Maple for Math Majors

Roger Kraft
Department of Mathematics, Computer Science, and Statistics
Purdue University Calumet
roger@calumet.purdue.edu

10. Some Grammar

10.1. Introduction

In this section we look at the "grammar" of Maple. In particular, we look at the notions of syntax, parsing, and semantics. These are important ideas for any computer programming language (in fact for any language). We will look at a number of examples that try to explain these concepts.

```
> 
```

10.2. Syntax, parsing, and semantics

The **syntax** of a language is the rules that define how a collection of words can be put together to form proper sentences. Another word for syntax is grammar. In the case of a Maple, its syntax defines how symbols are put together to construct proper inputs for Maple. **Parsing** is the process of looking at an input string of symbols and trying determine the distinct syntactic pieces of the string. Every time you type in a command to Maple and then hit the return key, Maple has to parse what you just typed before it can do anything else. The **semantics** of a syntactically correct Maple input is a fancy way of saying the "meaning" of the input.

To put it all briefly, the syntax of Maple determines which inputs to Maple are acceptable (i.e., don't have syntax errors), parsing is the process by which a given input is determined to be syntactically correct, and the semantics of Maple determines the meaning of a syntactically correct input.

You usually do not worry about all this until you get a syntax error, which means that Maple cannot make sense out of what you typed at its command prompt. For example:

```
> x=:5;
```

How does Maple find a syntax error? Maple **parses** each input line. It breaks up the input line into what it thinks are its basic pieces, and then determines if those pieces fit together properly. For example, with the input

```
x=:5;
```

Maple looks at the `x`, determines that it is a variable name. Then it looks at the equals sign which follows the variable name, so it thinks it is looking at an equation. But then Maple finds the colon which means terminate a command and suppress the output. But Maple never found the rest of the equation it was looking for, so it generated a syntax error pointing at the colon. The rest of the input, the "`5;`" never even got looked at.

Maple has a command called `parse` which takes as its input a string of characters and then parses

the string to see if it is syntactically correct. Let us give the string `` `x=:5;` `` to **parse** and see what it does.

```
> parse(`x=:5;`, statement);   # Notice the left-quotes!
```

Of course we get the same error message, since giving an input string to **parse** is exactlly the same thing as typing the input string at the Maple command prompt and then hitting return. Try fixing the syntax error and running **parse** again.

The above syntax error was caused by a "typing error". The Maple user probably meant to type **x:=5;** but transposed the colon with the equal sign. Not all typing errors lead to syntax errors. For example, the input

```
x:5;
```

could be a typing error where an equal sign was left out. How does Maple parse this input?

```
> x:5;
```

What did Maple do? It saw the **x** and determined that it is a variable name. Then it found the colon which terminates a command but suppresses output. So Maple evaluated **x** and suppressed the output. Then Maple found the **5;** which is another command, terminated by a semicolon this time, that tells Maple to evaluate 5 and so Maple outputed 5. So Maple treated **x:5;** as two commands on one line. There was no syntax error, but it is probably not what the user wanted.

How does Maple parse the following inputs, each of which is meant to be a typing error for **x2:=5**? Some of the inputs are syntactically correct. In that case, what are their semantics?

```
> x2: =5;
> x2==5;
> x 2:=5;
> x2;=5;
> x:2=5;   # Transpose the 2 and the colon.
> x2=5;
>
```

Here is a little observation about Maple's parsing of inputs. Sometimes Maple does not care about extra blank spaces, and sometimes it does. For example, you cannot put a space between the colon and the equals in the assignment operator (as we just saw above), but the following is syntactically correct.

```
>    sin      (   Pi   /
>  2   ^   2  )
>    ;
>
```

Here is a more sophisticated example of a typing error that may or may not lead to a syntax error. Suppose you wanted to input to Maple the command

```
> f := (x,y)->x^2+y^2;
```

but instead, you forgot the parentheses and input the following command. How does Maple parse this and what does it mean?

```
> f := x,y->x^2+y^2;
```

Here is a hint.

```
> f;
>
```

How does Maple parse the following commands? (You may need to look up the assignment statement to figure these out.)

```
> a, b:=0, 1, 2;
> a, b:=0, c:=1, 2;
> a, b:=0, 1;
>
```

How does Maple parse the following?

```
> 1/2/3/4;
> 1*2/3*4;
> 1*2/3/4;
```

Suppose the `/` was right associative. What would `1/2/3/4` evaluate to?

```
>
```

Let us compare the notions of syntax and semantics a bit more. We want to show that the same syntax can have different semantics, and also that different syntaxes can have the same semantics. Consider the following two examples.

What does the expression `x=1..5` mean? This is a syntactically correct way of saying that `x` is a variable that takes on the range of numbers from 1 to 5. Now look at the next two Maple inputs.

```
> seq(x^2, x=1..5);
> plot(x^2, x=1..5);
```

In the first command, `x=1..5` meant all the integers between 1 and 5, but in the second command `x=1..5` meant all real numbers between 1 and 5. So the meaning of the expression `x=1..5` changed depending on the context it was used in; the same piece of syntax had two different semantics.

Now consider the next example.

```
> letters := [l,m,n,o,p,q,r,s];
> letters[3..5];
> [op(3..5, letters)];
```

The last two commands are an example of two very different syntaxes that have the exact same semantics.

```
>
```

To give you another perspective of what we are talking about, here is an example of these kinds of ideas applied to the English language. Consider the following two sentences:

Time flies like an arrow.
Fruit flies like a banana.

Each of these sentences is syntactically correct English. However, each one can be parsed in several different ways, and each different parsing of one of the sentences gives it a different meaning (although some of the meanings may be nonsensical). These two sentences show that parsing and determining the meaning of English sentences can be pretty difficult. How would a computer figure out which way to "correctly" parse each of these sentences? We say that natural languages like English are **ambiguous**. Computer programming languages, like Maple, cannot be ambiguous. In every computer programming language, once an input to the language has been determined to be

syntactically correct, then it has to have only on meaning. In other words, the semantics of a syntactically correct input is well defined.

[>