

Static Single Assignment Form in the COINS Compiler Infrastructure

Masataka Sassa, Toshiharu Nakaya,
Masaki Kohama, Takeaki Fukuoka and
Masahito Takahashi
(Tokyo Institute of Technology)

Background

Static single assignment (SSA) form facilitates compiler optimizations
Compiler infrastructure facilitates compiler development.

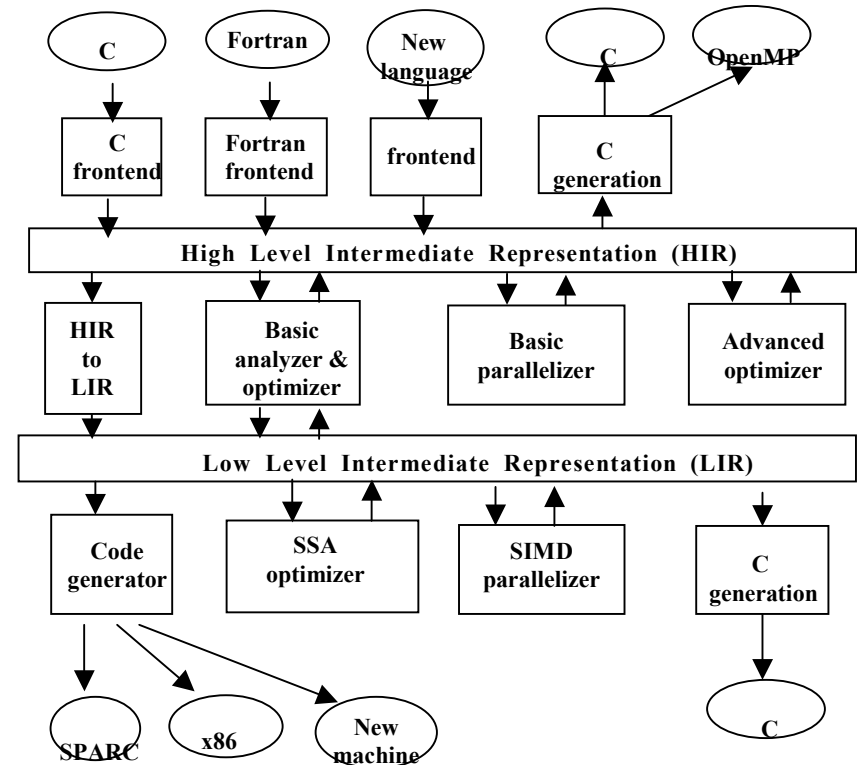
Outline

0. COINS infrastructure and the SSA form
1. Current status of optimization using SSA form in COINS infrastructure
2. A comparison of two major algorithms for translating from normal form into SSA form
3. A comparison of two major algorithms for translating back from SSA form into normal form

0. COINS infrastructure and Static Single Assignment Form (SSA Form)

COINS compiler infrastructure

- Multiple source languages
- Retargetable
- Two intermediate form, HIR and LIR
- Optimizations
- Parallelization
- C generation, source-to-source translation
- Written in Java
- 2000~ developed by Japanese institutions under Grant of the Ministry



Static Single Assignment (SSA) Form

1: $a = x + y$

2: $a = a + 3$

3: $b = x + y$

(a) Normal (conventional)
form (source program
or internal form)

1: $a_1 = x_0 + y_0$

2: $a_2 = a_1 + 3$

3: $b_1 = x_0 + y_0$

(b) SSA form

SSA form is a recently proposed internal representation where each use of a variable has a single definition point.

Indices are attached to variables so that their definitions become unique.

Optimization in Static Single Assignment (SSA) Form

```
1: a = x + y
2: a = a + 3
3: b = x + y
```

(a) Normal form

SSA
translation

```
1: a1 = x0 + y0
2: a2 = a1 + 3
3: b1 = x0 + y0
```

(b) SSA form

Optimization in SSA form (common
subexpression elimination)

```
1: a1 = x0 + y0
2: a2 = a1 + 3
3: b1 = a1
```

(c) After SSA form optimization

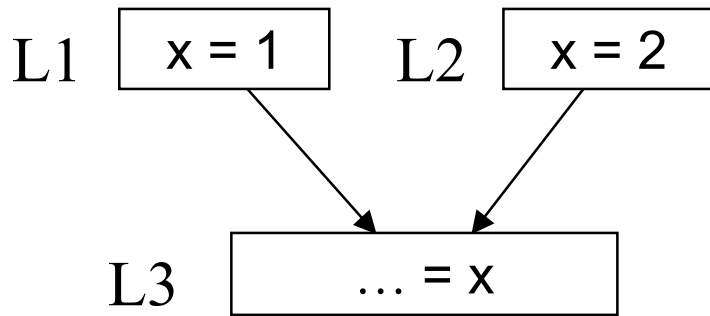
SSA back
translation

```
1: a1 = x0 + y0
2: a2 = a1 + 3
3: b1 = a1
```

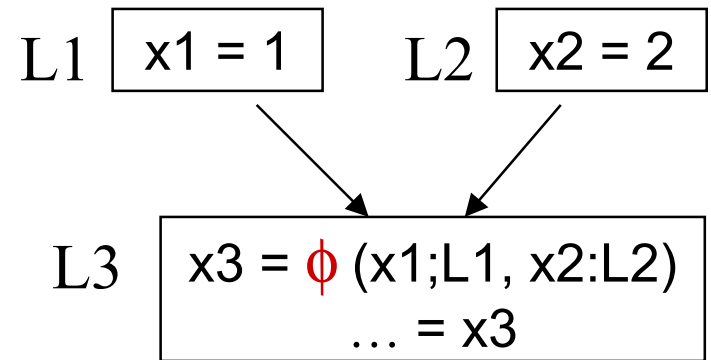
(d) Optimized normal form

SSA form is becoming increasingly popular in compilers, since it is suited for clear handling of dataflow analysis and optimization.

Translating into SSA form (SSA translation)

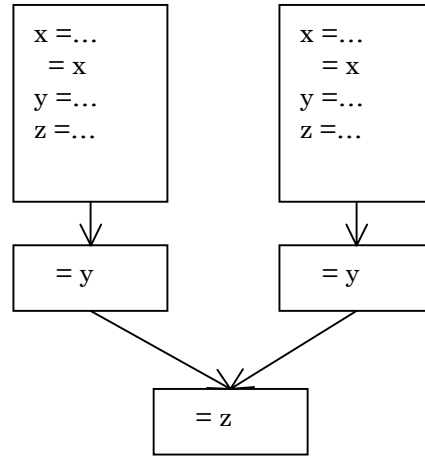


(a) Normal form

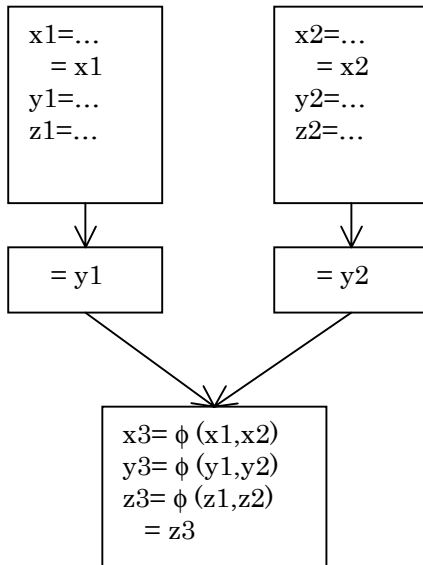


(b) SSA form

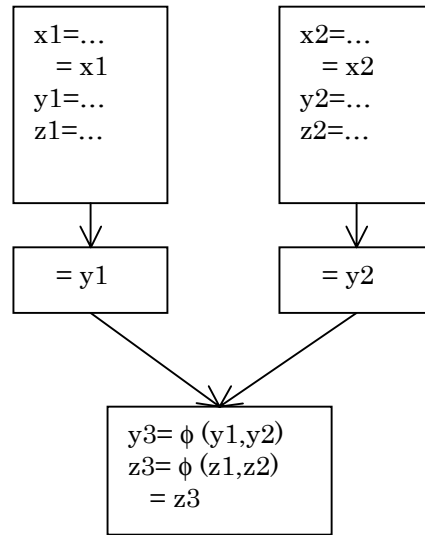
Translating into SSA form (SSA translation)



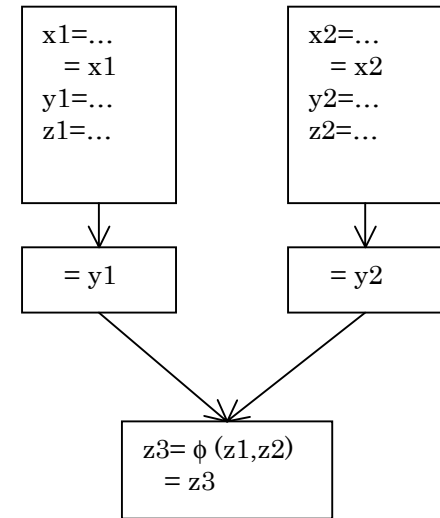
Normal form



Minimal SSA form

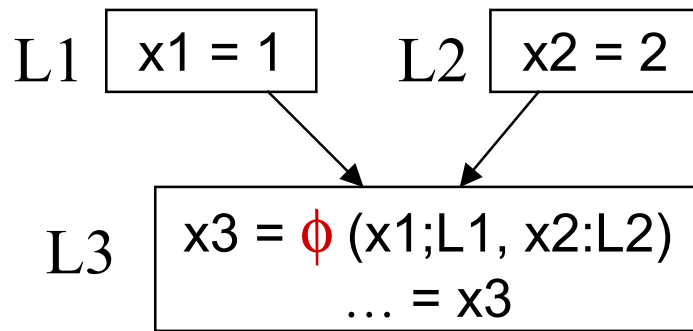


Semi-pruned SSA form

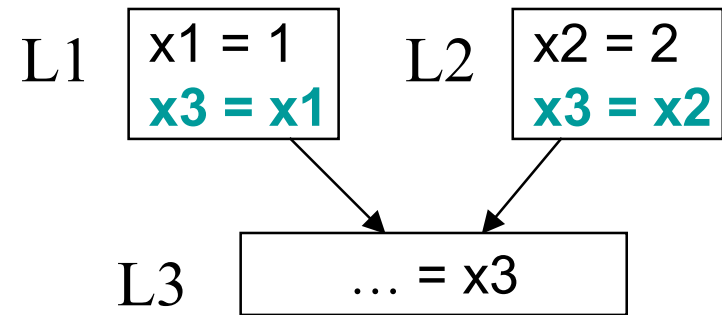


Pruned SSA form

Translating back from SSA form (SSA back translation)



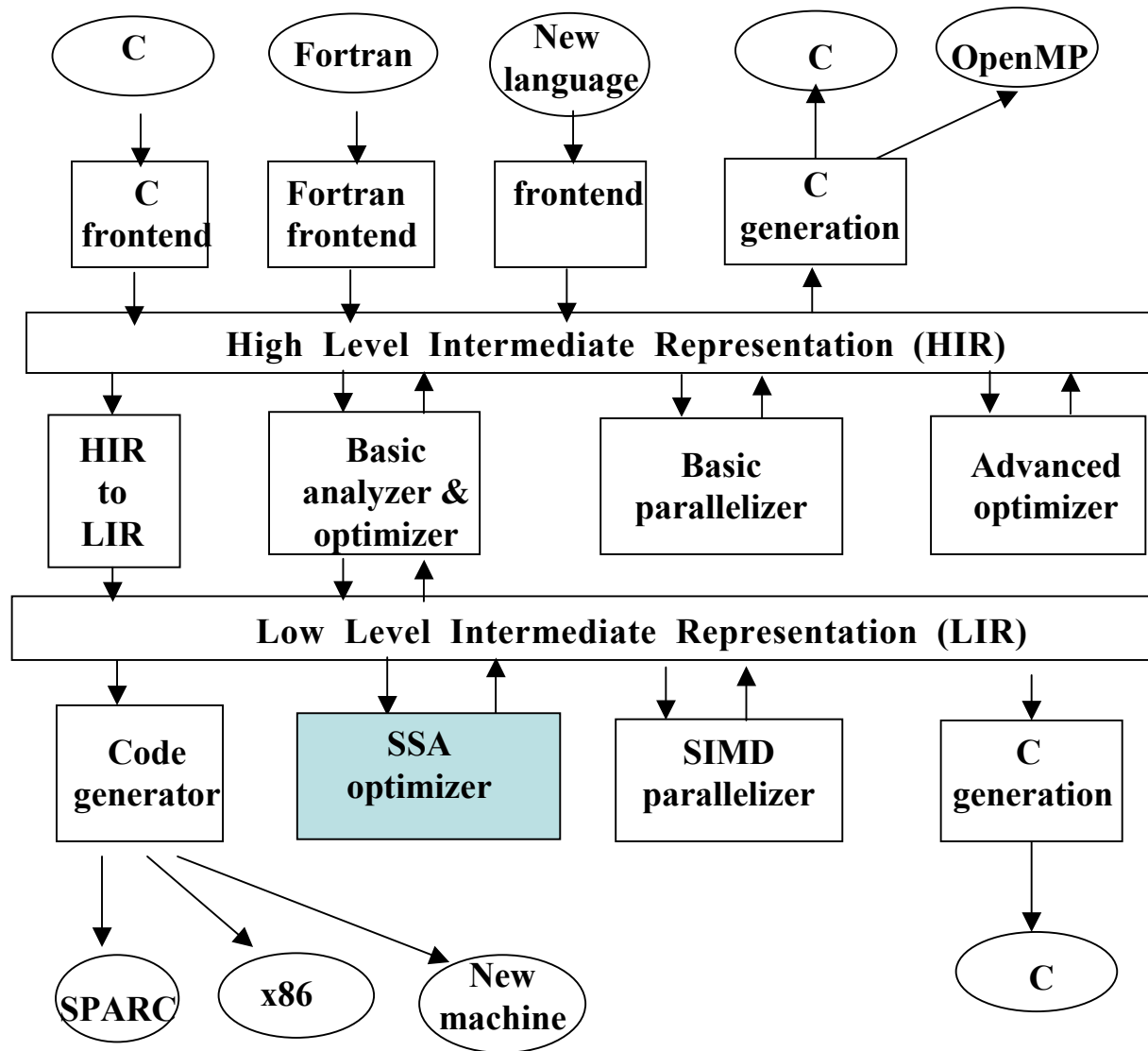
(a) SSA form



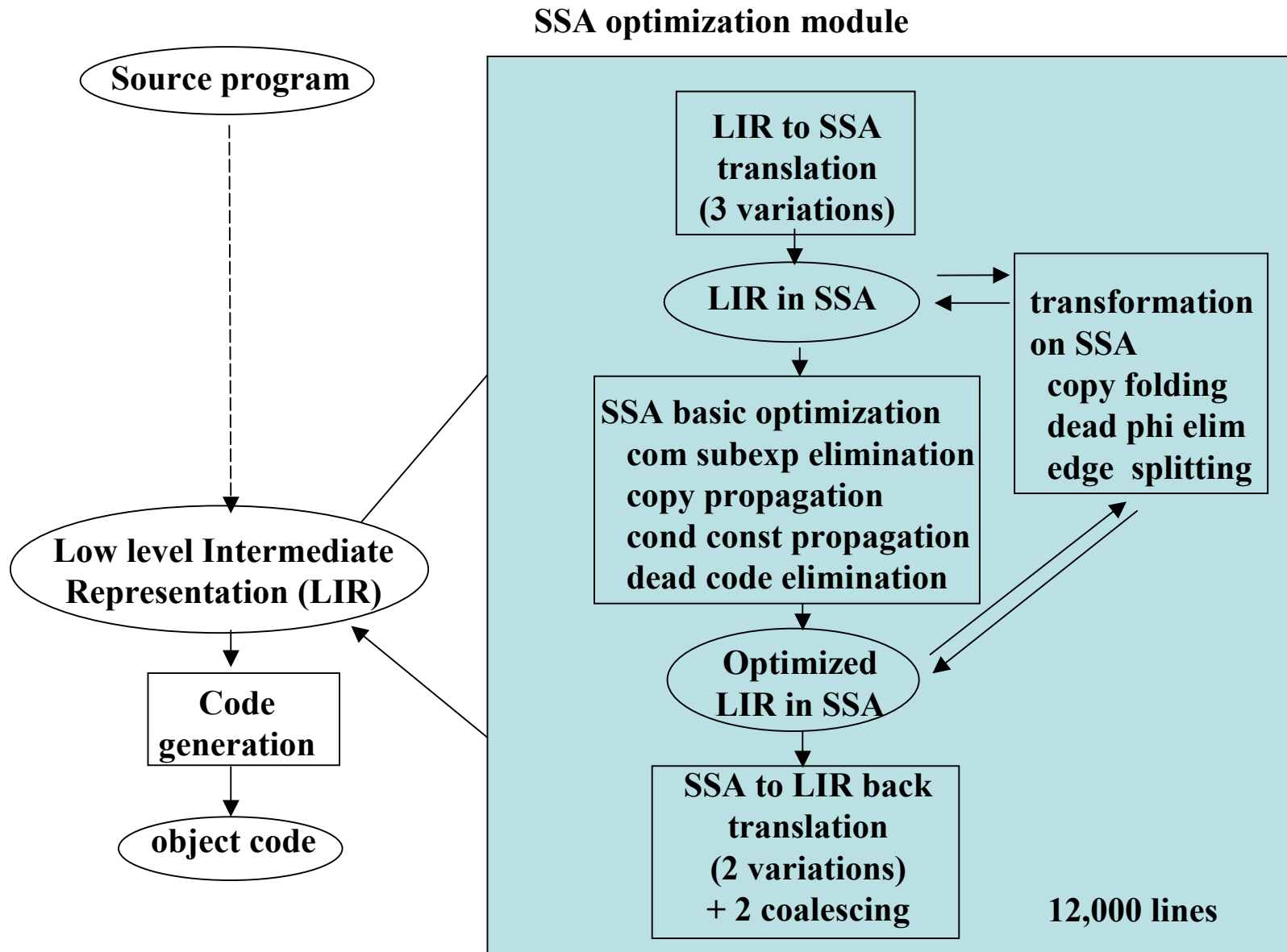
(b) Normal form

1. SSA form module in the
COINS compiler infrastructure

COINS compiler infrastructure



SSA optimization module in COINS



Outline of SSA module in COINS

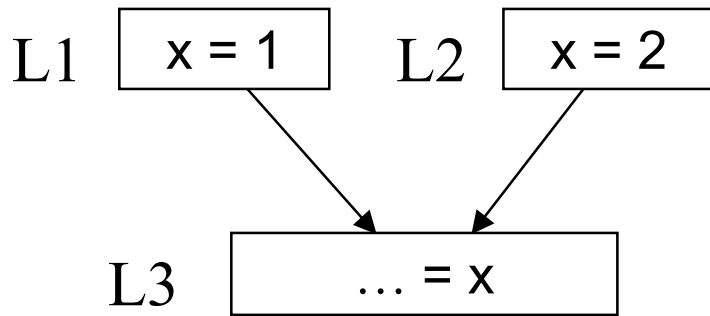
- Translation into and back from SSA form on Low Level Intermediate Representation (LIR)
 - SSA translation: Use dominance frontier [Cytron et al. 91]
 - SSA back translation: [Sreedhar et al. 99]
 - Basic optimization on SSA form: dead code elimination, copy propagation, common subexpression elimination, conditional constant propagation
- Useful transformation as an infrastructure for SSA form optimization
 - Copy folding at SSA translation time, critical edge removal on control flow graph ...
 - Each variation and transformation can be made selectively
- Preliminary result
 - 1.43 times faster than COINS w/o optimization
 - 1.25 times faster than gcc w/o optimization

2. A comparison of two major algorithms for SSA translation

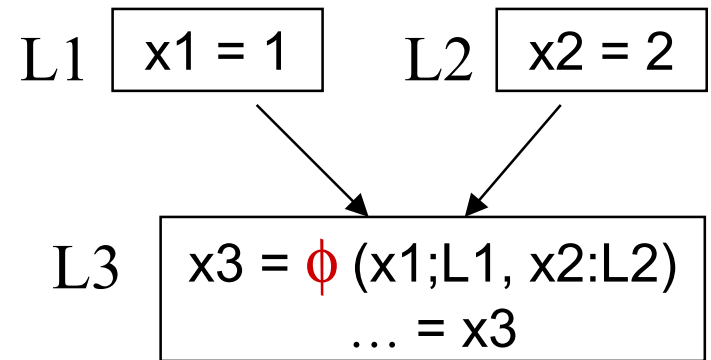
- Algorithm by Cytron [1991]
Dominance frontier
- Algorithm by Sreedhar [1995]
DJ-graph

Comparison made to decide the algorithm to be included in COINS

Translating into SSA form (SSA translation)

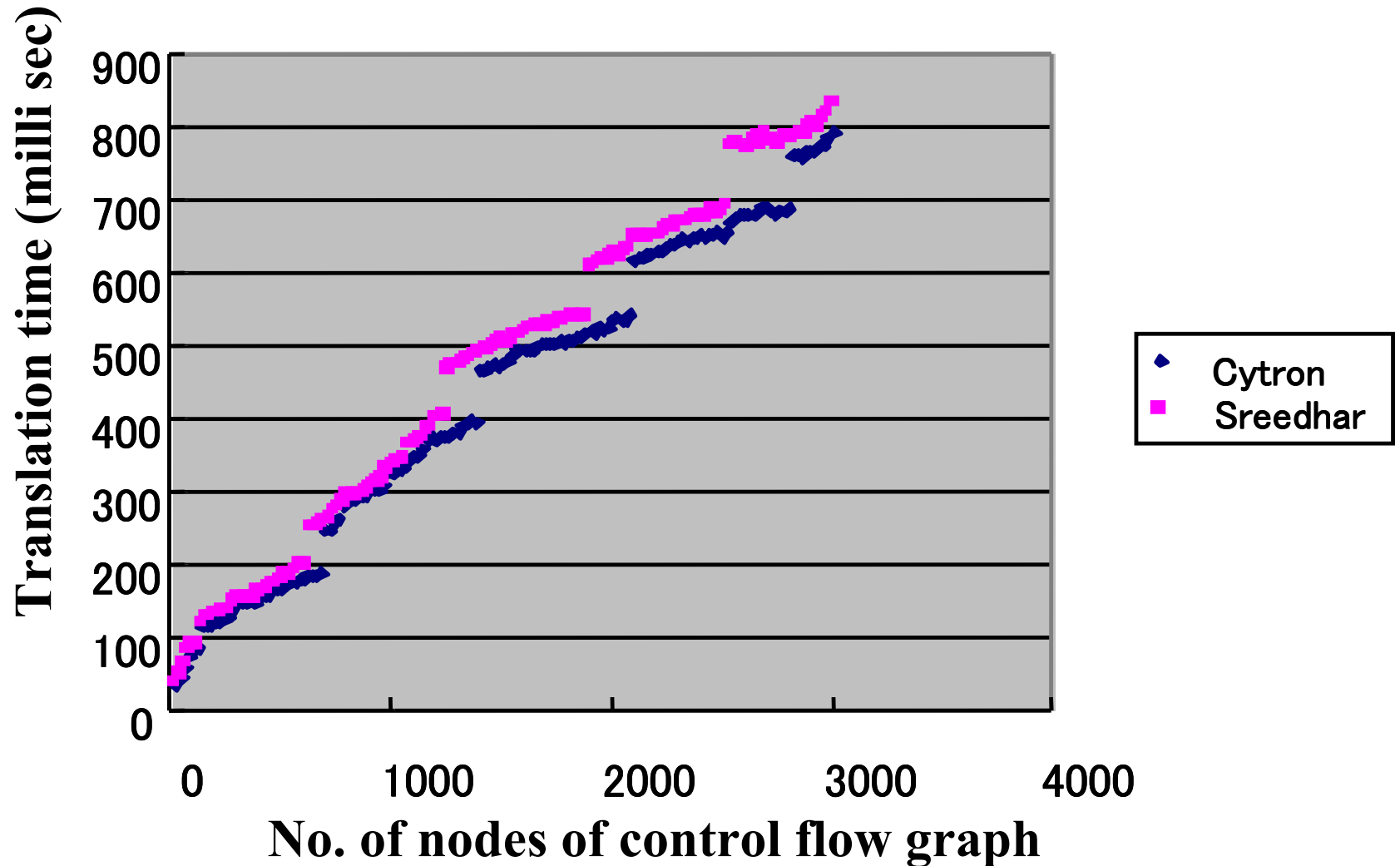


(a) Normal form



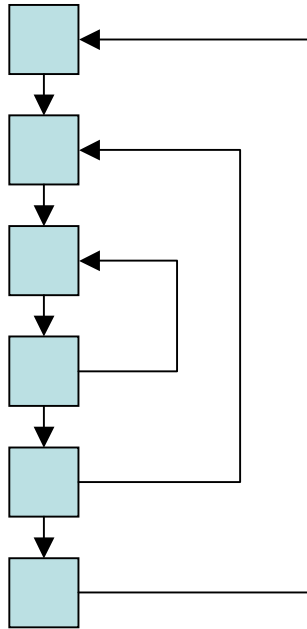
(b) SSA form

Usual programs

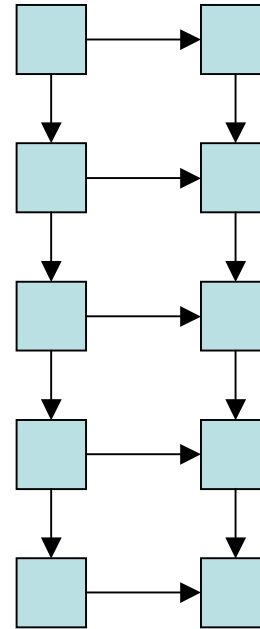


(The gap is due to the garbage collection)

Peculiar programs

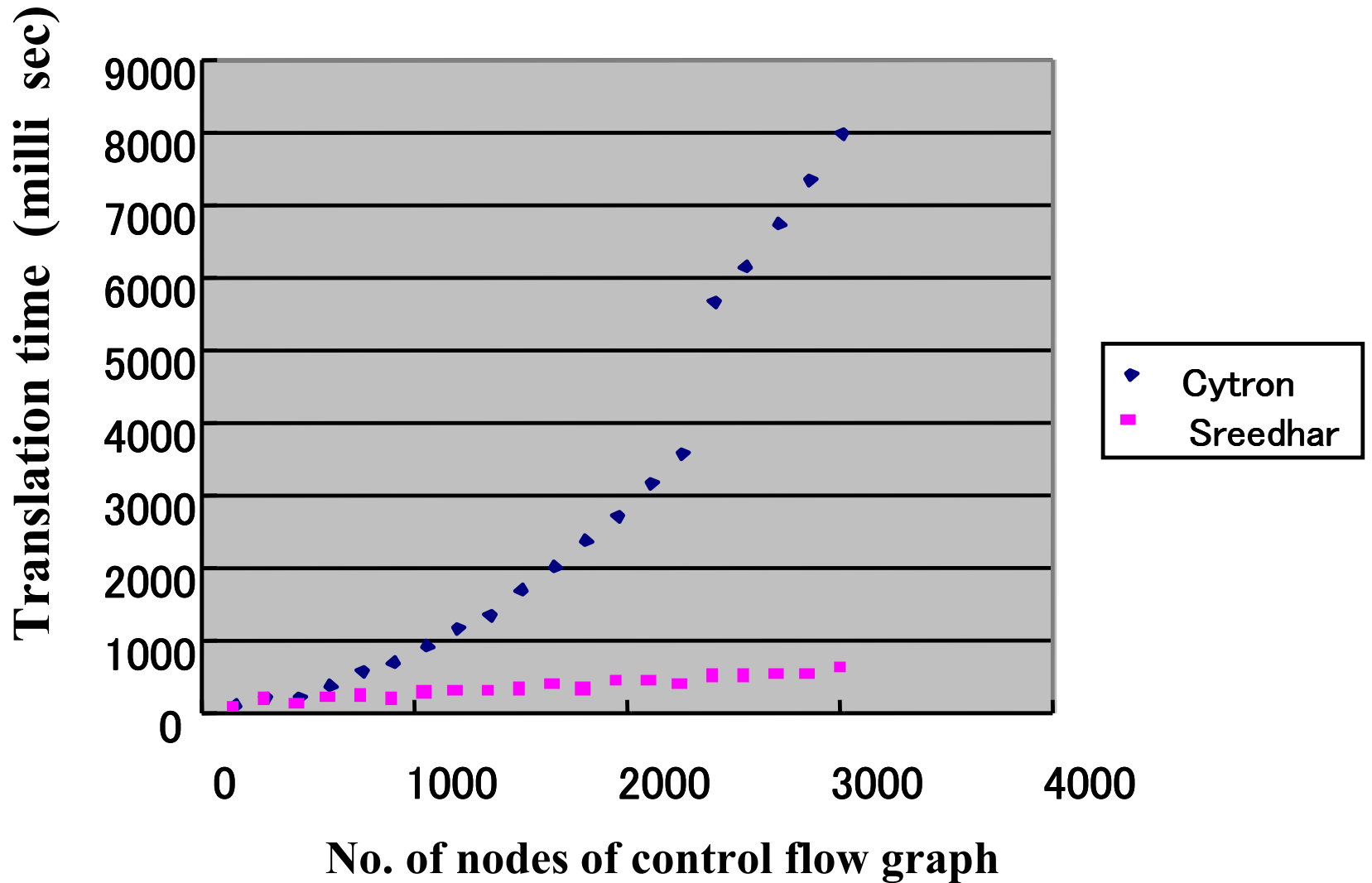


(a) nested loop

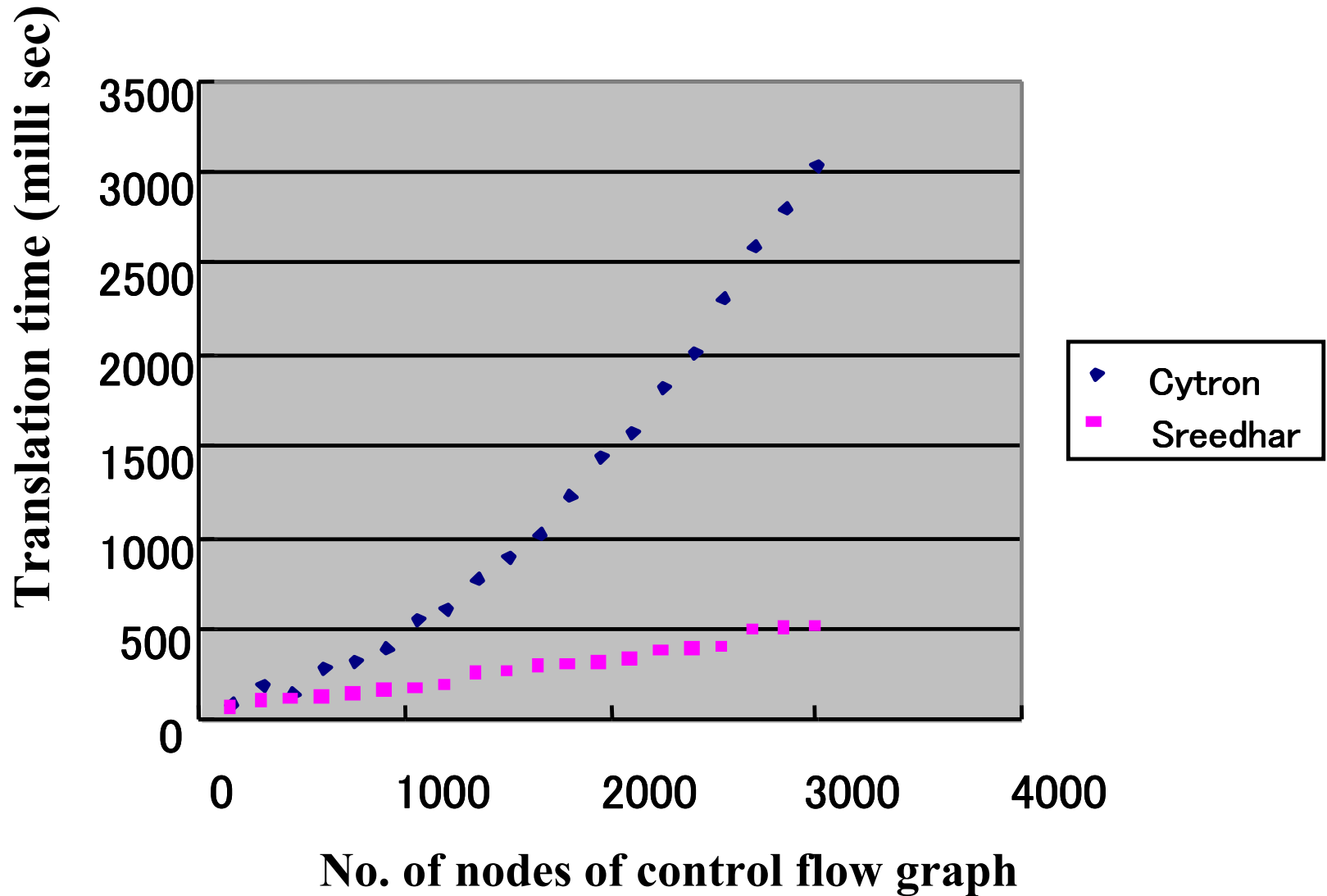


(b) ladder graph

Nested loop programs



Ladder graph programs

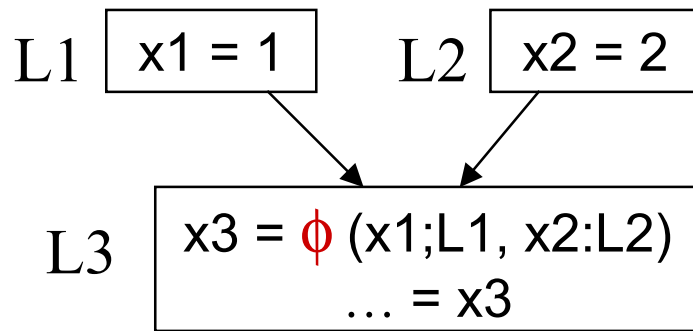


3. A comparison of two major algorithms for SSA back translation

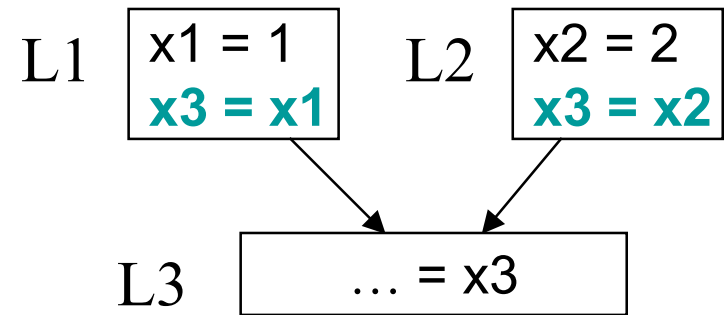
- Algorithm by Briggs [1998]
 Insert copy statements
- Algorithm by Sreedhar [1999]
 Eliminate interference

There have been no studies of comparison
Comparison made on COINS

Translating back from SSA form (SSA back translation)



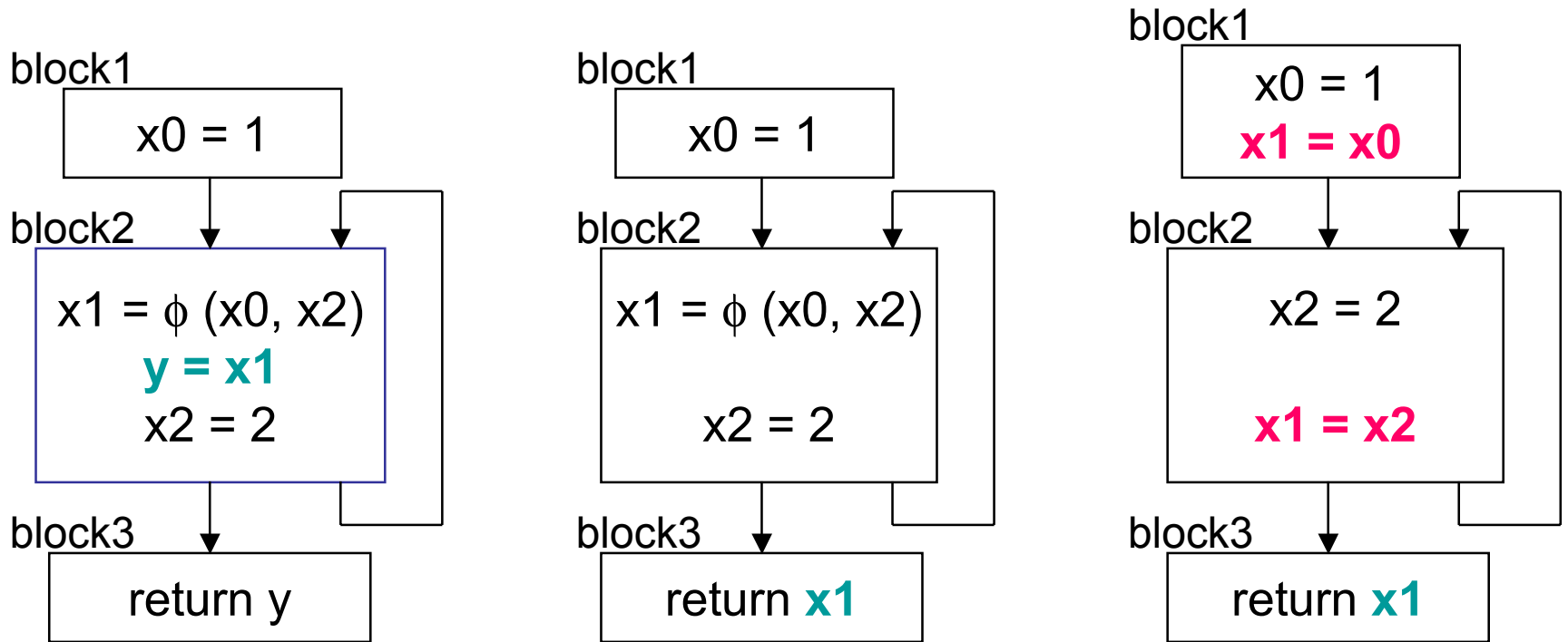
(a) SSA form



(b) Normal form

Problems of naïve SSA back translation

(lost copy problem)



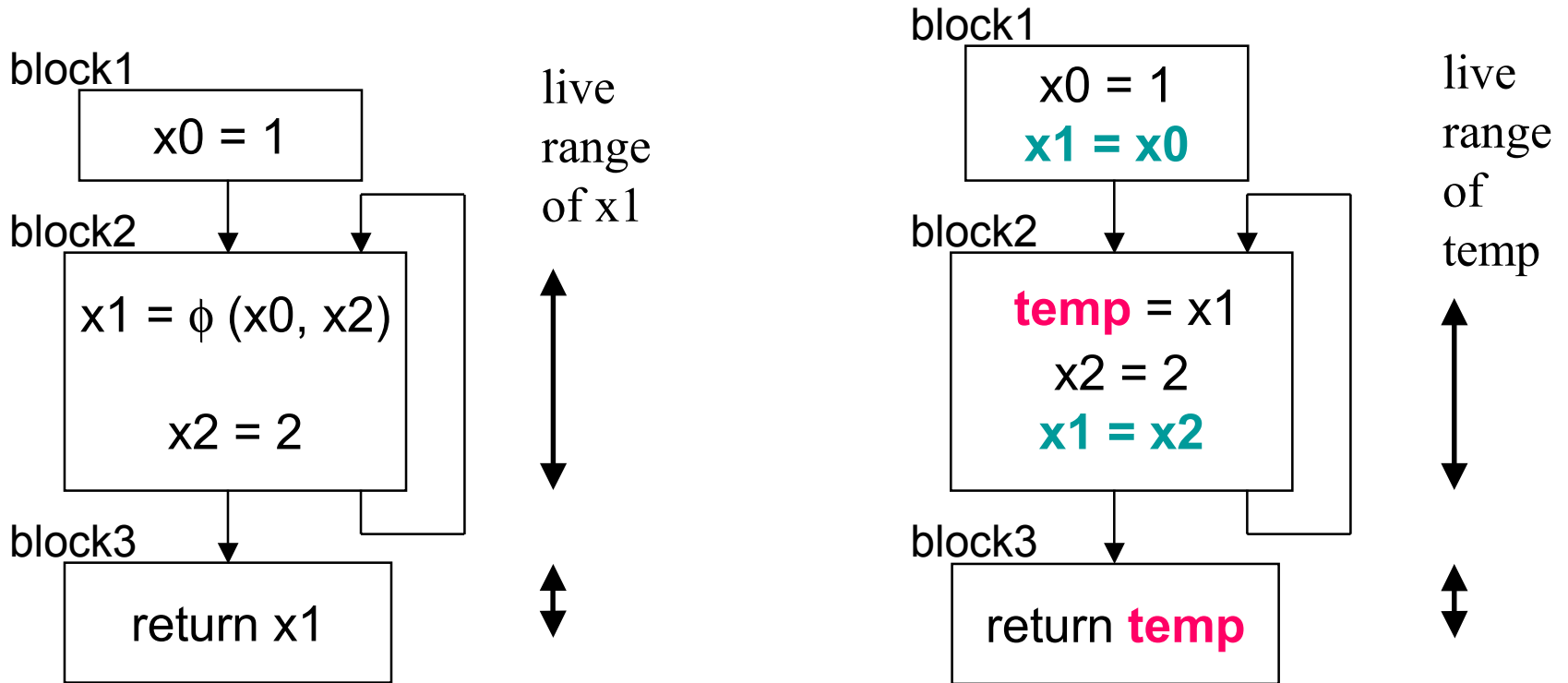
Copy propagation

Back translation
by naïve method

not correct

To remedy these problems...

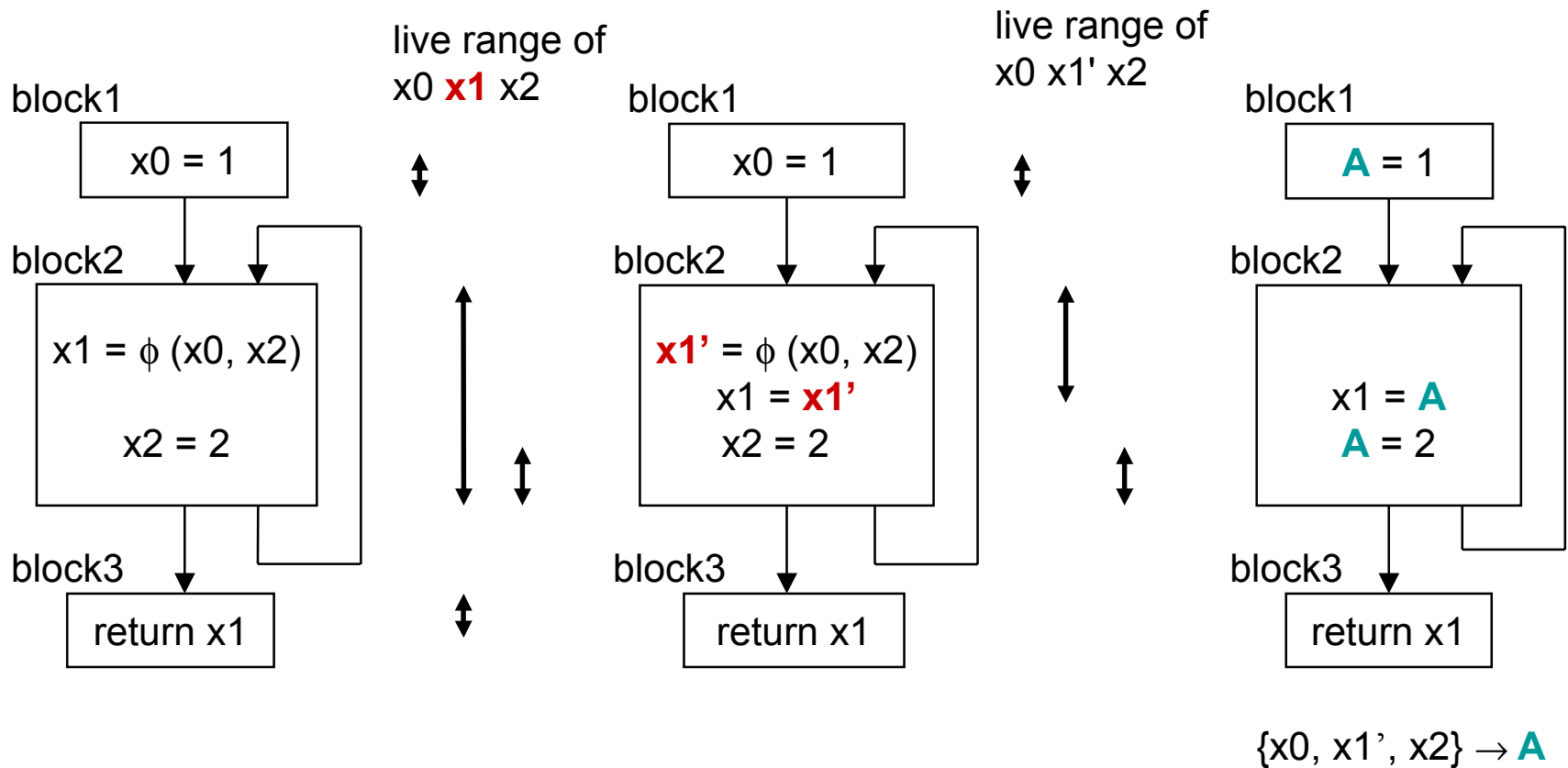
(i) SSA back translation algorithm by Briggs



(a) SSA form

(b) normal form after back translation

(ii) SSA back translation algorithm by Sreedhar



(a) SSA form

(b) eliminating interference

(c) normal form after back translation

Empirical comparison of SSA back translation

No. of copies (no. of copies in loops)

	SSA form	Briggs	Briggs + Coalescing	Sreedhar
Lost copy	0	3	1 (1)	1 (1)
Simple ordering	0	5	2 (2)	2 (2)
Swap	0	7	5 (5)	3 (3)
Swap-lost	0	10	7 (7)	4 (4)
do	0	9	6 (4)	4 (2)
fib	0	4	0 (0)	0 (0)
GCD	0	9	5 (2)	5 (2)
Selection Sort	0	9	0 (0)	0 (0)
Hige Swap	0	8	3 (3)	4 (4)

Previous work:

SSA form in compiler infrastructure

- SUIF (Stanford Univ.): no SSA form
- machine SUIF (Harvard Univ.): only one optimization in SSA form
- Scale (Univ. Massachusetts): a couple of SSA form optimizations. But it generates only C programs, and cannot generate machine code like in COINS.
- GCC: some attempts but experimental

Only COINS will have full support of SSA form as a compiler infrastructure

Summary

- SSA form module of the COINS infrastructure
- Empirical comparison of algorithms for SSA translation gave criterion to make a good choice
- Empirical comparison of algorithms for SSA back translation clarified there is no single algorithm which gives optimal result

Hope COINS and its SSA module help the compiler writer to compare/evaluate/add optimization methods