texttt{method}, \texttt{Foo}, \texttt{Bar}, \texttt{\{}, \texttt{\}}, \texttt{;}\}\\
&\quad P = \{\\
&\qquad N \to \texttt{Foo},\\
&\qquad N \to \texttt{Bar},\\
&\qquad M \to \texttt{method}N\texttt{;},\\
&\qquad I \to \texttt{interface}N\texttt{\{}M\texttt{\}},\\
&\qquad V_{\texttt{Foo}} \to \texttt{Foo},\\
&\qquad V_{\texttt{Bar}} \to \texttt{Bar},\\
&\qquad V_{\texttt{method}} \to \texttt{method},\\
&\qquad V_{\texttt{interface}} \to \texttt{interface},\\
&\qquad V_{\texttt{\{}} \to \texttt{\{},\\
&\qquad V_{\texttt{\}}} \to \texttt{\}},\\
&\qquad V_{;} \to \texttt{;},\\
&\qquad S \to N,\\
&\qquad S \to M,\\
&\qquad S \to I,\\
&\qquad S \to V_{\texttt{Foo}},\\
&\qquad S \to V_{\texttt{Bar}},\\
&\qquad S \to V_{\texttt{method}},\\
&\qquad S \to V_{\texttt{interface}},\\
&\qquad S \to V_{\texttt{\{}},\\
&\qquad S \to V_{\texttt{\}}},\\
&\qquad S \to V_{;}\\
&\qquad \}
\end{aligned}
$$

From the fragment grammar, we can now also generate a *Fragment Language*, containing all fragments which can be produced by that grammar:

$$\mathcal{F}(G) = \mathcal{L}(F(G)) = \bigcup_{g \in F(G)} \{w \in \Sigma^* | S_g \Rightarrow_g^* w\} = \bigcup_{g \in F(G)} \mathcal{L}(g)$$

Our original language is now a subset of the fragment language:

$$\mathcal{L}(G) \subseteq \mathcal{F}(G)$$

```
Artifact < Entity
Set      < Entity
Grammar  < Artifact
Language < Set
Framgent < Artifact

subsetOf    < Set * Set
elementOf   < Artifact * Language
conformsTo  < Artifact * Artifact
conformsTo  < Language * Artifact
partOf      < Artifact * Artifact

// an arbitrary grammar artifact
aGrammarArtifact : Grammar
aGrammarArtifact = "uri/to/grammar"

// a fragment of the grammar
aFragmentLanguage : Language
aFragmentLanguage conformsTo aGrammarArtifact

// a language generated be the grammar
aLanguage : Language
aLanguage = "uri/to/language"
aLanguage subsetOf aFragmentLanguage
aLanguage conformsTo aGrammarArtifact

// a artifact/word of the language
aLanguageArtifact : Artifact
aLangugaeArtifact = "uri/to/language/artifact"
aLanguageArtifact elementOf aLanguage
aLanguageArtifact conformsTo aGrammarArtifact

// a fragment of the artifact above
aFragment : Fragment
aFragment = "uri/to/language/artifact#fragment"
aFragment elementOf aFragmentLanguage
aFragment partOf aLanguageArtifact
aFramgnet conformsTo aGrammarArtifact
```