30. Imperative Scheme and LAML Constructs

This material is about functional programming in Scheme. So why do we include this section about imperative aspects of Scheme and LAML?

The reason is that Scheme in reality is a multi-paradigm programming language - although with strong historical roots in the functional paradigm. As a related observation, in Chapter 29 we saw that the distance between concepts in Scheme and the concepts in object-oriented programming is little too.

When Scheme is used for real world applications - like in the web domain - it is not possible to avoid use of imperative features. Just take, as an example, the handling of files on the harddisk. Some people may be able to regard file handling - deletion and file writing for instance - in a functional way. I prefer to think of these aspects as belonging to the well-known imperative paradigm.

In this brief side track chapter we will review some of the more important imperative aspects of Scheme.

30.1. Imperative Scheme Constructs

Lecture 9 - slide 2

We start by enumerating the main commands - and groups of commands - of the Scheme language. Be sure to notice the assignment syntactical form \texttt{set!}, which is the most important of them all.

\begin{quote}
\begin{center}
\begin{tabular}{|l|}
\hline
\textbf{The most fundamental imperative Scheme construct is the assignment \texttt{set!}} \\
\hline
\end{tabular}
\end{center}
\end{quote}

- Other imperative constructs:
  - \texttt{(begin e1 \ldots en)}
  - The iterative \texttt{do} control structure
  - The input output procedures
  - The list, string and vector mutators

\begin{quote}
\begin{center}
\begin{tabular}{|l|}
\hline
As a notational convention, most imperative abstractions in Scheme ends with \\
\texttt{"!"} \\
\hline
\end{tabular}
\end{center}
\end{quote}

As noticed above it is hard to avoid using imperative constructs when writing real-world Scheme programs in the web domain. But we should be careful. We do not want to mix traditional use of 'the imperative programming style' with functional programming. Thus, we should clearly avoid using \texttt{set!} side by side with all the functional means, which we have described in this material.

As a good rule of thumb, the imperative constructs in our programs should be kept at a minimum, and - most important - they should be used at a few places - typically at top level - such that the inner parts of the program are purely functional.
30.2. List mutators

In Chapter 6 we have studied the list concepts in Scheme (proper and improper lists) and we have seen the functional primitives for list construction and list selection - cons, car, cdr and the functions on top of these.

In this section we will mention and list the mutating functions, which are also available in Scheme.

Besides the list constructor cons and the list selectors car and cdr there are also list mutators called set-car! and set-cdr!

- The list mutator procedures:
  - The command (set-car! x y) changes the value of the car position of the cons cell referred by x. The car position is assigned to y
  - The command (set-cdr! x y) changes the value of the cdr position of the cons cell referred by x. The cdr position is assigned to y
  - Using the list mutators it is possible to make circular structures

Without use of the list mutators, and with use of structural equivalence predicates, it is not possible to tell the difference between a list structure and a copy of the list structure

If we make use of list mutators we can tell the difference between a list structure, LS, and a copy of LS. The way to do it is to mutate LS and observe that the copy is not changed. Using only the functional subset of Scheme (and disregarding the very discriminating equality predicates such as eq? and eqv?) a copy of LS serves the same purposes as LS itself.

30.3. String mutators

Strings can be mutated in Scheme. Below we mention the main procedure for this, string-set!, and we also look at a similar procedure, string-fill!.

A string can be mutated by the string-set! and the string-fill! procedures

- The string mutator procedures:
  - (string-set! str k chr) changes character number k in str to chr
  - (string-fill! str chr) changes every character in str to chr

There are similar functions that mutate the elements in vectors
30.4. Imperative features in LAML

As mentioned in the introduction to this chapter, we need imperative features to deal with the real-world needs in both static and more dynamic web authoring and programming.

In this section we enumerate some of the more important imperative aspects in the LAML software package.

<table>
<thead>
<tr>
<th>LAML needs imperative features for file IO, handling of LAML context information, and high level commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Overview of imperative features in LAML:</td>
</tr>
<tr>
<td>- The procedures write-text-file and read-text-file</td>
</tr>
<tr>
<td>- At a higher level procedures such as write-html which surrounds many HTML mirror documents in Scheme</td>
</tr>
<tr>
<td>- Many high level procedures which represent tools or commands</td>
</tr>
<tr>
<td>- Such as xml-parse, xml-pp, and schemesdoc</td>
</tr>
<tr>
<td>- File handling in general</td>
</tr>
<tr>
<td>- Definition of LAML context information</td>
</tr>
<tr>
<td>- fake-startup-parameters, laml-cd,...</td>
</tr>
</tbody>
</table>

In some situations we have internally in LAML used imperative patching of functional programs, because functional patching has been too difficult and too time consuming.

Unfortunately, we have not systematically used the 'exclamation mark' naming convention of LAML procedures.

30.5. References

[-] LAML tools and commands
    http://www.cs.auc.dk/~normark/scheme/distribution/laml/man/laml.html#SECTION7

[-] The Text Collection and Skipping Library

[-] Reading of writing of text files

[-] R5RS: Vectors in Scheme
    http://www.cs.auc.dk/~normark/prog3-03/external-material/r5rs/r5rs-html/r5rs_62.html#SEC64

[-] R5RS: Strings Scheme
    http://www.cs.auc.dk/~normark/prog3-03/external-material/r5rs/r5rs-html/r5rs_61.html#SEC63

[-] List and pairs in Scheme
 Input output procedures
http://www.cs.auc.dk/~normark/prog3-03/external-material/r5rs/r5rs-
html/r5rs_58.html#SEC60

 do
http://www.cs.auc.dk/~normark/prog3-03/external-material/r5rs/r5rs-
html/r5rs_35.html#SEC37

 begin
http://www.cs.auc.dk/~normark/prog3-03/external-material/r5rs/r5rs-html/r5rs_34.html

 Scheme assignment
http://www.cs.auc.dk/~normark/prog3-03/external-material/r5rs/r5rs-html/r5rs_30.html