Parsing Virtual Machine and Script Language

Bumble.sf.net/books/pars

1 "About "pep" or "nom"

Pep, `nom` or pep/nom is a virtual machine and script language for parsing text patterns (languages). Pep/nom is designed as an alternative to tools such as Lex, Yacc, Flex, Bison or ANTLR.

This folder contains various files and folders relating to the pattern-parser virtual machine and script language ("pep" - Parsing Engine for Patterns). This is an experimental, but powerful, and I believe, original approach to parsing context-free languages (text patterns). It may seem an outlandish claim, but this tool could revolutionise the way that software is created, and patterns are recognised.

2 Download

You can download a `.tar.gz` file of the entire system (plain `c` machine and interpreter - with debugger, translation scripts - in the `tr/` folder and example scripts in the `eg/` folder) Current download url: `HTTPS://SOURCEFORGE.NET/PROJECTS/BUMBLE/` The `README.txt` contains instructions for compiling the code. [last update to `tar.gz` file: 13 sept 2021]

I need to `put` this onto a Git host.

3 Documentation

The main documentation file about the machine and language is `/books/pars/pars-book.txt` (somewhat disorganised). An **html** version of that document can be seen at `/books/pars/pars-book.ht` which is generated by the "`pep`" script `/books/pars/eg/mark.html.pss` . Also the file "`pep.c`" `/books/pars/object/pep.c` source code contains documentation. The executable file /books/pars/pep also contains documentation about the machine which can be accessed by using the help commands in interactive mode (the -I switch). For example, the "*Com*" command in interactive mode, lists and describes all machine commands.

4 History

19 aug 2022

Made a magical interpret() method in the ***`perl`*** translator which will allow running of scripts.

Working on a simplified grammar for `tr/translate.perl.pss` which I hope to use in all the translator scripts. So far so good. Also introducing a new expression grammar for tests eg

```
(B"a",B"b").E"z" { ... }
```

This allows mixing AND and OR logic in tests. Also, a `pep` script that extracts all unique tokens from a script would be useful.

17 aug 2022

Looking at ANTLR example grammars, for ideas of simple languages such as "logo", "abnf", "bnf", "lambda", "tiny basic" Reforming grammars of the translators, writ-

ing good "*unescape*" and "*escape*" functions that actually walk and transform the workspace string. Converting ***perl*** translator to a parse method Need an "*esc*" command to change the escape char in all translators. The ***perl*** translator is almost ready to be an interpreter.

13 august 2022

Debugged the ***tcl*** translator- appears to be working well except for second generation scripts.
current tasks: finish translators, perl/c++/rust/tcl start translators: lisp/haskell/R (maybe) Write a new command "*until*" with no arguments.(done in some translators) Make the translators use a "*run*" or "*parse*" method, which can read and write to a variety of sources. Make the tape in `object/pep.c` dynamically allocated. See if begin { ++; } create space for a variable. And use this strategy for variable scope.

28 july 2022

Starting to create date-lists in `eg/mark.latex.pss` to render lists such as this one. Also, had the idea of a new test

```
F:file.txt:"int" { ... }
```

This would test if the file "`file.txt`" contain a line starting with "*int*" and ending with "*:*" + workspace. This test would allow checking variable types and declarations. It would also allow better natural language parsing, because a list of nouns/adj/verbs etc could be stored in a simple text file and looked up. Also, variable scope could be included in the file eg

```
int.global:x
int.fn:x
string.global:name
string.local:name
etc
```

Also, another test

```
F:name.txt: { ... }
```

Would check the file `name.txt` for a line which begins with the tape and ends with the workspace.

21 july 2022

A lot of work on the javascript translator `tr/translate.js.pss` 1st gen tests are working. Working on the ***rust*** translator and the `eg/sed.tojava.pss` translator.

13 july 2022

new ideas: create a lisp parser, create a brainf*** compiler (done) create a "*commonmark*" ***markdown*** translator. This should be not too hard, using the ideas in `eg/mark.latex.pss` will create a 'date list' format for `mark.latex.pss` and `mark.html.pss`

7 july 2022

Started a lisp parser `eg/lisp.pss` Worked on `eg/mark.latex.pss` which is now producing reasonable **pdf** output (from `.tex` via pdflatex). Also realised that the accumulator could be used to simplify the grammar by counting words.

5 july 2022

Developed a `sed` to ***java*** script, "`eg/sed.tojava.pss`" which has progressed well. Still lacking branching commands and some other `gnu sed` extensions.

30 june 2022
wrote a simple `sed` parser and formatter/explainer at `eg/sed.parse.pss` (commands a,i,c not parsed yet).

24 june 2022
Some work on the javascript and *perl* translators.

18 june 2022
Introducing an 'increment' method into the various machine classes in the target languages. This allows the 'tape' and 'marks' arrays to grow if required.

17 june 2022
Looking at translation scripts. Changing tape and mark arrays to be dynamically growable in various target languages.

14 sept 2021
reviewing documentation, tidying.

9 sept 2021
Working on the pl/0 scripts. `eg/plzero.pss` and `eg/plzero.ruby.pss` `eg/plzero.pss` now checks and formats a valid pl/0 program.

4 sept 2021
Working on the palindrome scripts `eg/pal.words.pss` and `eg/palindrome.pss` . Both are working well and can be translated to various languages (go, ruby, python, c, java) I would like to `add` hyphen lists to `mark.latex.pss` and date lists (such as this one)

28 aug 2021
Go translator now working well. I would like to write a translator for the Kotlin, R (the statistical language), *swift* rust. The script function pep.tt (in helpers.pars.txt) greatly helps debugging translation scripts.

20 aug 2021
More progress. A number of the translation scripts are now quite bug free and can be tested with the helper function pep.tt <langname>This script also tests 2nd generation script translation, which is very useful where the original `pep` engine is not available (for example, on a server).

15 july 2021
Continuing work. Starting many translation scripts such as `tr/translate.cpp.pss` and trying to debug and complete others.

14 july 2021
working on `tr/translate.c.pss` good progress. simple scripts translating and compiling and running. Did not eliminate dependencies so that scripts need to be compiled with `libmachine.a` in the `object/` folder.

5 july 2021
working on the Ruby translator in `tr/translate.ruby.pss` Should try to make a 'brew' package with *ruby* for pep.

17 june 2021
Some work on the Makefile. renamed `gh.c` to `pep.c` Made `pep` look for `asm.pp` in the current folder or else in the folder pointed to by the "ASMPP" environment variable. Need to `add` "*upper*" "*lower*" and "*cap*" to the translation scripts in `pars/tr/`

15 june 2021
things done:

- ⋆ implemented "*nochars*" "*nolines*" "*upper*" "*lower*" "*cap*" (capital case for

workspace) in machine.interp.c. nochars and nolines are already in a number of translation scripts.
- ⋆ clean up the pars folder (get rid of stray `gh.c` files etc).
- ⋆ fixed the `add` "\\" bug which was cause by a bad implementation of until in `machine.interp.c` (need to count preceding escape chars) Need to fix the same in the translation scripts

Here are some immediate tasks to make the `pep` engine more complete.

- ⋆ write a "make configure" script to install `pep` somewhere
- ⋆ fix up the website at `WWW.PEPTOOL.ORG` and include some docs there
- ⋆ try to write an **html** translator for the commonmark spec and contact jgm - the pandoc guy to try to generate some interest in pep.
- ⋆ write some code on rosettacode site. (done) Send to linguists.
- ⋆ write a go translator for a modern compiled script engine. (done)
- ⋆ finish *`tcl`* translator

8 june 2021

Have made some more good progress over the last few days. Modified the script `/books/pars/eg/json.check.pss` so that it recognises all **json** numbers.

Fixed `/books/pars/tr/translate.py.pss` so that it can translate scripts as well as itself. Started to fix /books/pars/tr/translate.tcl.pss. Still have an infinite loop when `.restart` is translated, and this is a general problem with the "*run-once*" loop technique (for languages that dont have labelled loops or goto statements, for implementing `.reparse` and .restart). The solution is a flag variable that gets set by `.restart` before the parse>label (see translate.ruby.pss)

The script `eg/mark.latex.pss` is progressing well. It transforms a markdown-ish format (like the current doc) into LaTeX. Need to do lists/images/tables/dates

18 april 2021

Having another look at this system. I still see enormous potential for the system, but dont know how to attract anyones attention! I updated the `eg/json.check.pss` script to provide helpful error messages with line+character numbers. Also, that script incorporates the scientific number format (crockford) in eg/json.number.pss. However, Crockfords grammer for scientific numbers seems much stricter than what is often allowed by **json** parsers such as the "*jq*" utility.

I became distracted by a bootable x86 forth stack-machine system I was coding at `/books/osdev/os.asm` That was also interesting, and I had the idea of somehow combining it with this. Hopefully these ideas will come to fruition.

I think the best idea would be to edit the `/books/pars/pars-book.txt` document, generate a pdf, `print` it out, and send it to someone who might be interested. This parsing/compiling system is revolutionary (I think), but nobody knows about it!!

15 december 2020

I have not done any work on this project since about august 2020 but the idea remains interesting. Finishing the "`translate.c.pss`" script would be good (done: sept 2021), make "`translate.go.pss`" for a more modern audience (done: sept 2021).

27 august 2020

Working on the script "`translate.c.pss`" to create `c` code from a `pep` script. I may try to eliminate dependency files and include all the required structures and functions in the script. That should facilitate converting the output to wide chars

"wchar".

11 august 2020

Ideas: write a `bash` script to test each script translator (such as `translate.tcl.pss` `translate.java.pss` ....) [done: the pep.tt function]

In the *`java`* translator, make the parse/compile script a method of the class, with the input stream as a parameter. So that the same method can be used to parse/-compile a string, a file, or stdin, among other things.

This technique can be used for any language but is easier with languages that support data-structures/classes/objects.

7 august 2020

Continuing to work on the scripts `translate.py.pss` and translate.tcl.pss. Had the idea to split the `pars-book.txt` into separate manpages just like the *`tcl`* system "man 3tcl string" etc.

24 july 2020

Made great progress on the script "`translate.java.pss`" which could become a template for a whole set of scripts for translating to other languages.

23 july 2020

continuing to work on `translate.java.pss` Still need to convert the push/pop code and test and debug. Many methods have been in-lined and the Machine class code is now in the script.

22 july 2020

Rethinking the translation scripts `/books/pars/tr/translate.java.pss` `/books/pars/tr/tran`
These scripts can be greatly simplified. I will remove all trivial methods from the Machine object and use the script to emit code instead. Hopefully `translate.java.pss` will become a template for other similar scripts. Also, I will include the Machine object within the script output so that there will be no dependency on external code.

20 july 2020

Wrote the script `/books/pars/eg/json.number.pss` which parses and checks numbers in **json** scientific format (Eg -0.00012e+012) This script can be included in the script `eg/json.parse.pss` to provide a reasonable complete **json** parser/checker.

3 july 2020

Working on the script `/books/pars/eg/mark.html.pss` The script is working reasonably well for transforming the `pars-book.txt` file into html. It can be run with:

```
pep -f eg/mark.html.pss pars-book.txt > pars-book.html
```

15 june 2020

Cleaning up the files in the `/books/pars/` folder tree. Renaming the executable to "`pep`" from "pp". I think "`pep`" will be the tools definitive name.

14 june 2020

I will rename the tool and executable to "`pep`" which would stand for "parsing engine for patterns". I think it is a better name than "*pp*" and only seems to conflict with "python enhancement process" in the unix/linux world.

Wrote a substantial part of the script `/books/pars/eg/json.parse.pss` which can parse and check the **json** file format. However, the parser is incomplete because at the moment it only accepts integer numbers. Recursive object and array parsing is working.

I will try to improve the `mark.html.pss` "*`markdown`*" transform script. I would

still like to promote this parsing VM since I think it is a good and original idea.

23 august 2019

Did some work on `mark.html.pss`

20 august 2019

Cleaned up memory leaks (with valgrind). Also some one-off errors and invalid read/writes. The double-free segmentation fault seems to be fixed. Still need to fix a couple of memory bugs in interpret() (one is in the UNTIL command).

17 august 2019

Trying to clean up the `pars-book.txt` file which is the primary documentation file for the project.

Posted on comp.compilers and `comp.lang.c` to see if anyone might find this useful or interesting...

16 august 2019

The implementation at `HTTP://BUMBLE.SOURCEFORGE.NET/BOOKS/PARS/OBJECT` has arrived at a usable beta stage (barring a segmentation fault when running big scripts).

22 February 2015

(approximately)

Started the current implementation in the `c` language. I created a simple loop to test each new command as it was added to the machine, and this proved a successful strategy as it motivated me to keep going and debug as I went.

2009

Wrote an incomplete `c` version of this machine called "chomski".

2006 - 2014

Wrote incomplete versions in *c++* and java. The *java* Machine object at `/books/pars/object.java/` got to a useful stage and will be a useful target for a script, very similar to `/books/pars/tr/translate.c` (and will be called "`translate.java.pss`"). This script creates compilable *java* code using the *java* Machine object. In fact, we will be able to run this script on itself (!). In other words we can run:

```
pep -f tr/translate.java.pss tr/translate.java.pss
```

The output will be compilable *java* code that can compile any parse machine script into compilable *java* code. Having this *java* system we are able to use unicode characters in scripts.

It will be interesting to see how much slower the *java* version is.

2005

Started to think about a tape/stack parsing machine.

5 Roadmap

I am keen to try to publish this language and idea further, because I think that it has great potential. Here is a list of things which I will try to do, to make the system more credible.

6 Tasks

- ⋆ write a `pep` script that extracts all unique tokens from script.
- ⋆ write script that transforms `pep` to BNF ignoring attributes.
- ⋆ finish translation scripts for rust/haskell/c++ ...
- ⋆ `add` an exit code to quit;
- ⋆ make "*w*" take a filename argument
- ⋆ escape should escape all chars in string eg "escape '${}'"
- ⋆ fix the escape and unescape code in `machine.c`
- ⋆ `get` a domain name such as peptool.org, (maybe peplang.org or pepnomlang.org or nomlang.org) [done]
- ⋆ `add` a list syntax to `/books/pars/eg/mark.html.pss` and `mark.latex.pss`
- ⋆ Also a definition list syntax. So a paragraph with lines starting with 'o-' or d/- u/- [done]
- ⋆ collect some artwork/screenshots/diagrams etc to go into the 'pars-book.txt' documentation file.
- ⋆ convert `mark.html.pss` into a version that generates LaTeX, [done] (eg/mark.latex.pss)
- ⋆ comprehensively edit and proof-read the 'parse-book.txt' file
- ⋆ using `mark.latex.pss` create a **pdf** version of the booklet.
- ⋆ `print` and bind a limited edited of the booklet. Send it to people who may be interested.
- ⋆ work on all the translation scripts so that I can translate scripts into many other languages (and so, support unicode) [aug 2022: java/go/ruby/python/c/tcl/js done. `perl/rust/` etc need to be debugged]
- ⋆ solve the problem of attribute grammars, how do we do type checking and variable definition validity checks? There are a number of possible solutions, include a string 'type' stack which works just like the token stack (but with no accompanying tape array). Another solution is just to use the "*mark*" and "*here*" commands to check tape cells. But we need a test that checks if the tape is contained in the workspace or vice-versa.

7 Bugs

There appears to be a problem in growProgram in `program.c` called by `machine.c` Throwing a seg fault.

8 Compiling the code

In the `object/` folder there is a Makefile which can be used to compile the `c` interpreter code.

The file `/books/pars/helpers.pars.sh` contains `bash` functions to compile the `c` source code. The most important `bash` functions are

peplib() which compiles the object `c` files into a static library `/books/pars/object/libmachine.a` (for linking into executable compiled scripts)

ppco() compiles all `c` source files into the executable "*pp*"

ppcl() compiles standalone executable scripts generated by `compilable.c.pss`

ppjjff() compile or translate a script into *java* which can be run with the code in `object.java`

9 Important files

`/books/pars/object/*.c` implementation of the machine and program `c` objects

`/books/pars/object/pep.c` implementation of the interactive script interpreter and debugger This version uses only plain 8 bit characters (char). However this problem can be overcome by using a translation script such as `translate.java.pss` into a language which supports unicode.

`/books/pars/compile.pss` implementation (compiler) of the script language in the script language itself. This was originally "*bootstrapped*" by `ar.compile/asm.handcode.pp`

`/books/pars/asm.pp` implementation of the script language in "*assembler*" format This is now generated from from the `compile.pss` script by running

```
pep -f compile.pss compile.pss > asm.new.pp; cp asm.new.pp asm.pp
```

The original "*bootstrap*" script compiler can be seen at `/books/pars/ar.compile/asm.handcode.pp`

`/books/pars/helpers.pars.sh` various `bash` functions to run and compile the `c` code and scripts.

`/books/pars/tr/translate.java.pss` a script which generates compilable *java* code for any script (including itself). This script shows great potential but needs to be more completely debugged (as of 25 july 2020)

`/books/pars/eg/` some `pep` scripts which demonstrate uses of the language and virtual machine.

`/books/pars/eg/exp.tolisp.pss` A script which converts arithmetic expressions into a lisp-like format

`/books/pars/eg/natural.language.pss` a very simple and limited natural language (english) recogniser.

`/books/pars/eg/mark.html.pss` Converts a "mark-down"-like text document format into **html** This is used to generate the file "`pars-book.html`"

`/books/pars/eg/json.parse.pss` A script that recognisers and checks a subset of the **json** format (only integer numbers recognised until I integrate the script `json.number.pss` into it. This can be translated to (for example) *java* and executed with

```
pep -f translate.java.pss eg/json.parse.pss > Machine.
  ⇒ java
javac Machine.java
echo "[1,2,[0,0],{'name':'bob', 'age':22}]" | java
  ⇒ Machine
```

Or it can be executed directly with

```
pep -f eg/json.parse.pss -i "[1,2,[0,0],{'name':'bob', 'age':22}]"
```

10 Attribute grammars and pep

The problem of "*attribute*" grammars is an important one, and needs to be solved in order to make `pep` a viable option for compiling computer languages. Let us say that gender, or number are attributes of adjectives or nouns. Also, the type of a variable or expression is an attribute of that expression.

These attributes need to "*agree*" when tokens are resolved/reduced: that is "los mujeres" is grammatically incorrect because "*los*" has a masculine attribute and "*mujeres*" has a feminine attribute.

The solution may be a type stack with an item on the stack for each "*scope*" (procedure, subprocedure etc). No, a fake stack can be made in a tape cell.

11 Changes that need to be made

Make the following commands:

```
'w "name.txt"; write the workspace to the file name
W "name.txt"; append the workspace to the file "name.txt"
W; append the workspace to the file name in the tape cell.
q 4; exit with code 4
```

Maybe organise better the `c` code: the struct Program could be removed as a member of the struct Machine.