Evolutionary Program Sketching

Iwo Błądek and Krzysztof Krawiec
{ibladek|kkrawiec}@cs.put.poznan.pl
Institute of Computing Science, Poznan University of Technology, Poland

Overview

- Evolving partial programs with *holes* (sketches) [1].
- Evolution responsible for program *structure*, SMT solver fills in the gaps with short code pieces.
- This paper: single-instruction holes.
- Improves over standard GP and works particularly well for constants.

https://github.com/iwob/EPS

SMT solving (Satisfiability Modulo Theories)

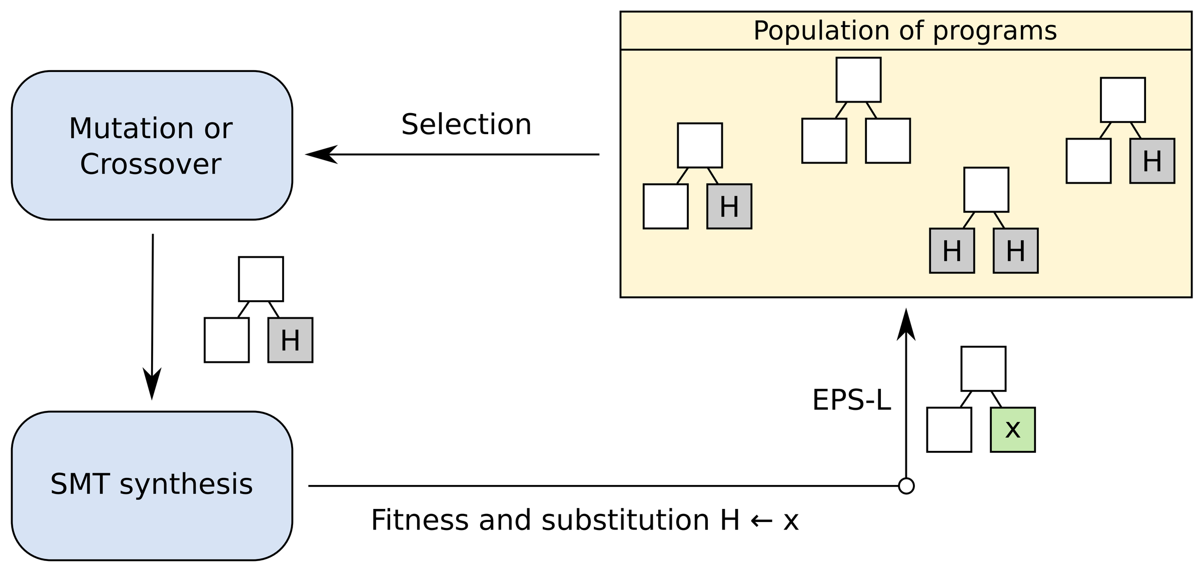
- Checking if a formula is satisfiable under a certain theory T.
- *Theory* describes mathematical objects and semantics of functions that operate on them (e.g. integer numbers and $+, -, *, \div$, *mod*) and includes decision procedures for reasoning.
- *Model* contains valuation of variables which make the formula satisfiable.

Examples:

$(10 \cdot x = 20) \wedge a$	$(x < y) \wedge (y > x)$	$x^2 + 1 \leq 2 \cdot x$	$\forall_{x,y}\ (x+y)^2 > x^2 + y^2$
SAT $x = 2,\ a = true$	UNSAT	SAT $x = 1$	UNSAT

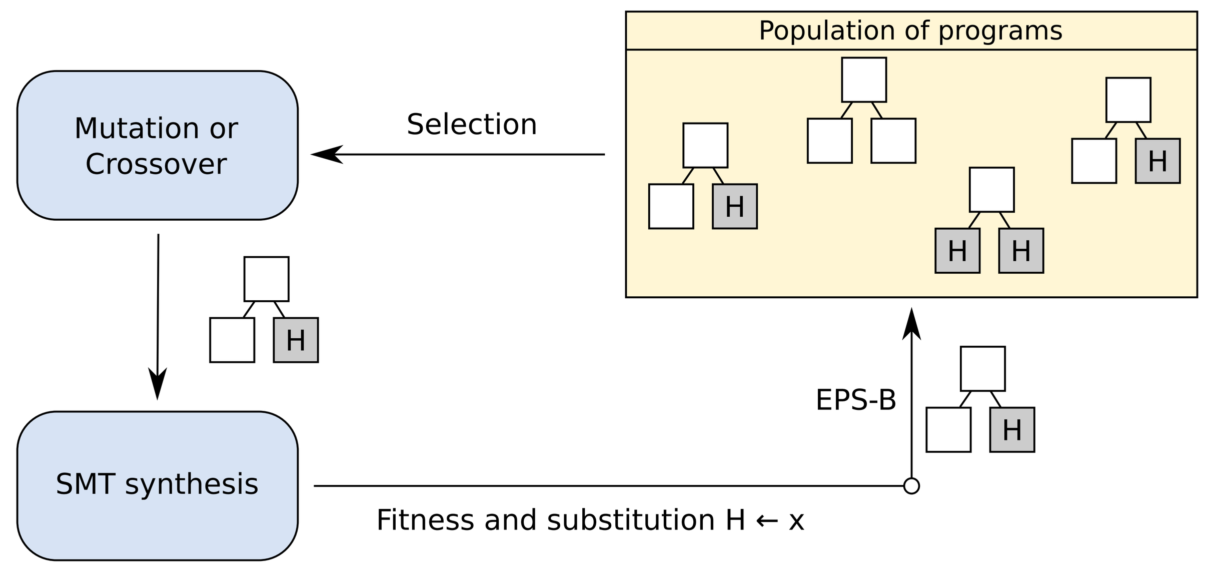
EPS-L ("Lamarckian" EPS)

Holes are filled with the content discovered by the solver.
New holes may be introduced only by mutation.



EPS-B ("Baldwinian" EPS)

Holes remain in the program.
Content discovered by the solver is used only for computing fitness.



Evaluation in EPS

Sketch of the program
$(int_0 \cdot x) \cdot (int_1 \cdot x) \cdot int_2$

Target function
$x^2 - 1$

Benchmarks (x, out)
$(-3, 8), (-2, 3), (-1, 0), (0, -1), (1, 0), (2, 3), (3, 8)$.

⇒

SMT-LIB script

```
(set-option :produce-assignments true)
(set-logic QF_NIA)
; Free variables
(declare-fun int0 () Int)
(declare-fun int1 () Int)
(declare-fun int2 () Int)

; ------------------------------------------------------------------
; CONSTRAINTS FOR ALL TEST CASES:
; ------------------------------------------------------------------

; DECLARATIONS: Local variables (T0)
(define-fun x_T0 () Int -3)
(declare-fun res_T0 () Int)
; PROGRAM (T0)
(assert (! (= res_T0 (* (* (* int0 x_T0) (+ x_T0 int1)) int2))
        :named T0_p0))
; POSTCONDITION (T0)
(define-fun itest0 () Int (ite (! (= res_T0 8)
                                :named pass_itest0) 1 0))
; ------------------------------------------------------------------
... (other test cases) ...
; ------------------------------------------------------------------
; DECLARATIONS: Local variables (T6)
(define-fun x_T6 () Int 3)
(declare-fun res_T6 () Int)
; PROGRAM (T6)
(assert (! (= res_T6 (* (* (* int0 x_T6) (+ x_T6 int1)) int2))
        :named T6_p0))
; POSTCONDITION (T6)
(define-fun itest6 () Int (ite (! (= res_T6 8)
                                :named pass_itest6) 1 0))
; ------------------------------------------------------------------

(declare-fun fitness () Int)
(assert (= fitness (+ itest0 ... itest6)))
(maximize fitness)(check-sat)
```

⇒ **SMT solver** Z3 ⇒

Model and fitness

```
sat
(objectives
  (fitness 2)
)
((int0 0))
((int1 1))
((int2 (- 1)))
```

⇒

EPS-L
$(0 \cdot x) \cdot (1 \cdot x) \cdot (-1)$

EPS-B
$(int_0 \cdot x) \cdot (int_1 \cdot x) \cdot int_2$

Experiment configuration

Evolution parameters:

- Runs per configuration: 100
- Number of generations: 100
- Population size: 250
- Mutation (0.5) and crossover (0.5)
- Solver timeout [ms]: 1500

Tested variants:

- GP: standard GP.
- GP_T: GP with time limit similar to EPS.
- GP_{5000}: GP with population size 5000.
- c: holes can be filled with constants.
- v: holes can be filled with input variables.
- cv: both constants and variables.

Results

Number of optimal solutions found (per 100 runs)

	GP			EPS-L			EPS-B		
	GP	GP_T	GP_{5000}	c	v	cv	c	v	cv
Keijzer12	0	0	5	0	0	1	39	1	0
Koza1	19	68	96	33	-	32	100	-	100
Koza1-p	0	0	0	5	-	3	100	-	100
Koza1-2D	1	12	20	2	0	11	80	21	23
Koza1-p-2D	0	0	0	0	0	1	75	0	0

Average runtime [s]

	GP			EPS-L			EPS-B		
	GP	GP_T	GP_{5000}	c	v	cv	c	v	cv
Keijzer12	15	11331	493	772	488	1579	15440	21173	28354
Koza1	5	291	46	700	-	801	652	-	696
Koza1-p	5	963	344	892	-	972	978	-	982
Koza1-2D	16	7636	432	793	479	1791	9077	16281	23034
Koza1-p-2D	15	9206	515	750	511	1726	11986	12391	27875

Acknowledgments

Grant 2014/15/B/ST6/05205 funded by the National Science Centre, Poland.

References

[1] Armando Solar-Lezama. "Program sketching". In: *International Journal on Software Tools for Technology Transfer* 15.5 (2013), pp. 475–495.