Stacks in TEX

Pedro Quaresma

Abstract

There are several situations where we need to "forward reference" something that it is not yet available. For example, when we say something like "as we will see in chapter ..." and when we make a bibliographic citation. Those situations are well treated in LATEX by the use of auxiliary files.

A different situation arises if we want to have a LATEX environment where one or more commands depend on the arguments given to other commands; that is, the values of the arguments of one command are taken from the arguments of another. We can also use an auxiliary file as a way of communication between commands but that implies that we have to process the document twice (at least) in order to complete the environment.

In this paper we describe an implementation of stacks in TEX as a way to solve the problem just described. One command puts the information in the stack, the other command takes the information from the stack, and with this approach we manage to establish communication between commands while processing the document only once.

1 Introduction

In 1990 I was a PhD student in Computer Science and felt the need of a LATEX style file for producing diagrams, namely those used in Category Theory [4], e.g.

$$A \xrightarrow{\quad f \quad} B$$

So, I created a style file whose first version used the LATEX picture environment as the graphical engine, and in a later version switched to PICTEX because of its better capabilities. Since the first version the two main goals of the DCpic [5] package were:

- to have a TEX-only format, in order to have good portability properties;
- to have a simple notation, a notation close to the graph notation where we describe a graph as a set of nodes (objects), and a set of arrows (morphisms) with each arrow having an initial and a end node.

So DCpic implements an environment `\begindc ...\enddc`, with the command `\obj(x,y){`*text*`}` for the nodes, and the command `\mor(x1,y1)(x2,y2){`*text*`}` for the arrows. The diagram pictured above has the following specification:

```
\begindc
\obj(1,1){$A$}
\obj(3,1){$B$}
\mor(1,1)(3,1){$f$}
\enddc
```

The syntax is one of the simplest, if not the simplest, among packages of this type [2], but it deviates from the graph notation because the arrows specification is done in absolute terms and not in relative terms, i.e., it does not state the initial and end node of each arrow, but rather their positions. Since version 2 we began to look for a specification syntax that would allow the following specification for the diagram:

```
\begindc
\obj(1,1){$A$}
\obj(3,1){$B$}
\mor{$A$}{$B$}{$f$}
\enddc
```

But this implies that the coordinates of objects A and B must be passed to the `\mor` command. We also want to pass the dimensions of the (box) objects to be able to adjust the arrow length accordingly.

How to do this? I saw two possible solutions:

- the `\obj` command writes all the information about its object in an auxiliary file, after which `\mor` reads the information from that file.
- the `\obj` command writes all the information about its object in an auxiliary structure kept in memory, after which `\mor` reads the information from that structure.

The first solution seemed to me too complicated to the problem at hand; it is also inefficient because for a large diagram we have to open and close the auxiliary file too many times (or enforce a strict separation between objects and morphisms). Because of this we chose the second approach and decided to implement stacks.

Using stacks the solution is very simple: `\obj` pushes the information onto the stack, and `\mor` looks for it on the stack and whenever necessary pops it from the stack.

2 Stacks

To implement stacks we need a structure in which to place the elements and functions to operate on the stack. Using the *Maude* [1, 3] syntax we have:

```
fmod STACK is
 sorts Elem NeStack Stack .
 subsorts Elem < NeStack < Stack .
 op empty : -> Stack .
 op push : Elem Stack -> NeStack .
 op pop : NeStack -> Stack .
 op top : NeStack -> Elem .
 op isempty : Stack -> Bool .
 var S : Stack .
```

```
 var E : Elem .
 eq pop(push(E,S)) = S .
 eq top(push(E,S)) = E .
 eq isempty(empty) = true .
 eq isempty(push(E,S)) = false .
endfm
```

That is, we have *elements* and *stacks* of elements; *empty* gives us the empty stack; *push* puts an element on top of the stack; *pop* deletes the element on top of the stack; *top* sees (and does not delete) the element on top of the stack; and *isempty* returns whether the stack is empty or not.

2.1 Stacks in TEX

Not having a predefined type structure in TEX that can support stacks, we decided to implement stacks as a macro. We began by defining (initializing) it as the empty stack, that is, a stack that contains only a sentinel, the "end-of-stack" element.

```
\def\emptystack{:}
\let\stack=\emptystack
```

The elements of the stacks will be other TEX elements, e.g., we can put `\$x\$` on the stack. The stack is now a structure that may contain almost anything.

The implementation of the the functions is now a matter of redefinition of the macro `\stack`.

The "push" function has one argument only, the element to be pushed; the result is the stack with that element on top of it.

```
\def\push#1{%
        \edef\stack{#1.\stack}%
        }
```

(The dot serves as an element separator.)

The "topstack" function has no arguments; the result is the element that it is on the top of the stack. We use an auxiliary function and the "expandafter" command to control the expansion of the arguments.

```
\def\topaux#1.#2:{#1}
\def\topstack{\expandafter\topaux\stack}
```

The "pop" function also has no arguments; the result is the stack without the top element. It is very similar to `\topstack`.

```
\def\popaux#1.#2:{\def\stack{#2:}}
\def\pop{\expandafter\popaux\stack}
```

To implement the "isempty" predicate we need to define the appropriate "if". In DCpic we have opted for two "ifs", one to test if the stack is empty, and the other to test if the stack is not empty.

```
\newif\ifisempty
\newif\ifnisempty

\def\isempty#1{%
   \let\arg=#1\relax
   \if:\arg\ \isemptytrue
   \else \isemptyfalse\fi}

\def\nisempty#1{%
   \let\arg=#1\relax
   \if:\arg\ \nisemptyfalse
   \else \nisemptytrue\fi}
```

As you can see, this is a one-stack-at-a-time implementation; we begin by defining the object that we call `\stack` and then all the operations are done on that object. This does not mean that we can only have one stack in the document. We can create other stacks by saying, e.g., `\let\secondstack=\stack` and later `\let\stack=\secondstack`, but all the operations are still done with `\stack`.

2.2 Using Stacks

The Polish Notation To illustrate the use of these stacks in TEX, let's pretend to calculate arithmetic expressions in Reverse Polish Notation (postfix notation). Our example will be this expression:

$$2\ 1\ +\ 3\ \times$$

Start by pushing all the elements into the stack:

```
\push{2}
\push{1}
\push{+}
\push{3}
\push{$\times$}
```

After this, the stack has this form:

Stack = ×.3.+.1.2.:

Loop until the stack is empty We can construct a cycle that will stop when the stack is empty:

```
\loop
        Pop, \pop \quad Stack=\stack
        \nisempty\stack
        \ifnisempty
        {\endgraf }
\repeat
```

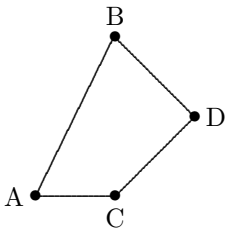
Using the stack from the previous example:

Stack=×.3.+.1.2.:
Pop, Stack=3.+.1.2.:
Pop, Stack=+.1.2.:
Pop, Stack=1.2.:
Pop, Stack=2.:
Pop, Stack=:

Stacks in the DCpic package The use of stacks allows a very simple notation for the specification of our diagrams in LATEX. For example:

```
\begindc{\undigraph}[15]
\obj(1,1){A}[\west]
\obj(3,5){B}
\obj(3,1){C}[\south]
\obj(5,3){D}[\east]
\mor{A}{B}{}
\mor{A}{C}{}
\mor{B}{D}{}
\mor{C}{D}{}
\enddc
```

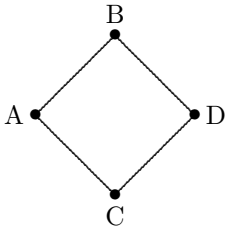
gives



But, if we realize that we misplaced "A" we can correct that modifying the "A" coordinates only:

```
\begindc{\undigraph}[15]
\obj(1,3){A}[\west]
... the rest remains the same ...
\enddc
```

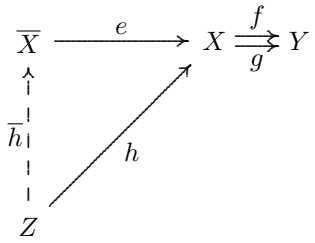
gives



We can put almost anything in the stack, for example:

```
\begindc{\commdiag}[2]
\obj(1,1){$Z$}
\obj(1,36){$\overline{X}$}
\obj(36,36){$X$}
\obj(52,36){$Y$}
\mor{$Z$}{$\overline{X}$}{$\overline{h}$}%
  [\atleft,\dasharrow]
\mor{$Z$}{$X$}{$h$}[\atright,\solidarrow]
\mor{$\overline{X}$}{$X$}{$e$}
\mor(36,37)(52,37)[8,8]{$f$}
\mor(36,35)(52,35)[8,8]{$g$}[\atright,%
  \solidarrow]
\enddc
```

gives



As you can see, the communication between `\obj` and `\mor` does not follow a strict first-in-last-out discipline; so what we do is to preserve the stack before a pop operation and recover the value afterwards. A list structure would be more appropriate for DCpic, but the simplicity of the stack implementation justifies, in our view, a little loss of efficiency.

3 Conclusions

Using an auxiliary structure like the stack gives us the possibility of organizing our TEX programs with intercommunicating macros. The communication is established using the global variable `\stack`.

We decided not to deal with the error situations (e.g., a pop of the empty stack) in our implementation of stacks. The `\mor` command analyzes the stack to see whether it is empty; if so, then it writes an error message in the output and tries a naive error recovery using some default values.

The use of stacks in DCpic allows us to have a very simple notation for the graphs without a "visible" burden to the user. We are certain that this approach will be useful in other situations.

References

[1] Manuel Clavel, Francisco Durán, Steven Eker, Lincoln Patrick, Narciso Martí-Oliet, Meseguer José, and Carolyn Talcott. *Maude Manual*. Computer Science Laboratory, SRI International, April 2005. Version 2.1.1.

[2] Gabriel Valiente Feruglio. Typesetting commutative diagrams. *TUGboat*, 15(4):466–484, 1994.

[3] Joseph Goguen and Timothy Winkler. Introducing OBJ. Technical Report SRI-CSL-88-9, SRI International, Computer Science Lab, August 1988.

[4] Benjamin Pierce. *Basic Category Theory for Computer Scientists*. Foundations of Computing. The MIT Press, London, England, 1998.

[5] Pedro Quaresma. DCpic, commutative diagrams in a (LA)TEX document. In *Proceedings of the EuroTEX 2001 conference*, Rolduc, The Netherlands, September 2001.