Simultaneous Synthesis of Multiple Functions using Genetic Programming with Scaffolding

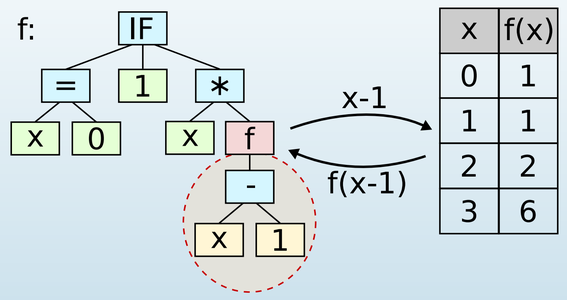
Iwo Bladek and Krzysztof Krawiec
{ibladek|kkrawiec}@cs.put.poznan.pl
Institute of Computing Science, Poznan University of Technology, Poland

Overview

- **Scenario:** several functions to be synthesized, potentially dependent on each other (*multisynthesis*).
- **Sequential** and **parallel** approaches to multisynthesis.
- A novel application of *scaffolding* to calls to other functions being synthesized.
- Computational experiments prove **parrallel multisynthsis more effective**.

Scaffolding

A technique devised originally to facilitate evolution of recursive programs [1].



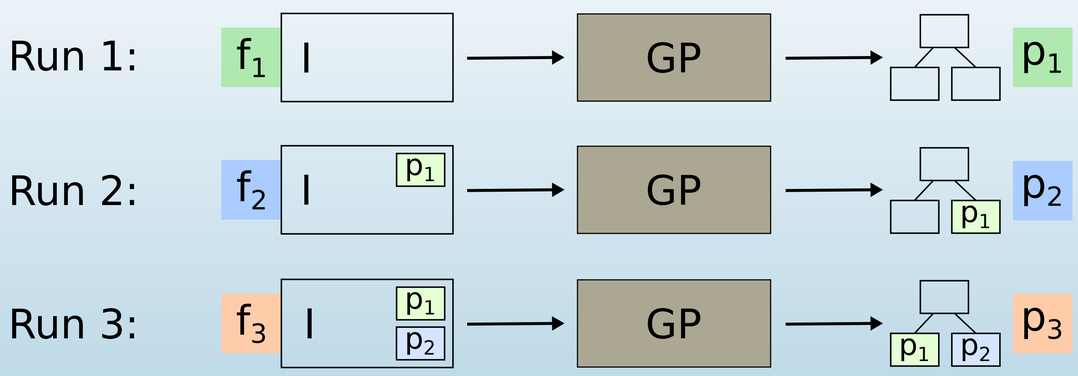
x	f(x)
0	1
1	1
2	2
3	6

The principle: If the argument of a recursive call occurs on the list of fitness cases, return the associated output (rather than recursively calling the program).

Key observation: Scaffolding can also be used in *multisynthesis*.

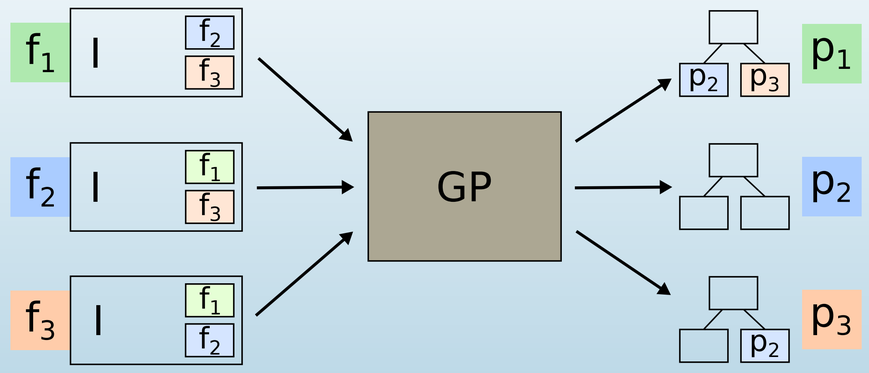
Sequential Synthesis (SEQ)

Functions are synthesized in an order provided by the user. Subsequent runs add to instruction set all previously synthesized programs (I – the common instruction set).



Parallel Synthesis (PAR)

Functions are synthesized simultaneously. Each function may call other functions, but cyclical calls are not allowed. GP is expected to discover potential dependencies between the functions being synthesized.



Benchmarks and Program Representation

- **Instruction set:** a subset of the Scala programming language.
- **Benchmarks:** various generic methods of the *List* class. Each benchmark has three functions to be synthesized:
  - one dependent function,
  - two helper functions.

Benchmark	Dependent function	Helper functions
`last`	`last(list:List[T]):T`	`at,size`
`patch`	`patch(list:List[T],d:Int,p:List[T],u:Int):List[T]`	`drop,take`
`slice`	`slice(list:List[T],d:Int,u:Int):List[T]`	`drop,take`
`splitAt`	`splitAt(list:List[T],i:Int):(List[T],List[T])`	`drop,take`

- **Program representation**: purely functional; only function calls, immutable values and constants.
- **Solution:** a triple of programs.

Exemplary synthesized solution:

```
splitAt:  tuple(take(list, i), drop(list, i))
drop:     slice(list, n, *(7, 4))
take:     takeRight(reverse(takeRight(reverse(list), n)), n)
```

Configuration

- Population size: 500
- Evaluation budget: 150000
- Mutation (0.5) and crossover (0.5)
- Runs per variant: 25

Tested variants:

- $*_{opt}$: optimal ordering of functions.
- $*_{exp}$: randomized ordering of functions.
- $*_{nsga}$: NSGA2 selection with 3 criteria: numbers of tests passed for each target function.
- $*^{s}$: scaffolding with an oracle used for calls to other functions.

Results

- **The table:** average numbers of synthesized functions in best-of-run programs.
- **Conclusions:** parallel synthesis gave on average better results on the number of correctly synthesized functions than SEQ_{exp}.
- The presence of multiple contracts (constraints) changes the fitness landscape.

Method	`last`	`patch`	`slice`	`splitAt`	Average
SEQ_{opt}	2.23	2.13	1.36	2.69	2.10
SEQ_{exp}	1.11	1.73	0.97	1.56	1.34
PAR	2.00	1.50	0.72	1.80	1.51
PAR^{s}	1.90	1.47	0.59	1.94	1.48
PAR_{nsga}	2.03	1.47	0.53	2.17	1.55
PAR^{s}_{nsga}	2.18	1.40	1.05	1.88	1.63

Acknowledgments

Work supported by grant 2014/15/B/ST6/05205 funded by the National Science Centre, Poland.

References

[1] A. Moraglio, F. Otero, C. Johnson, S. Thompson, and A. Freitas. Evolving recursive programs using non-recursive scaffolding. In *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, pages 2242–2249, Brisbane, Australia, 10-15 June 2012.