{Setɪy}: Optimizing a Set Language — Project Proposal

Adam Blank (adamblan@cs) and Andrey Kostov (akostov@andrew)

Web Page

The project web page can be found at:

`http://countablethoughts.com/setty`

This page will be periodically updated with our progress and any write-ups.

Description

The {Setɪy} Programming Language

{Setɪy} is a toy programming language that Adam created to supplement teaching logic and sets to computer science freshmen. A typical {Setɪy} program looks like the following:

```
1  is_prime(x) := x ≠ 0 ∧ x ≠ 1 ∧ ¬∃(a ∈ [x−1] \ {1}). ∃(b ∈ [x−1] \ {1}). x = a∗b
2  prime_factors(n) := {x | x ∈ [n] ∧ is_prime(x) ∧ ∃(y ∈ [n]). n = x∗y}
3
4  U := 10
5
6  for i to U:
7     print i, "is prime?", is_prime(i)
8
9  for i to U:
10    print i, prime_factors(i)
```

In {Setɪy}, the only primitive type available to users is a set. Natural numbers, pairs, lists, etc. can all be implemented on top of them.

Optimization Opportunities

Set Re-use. Currently, the compiler creates a new set every time the user requests one. So, a {Setɪy} program:

```
1  s := {}
2  t := {}
3  r := {}
4  print s, t, r
```

would create three empty sets. Since sets are not mutated, this is extremely inefficient. For this optimization, we will make the {Setɪy} compiler (1) re-use sets whenever possible and (2) keep track of liveness of sets in preparation for the other two optimizations. Since sets are malloced, this optimization should save a significant amount of both time and space.

Inference of Universe. In many of the high-level constructs (universal, existential, and set comprehension statements), the {Setɪy} compiler needs a universe for any free variables.

Often, the user specifies these directly, but this can be painful (and users often specify larger sets than necessary). The goal of this optimization is two-fold: (1) automatically infer the universe of free variables whenever possible, (2) refine both these predictions and user-specified universes using information from the *set re-use* pass.

Purity Checking. The vast majority of {Set1y} functions are pure. It would save a significant amount of time if we automatically memoized function values upon determining that the function is pure.

Metrics for Evaluation

We plan to test our optimizations by looking at (1) the number of dynamic instructions run as in assignment 3, and (2) the memory footprint of various test cases. Several interesting {Set1y} programs already exist for testing correctness of the interpreter and compiler. As part of the project, we will develop more tests that form a reasonable benchmark for the compiler as well as our optimizations.

Goals

75% Goal. Our 75% goal is to implement **one** of *Set Re-use*, *Optimizing QBFs*, and *Inference of Universe*.

100% Goal. Our 100% goal is to implement **two** of *Set Re-use*, *Optimizing QBFs*, and *Inference of Universe*.

125% Goal. Our 100% goal is to implement **three** of *Set Re-use*, *Optimizing QBFs*, and *Inference of Universe*.

Logistics

Plan of Attack

Week	Tasks	
1	Continue working on the compiler	(**Adam**)
	Review compiler code-base	(**Andrey**)
2	Begin work on set-reuse	(**Adam**, **Andrey**)
3	Finish set-reuse; begin inference of universe	(**Adam**, **Andrey**)
4	Finish inference of universe	(**Adam**, **Andrey**)
	Begin purity checking	(**Adam**, **Andrey**)
5	Finish purity checking	(**Adam**, **Andrey**)
	Begin benchmark suite	(**Adam**, **Andrey**)
6	Finish benchmark suite; fix bugs; reporting	(**Adam**, **Andrey**)

Milestone

Our milestone is completing our 75% goal.

Resources Needed

The {Setɻy} compiler is written in python and targets llvm. As such, it depends on `ply` (python bindings for `yacc` and `flex`), `llvm-py` (python bindings for `llvm`), `llvm`, and `python`. The toolchain is already set up and working on several machines.

Work to Date

Adam has written a parser, interpreter, and partial compiler for {Setɻy}. Parts of the compiler that are unimplemented (e.g. set comprehensions) are blocking later work, but we can begin on the first goal immediately.

References

Literature on set-based languages is (un)fortunately sparce. There are several papers ([1][2]) about one existing language (SETL) which we have read:

[1] Stefan M. Freudenberger, Jacob T. Schwartz, and Micha Sharir. "Experience with the SETL Optimizer". In: *ACM Trans. Program. Lang. Syst.* 5.1 (Jan. 1983), pp. 26–45. ISSN: 0164-0925. DOI: 10.1145/357195.357197. URL: http://doi.acm.org/10.1145/357195.357197.

[2] J. T. Schwartz. "Automatic and semiautomatic optimization of SETL". In: *SIGPLAN Not.* 9.4 (Mar. 1974), pp. 43–49. ISSN: 0362-1340. DOI: 10.1145/942572.807044. URL: http://doi.acm.org/10.1145/942572.807044.