Poor Man's Recognizer

Klaus Schleisiek
kschleisiek at freenet.de

Classical Parser

```
: host-compiler ( addr len -- )
\ search the host's dictionary
   2dup find-name ?dup
   IF  nip nip   State @
      IF  name>comp  ELSE  name>int  THEN
      execute
   EXIT THEN
\ try to convert to an integer number
   2dup 2>r snumber? ?dup 0=
   IF  2r> interpreter-notfound  EXIT THEN
   2rdrop  State @
   IF  0> IF  swap postpone Literal  THEN
       postpone Literal
   EXIT THEN
   drop
;
```

Classical Parser

- The classical parser describes the entire parsing process in one single definition, which results in a clumsy control structure.
- It explicitly includes all stages after tokenization (**`name`**):
  - dictionary search (**`find-name, snumber?`**),
  - **`xt`** generation depending on **`State`**,
  - and the final **`execute`**.
- Every stage of the parsing process is contained in a single definition, which enhances readability.

Poor Man's Recognizer

- In order to overcome the clumsy control structure, which becomes worse with every additional 'recognizer' clause, we want to be able to write **`host-compiler`** down as follows:

```
: host-compiler ( addr len -- )
   host-find host-number not-found
;
```

- All 'recognizers' in the definition of **`host-compiler`** have identical stack behaviour:

```
: <recognizer> ( addr len -- addr len | rdrop )
```

- When a match was found, we just leave **`host-compiler`** by way of an **`rdrop`**. Otherwise, **`addr len`** remains on the stack and the next recognizer is called until we finally bump into **`not-found`**.

Poor Man's Recognizer

```
: host-find ( addr len -- addr len | rdrop )
   2dup find-name ?dup 0= ?EXIT   rdrop nip nip
   State @ IF  name>comp  ELSE  name>int  THEN
   execute ;
: host-number ( addr len -- addr len | rdrop )
   2dup 2>r snumber? ?dup 0= IF  2r> EXIT THEN
   rdrop rdrop rdrop
   State @ IF  0> IF  swap postpone Literal  THEN
               postpone Literal
   EXIT THEN
   drop ;
: host-compiler ( addr len -- )
   host-find host-number not-found
;
```

Note on Parsing

- I pondered on gforth's **`[ ]`** and **`parser`**. They are defined in gforth as follows:

```
: ]  ( -- ) ['] compiler is parser     1 State ! ;
: [  ( -- ) ['] interpreter is parser  0 State ! ;
```

- Bad luck if you need to modify the parser e.g. in a cross-compiler environment that either compiles code into the host or into the target, which requires different parsers. **`[`** and **`]`** will clobber your **`parser`** setting!

Note on Parsing

- A more useful implementation looks like this:

```
: ]  ( -- )   1 State ! ;
: [  ( -- )   0 State ! ;
: host-parser ( addr len -- )
   State @ IF  compiler  ELSE  interpreter  THEN
;
```