Autoconstructive Evolution: Push, PushGP, and Pushpop

Lee Spector
Cognitive Science
Hampshire College
Amherst, MA 01002

lspector@hampshire.edu
http://hampshire.edu/lspector

Overview

Autoconstructive Evolution, self-construction of the evolutionary process

The **Push** programming language for evolutionary computation

PushGP, a genetic programming system that evolves Push programs

Pushpop, an autoconstructive evolution system that evolves Push programs

	Results	Potential (?)
Push	* (large)	* (small)
PushGP	*	*
Pushpop	* (small)	* (large)

Autoconstructive Evolution

....

Individuals make their own children.

The machinery of reproduction and diversification (and thereby the machinery of evolution) evolves.

Radical self-adaptation.

Making a living,
Making babies

....

Individuals, like natural organisms, must **both** make a living in the world and produce offspring.

....

Making a living = performing well on a environment/problem-specific fitness test.

Producing offspring = generating code.

Children

Produced as output from parents' code.

Problem-solving and child-producing code may be integrated and interdependent.

May use arbitrary computational processes built in an expressive, Turing-complete programming language.

Hypotheses

Autoconstructive evolution systems can be valuable sources of data on the nature of life and evolution.

Autoconstructive evolution systems can out-perform traditional evolutionary computation systems by adapting their reproductive mechanisms to their representations and problem environments.

Advance praise for the Push programming language

....

Multiple data types with ***no*** *constraints on code generation or manipulation!* (Compare to Strongly Typed Genetic Programming [Montana].)

Arbitrary modularity with ***no*** *constraints on code generation or manipulation!* (Compare to Automatically Defined Functions [Koza] or Automatically Defined Macros [Spector].)

You'll ***never*** *need to pre-specify the module architecture!* ***No*** *extra machinery required for architecture evolution!* (Compare to Architecture-Altering Operations [Koza].)

Explicit and arbitrary recursion? ***No problem!*** (in principle) (Compare to the work of Yu and others)

Ontogenetic development, evolved adaptivity, and diversifying self-replication? ***Push makes it easy!*** (in principle) (Compare to Ontogenetic Programming [Spector], TIERRA [Ray], Avida [Adami].)

The Push programming language for evolutionary computation

....

Goals:

- multiple data types
- modularity
- Turing completeness
- recursion
- code manipulation
- **uniform syntax**

Push

Stack-based, like Forth or Postscript

Multiple stacks, one for each type

Types are hierarchical

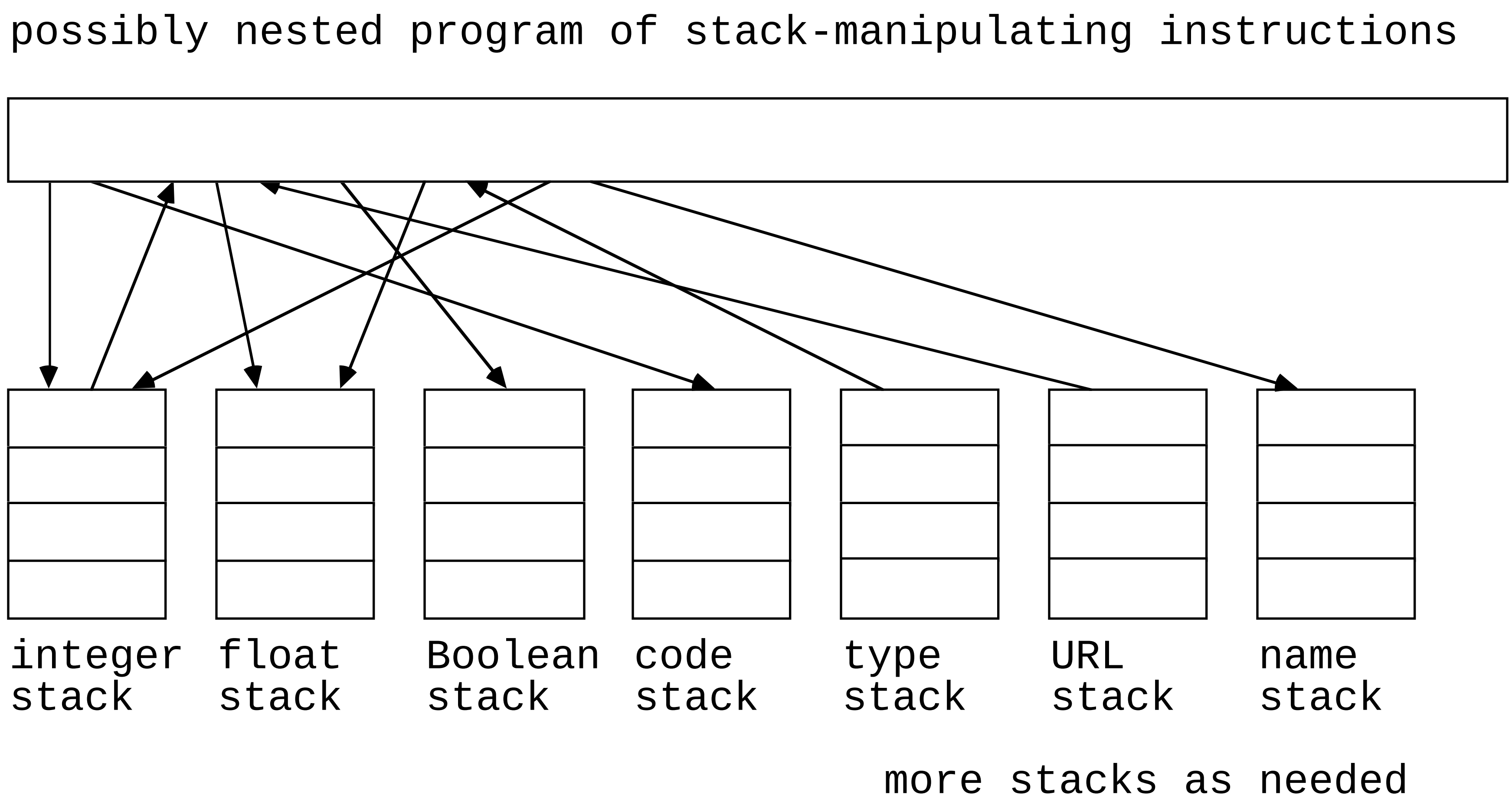
Type constants on a **type** stack/bottom

Missing argument? NOOP

Code type/stack -> advanced features

Runtime resource limits

Push architecture



Push examples

```
(integer 2 3 +)

(integer 2 3 + float 2.72 3.14 +)

(2 3 2.72 3.14 integer + float +)

(2.72 integer 2 3.14 3 + float +)

((integer) (2 (3)) +)

(code quote (integer 2 3 +) do)
```

Factorial in Push

```
(quote (pop 1)
 quote (code dup
        integer dup
        1 - do *)
 integer dup 2 < if)
```

Factorial with Names

```
(code
  quote (quote (pop 1)
         quote (integer dup 1 -
                code factorial get do
                *)
         integer dup 2 < if)
  factorial set
  factorial get do)
```

The Push type hierarchy

```
- push-base-type: dup, pop, swap, rep, =[boolean],
                  set[name], get[name], convert[type],
                  pull[integer], noop
 - number: +, -, *, /, >[boolean], <[boolean]
   - integer: rand, pull, /
   - float: rand
 - boolean: not, and, or, nand, nor, rand
 - expression: quote, car, cdr, cons, list, append, subst,
               container, length[integer], size[integer],
               atom[boolean], null[boolean], nth[integer],
               nthcdr[integer], member[boolean],
               position[integer], contains[boolean],
               insert[integer], extract[integer],
               instructions[type], perturb[integer],
               other[integer], other-tag[float],
               elder[integer], neighbor[integer],
               rand[integer]
   - code: do, do*, if[boolean], map
   - child:
 - type: rand
 - name: rand
```

Inheritance, multi-stack access, subsets

PushGP: GP for Push programs

≈ Standard Koza-style GP but evolves Push programs

Uniform code generation

Crossover: expression swapping or uniform crossover on terminals

Mutation: replacement, perturbation

Tournament selection.

Networked on a 16-node cluster.

PushGP: symbolic regression

[Robinson, 2001]

....

Cases	50 from $x^6-2x^4+x^2$
Popsize	4000
Max Gens	51
Max Length	50 points
Instr Set	+ - * / dup ERC
Operators	90% xover, 10% duplication
Input	x value on integer stack
Fitness	Sum of error for all cases
Termination	error for each case < 0.01

One result:

```
(dup (dup dup * (*) - /) dup *)
```

PushGP: multiple data types and the ODD problem

The ODD problem: Is a given integer odd? (integer->Boolean)

An odd solution:

```
((nth) atom (insert) pull)
```

Uses its own code as an auxiliary data structure.

Multiple data types can sometimes be used to synergistic advantage.

PushGP: PARITY and modularity

....

The EVEN PARITY problem: Is the number of "on" bits in the input even? (Boolean->Boolean)

EVEN PARITY can be decomposed into smaller parity problems; ADFs provide advantages [Koza].

PushGP: solution (simplified) to EVEN 4-PARITY

....

```
(quote
 (x x (x ((x) x)))
 (list
  (x)
  ((x) (x quote (dup nand) if) nil)
  (x x)
  ((quote) ((x) x x) x (map nor))))
```

Modular?

Recursive (via map)

Heavy code re-use (but not "human-style"!)

PushGP results, continued

Initial data on PushGP and larger parity problems: scaling of difficulty compares favorably with ADFs.

EVEN N-PARITY for bounded (but not yet unbounded) *N*.

Data/variants/comparisons to literature: Alan Robinson’s thesis [Robinson, 2001]; also [Spector and Robinson, in preparation].

Real interest lies in application to new kinds of problems.

PushGP: current work

....

Unbounded recursion (e.g. factorial)

Seek human-competitive results in:

- quantum computation
- integer sequence induction
- agents in virtual worlds

Pushpop: autoconstructive evolution of Push programs

....

Children: at the end of each program execution the top of the “child” stack is a potential child.

Selection: The children of the better parents are more likely to survive.

Sex: Access to code of other programs, for execution and/or reproduction. Access based on geography, fitness, and/or genetics. Any number of "genders" is possible.

Pushpop: Diversity management

Extreme measures are required.

Syntactic diversity: no clones

Semantic diversity:

- Limit number of children from identically-performing parents
- Vary fitness components geographically

Reproductive competence

Pushpop: Results

....

Reproductive competence is easily achieved.

Fitnesses generally improve.

Simple problems can be solved.

Evolutionary mechanisms evolve.

Some emergent features resemble those of natural and/or engineered self-adaptive systems, for example in the dynamics of reproductive strategies

Fitness-improvement often stagnates.

What is necessary for the emergence of robust, fitness-progressive evolution?

....

Hypotheses under exploration:

Spatial irregularity (neighbors, local climate, local problems)

Environmental dynamism (comets, seasons, climate change, cooling of the universe)

Thermodynamic constraints (information budgets)

Relations to TIERRA/Avida

Problem-solving orientation

Higher-level language

Fully endogenous diversification

Summary/Conclusions

Push supports novel evolutionary computation paradigms.

PushGP evolves Push programs to solve many types of problems. Modularity and other advanced programming features arise naturally.

Pushpop is an autoconstructive evolution system in which Push programs solve problems while constructing their own children and thereby their own evolutionary mechanisms.

Cliffhangers

....

Can Pushpop fulfill the hypothesized promise of autoconstructive evolution systems to out-perform traditional evolutionary computation systems by adapting their reproductive mechanisms to their representations and problem environments?

Can Pushpop fulfill the hypothesized promise of autoconstructive evolution systems to provide useful data on the nature of life and evolution?

Come to GECCO-2002 (New York) to find out! (or maybe GECCO-2003 or 4 or...)