

Program Transformation and Analysis

Assignment 3

Benjamin Brandt Ohrt, `zpn492`

May 19, 2019

1 Introduction

This is the fourth of five weekly assignments in the course Program Transformation and Analysis (PAT) at Copenhagen University. The course professor is Robert Glück. The course is held in block 4, 2019.

In this assignment the focus is on partial evaluation.

2 Partial Evaluation

Partial evaluation is a method for program optimization by specialization[2]. To specialize an interpreter with respect to a given program and input. The specialization is to execute the program as far as possible with the known variables (static/dynamic) and ending with a new residual program [1, p. 69].

An example of partial evaluation is given by William and Ralf in their paper 'Tutorial on Online Partial Evaluation'[3]. Let

$$\text{if } x > y \text{ then } (10 + y)/x \text{ else } y \quad (1)$$

be an expression with two dynamic variables in a simple functional language. If y was to be known as the value 0 the expression could be specialized to the following residual expression[3, p. 2]:

$$\text{if } x > 0 \text{ then } 10/x \text{ else } 0 \quad (2)$$

3 Online Partial Evaluation

Partial evaluation in three steps:

- Collect all reachable states
see which blocks of the program we need and what data will arrive at each block.
- Program point specialization
for each reachable state (l, σ) , create a version of l , that is specialized wrt σ .
- Transition compression
remove trivial jumps (gotos)
'Definition 4.5 Let pp be a label occuring in program p , and consider a jump `goto pp`. The replacement of this `goto` by a copy of the basic block labelled pp is transition compression.'[1, p. 81]

4 Assignment

Exercise 1

Show the main steps of specializing the Turing-machine interpreter with respect to the Turing-program into the target program. Use online or offline FCL-partial evaluator.

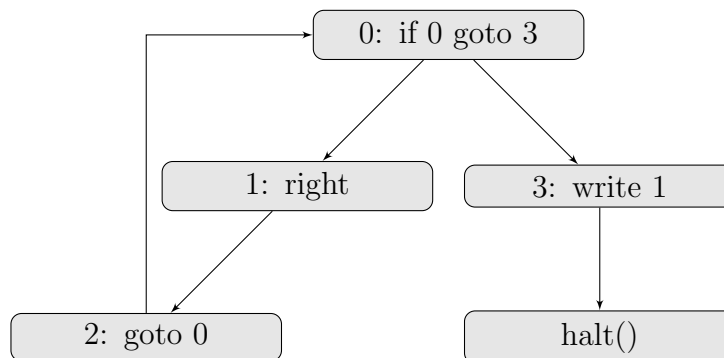
```
prg(String s):  
    0: if 0 goto 3  
    1: right  
    2: goto 0  
    3: write 1
```

We use the online FCL-partial evaluation method to specialize the program above. **First we find all reachable states.**

Table over states

	0 (goto loop/jump)	
1 (loop)		3 (jump)
2		halt()
0 (goto loop/jump)		

Above table shows which states are reachable from another. Ex. State 0, can reach 1 or 3. 1 can reach 2 and 2 can reach 0.



Second we specialize each block with respect to the program prg and the Turing-interpreter. The specialization is bluntly to replace dynamic values in the interpreter with static information known from program.

Original block:
 0: if 0 goto 3

Specialized wrt [input->S, symbol->0, nextlabel->3]:
 (0, [input->S, symbol->0, nextlabel->3]) :
 if 0 = firstsym(Right) goto 3 else 1;

Original block:
 1: right

Specialized wrt [input->S]:
 (1, [input->S]) :
 Left := cons(firstsym(Right), Left);
 Right:= tl(Right);
 Goto 2;

Original block:
 2: goto 0

Specialized wrt [input->S, nextlabel->0]:
 (2, [input->S, nextlabel->0]) :
 Goto 0;

Original block:
 3: write 1

Specialized wrt [input->S, symbol->1, nextlabel->stop]:
 (3, [input->S, symbol->1]) :
 Right:= cons(1, tl(Right));
 return right;

Ending with a specialized program as below:

(0, [input→ S, symbol→ 0, nextlabel→ 3]) :	if 0 = firstsym(Right) goto 3 else 1;
(1, [input→S]) :	Left := cons(firstsym(Right), Left); Right:= tl(Right); Goto 2;
(2, [input→S, nextlabel→0]) :	Goto 0;
(3, [input→S, symbol→1]) :	Right:= cons(1, tl(Right)); return right;

As the third and last step, we use transition compression on trivial jumps. Its shown in above specialization that the only purpose of instruction 2 is to move the instruction pointer to state 0, and can therefore be removed. So instead of letting state 2 follow up on state 1 we extend state 1 with the combination of state 1 and 0.

(0, [input→ S, symbol→ 0, nextlabel→ 3]) :	if 0 = firstsym(Right) goto 3 else 1;
(1, [input→S,symbol→0]) :	Left := cons(firstsym(Right), Left); Right:= tl(Right); if 0 = firstsym(Right) goto 3 else 1;
(3, [input→S, symbol→1]) :	Right:= cons(1, tl(Right)); return right;

The transformation, performs The First Futamura projection, which takes an S-interpreter $[int]$ written in a language L and s as a S-program and specializes the interpreter with respect to s and returns a residual target program.

$$\begin{aligned}
[s] &= [int]_L[s] \\
&= [([mix]_L, [int, s])]_L \\
&= [target]_L
\end{aligned}$$

As shown above, the specialized program is written the same language as the interpreter.

Exercise 2

Answer exercise 4.5[1]: Specialize the Turing interpreter with respect to the following program:

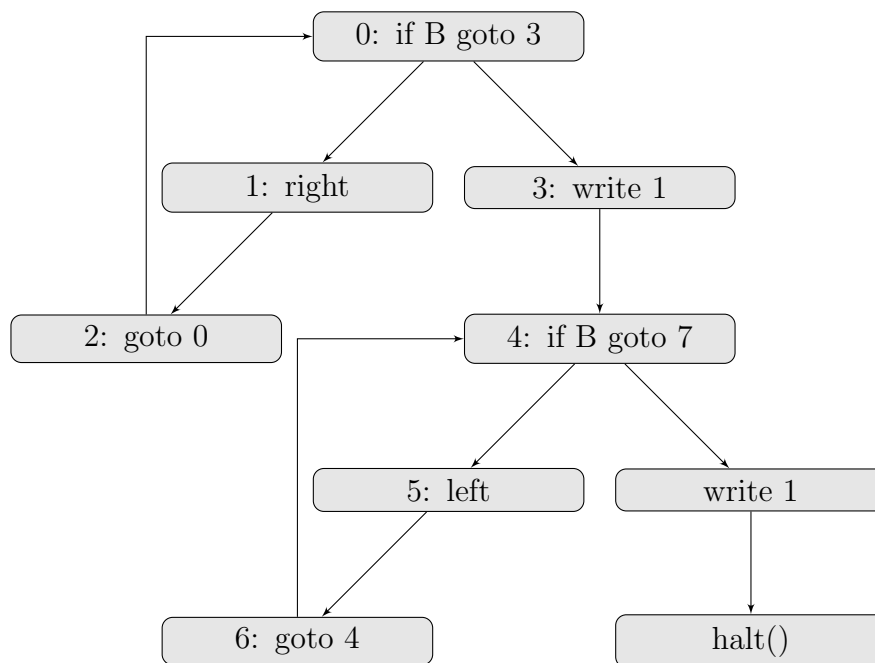
```

0: if B goto 3
1: right
2: goto 0
3: write 1
4: if B goto 7
5: left
6: goto 4
7: write 1

```

Find all reachable states.

	0 (goto loop/jump)		
1 (loop)		3 (jump)	
2		4 (goto loop1/jump1)	
0 (goto loop/jump)	5 (loop1)		7 (jump1)
	6		halt()
	4 (goto loop1/jump1)		



Second we specialize each block

Original block:

```
0: if B goto 3
```

Specialized wrt [input->S, symbol->B, nextlabel->3]:
if B = firstsym(Right) goto 3 else 1;

Original block:

```
1: right
```

Specialized wrt [input->S]:

```
(1, [input->S]) :  
    Left := cons(firstsym(Right), Left);  
    Right:= tl(Right);  
    Goto 2;
```

Original block:

```
2: goto 0
```

Specialized wrt [input->S, nextlabel->0]:

```
(2, [input->S, nextlabel->0]) :  
    new_tail(0, Q);  
    Goto 0;
```

Original block:

```
3: write 1
```

Specialized wrt [input->S, symbol->1, nextlabel->4]:

```
(3, [input->S, symbol->1]) :  
    Right:= cons(1, tl(Right));  
    Goto 4
```

Original block:

```
4: if B goto 7
```

Specialized wrt (4, [input->S, symbol->B, nextlabel->7]):
if B = firstsym(Right) goto 7 else 5;

Original block:

```
5: left
```

Specialized wrt [input->S]:
(5, [input->S]) :
Right := cons(firstsym(Right), Right);
Left:= tl(Left);
Goto 6;

Original block:

```
6: goto 4
```

Specialized wrt [input->S, nextlabel->4]:
(4, [input->S, nextlabel->4]) :
Goto 4;

Original block:

```
7: write 1
```

Specialized wrt [input->S, symbol->1, nextlabel->stop]:
(7, [input->S, symbol->1]) :
Right:= cons(1, tl(Right));
return right;

Resulting in the following specialized program:

(0, [input→ S, symbol→ B, nextlabel→ 3]) :	if B = firstsym(Right) goto 3 else 1;	
(1, [input→S]) :	Left := cons(firstsym(Right), Left); Right:= tl(Right); Goto 2;	
(2, [input→S, nextlabel→0]) :	Goto 0;	
(3, [input→S, symbol→1]) :	Right:= cons(1, tl(Right)); Goto 4;	
(4, [input→ S, symbol→ B, nextlabel→ 7]) :	if B = firstsym(Right) goto 7 else 5;	
(5, [input→S]) :	Right := cons(firstsym(Left), Right); Left:= tl(Left); Goto 6;	
(6, [input→S, nextlabel→4]) :	Goto 4;	
(7, [input→S, symbol→1]) :	Right:= cons(1, tl(Right)); return right;	

Third we use transition compression to eliminate trivial jumps.

Again state 2 can be replaced with the combination of state 1 + 0. And state 6 can be replaced by state 5 + 4. We see that state 3 and 4 can be merged as an initial state before the second loop. Which makes the specialization of state 1, 3 and 5 the following:

(0, [input→S, symbol→B, nextlabel→ 3]) :	if B = firstsym(Right) goto 3 else 1;
(1, [input→S, symbol→B]) :	Left := cons(firstsym(Right), Left); Right:= tl(Right); if B = firstsym(Right) goto 3 else 1;
(3, [input→S, symbol→{1,B}]) :	Right:= cons(1, tl(Right)); if B = firstsym(Right) goto 7 else 5;
(5, [input→S, symbol→B, nextlabel→ 7]) :	Right := cons(firstsym(Left), Right); Left:= tl(Left); if B = firstsym(Right) goto 7 else 5;
(7, [input→S, symbol→1]) :	Right:= cons(1, tl(Right)); return right;

Which produces the

Exercise 3

Answer exercise 4.7[1]: Will specialization of the Turing interpreter (Figure 4.4[1]) with respect to a program p terminate for all Turing programs p ?

References

- [1] Jones N.D., Gomard C.K., Sestoft P., *Partial Evaluation and Automatic Program Generation*. Prentice Hall 1993. <https://www.itu.dk/people/sestoft/pebook/pebook.html> (Links to an external site.)
- [2] https://en.wikipedia.org/wiki/Partial_evaluation
- [3] William Cook, Ralf Lämmel, *Tutorial on Online Partial Evaluation*. www.cs.utexas.edu/~wcook/tutorial/PEnotes.pdf