A Formal Description of NoBeard

v 01.01.00

P. Bauer

HTBLA Leonding
Limesstr. 14 - 18
4060 Leonding
Austria

Contents

Chapter 1

Introduction

According to a web article (see [1]) the popularity of programming languages is strongly related to the fact whether its inventor(s) is/are are bearded m[ae]n or not. Well, the main aim of the programming language NoBeard is not to be popular, moreover it should give the reader a clear insight how the main principles of compiler construction are.

This report aims to give a formal description of the programming language NoBeard. Please note that only the parts necessary for your work can be trusted. In the next versions, more and more information relevant for your assignments will be available.

In case of typos, misleading wording or other problems, please feel free to contact me. Thanks for your help.

Chapter 2

The Programming Language

2.1 Lexical Structure

NoBeard programs are written in text files of free format, i.e., there is no restriction concerning columns or lines where the source text has to be. In this section the scanner relevant terms for NoBeard are denoted in the form of regular expressions with the extension that we allow "definitions" of non-terminals. This means in particular that if we define a term (e.g. *letter* as it can be seen in the next section) this term can be used in subsequent definitions and is rewritten as given in its original definition.

2.1.1 Character Sets

letter `'[A-Za-z]'`

digit `'[0-9]'`

2.1.2 Keywords

There is only one keyword, namely `PUT`.

2.1.3 Token Classes

ident `['letter(letter | digit)*']`

number `['digit digit*]`

2.1.4 Single Tokens

The characters "`+`", "`-`", "`*`", "`/`", "`:=`", "`;`", "`(`", and "`)`" are mapped to single tokens.

2.1.5 Semantics

- NoBeard is a case sensitive language. For example, the names "myVar", "myvar", and "MYVAR" denote three different identifiers.

- Constants may only be between 0 and 65535 ($2^{16} - 1$).
- No symbol may span over more than one line.

2.2 Syntax

The following context free grammar gives the syntax of NoBeard. The well-known EBNF notation is used.

NoBeard	=	"unit" ident ";" Block ident ";".
Block	=	"do" StatSeq "done".
StatSeq	=	Stat {Stat}.
Stat	=	VarDecl ";"
		| Assignment ";"
		| PutStat ";"
		| ε.
VarDecl	=	Type ident ["=" Expr].
Type	=	TypeSpec [ArraySpec].
TypeSpec	=	SimpleType .
ArraySpec	=	"[" number "]".
SimpleType	=	"int" | "char" | "bool".
Assignment	=	Reference "=" Expr .
PutStat	=	"PUT" "(" Expr ["," Expr] ")"
		| "PUTLN.
Reference	=	ident ["[" Expr "]"].
Expr	=	[AddOp]Term {AddOp Term}.
Term	=	Fact {MulOp Fact}.
Fact	=	Reference | number | string
		| FuncCall | "(" Expr ")".
AddOp	=	"+" | "-".
Mulop	=	"*" | "/" | "%".

Chapter 3

The NoBeard Machine

3.1 Overview

The virtual machine being target for NoBeard programs is a stack machine with instructions of variable length. The word width of the NoBeard machine is 4 bytes.

dat[MAX_DATA] The data memory is byte addressed and is separated into three parts: String constants (not used yet), activation records and expression stack.

The expression stack is addresses word-wise only.

3.2 List of Assembler Instructions

Opcode	Mnemonic	Arguments	Description	Size
0	LIT	n	Load Literal **Push**(n);	3 Bytes

1	LA	d, a	Load Address
			base = **db**;
			for (i= 0; i < d; i++) {
			base = dat[base ... base + 3];
			}
			adr = base + a;
			Push(dat[addr ... addr + 3];
				4 Bytes

Chapter 4

Symbol List

```
1  unit M;
2    function A(int a);
3      int b;
4
5      int function B(char c);
6        int d;
7      do
8          # some code
9      done B;
10
11     char function C;
12       int e;
13     do
14       # some more code
15     done C;
16   do
17     # some code on A
18   done A;
19 do
20   # this is the main of unit M
21 done M;
```

After parsing line 1 the symbol list looks as follows:

name	kind	type	size	addr	level
M	PROCKIND	UNITTYPE	0	0	0

After parsing line 2 a snapshot on the symbol list looks like

name	kind	type	size	addr	level
M	PROCKIND	UNITTYPE	0	0	0
A	PROCKIND	UNITTYPE	0	0	1
a	PARKIND	SIMINT	4	32	2

After parsing line 3

name	kind	type	size	addr	level
M	PROCKIND	UNITTYPE	0	0	0
A	PROCKIND	PROCTYPE	4	0	1
a	PARKIND	SIMINT	4	32	2
b	VARKIND	SIMINT	4	36	2

After parsing line 6 being somewhere between line 7 and the end of line 9.

name	kind	type	size	addr	level
M	PROCKIND	UNITTYPE	0	0	0
A	PROCKIND	PROCTYPE	4	0	1
a	PARKIND	SIMINT	4	32	2
b	VARKIND	SIMINT	4	36	2
B	PROCKIND	PROCTYPE	0	0	2
c	PARKIND	SIMCHAR	1	32	3
d	VARKIND	SIMINT	4	36	3

Chapter 5

Attributed Grammar

Leongage	=		sem EmitOp(INC); Emit2(0); int inc_addr = 1; endsem
		Stat ";" {Stat ";"}.	
Stat	=	ident ":=" $\text{Expr}_{\uparrow op}$ |	
		"PUT" $\text{Expr}_{\uparrow op}$.	
$\text{Term}_{\uparrow op}$	=	$\text{Fact}_{\uparrow op}$	
		{("*"	sem opcode = mul endsem
		| "/"	sem opcode = div endsem
		)	semEmitOp(↓opcode)endsem
		$\text{Fact}_{\uparrow op}$	semLoadVal(↓op)endsem

Bibliography

[1] T. Khason. Computer languages and facial hair – take two. http://khason.net/blog/computer-languages-and-facial-hair--take-two/, April 2008.