SE 550: Lecture 3

Overview

Protocol specification

Grammars

BNF and Extended BNF

LL(1) grammars

Summary
Protocol specification

What is the IETF? What is an RFC?

What is a protocol specification?

Specifications have been very important in some fields (e.g. communication protocols, compilers) but not in others (e.g. application file formats). Why?

What information does a protocol specification contain?

What is a formal specification? Why are formal specifications better than informal specifications?
Example: HTTP

How can we find the specification of HTTP?

Part of the specification:

\[
date1 = 2DIGIT \ SP \ month \ SP \ 4DIGIT
\]

\[
month = "Jan" \ | \ "Feb" \ | \ "Mar" \ | \ "Apr"
\ |
"May" \ | \ "Jun" \ | \ "Jul" \ | \ "Aug"
\ |
"Sep" \ | \ "Oct" \ | \ "Nov" \ | \ "Dec"
\]

\[
SP = <US-ASCII SP, space (32)>
\]

\[
DIGIT = <any US-ASCII digit "0"..'9'>
\]

What does this define?

Which of these are date1s: Fred Flintstone? 25 Sep 2000? 25 Sep 00?
Example: IMAP

Real-world example.

[ajeffrey@klee se550]$ telnet bach.cs.depaul.edu imap
Trying 140.192.33.6...
Connected to bach.cs.depaul.edu.
Escape character is '^[].'
* OK Microsoft Exchange IMAP4rev1 server...
  1 login foobar {8}
+ Ready for additional command text.
  {abcdef}
  1 BAD Protocol Error: "Literal end without corresponding literal begin"

So MS Exchange doesn’t allow a password {...}

Is this OK or not?

Without a specification, we just argue about it.
Example: IMAP

Part of the IMAP specification:

```plaintext
login ::= "LOGIN" SPACE userid SPACE password

password ::= astring

astring ::= atom / string

string ::= quoted / literal

literal ::= "{" number "}" CRLF *CHAR8

CHAR8 ::= <any 8-bit octet except NUL, 0x01 - 0xff>

So who was right?
Grammars

Unfortunately, nobody agrees on syntax for grammars...

The style of the HTTP protocol spec:

name = first "Flintstone"
first = "Fred" | "Wilma"

The style of the IMAP protocol spec:

name ::= first "Flintstone"
first ::= "Fred" / "Wilma"

The style used in Johnsonbaugh (Backus Naur Form):

<name> ::= <first> Flintstone
<first> ::= Fred | Wilma

We’ll use BNF.
Math preliminaries

Reminders of some math you should already know...

- What is the set $\emptyset$?
- What is the set $S \times T$?
- What is the set $S \cup T$?
- What is the set $S^*$?
- What is the string $\lambda$?

Examples:

- What is the set $\{a,b\} \times \{b,c\}$?
- What is the set $\{a,b\} \cup \{b,c\}$?
- What is the set $\{a,b\}^*$?
- Is $\lambda \in \emptyset$?
- Is $\lambda \in \{a,b\}^*$?
Grammars

A context free grammar is given by:

- A set $T$ of terminal symbols.
- A set $N$ of nonterminal symbols.
- A set $P \subseteq N \times (N \cup T)^*$ of productions.
- A starting symbol $\sigma \in N$.

We normally write $a$ for terminals, $A$ for nonterminals, $\alpha$ for strings in $(N \cup T)^*$, and $A \rightarrow \alpha$ for productions $(A, \alpha)$.

In BNF style, we write $A ::= \alpha_1 | ... | \alpha_n$ when $A \rightarrow \alpha_1$, ... $A \rightarrow \alpha_n$ are all the productions for $A$. By convention, a BNF is written with the start symbol first.

What are the terminals, nonterminals, productions and start state of the BNF:

\[
\text{<name>} ::= \text{<first>} \text{ Flintstone}
\]
\[
\text{<first>} ::= \text{ Fred} | \text{ Wilma}
\]
Grammars

A *language* is a set of strings.

We need to know the language recognized by a grammar. (Why?)

A *derivation* $\alpha \Rightarrow \beta$ is given by two rules:

- If $B \rightarrow \beta$, then $\alpha B \gamma \Rightarrow \alpha \beta \gamma$.
- If $\alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n$ then $\alpha_1 \Rightarrow \alpha_n$.

In the BNF:

```
<name> ::= <first> Flintstone
<first> ::= Fred | Wilma
```

what are the derivations of `<name>`?
Grammars

The language recognized by a grammar is defined:

\[ L(G) = \{ \alpha \in T^* \mid \sigma \Rightarrow \alpha \} \]

What is the language accepted by the BNF:

\[
\begin{align*}
\text{name} & ::= \text{first} \text{ Flintstone} \\
\text{first} & ::= \text{Fred} \mid \text{Wilma}
\end{align*}
\]
Example grammars

What is the language accepted by:

\[ <\text{foo}> ::= a \mid b \ <\text{foo}> \]

What is the language accepted by (recall \( \lambda \) is the empty string):

\[ <\text{bar}> ::= \lambda \mid a \ b \ <\text{bar}> \]

What is the language accepted by:

\[ <\text{baz}> ::= \lambda \mid a \mid b \mid a <\text{baz}> a \mid b <\text{baz}> b \]
Example grammars

What about:

date1 = 2DIGIT SP month SP 4DIGIT

month = "Jan" | "Feb" | "Mar" | "Apr"
  | "May" | "Jun" | "Jul" | "Aug"
  | "Sep" | "Oct" | "Nov" | "Dec"

SP    = <US-ASCII SP, space (32)>

DIGIT = <any US-ASCII digit "0".."9">
Extended BNF

Common extensions to BNF:

- $A ::= \alpha^*$ (sequence)
- $A ::= \alpha^+$ (non-empty sequence)
- $A ::= \alpha^?$ (option)

For example from the HTTP protocol (syntax changed to use EBNF):

```plaintext
<chunk-extension> ::= ( ";" <chunk-ext-name> ( ";" <chunk-ext-val> )? )*
```

what does this grammar specify?
Extended BNF

EBNF extensions are just syntax sugar, they don’t add any expressive power.

For example, we can translate away any use of * as:

\[ A ::= \alpha^* \]

becomes:

\[ A ::= \lambda | \alpha A \]

What about + and ?

How do you translate this HTTP spec fragment into BNF:

\[ <\text{chunk-extension}> ::= (";" <\text{chunk-ext-name}> ("=" <\text{chunk-ext-val}> )? )* \]
LL(1) grammars

An important class of grammars are called LL(1) grammars (Left-to-right parsing, Leftmost-derivation, 1-token lookahead).

A grammar is LL(1) when:

\[ A \rightarrow \alpha_1 \Rightarrow a\beta_1 \text{ and } A \rightarrow \alpha_2 \Rightarrow a\beta_2 \]
then \( \alpha_1 = \alpha_2 \)

Example which is not LL(1):

\[
<\text{foo}> ::= a\ b | a\ c
\]

Fixed grammar which is LL(1):

\[
<\text{foo}> ::= a\ <\text{bar}>
<\text{bar}> ::= b | c
\]

[Note: this definition of LL(1) only works for grammars where \( A \rightarrow \lambda \). Adventurous students can try to fix this!]
**LL(1) grammars**

A grammar is LL(1) when:

\[
\begin{align*}
\text{If } A &\to \alpha_1 \Rightarrow a \beta_1 \\
\text{and } A &\to \alpha_2 \Rightarrow a \beta_2 \\
\text{then } \alpha_1 &\neq \alpha_2
\end{align*}
\]

Is this grammar LL(1):

\[
<\text{foo}> ::= a \ b \ c \mid a \ c \ b
\]

Is this grammar LL(1):

\[
<\text{foo}> ::= a \ b \ c \mid c \ b \ a\]

Is this grammar LL(1):

\[
<\text{foo}> ::= a \ <\text{bar}>
<\text{bar}> ::= b \ c \mid c \ b
\]
LL(1) grammars

A grammar is LL(1) when:

If \( A \rightarrow \alpha_1 \Rightarrow a \beta_1 \)
and \( A \rightarrow \alpha_2 \Rightarrow a \beta_2 \)
then \( \alpha_1 = \alpha_2 \)

Is this grammar LL(1):

\[
<\text{foo}> ::= <\text{bar}> | <\text{baz}>
\]
\[
<\text{bar}> ::= a \ b
\]
\[
<\text{baz}> ::= a \ c
\]

Is this grammar LL(1):

\[
<\text{foo}> ::= <\text{bar}> | <\text{baz}>
\]
\[
<\text{bar}> ::= a \ b
\]
\[
<\text{baz}> ::= b \ a
\]
LL(1) grammars

Is this grammar LL(1):

\[
\begin{align*}
\langle \text{foo} \rangle & ::= \langle \text{bar} \rangle \ a \ | \ a \\
\langle \text{bar} \rangle & ::= b \ | \ b \ \langle \text{bar} \rangle
\end{align*}
\]

Is this grammar LL(1):

\[
\begin{align*}
\langle \text{foo} \rangle & ::= \langle \text{bar} \rangle \ a \ | \ a \\
\langle \text{bar} \rangle & ::= \lambda \ | \ b \ \langle \text{bar} \rangle
\end{align*}
\]

Is this grammar LL(1):

\[
\begin{align*}
\langle \text{foo} \rangle & ::= \langle \text{bar} \rangle \ a \\
\langle \text{bar} \rangle & ::= \lambda \ | \ b \ \langle \text{bar} \rangle
\end{align*}
\]
**LL(1) grammars**

Why do we care whether a grammar is LL(1) or not?

For more information than you want to know about LL(1) grammars, read any book on compilers, for example *Modern Compiler Implementation in Java*, Appel, Cambridge University Press, 1998.

Also see tools such as JavaCC
Example

Design a protocol which allows a client to find the size of a file.

An example run of the protocol (C> = sent by client, S> = sent by server):

C> SIZE /home/httpd/html/se550/index.html
S> FOUND /home/httpd/html/se550/index.html 312 BYTES
C> SIZE /home/httpd/html/se550/index.htm
S> NOT FOUND
C> QUIT
S> OK

Is the grammar in BNF? Is the grammar LL(1)?

Here’s a sample solution.
**Summary**

Protocol specification makes use of grammars.

Grammars are formal definitions of a language (for example the request and response messages in a communication protocol).

Grammars can be specified in BNF or EBNF. EBNF can be translated down to BNF.

Some grammars are LL(1).

*Reading*: Metamata/Webgain/Sun’s [JavaCC](http://javacc.java.sun.com) documentation.