PS Interpreter Issues

CptS 355 - Programming Language Design

Washington State University

Home

Kliks

Calendar

Syllabus

Resources

People

Project turn-in

A PostScript Interpreter

Sketch the implementation of a PostScript interpreter in pseudo-code and diagrams, and taking into account the handling of

- numbers
- arithmetic and boolean operations
- if, ifelse
- {}
- def
- /name
- name
- dict
- begin
- end

and without worrying about graphics-related components and operations.

How will you read the input and break it up into meaningful components, e.g. /name, or a number, or {?

Pay attention to what you might not understand well enough to describe well and to see if you noticed anything about interpretation of PS that could not be handled by the mechanisms described so far.

The primary misunderstandings usually have to do with distinguishing between dictionaries and dictionary entries, with distinguishing between the behavior of code and non-code values as the values of names in dictionaries, and the rules for looking up names.

1. A dictionary is not the same as a dictionary entry. The dictionary stack in the interpreter is a stack of dictionaries. Each dictionary contains a number of dictionary entries. begin pushes a dictionary on the dictionary stack, end removes a dictionary from the dictionary stack. def creates or modifies an entry in the top dictionary on the dictionary stack.
2. When a name is encountered in the program its value is looked up in the dictionary stack (see next the next item). If the value of the name is a non-code value it is pushed on the operand stack. If it is a code value it is executed as a subroutine.
3. Names are looked up in the dictionary stack by first looking in the top (warmest) dictionary, then the one below it, and so forth down the dictionary stack. If the name is not defined in any of the dictionaries it is an error. def always creates or modifies the topmost (warmest) dictionary.

A small misunderstanding had to do with the handling of {}. Braces may be nested so in reading a block of code it is important to read all the way to from the { to the *matching* }. (Cover the algorithm for matching braces/parentheses by counting.)

Finally, although we have talked about calling code as a subroutine and it seems natural enough, we discover that the PS abstract machine as we had previously described does not have a mechanism for keeping track of where we are executing in multiple code blocks. Thus we need to include in the AM an *execution stack* that maintains the execution point in each active code block so we can return to it when we finish executing a routine that has been called. Calling a code block can also be modelled as *copying* that code block to the point of the call. (Illustrate the above using the fact function.)

(c) 2003 Curtis Dyreson, (c) 2004-2006 Carl H. Hauser *E-mail questions or comments to Prof. Carl Hauser*