EBF++

Pandu Rendradjaja, John Tran, John Wilkey
58+.17-.

The problem

Programming in BF is hard due to:

- a minimal set of language features
- and no support for abstraction

Programming in EBF (an extended language that compiles into BF) is still hard, due to:

- limited abstraction
- no debugging tools for EBF (rather than BF)

Domain Analysis

We aim to bring the BF language family closer to containing a useful general-purpose programming language. Writing large programs in BF is possible, but painful.

In particular, we want to make it easier to write abstracted code, (via an extended macro system) and to write code that uses memory in “array” and “struct” layouts. (via language support for these)

Our language extensions

```
{pour \ src \ dst \\    ; define a macro with arguments
  %src [- %dst+ %src]}

:a :b                   ; define variables
$a 3+ $b 2+             ; initialize variables

&{pour / $a / $b}       ; use macro. expands to
                        ; $a [- $b+ $a]

:=Triple {a b c}                ; declare a struct type
::list Triple {1 2 3 / 1 4 9}   ; def. and init. an array

$:list 1                        ; list[1]
$$a                             ; list[1].a
$$b                             ; list[1].b
```

Demo: a factorial macro

```
!'stdlib'              ; import standard library

; define macro that sets dst = n!
{factorial \ n \ dst \ t0 \ t1 \ t2 \ t3 \\
  %t3+                                    ; t3 = 1
  %n(                                     ; while n
    &{mul / %n / %t3 / %t0 / %t1 / %t2}   ;   t3 *= n
    %n-)                                  ;   n -= 1
  &{pour / %t3 / %dst}}                   ; dst = t3

:a:b:t0:t1:t2:t3     ; declare variables a...t3
$a 5+                ; a = 5

; invoke macro to calculate 5!, storing into variable b
&{factorial / $a / $b / $t0 / $t1 / $t2 / $t3}
```

Implementation

First, we ported the existing EBF compiler (written in EBF) to JavaScript. We wrote a Jison grammar to create the parser.

Then we extended this compiler with our additional features to create an **EBF++ compiler**.

We used the same parser for an HTML/JavaScript **EBF++ interpreter and debugger**.

The interpreter-debugger is at its core a BF interpreter that uses the EBF++ compiler to determine how to execute each EBF++ instruction.

An implementation trick we're proud of

Every array index is assigned an illegal variable name; the compiler uses these internally to resolve “static” array accesses.

A mistake

$$goto_member is confusing; it depends on the last array access.

What we would do differently

- Approach: PL design is *hard*. We should have been writing programs at every step.
- Language: Appending to arrays should have language support. Also, there could be real variables *in EBF++*, in addition to variables for cell locations in BF.