NonCFG– A research language

# Idea

Ideas to research:

* containers (contained execution for modular yet secure execution)
* grammars
  + non context free grammars
  + simple grammars based on separators (in general in functional charater classes)
  + loose grammar
  + macros, meta, reflection

Here we study non context free grammars. More specifically we assume a general CFG for base language and concrete programs with named variables, functions as non CFGs. The research question:

* Can we describe the base language as an abstract CFG (kinda template language) of which concrete program are/generates CFG instances? That is we try to consider programs having their own CFG grammar which is an instance of the more general base language grammar.
* More specifically we only study names (declarations and references) which are basic language constructs. We study names in a simple manner:
  + first with no namespaces, then
  + with simple namespaces such as occuring in import statements which support aliases such as in Python (import … as …)
  + we use a simple procedural language (no OO yet, where name references can happen through variables (p = Person(…); name = p.name))

Add sample

# Basic language

* **first order**
  + **values - CFG**
    - literals
    - primitive values: boolean, numeric, string
    - sets, lists, tuples (aka structs)
  + **variables – non CFG in classic approach**
    - declaration
    - reference
    - assignment
  + **operators**
* **second order**
  + **functions – non CFG in classic approach**
    - declaration
    - reference
  + **function variables – non CFG in classic approach**
    - assignment
* **modules**
  + import

# First order

**values**

true, 1, "Hello world", [1,2,3]

**variables**

b = true

x := 1

text = "Hello world"

l = [1,2,3] # list with 3 elements

s = {1,2,3} # set with 3 elements

t = (1,2,3) # 3 dim vector

**operators**

=, +, \*, stb

y = 1 + 2

z := x + 1

z == x + 1

v = x < z # always true

0 > l # [0,1,2,3]

l < 4 # [0,1,2,3,4]

# Second order

TODO

# Modules

TODO

# Machine

* A single program machine hat can run one program at a time
* A single processor machine with linear execution
* Memory is fully used to the program (since only one program runs, memory is private to it)
* A console to talk to he outside world.
* No IPC

Design

* Memory
  + set
  + get
* Processor
  + execute
  + compute
* CLI
  + print
  + read

|  |  |
| --- | --- |
| **Program** | **Memory** |
| x = 1 | first snapshot  mem[0.1, x] = 1  cli[0.0] = [] |
| y = 0 | mem[0.1, x] = 1  mem[0.2, y] = 0  cli[0.0] = [] |
| x = y | new snapshot (mem)  mem[1.0, y] = s[0.2] = 0  mem[1.1, x] = s[0.2] = 0  cli[1.0] = cli[0.0] |
| print(x) | new snapshot (cli)  mem[2.0, y] = s[1.0] = s[0.2] = 0  mem[2.0, x] = s[1.1] = s[0.2] = 0  cli[2.0] = [0] |

modes:

* interpreted
* transpiled (e.g. to Python)
* machine code
  + internal ABI
  + OS/C ABI