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? 25 September 2000? 2 Sep 2000? 02 Sep 2000? 55 Sep 2000? 55 Sep 1066?

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:

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 λ?

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, α for strings in $(N \cup T)^*$, and $A \rightarrow \alpha$ for productions (A,α).

In BNF style, we write A ::= α_1 | ... | α_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:

<name> ::= <first> Flintstone
<first> ::= Fred | 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 $\alpha_1 \Rightarrow ... \Rightarrow \alpha_n$ 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:

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

Example grammars

What is the language accepted by:

<foo> ::= a | b <foo>

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

<bar> ::= λ | a b <bar>

What is the language accepted by:

<baz> ::= λ | a | b | a<baz>a | b<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 ::= α* (sequence)
- A ::= α+ (non-empty sequence)
- A ::= α? (option)

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

<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 ::= α*

becomes:

A ::= λ | α A

What about + and ?

How do you translate this HTTP spec fragment into BNF:

<chunk-extension> ::= (";" <chunk-ext-name> ("=" <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:

> If $A \rightarrow \alpha_1 \Rightarrow \mathrm{a}\beta_1$ and $A \rightarrow \alpha_2 \Rightarrow \mathrm{a}\beta_2$
> then $\alpha_1 = \alpha_2$

Example which is not LL(1):

> <foo> ::= a b | a c

Fixed grammar which is LL(1):

> <foo> ::= a <bar>
> <bar> ::= b | c

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

LL(1) grammars

A grammar is LL(1) when:

If $A \rightarrow \alpha_1 \Rightarrow \text{a}\beta_1$
and $A \rightarrow \alpha_2 \Rightarrow \text{a}\beta_2$
then $\alpha_1 = \alpha_2$

Is this grammar LL(1):

<foo> ::= a b c | a c b

Is this grammar LL(1):

<foo> ::= a b c | c b a

Is this grammar LL(1):

<foo> ::= a <bar>
<bar> ::= b c | c b

LL(1) grammars

A grammar is LL(1) when:

If $A \rightarrow \alpha_1 \Rightarrow \text{a}\beta_1$
and $A \rightarrow \alpha_2 \Rightarrow \text{a}\beta_2$
then $\alpha_1 = \alpha_2$

Is this grammar LL(1):

<foo> ::= <bar> | <baz>
<bar> ::= a b
<baz> ::= a c

Is this grammar LL(1):

<foo> ::= <bar> | <baz>
<bar> ::= a b
<baz> ::= b a

LL(1) grammars

Is this grammar LL(1):

<foo> ::= <bar> a | a
<bar> ::= b | b <bar>

Is this grammar LL(1):

<foo> ::= <bar> a | a
<bar> ::= λ | b <bar>

Is this grammar LL(1):

<foo> ::= <bar> a
<bar> ::= λ | b <bar>

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 documentation.