Superoptimization

How fast can your code go?

James Pallister
University of Bristol & Embecosm

What is superoptimization?

Unoptimized
code

What is superoptimization?

Unoptimized
code

Compiler
optimized
code

What is superoptimization?

Unoptimized code

Compiler optimized code

Superoptimized code

Plan for today

What is superoptimization?

Latest developments

The GNU Superoptimizer

Plan for today

What is superoptimization?

Latest developments

The GNU Superoptimizer

Superoptimization in action

```
int sign(int n)
{
  if(n > 0)
    return 1;
  else if(n < 0)
    return -1;
  else
    return 0;
}
```

Superoptimization in action

```
int sign(int n)
{
   if(n > 0)
      return 1;
   else if(n < 0)
      return -1;
   else
      return 0;
}
```

Superoptimization in action

```c
int sign(int n)
{
   if(n > 0)
      return 1;
   else if(n < 0)
      return -1;
   else
      return 0;
}
```

```
    cmp.l   d0, 0
    ble     L1
    move.l  d1, 1
    bra     L3
L1:
    bge     L2
    move.l  d1, -1
    bra     L3
L2:
    move.l  d1, 0
L3:
```

Superoptimization in action

```c
int sign(int n)
{
  if(n > 0)
    return 1;
  else if(n < 0)
    return -1;
  else
    return 0;
}
```

```asm
add.l  d0, d0
subx.l d1, d1
negx.l d0
addx.l d1, d1
```

University of BRISTOL

EMBECOSM®

Superoptimization in action

```c
int sign(int n)
{
  if(n > 0)
    return 1;
  else if(n < 0)
    return -1;
  else
    return 0;
}
```

```
   cmp.l   d0, 0
   ble     L1
   move.l  d1, 1
   bra     L3
L1:
   bge     L2
   move.l  d1, -1
   bra     L3
L2:
   move.l  d1, 0
L3:
```

```
add.l   d0, d0
subx.l  d1, d1
negx.l  d0
addx.l  d1, d1
```

How does it work?

```
d0  ← n

add.l  d0, d0
subx.l d1, d1
negx.l d0
addx.l d1, d1

d1  → sign(n)
```

University of BRISTOL

EMBECOSM®

How does it work?

```
d0   ← n

add.l   d0, d0
subx.l  d1, d1
negx.l  d0
addx.l  d1, d1

d1  → sign(n)
```

x	d0	d1
0	-3	

x	d0	d1
0	0	

x	d0	d1
0	2	

University of BRISTOL

EMBECOSM®

How does it work?

```
d0  ← n

add.l  d0, d0
subx.l d1, d1
negx.l d0
addx.l d1, d1

d1  → sign(n)
```

x	d0	d1
0	-3	
1	-6	

x	d0	d1
0	0	
0	0	

x	d0	d1
0	2	
0	4	

How does it work?

```
d0  ← n

add.l  d0, d0
subx.l d1, d1
negx.l d0
addx.l d1, d1

d1  → sign(n)
```

x	d0	d1
0	-3	
1	-6	
0	-6	-1

x	d0	d1
0	0	
0	0	
0	0	0

x	d0	d1
0	2	
0	4	
0	4	0

How does it work?

```
d0   ← n

add.l   d0, d0
subx.l  d1, d1
negx.l  d0
addx.l  d1, d1

d1   → sign(n)
```

x	d0	d1
0	-3	
1	-6	
0	-6	-1
1	6	-1

x	d0	d1
0	0	
0	0	
0	0	0
0	0	0

x	d0	d1
0	2	
0	4	
0	4	0
1	-4	0

How does it work?

```
d0  ← n

add.l   d0, d0
subx.l  d1, d1
negx.l  d0
addx.l  d1, d1

d1  → sign(n)
```

x	d0	d1
0	-3	
1	-6	
0	-6	-1
1	6	-1
0	6	-1

x	d0	d1
0	0	
0	0	
0	0	0
0	0	0
0	0	0

x	d0	d1
0	2	
0	4	
0	4	0
1	-4	0
0	-4	1

University of BRISTOL

EMBECOSM®

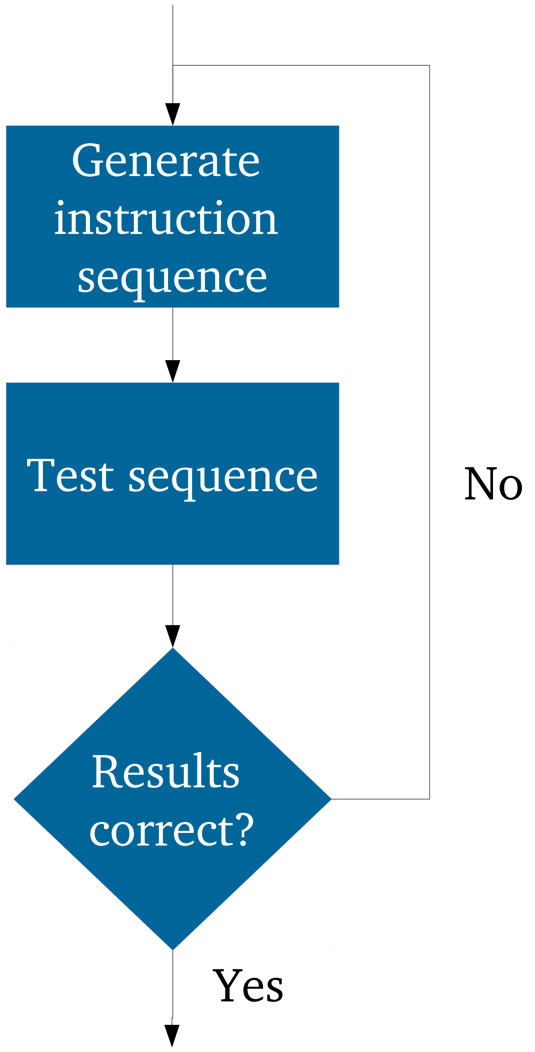
How does it work?

	x	d0	d1	x	d0	d1	x	d0	d1
`d0 ← n`	0	-3		0	0		0	2	
`add.l d0, d0`	1	-6		0	0		0	4	
`subx.l d1, d1`	0	-6	-1	0	0	0	0	4	0
`negx.l d0`	1	6	-1	0	0	0	1	-4	0
`addx.l d1, d1`	0	6	-1	0	0	0	0	-4	1
`d1 → sign(n)`			-1			0			1

University of BRISTOL

EMBECOSM®

“A look at the smallest program”



Massalin, H. (1987). Superoptimizer - A Look at the Smallest Program. ACM SIGARCH Computer Architecture News, 122–126.

A recent example: Atmel AVR

A recent example: Atmel AVR

Is x a power of 2?

A recent example: Atmel AVR

Is x a power of 2?

```
r = !((x-1) & x) && x;
```

A recent example: Atmel AVR

Is x a power of 2?

```
r = !((x-1) & x) && x;
```

Compiled code

```
mov     r20, r24
ldi     r21, 0
mov     r22, r20
mov     r19, r21
subi    r22, 1
sbc     r19, r1
and     r22, r20
and     r19, r21
or      r22, r19
brne    .+0
ldi     r25, 0x01
cpse    r24, 1
rjmp    .+0
ldi     r25, 0
mov     r24, r25
ret
ldi     r24, 0
ret
```

A recent example: Atmel AVR

Is x a power of 2?

```
r = !((x-1) & x) && x;
```

Compiled code

```
mov     r20, r24
ldi     r21, 0
mov     r22, r20
mov     r19, r21
subi    r22, 1
sbc     r19, r1
and     r22, r20
and     r19, r21
or      r22, r19
brne    .+0
ldi     r25, 0x01
cpse    r24, 1
rjmp    .+0
ldi     r25, 0
mov     r24, r25
ret
ldi     r24, 0
ret
```

Superoptimized code

```
mov     r1,r0
dec     r1
eor     r0,r1
sub     r1,r0
adc     r1,r1
and     r1,r0
ret
```

University of BRISTOL

EMBECOSM®

Superoptimization fundamentals: Enumeration

Superoptimization fundamentals: Enumeration

Generating the sequences of instructions

Superoptimization fundamentals: Enumeration

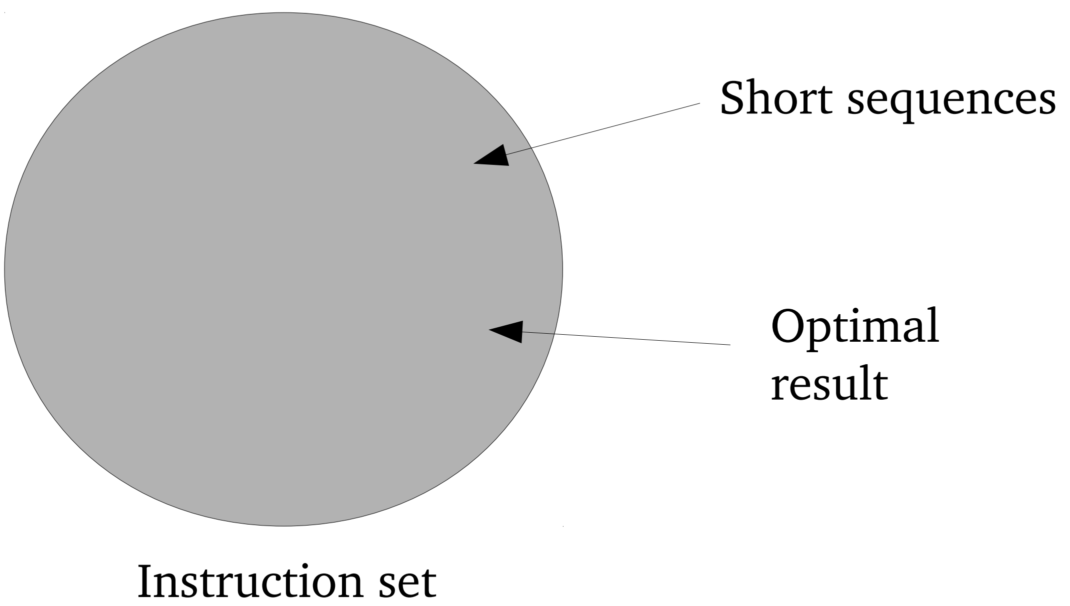
Generating the sequences of instructions

- But doing them *all* takes far too long

Superoptimization fundamentals: Enumeration

Generating the sequences of instructions

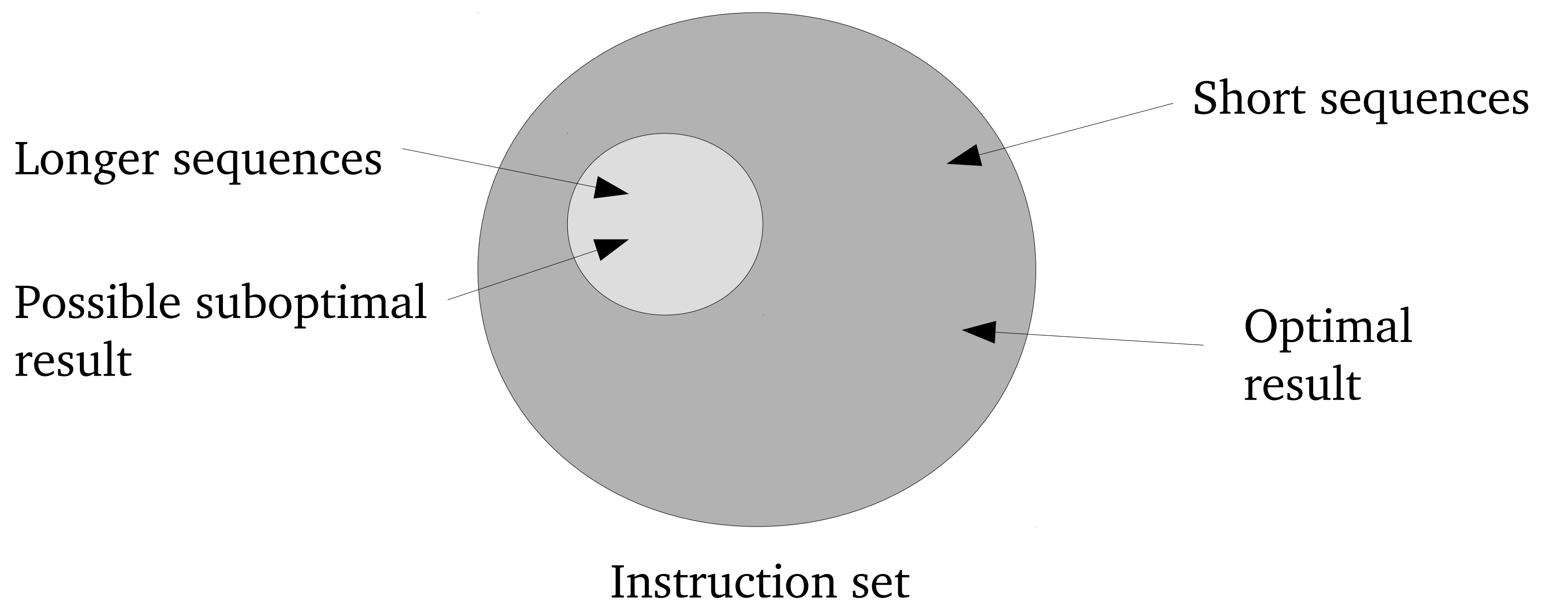
– But doing them *all* takes far too long



Superoptimization fundamentals: Enumeration

Generating the sequences of instructions

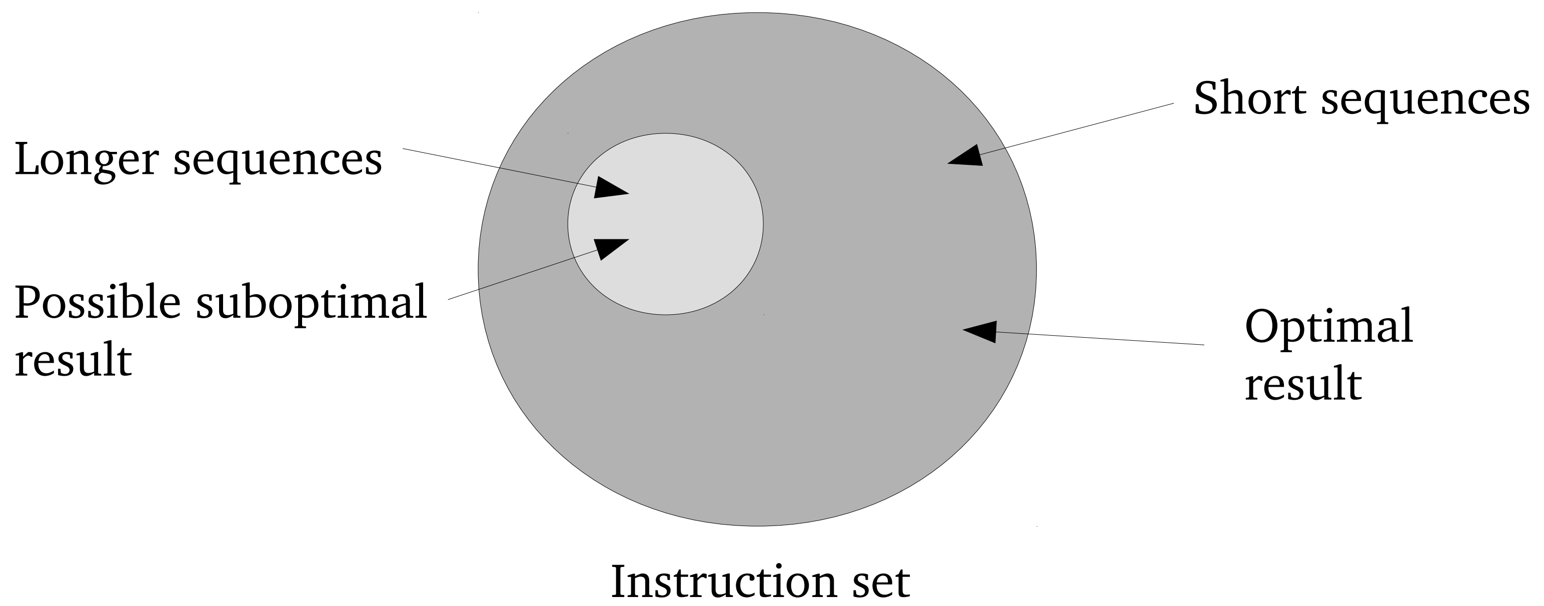
– But doing them *all* takes far too long



Superoptimization fundamentals: Enumeration

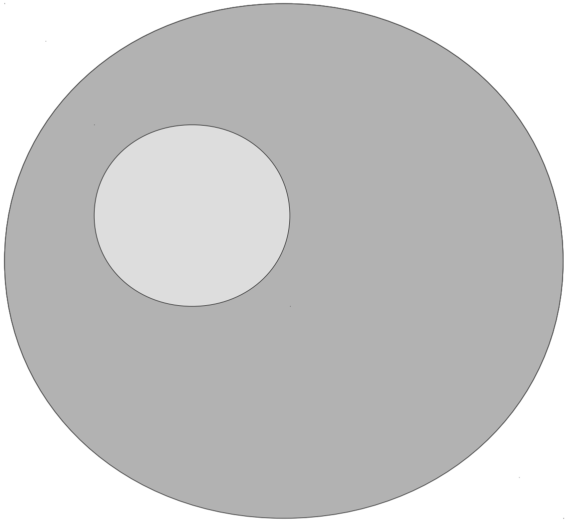
Generating the sequences of instructions

- But doing them *all* takes far too long



How to select the sequences of instructions?

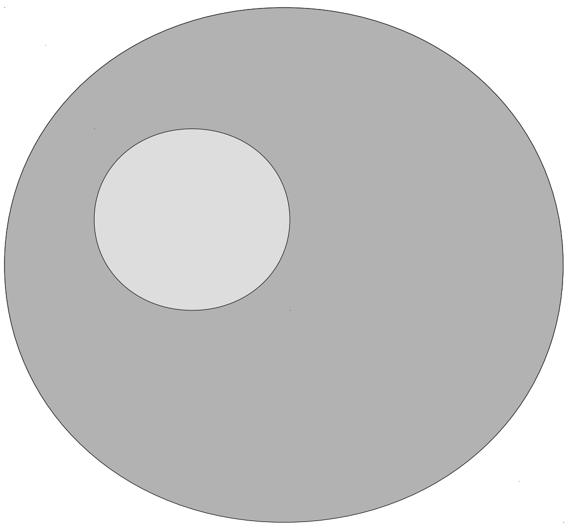
Superoptimization fundamentals: Pruning



Instruction set

Superoptimization fundamentals: Pruning

Not all instruction sequences are valid.

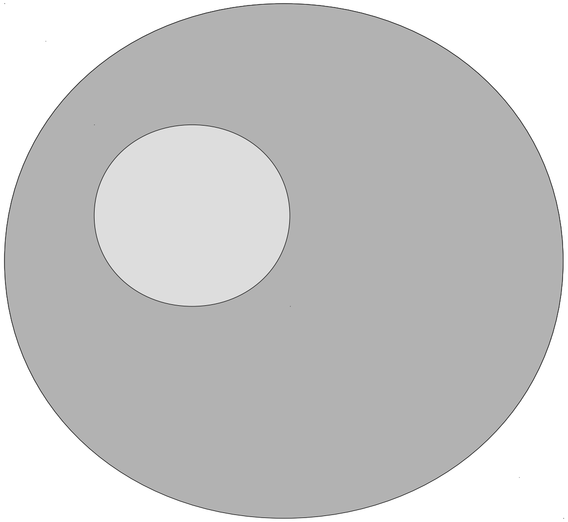


Instruction set

Superoptimization fundamentals: Pruning

Not all instruction sequences are valid.

How do we quickly ignore bad sequences?



Instruction set

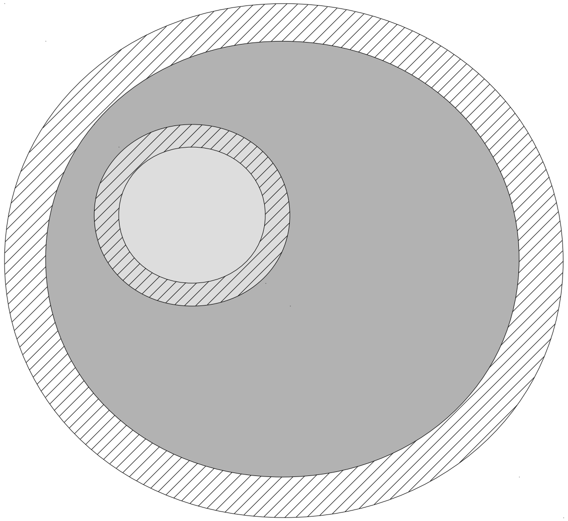
University of BRISTOL

EMBECOSM®

Superoptimization fundamentals: Pruning

Not all instruction sequences are valid.

How do we quickly ignore bad sequences?

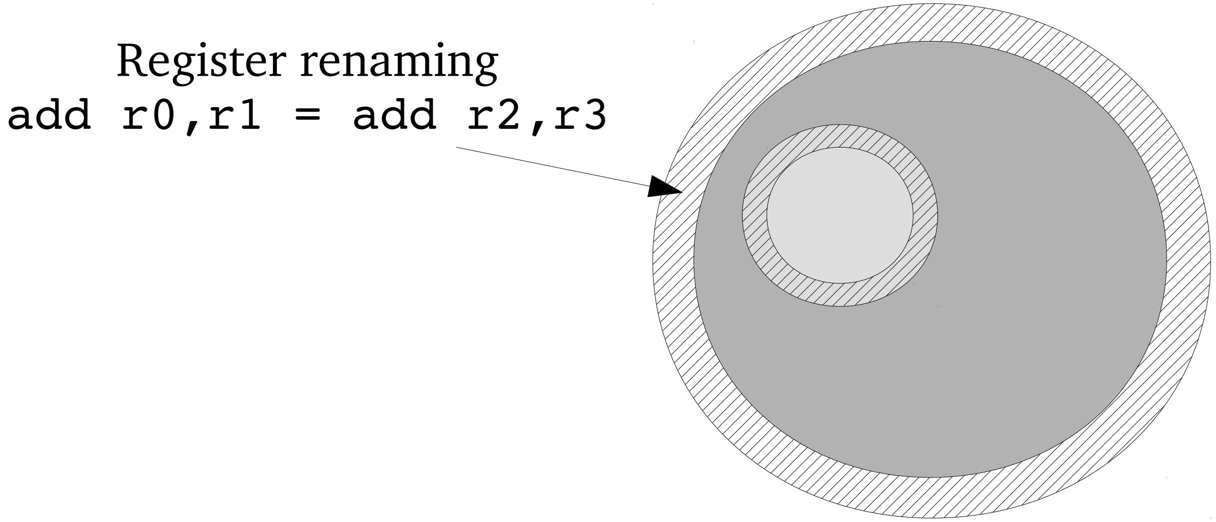


Instruction set

Superoptimization fundamentals: Pruning

Not all instruction sequences are valid.

How do we quickly ignore bad sequences?

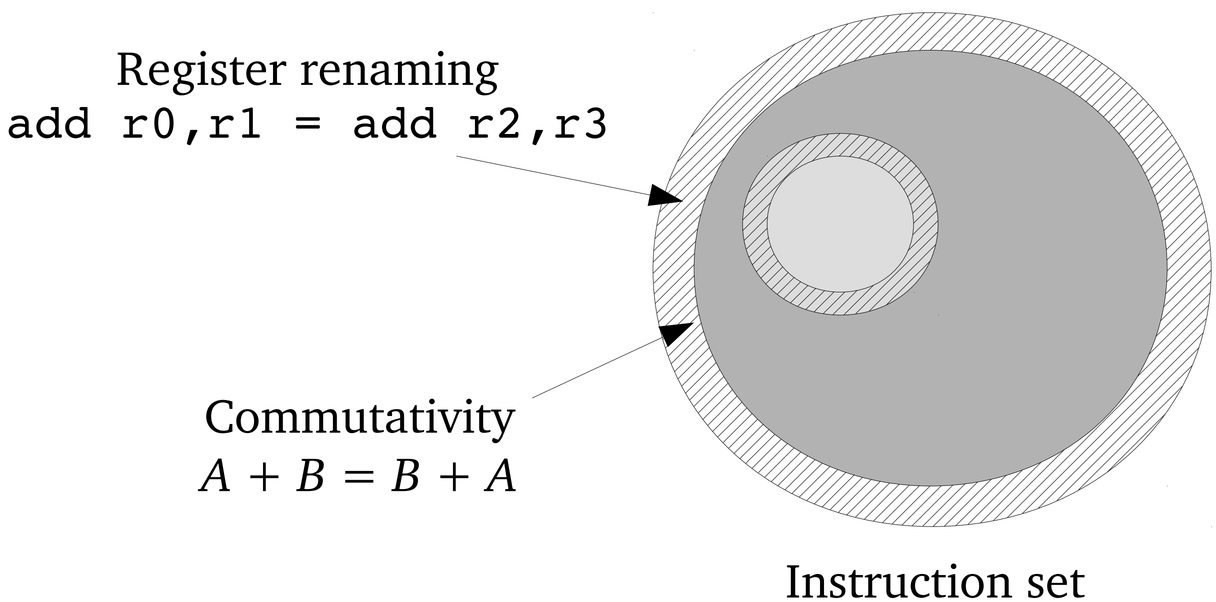


Instruction set

Superoptimization fundamentals: Pruning

Not all instruction sequences are valid.

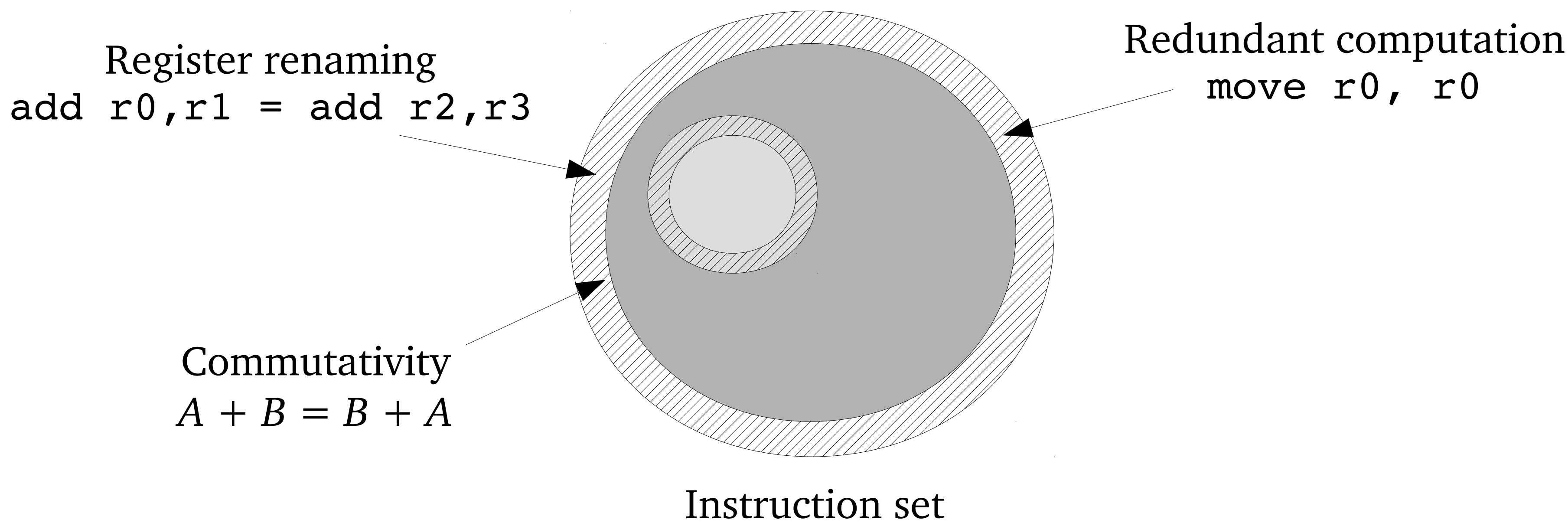
How do we quickly ignore bad sequences?



Superoptimization fundamentals: Pruning

Not all instruction sequences are valid.

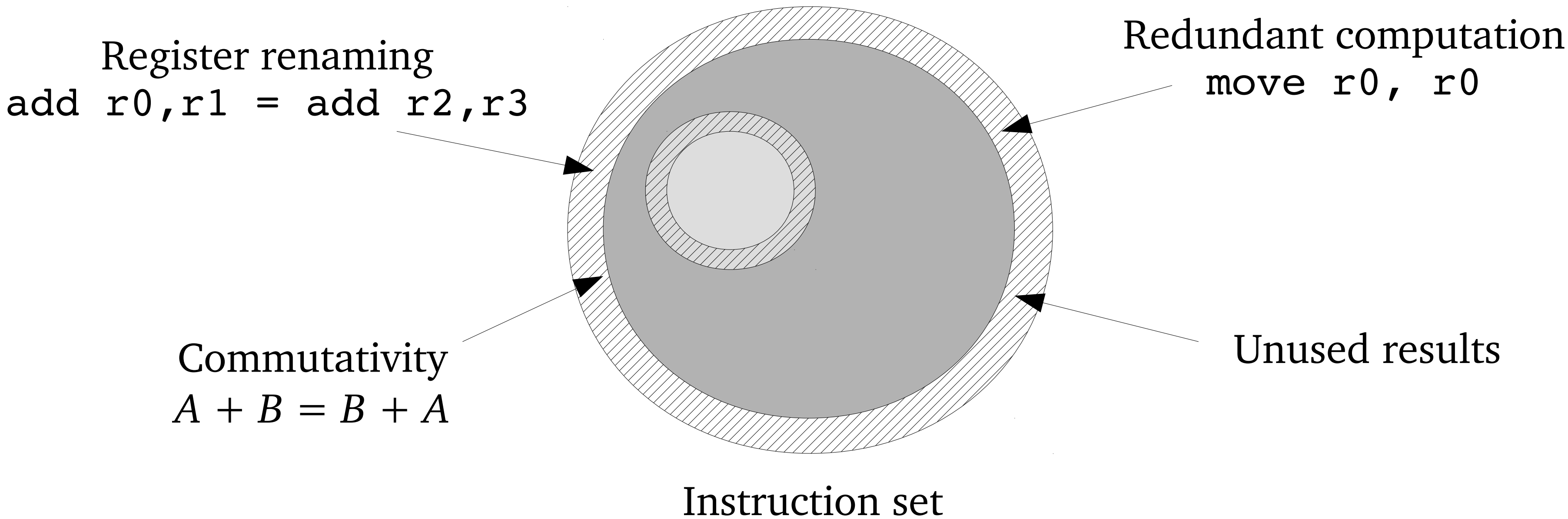
How do we quickly ignore bad sequences?



Superoptimization fundamentals: Pruning

Not all instruction sequences are valid.

How do we quickly ignore bad sequences?



Superoptimization fundamentals: Testing

Superoptimization fundamentals: Testing

Is the sequence correct?

Superoptimization fundamentals: Testing

Is the sequence correct?

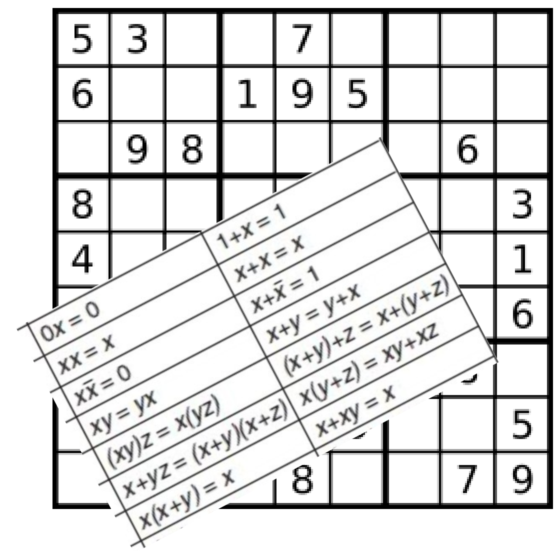
Testing
(simulation)

Superoptimization fundamentals: Testing

Is the sequence correct?

Testing
(simulation)

Mathematical proof
(symbolic solving)



University of BRISTOL

EMBECOSM®

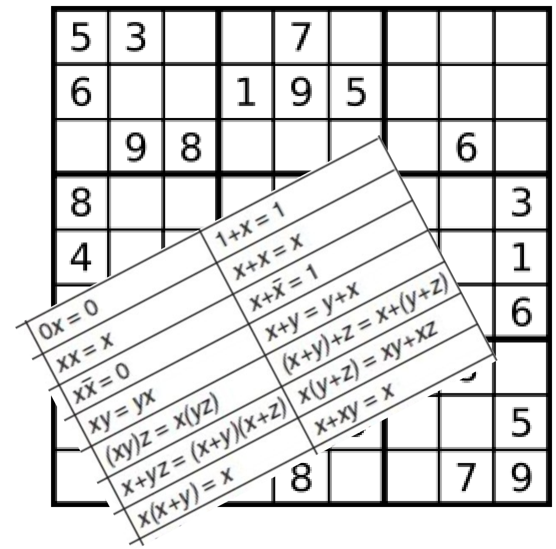
Superoptimization fundamentals: Testing

Is the sequence correct?

Testing
(simulation)

1. Choose some input
2. Run/simulate
3. Check output

Mathematical proof
(symbolic solving)



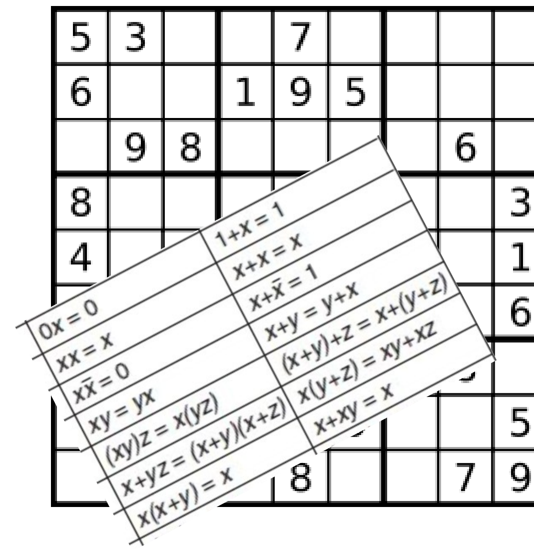
Superoptimization fundamentals: Testing

Is the sequence correct?

Testing
(simulation)

1. Choose some input
2. Run/simulate
3. Check output

Mathematical proof
(symbolic solving)



Formal verification
Proves the sequence correct
Slow

Superoptimization fundamentals: Testing

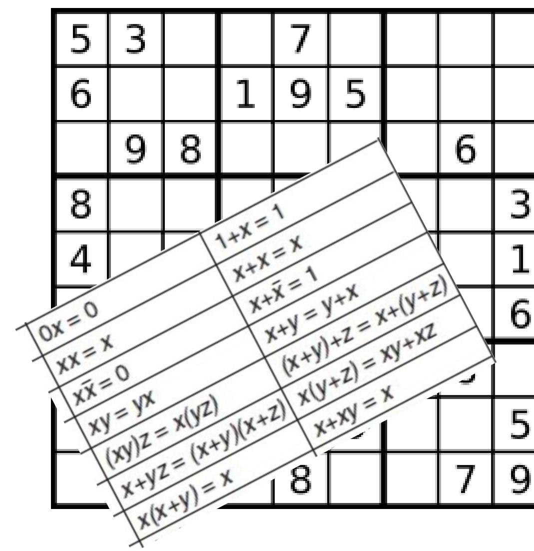
Is the sequence correct?

Testing
(simulation)

1. Choose some input
2. Run/simulate
3. Check output

Use Both

Mathematical proof
(symbolic solving)



Formal verification
Proves the sequence correct
Slow

Superoptimization fundamentals: Costing

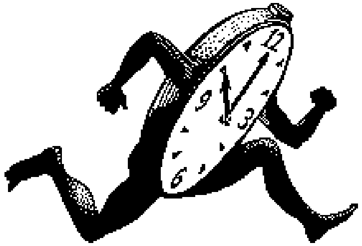
Superoptimization fundamentals: Costing

Which sequence is the best?

Superoptimization fundamentals: Costing

Which sequence is the best?

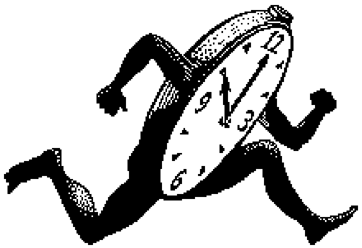
Execution time



Superoptimization fundamentals: Costing

Which sequence is the best?

Execution time

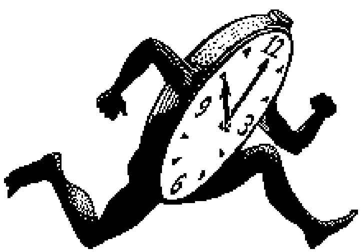


Code size

Superoptimization fundamentals: Costing

Which sequence is the best?

Execution time



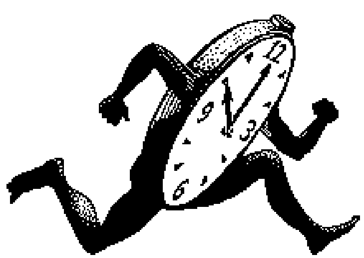
Code size

Energy consumption

Superoptimization fundamentals: Costing

Which sequence is the best?

Execution time



Code size

Energy consumption

If you can enumerate the instructions in cost order, the first correct sequence is the optimal sequence.

Plan for today

What is superoptimization?

Latest developments

The GNU Superoptimizer

University of BRISTOL

EMBECOSM®

Search space pruning

Search space pruning

Restrict parameters

- Registers
  - 50% of instruction sequences of length 8 use less than 4 registers
- Immediate constants
  - Frequently used constants: -16 to +16, 2^n, 2^n-1

Search space pruning

Restrict parameters

- Registers
  - 50% of instruction sequences of length 8 use less than 4 registers
- Immediate constants
  - Frequently used constants: -16 to +16, 2^n, 2^n-1

Remove meaningless constructs

- `mov r0, r0`
- `add r0, r0, #0`

Search space pruning

Restrict parameters

- Registers
  - 50% of instruction sequences of length 8 use less than 4 registers
- Immediate constants
  - Frequently used constants: -16 to +16, 2^n, 2^n-1

Remove meaningless constructs

- `mov r0, r0`
- `add r0, r0, #0`

Canonical form

Canonical form

`mov r1, r0` has many equivalent versions

Canonical form

`mov r1, r0` has many equivalent versions

Rename each register so they appear in sequence:

```
mov r1, r0   →
mov r4, r2   →   mov r1, r0
mov r2, r8   →
```

Canonical form

`mov r1, r0` has many equivalent versions

Rename each register so they appear in sequence:

```
mov r1, r0
mov r4, r2    →    mov r1, r0
mov r2, r8
```

With 16 registers this replaces 16*15 equivalent versions

Canonical form

```
add r4, r8, r1            add r2, r1, r0
orr r8, r4, #1    --->    orr r1, r2, #1
sub r1, r2, #8            sub r0, r3, #8
```

Canonical form

```
add r4, r8, r1
orr r8, r4, #1
sub r1, r2, #8
```

→

```
add r2, r1, r0
orr r1, r2, #1
sub r0, r3, #8
```

Single three operand instruction:

```
add rX, rX, rX
```

5 unique forms

→

```
add r0, r0, r0
add r0, r0, r1
add r0, r1, r0
add r0, r1, r1
add r0, r1, r2
```

Canonical form – reduction

Canonical form – reduction

Data processing instructions

- 16 ops, each using 3 of 16 possible registers.
- E.g.

```
add r0, r1, r2
sub r3, r4, r5
```

Canonical form – reduction

Data processing instructions

- 16 ops, each using 3 of 16 possible registers.
- E.g.

```
add r0, r1, r2
sub r3, r4, r5
```

Instructions	Normal	Canonical	Canonical (4 registers)
1	65,536	80	80
2	4,294,967,296	51,968	47,872
3	281,474,976,710,656	4,157,669,376	45,264,896
4	18,446,744,073,709,551,616	276,142,292,992	45,880,115,200

Canonical form – reduction

Data processing instructions

- 16 ops, each using 3 of 16 possible registers.
- E.g.

```
add r0, r1, r2
sub r3, r4, r5
```

Instructions	Normal	Canonical	Canonical (4 registers)
1	65,536	80	80
2	4,294,967,296	51,968	47,872
3	281,474,976,710,656	4,157,669,376	45,264,896
4	18,446,744,073,709,551,616	276,142,292,992	45,880,115,200

@200,000 tests/second   2.9 million years

Canonical form – reduction

Data processing instructions

- 16 ops, each using 3 of 16 possible registers.
- E.g.

```
add r0, r1, r2
sub r3, r4, r5
```

Instructions	Normal	Canonical	Canonical (4 registers)
1	65,536	80	80
2	4,294,967,296	51,968	47,872
3	281,474,976,710,656	4,157,669,376	45,264,896
4	18,446,744,073,709,551,616	276,142,292,992	45,880,115,200

@200,000 tests/second — 2.9 million years (Normal, 4 instructions); 16 days (Canonical, 4 instructions)

Canonical form – reduction

Data processing instructions

- 16 ops, each using 3 of 16 possible registers.
- E.g.

```
add r0, r1, r2
sub r3, r4, r5
```

Instructions	Normal	Canonical	Canonical (4 registers)
1	65,536	80	80
2	4,294,967,296	51,968	47,872
3	281,474,976,710,656	4,157,669,376	45,264,896
4	18,446,744,073,709,551,616	276,142,292,992	45,880,115,200

@200,000 tests/second — 2.9 million years (Normal, 4 instructions); 16 days (Canonical); <3 days (Canonical, 4 registers)

Instruction costing

University of BRISTOL

EMBECOSM®

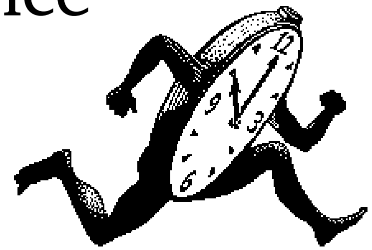
Instruction costing

Sequence cost is simple if code size is to be minimised

Instruction costing

Sequence cost is simple if code size is to be minimised

Difficult to accurately measure the performance of short sequences of instructions.



- Pipeline modelling
- Cycle accurate simulation

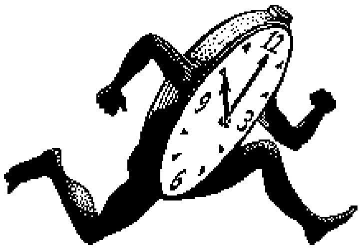
University of BRISTOL
EMBECOSM®

Instruction costing

Sequence cost is simple if code size is to be minimised

Difficult to accurately measure the performance of short sequences of instructions.

- Pipeline modelling
- Cycle accurate simulation



Energy

- Total Software Energy and Reporting (TSERO)

Instruction sets

Instruction sets

Characteristics of the instruction set affect how well a superoptimizer will perform.

Instruction sets

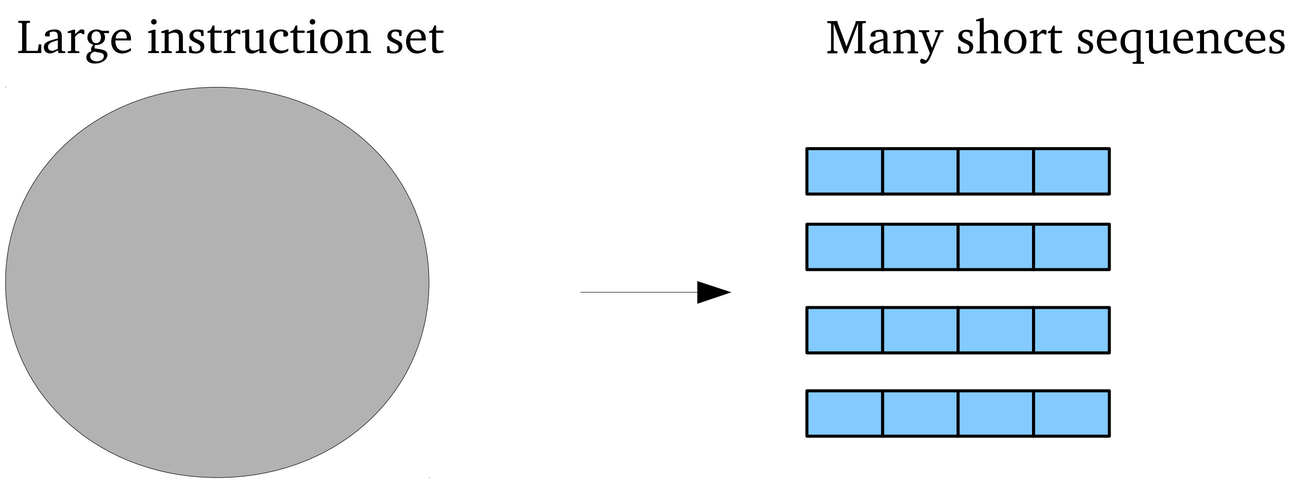
Characteristics of the instruction set affect how well a superoptimizer will perform.

Smaller instruction set → fewer optimal sequences (?)

Instruction sets

Characteristics of the instruction set affect how well a superoptimizer will perform.

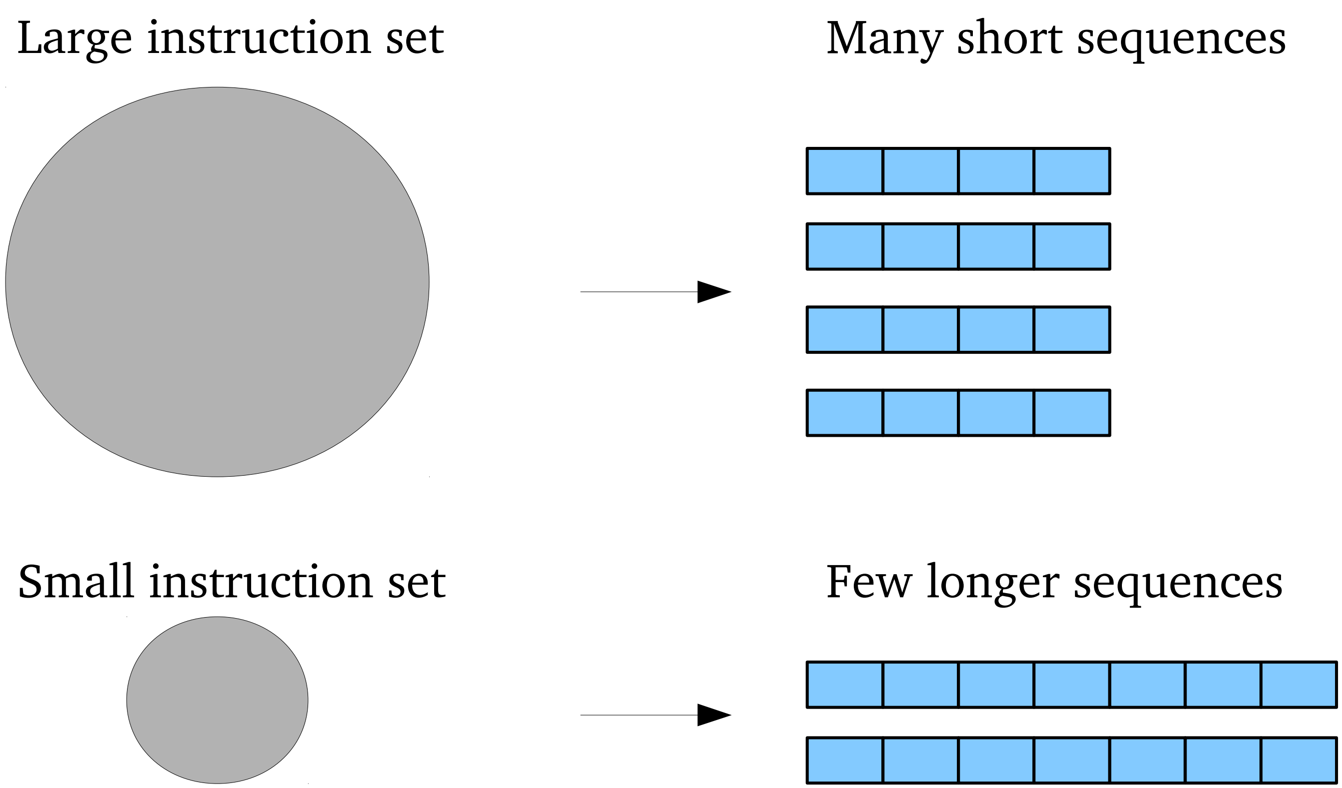
Smaller instruction set → fewer optimal sequences (?)



Instruction sets

Characteristics of the instruction set affect how well a superoptimizer will perform.

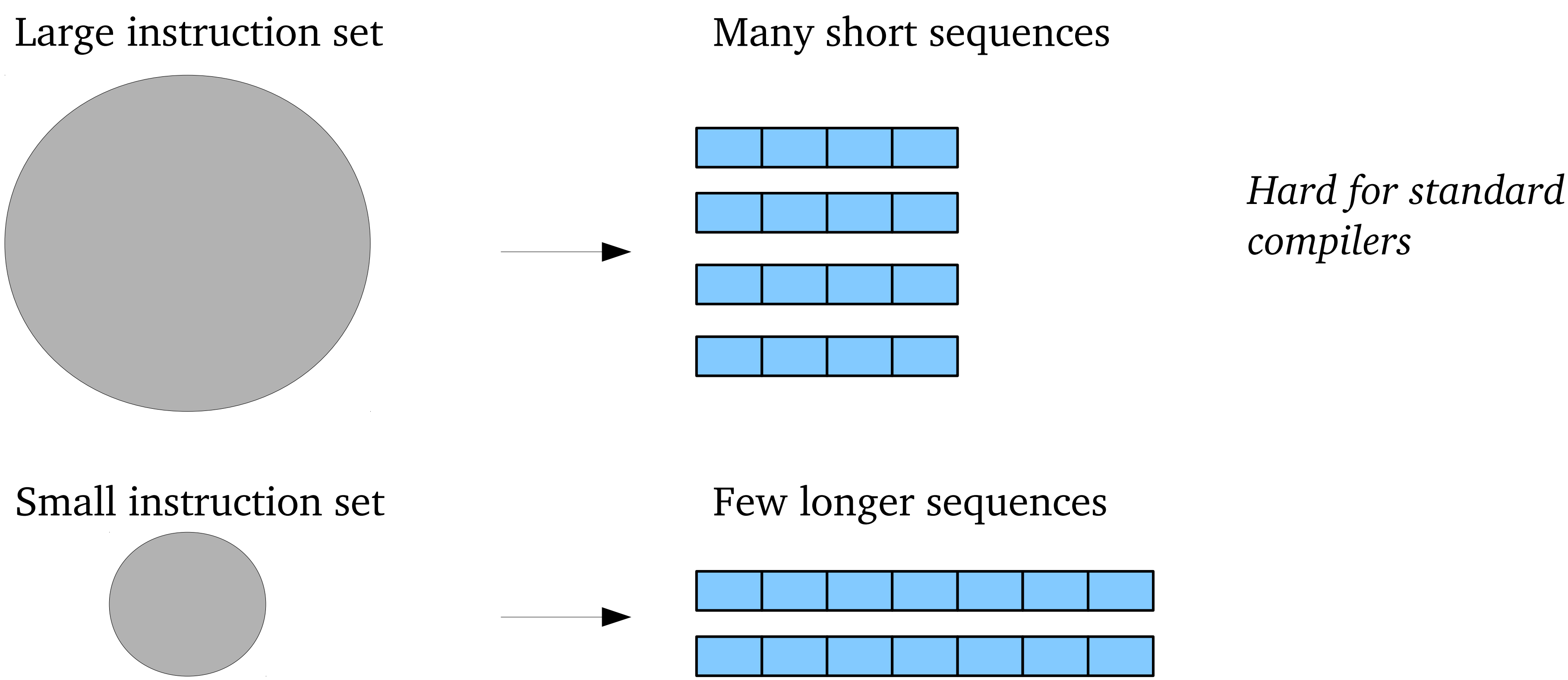
Smaller instruction set → fewer optimal sequences (?)



Instruction sets

Characteristics of the instruction set affect how well a superoptimizer will perform.

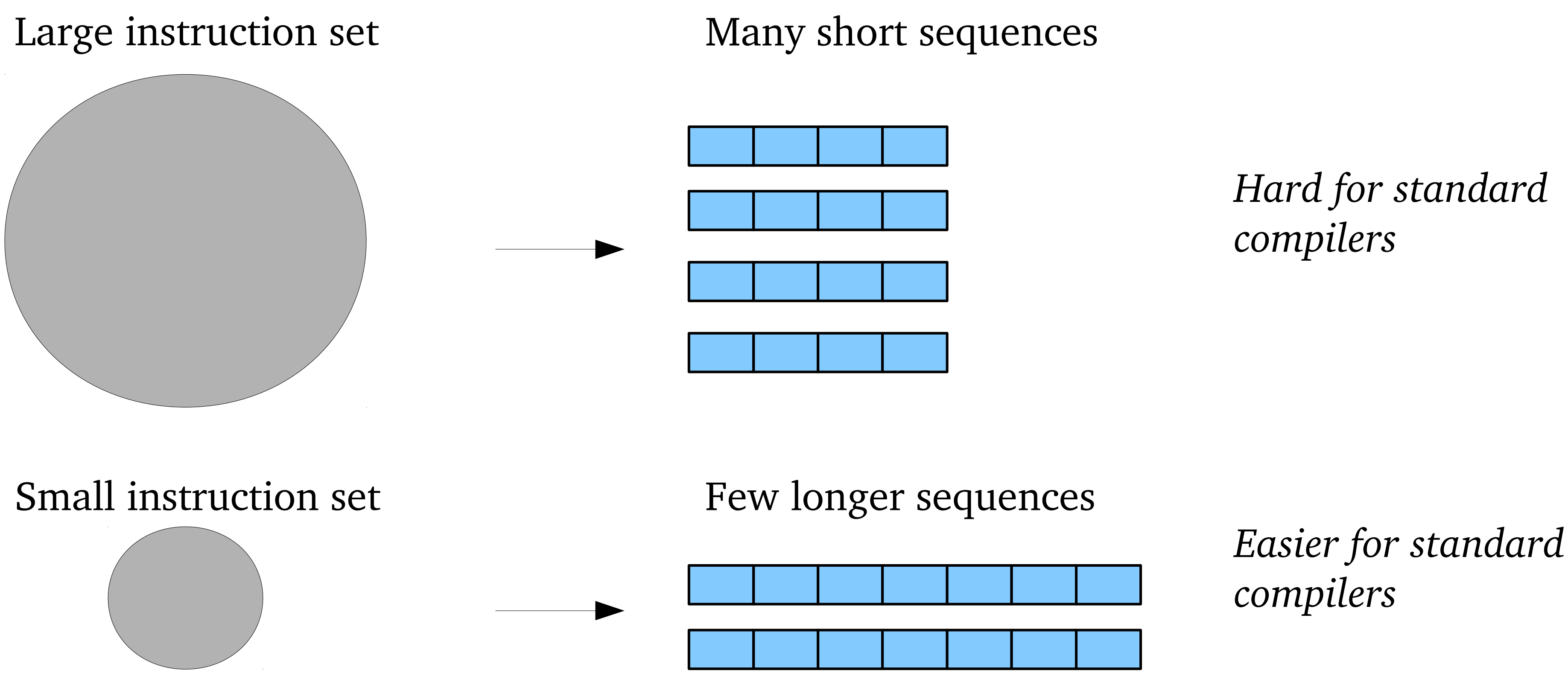
Smaller instruction set → fewer optimal sequences (?)



Instruction sets

Characteristics of the instruction set affect how well a superoptimizer will perform.

Smaller instruction set → fewer optimal sequences (?)



Superoptimizers

University of BRISTOL

EMBECOSM®

Superoptimizers

GNU SuperOptimizer (GSO)

Superoptimizers

GNU SuperOptimizer (GSO)

Denali

Superoptimizers

GNU SuperOptimizer (GSO)

Denali

Peephole Superoptimizer

Superoptimizers

GNU SuperOptimizer (GSO)

Denali

Peephole Superoptimizer

Stochastic Superoptimization

Superoptimizers

- GNU SuperOptimizer (GSO)
- Denali
- Peephole Superoptimizer
- Stochastic Superoptimization
- Other
  - A Hacker's Assistant (similar to GSO)
  - TOAST (similar to Denali)

GNU SuperOptimizer

Focuses on elimination of short basic blocks

Granlund, T., & Kenner, R. (1992). Eliminating branches using a superoptimizer and the GNU C compiler. PLDI '92 Proceedings of the ACM SIGPLAN 1992 conference on Programming language design and implementation.

GNU SuperOptimizer

Focuses on elimination of short basic blocks

```c
unsigned a, b, c;

if(a < b)
    c++;
```

→

```
    cmp     eax, ebx
    jge     L1
    add     ecx, #1
L1:
    ...
```

Granlund, T., & Kenner, R. (1992). Eliminating branches using a superoptimizer and the GNU C compiler. PLDI ’92 Proceedings of the ACM SIGPLAN 1992 conference on Programming language design and implementation.

GNU SuperOptimizer

Focuses on elimination of short basic blocks

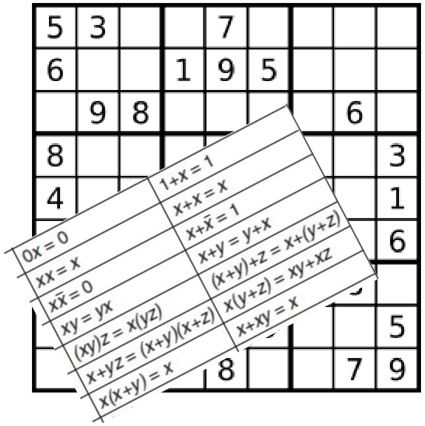
```
unsigned a, b, c;

if(a < b)                      cmp   eax, ebx
    c++;             →         jge   L1
                               add   ecx, #1
                           L1:
                               ...

c = (a < b) + c;     →         subl  eax, edx
                               adcl  ecx, #0
```

Granlund, T., & Kenner, R. (1992). Eliminating branches using a superoptimizer and the GNU C compiler. PLDI '92 Proceedings of the ACM SIGPLAN 1992 conference on Programming language design and implementation.

Denali



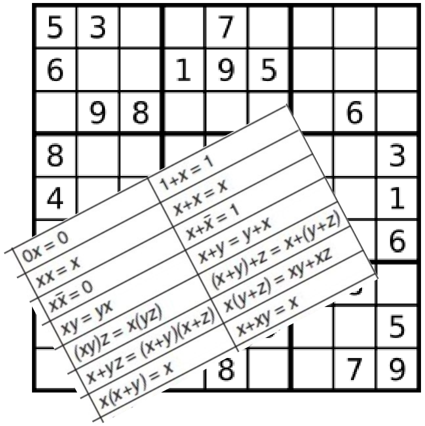
Joshi, R., Nelson, G., & Randall, K. (2001). Denali : a goal-directed superoptimizer. Proceedings of the ACM 2000 Conference on Programming Language Design and Implementation (pp. 304–314).

University of BRISTOL

EMBECOSM®

Denali

Rule based program generation:

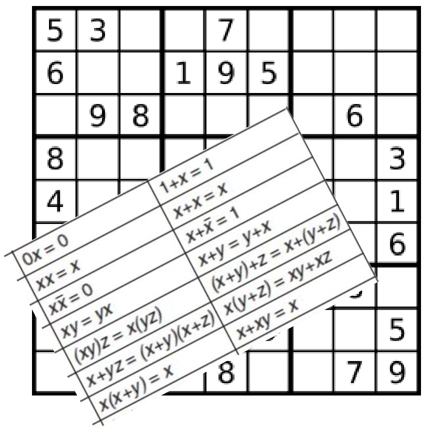


Joshi, R., Nelson, G., & Randall, K. (2001). Denali : a goal-directed superoptimizer. Proceedings of the ACM 2000 Conference on Programming Language Design and Implementation (pp. 304–314).

Denali

Rule based program generation:

- Prove the condition: there exists no instruction sequence that solves F(x) in < K cycles

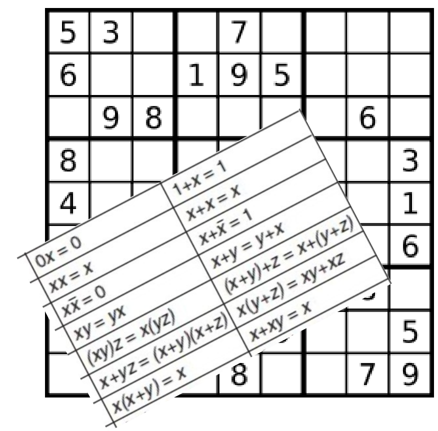


Joshi, R., Nelson, G., & Randall, K. (2001). Denali : a goal-directed superoptimizer. Proceedings of the ACM 2000 Conference on Programming Language Design and Implementation (pp. 304–314).

Denali

Rule based program generation:

- Prove the condition: there exists no instruction sequence that solves F(x) in < K cycles
- Failure yields an example sequence

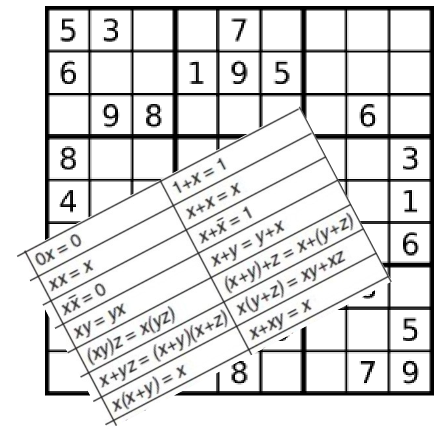


Joshi, R., Nelson, G., & Randall, K. (2001). Denali : a goal-directed superoptimizer. Proceedings of the ACM 2000 Conference on Programming Language Design and Implementation (pp. 304–314).

Denali

Rule based program generation:

- Prove the condition: there exists no instruction sequence that solves F(x) in < K cycles
- Failure yields an example sequence
- Use hand-written transformation rules

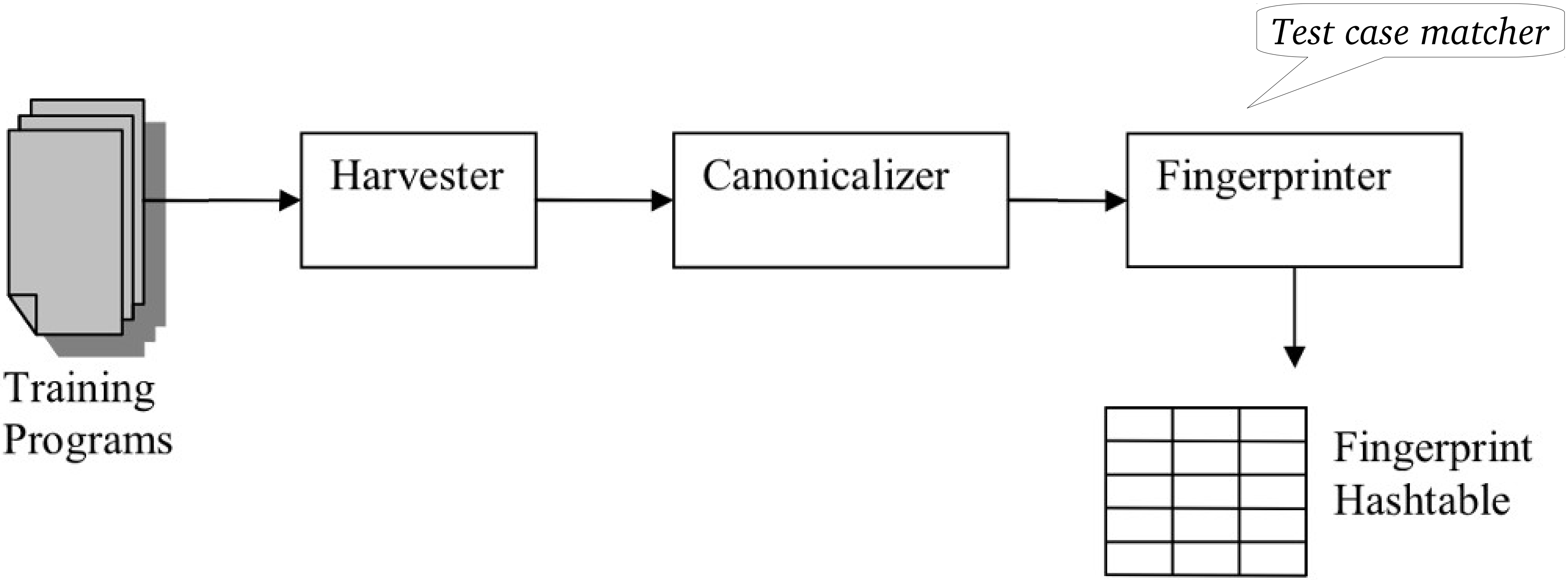



```
x * 2  ↔  x << 1
 a + b  ↔  b + a
```

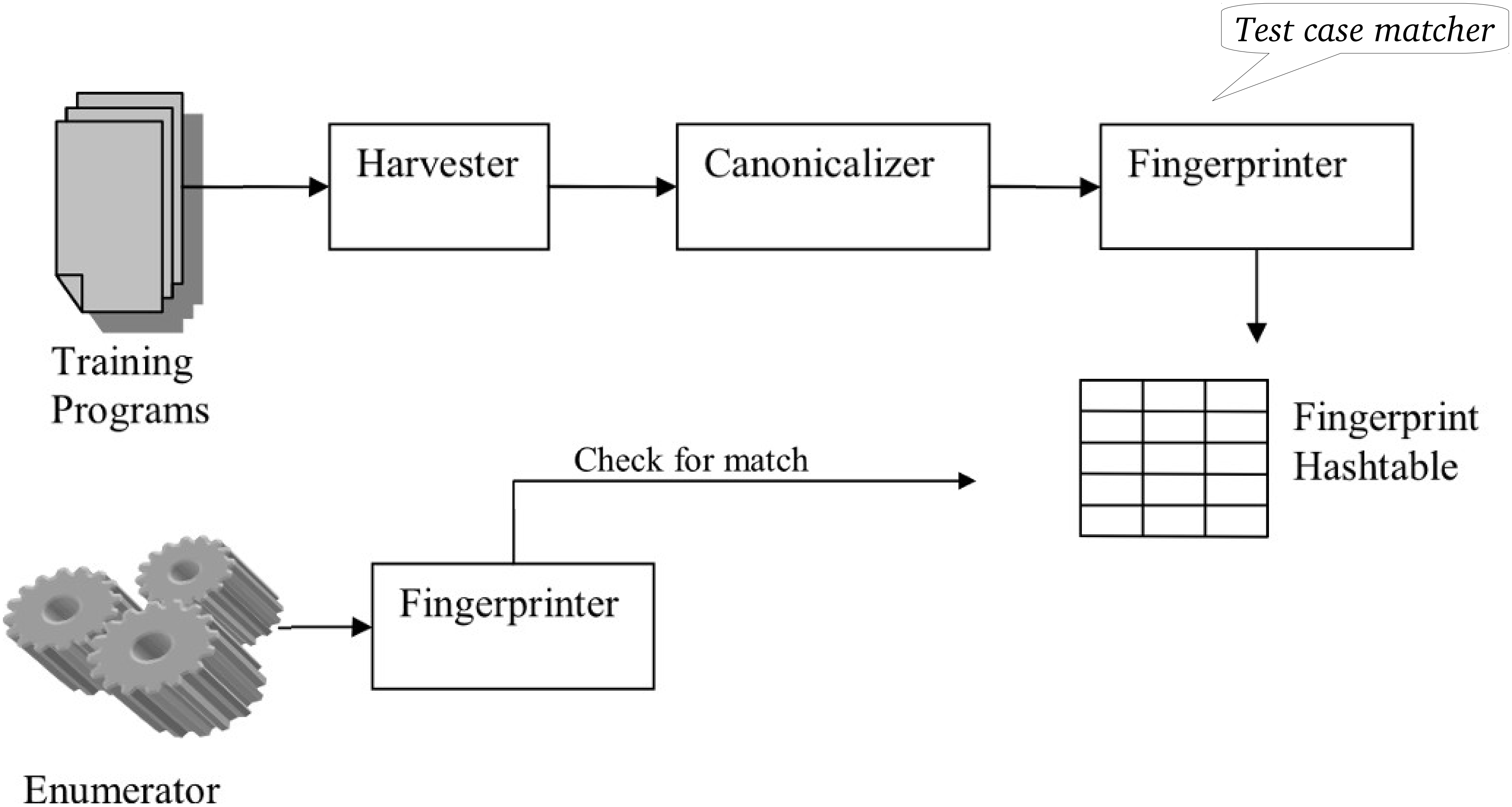
Joshi, R., Nelson, G., & Randall, K. (2001). Denali : a goal-directed superoptimizer. Proceedings of the ACM 2000 Conference on Programming Language Design and Implementation (pp. 304–314).

Peephole Superoptimizers

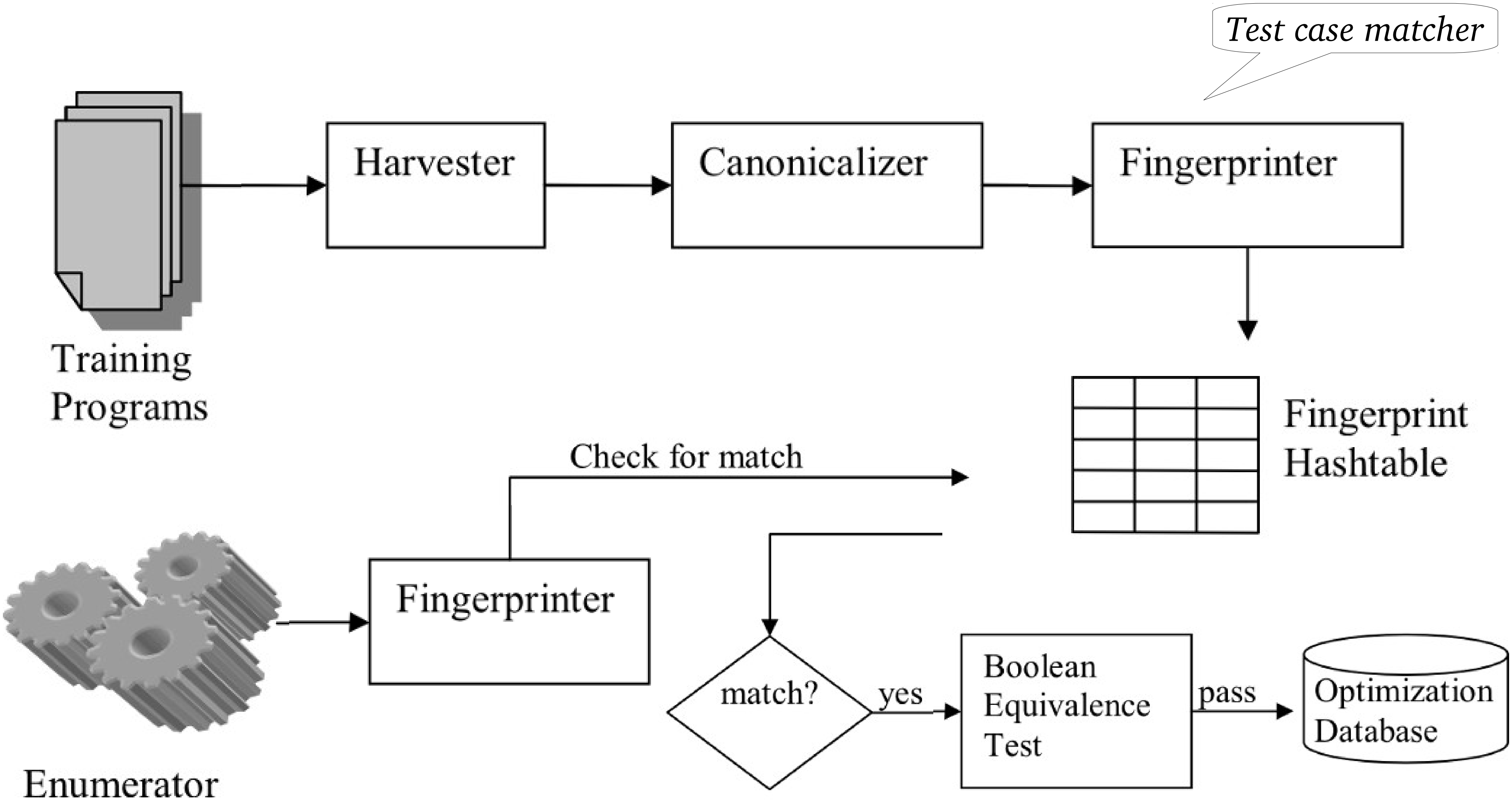
Peephole Superoptimizers



Peephole Superoptimizers



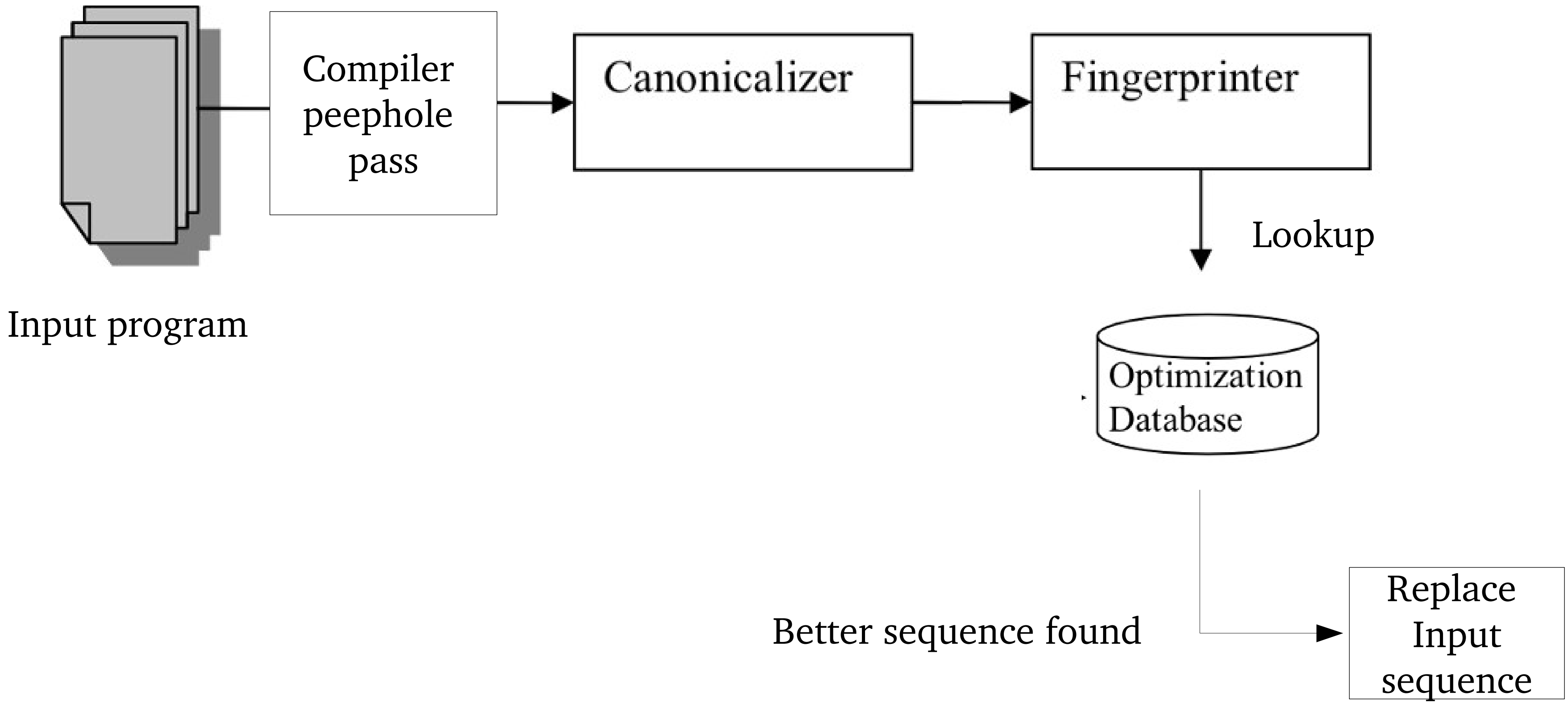
Peephole Superoptimizers



University of BRISTOL

EMBECOSM®

Peephole Superoptimizers



Stochastic superoptimization

Schkufza, E., Sharma, R., & Aiken, A. (2013). Stochastic superoptimization. Architectural Support for Programming Languages and Operating Systems, 305.

University of BRISTOL

EMBECOSM®

Stochastic superoptimization

A different approach to instruction sequence enumeration

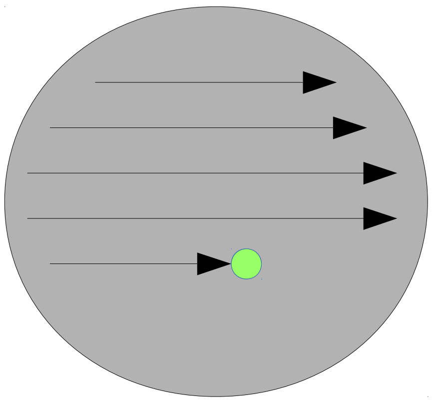
Schkufza, E., Sharma, R., & Aiken, A. (2013). Stochastic superoptimization. Architectural Support for Programming Languages and Operating Systems, 305.

University of BRISTOL

EMBECOSM®

Stochastic superoptimization

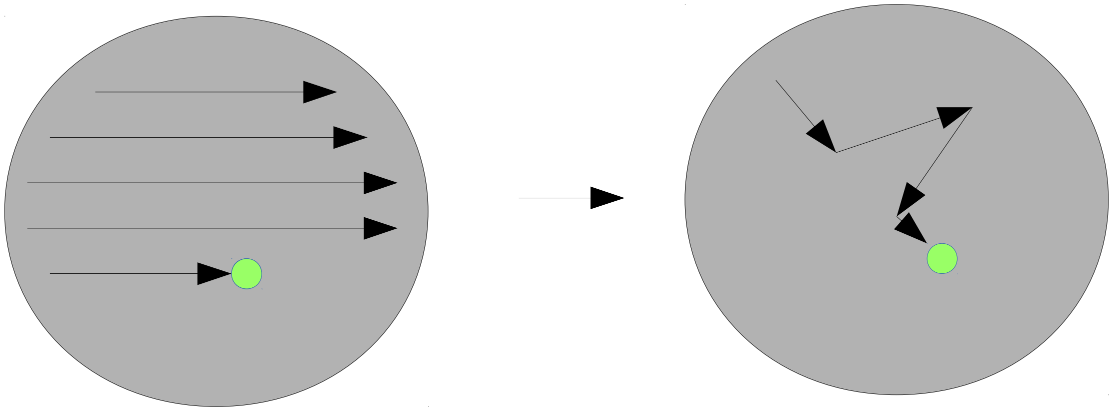
A different approach to instruction sequence enumeration



Schkufza, E., Sharma, R., & Aiken, A. (2013). Stochastic superoptimization. Architectural Support for Programming Languages and Operating Systems, 305.

Stochastic superoptimization

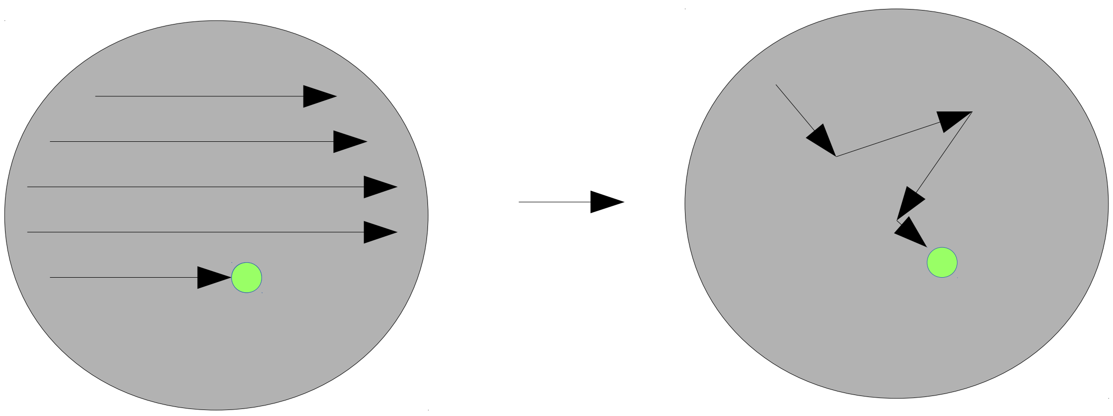
A different approach to instruction sequence enumeration



Schkufza, E., Sharma, R., & Aiken, A. (2013). Stochastic superoptimization. Architectural Support for Programming Languages and Operating Systems, 305.

Stochastic superoptimization

A different approach to instruction sequence enumeration



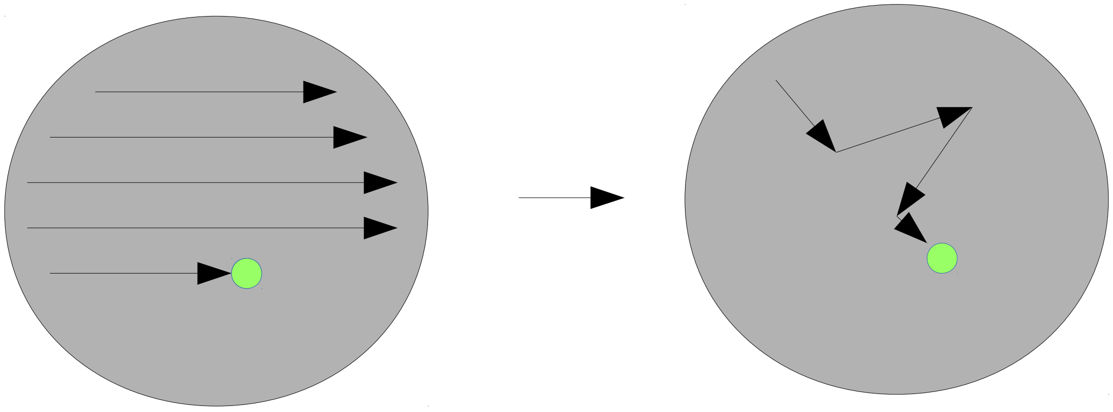
Longer sequences of instructions

- Sequences of >14 instructions were considered

Schkufza, E., Sharma, R., & Aiken, A. (2013). Stochastic superoptimization. Architectural Support for Programming Languages and Operating Systems, 305.

Stochastic superoptimization

A different approach to instruction sequence enumeration



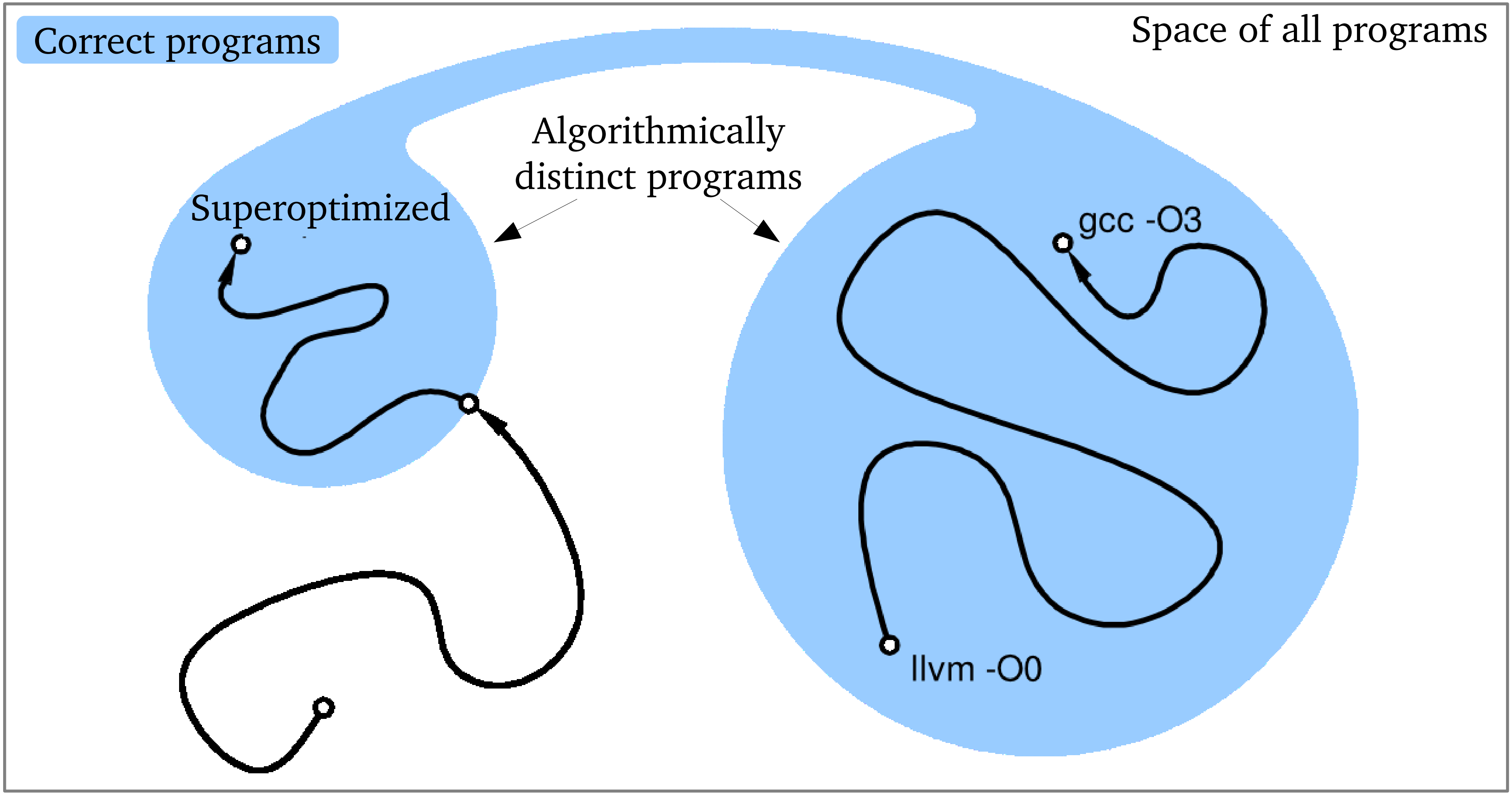
Longer sequences of instructions

- Sequences of >14 instructions were considered
- E.g. OpenSSL Montgomery multiplication 60% faster

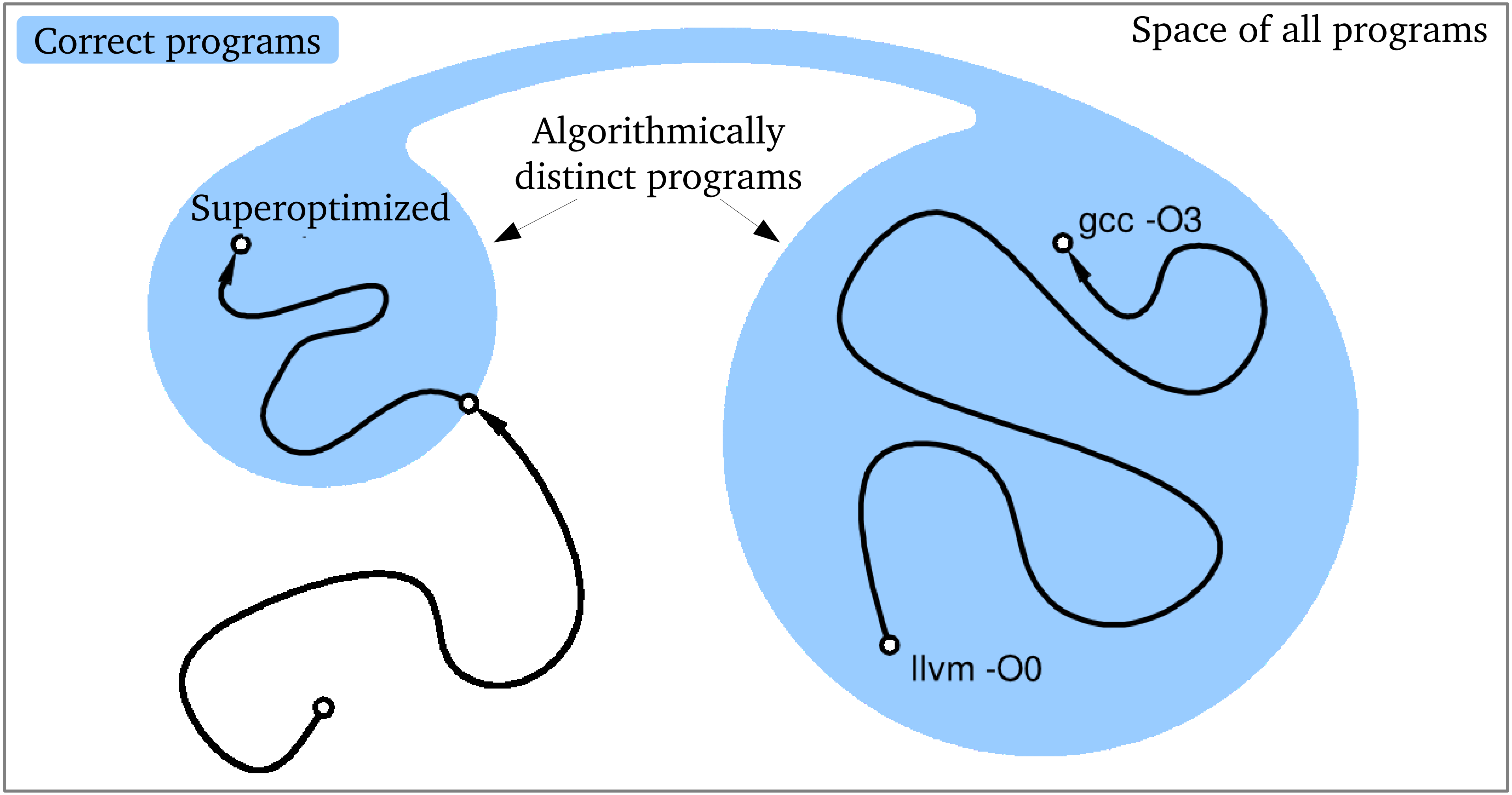
Schkufza, E., Sharma, R., & Aiken, A. (2013). Stochastic superoptimization. Architectural Support for Programming Languages and Operating Systems, 305.

Discovering new algorithms

Discovering new algorithms



Discovering new algorithms



Stochastic superoptimization's longer sequences make this more likely

Plan for today

What is superoptimization?

Latest developments **NEW**

The GNU SuperOptimizer

GSO: What can it do?

The sign function, AVR:

```
cp    r1, r24
brlt .+14
ldi   r25, 0x01
cpse r24, r1
rjmp .+2
ldi   r25, 0x00
mov   r24, r25
neg   r24
rjmp .+2
ldi   r24, 0x01
```

```
add r0, r0
mov r1, r0
sbc r1, r0
sub r1, r0
adc r1, r0
```

Compiler (-Os)
- 11 instructions
- 4-10 cycles (if r1 initialised to 0)

Superoptimizer:
- 5 instructions
- 5 cycles

How does it work?

Iterative deepening depth first search

- 1 instruction

How does it work?

Iterative deepening depth first search

- 1 instruction

add r0, r0 → Check

How does it work?

Iterative deepening depth first search

1 instruction

add r0, r0 → Check
sub r0, r0 → Check

How does it work?

Iterative deepening depth first search

1 instruction

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

How does it work?

- Iterative deepening depth first search
  - 2 instructions

How does it work?

Iterative deepening depth first search

- 2 instructions

```
add r0, r0
```

How does it work?

Iterative deepening depth first search

2 instructions

add r0, r0 → Check

add r0, r0

How does it work?

Iterative deepening depth first search

2 instructions

add r0, r0 → add r0, r0 → Check

sub r0, r0 → Check

How does it work?

Iterative deepening depth first search

2 instructions

add r0, r0 →

add r0, r0 → Check

sub r0, r0 → Check

mul r0, r0 → Check

How does it work?

Iterative deepening depth first search

2 instructions

add r0, r0
sub r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

How does it work?

Iterative deepening depth first search

2 instructions

add r0, r0
sub r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

add r0, r0 → Check

How does it work?

Iterative deepening depth first search

2 instructions

add r0, r0
sub r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

add r0, r0 → Check
sub r0, r0 → Check

How does it work?

Iterative deepening depth first search

2 instructions

add r0, r0
sub r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

How does it work?

Iterative deepening depth first search

2 instructions

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

add r0, r0
sub r0, r0
mul r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

How does it work?

Iterative deepening depth first search

2 instructions

add r0, r0
sub r0, r0
mul r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

...

University of BRISTOL

EMBECOSM®

How does it work?

- Iterative deepening depth first search
  - 3 instructions

How does it work?

Iterative deepening depth first search

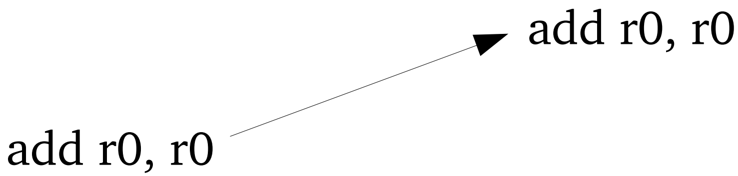
- 3 instructions

```
add r0, r0
```

How does it work?

Iterative deepening depth first search

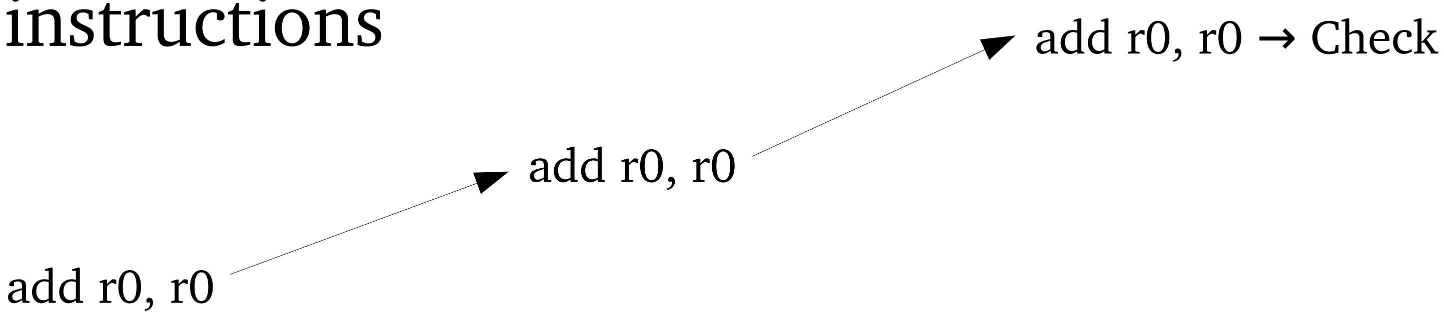
- 3 instructions



How does it work?

Iterative deepening depth first search

3 instructions



How does it work?

Iterative deepening depth first search

3 instructions

add r0, r0 → add r0, r0 → add r0, r0 → Check

sub r0, r0 → Check

How does it work?

Iterative deepening depth first search

3 instructions

add r0, r0 → add r0, r0 →

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

University of BRISTOL

EMBECOSM®

How does it work?

Iterative deepening depth first search

3 instructions

add r0, r0

add r0, r0
sub r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

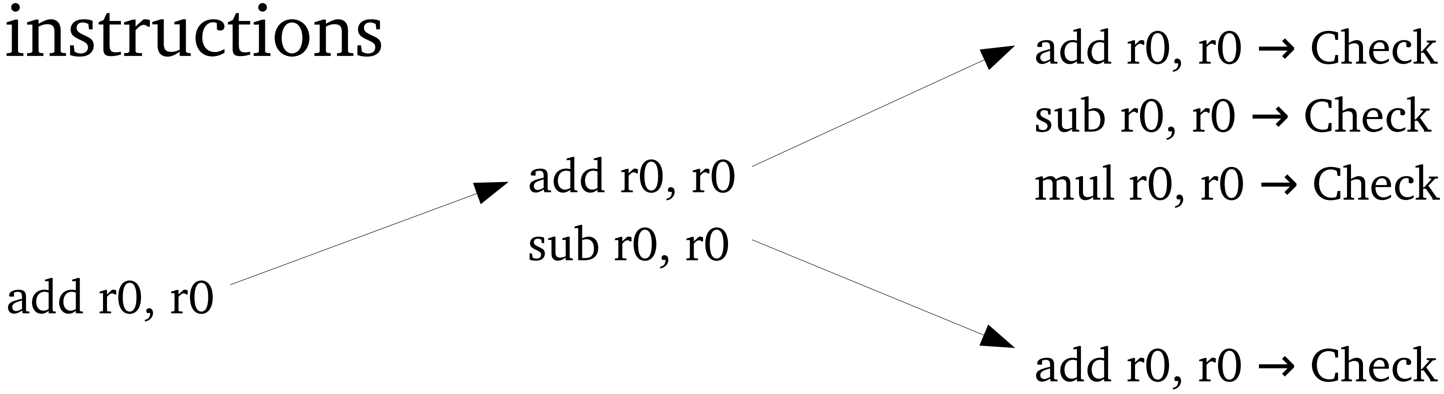
University of BRISTOL

EMBECOSM®

How does it work?

Iterative deepening depth first search

3 instructions



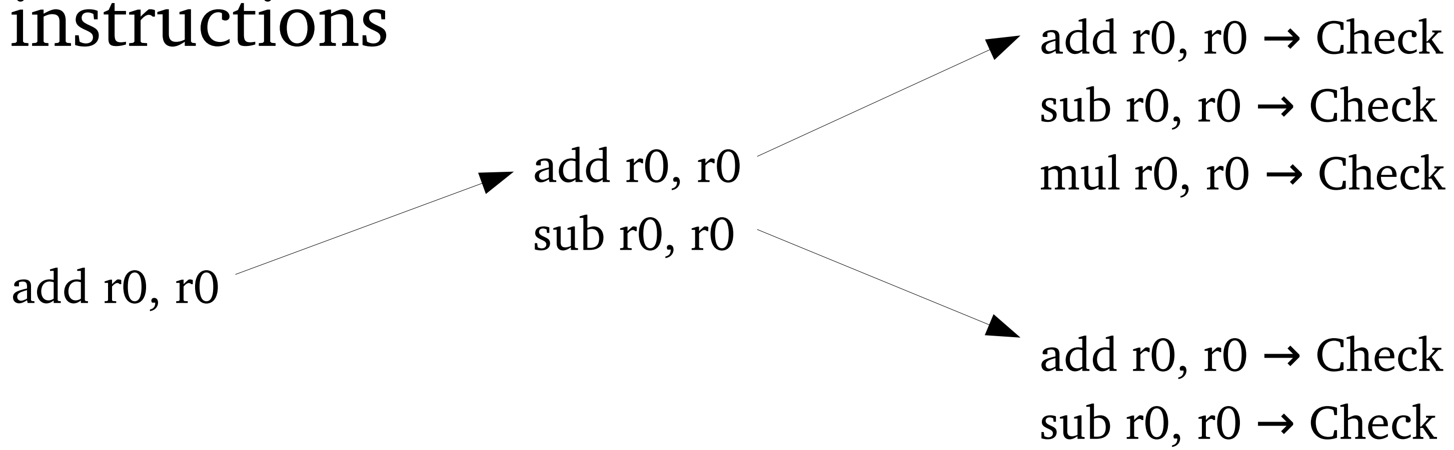
University of BRISTOL

EMBECOSM®

How does it work?

Iterative deepening depth first search

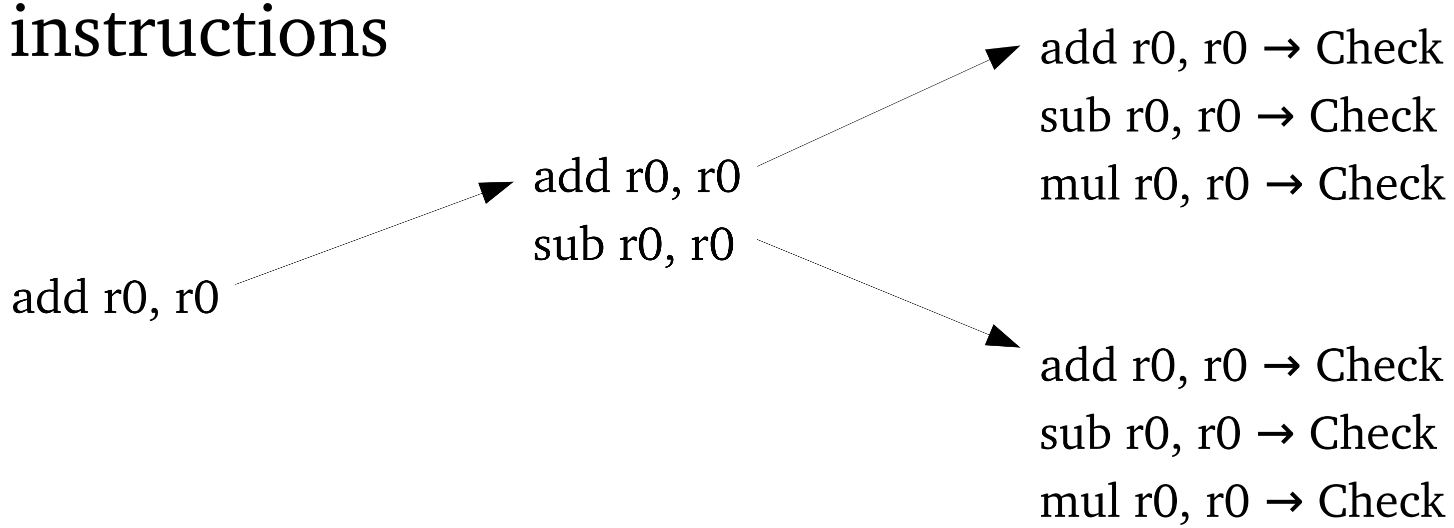
3 instructions



How does it work?

Iterative deepening depth first search

3 instructions



How does it work?

Iterative deepening depth first search

3 instructions

- add r0, r0
  - add r0, r0
    - add r0, r0 → Check
    - sub r0, r0 → Check
    - mul r0, r0 → Check
  - sub r0, r0
    - add r0, r0 → Check
    - sub r0, r0 → Check
    - mul r0, r0 → Check
  - mul r0, r0

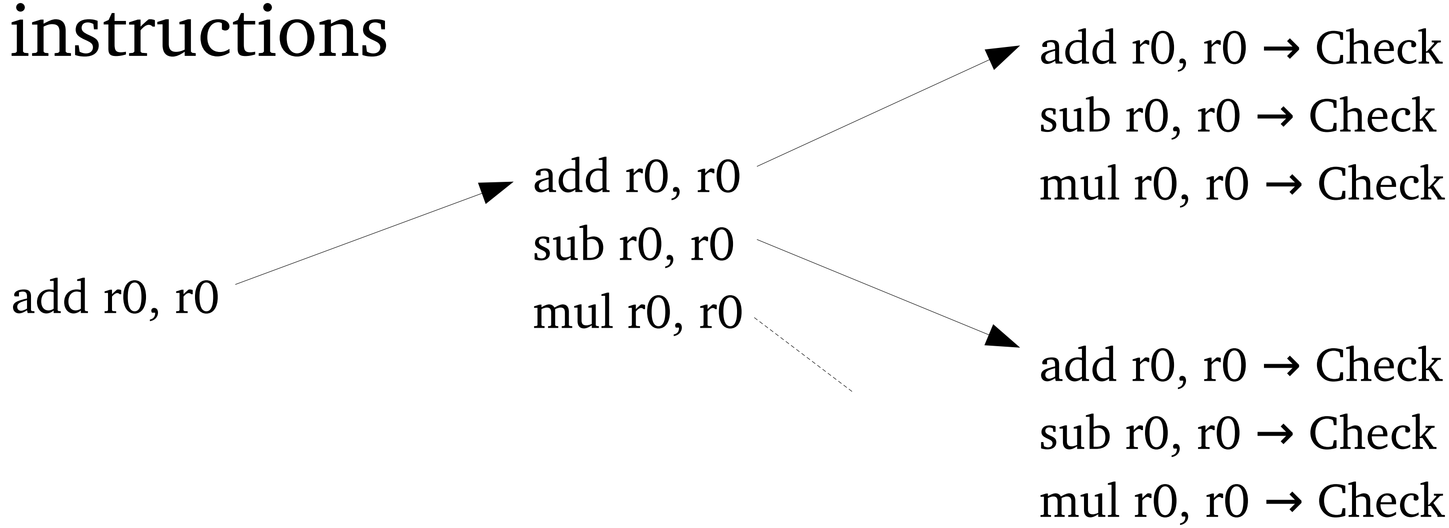
University of BRISTOL

EMBECOSM®

How does it work?

Iterative deepening depth first search

3 instructions



...

How does it work?

Iterative deepening depth first search

3 instructions

add r0, r0
sub r0, r0

add r0, r0
sub r0, r0
mul r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

...

How does it work?

Iterative deepening depth first search

3 instructions

add r0, r0
sub r0, r0

add r0, r0
sub r0, r0
mul r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

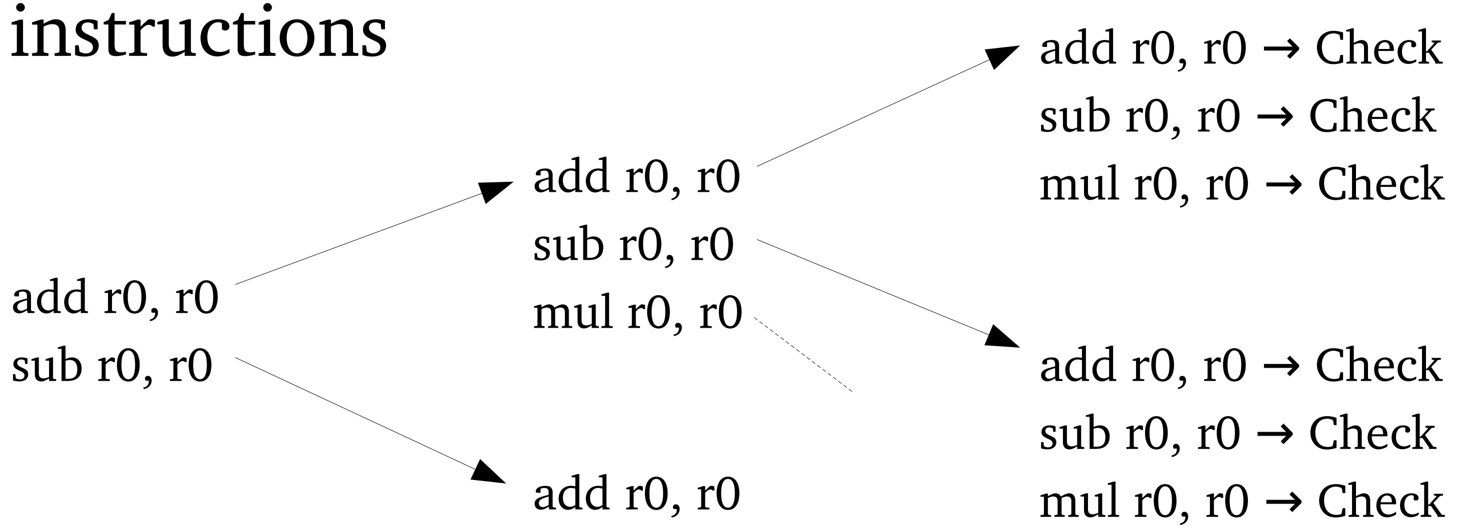
add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

...

How does it work?

Iterative deepening depth first search

3 instructions

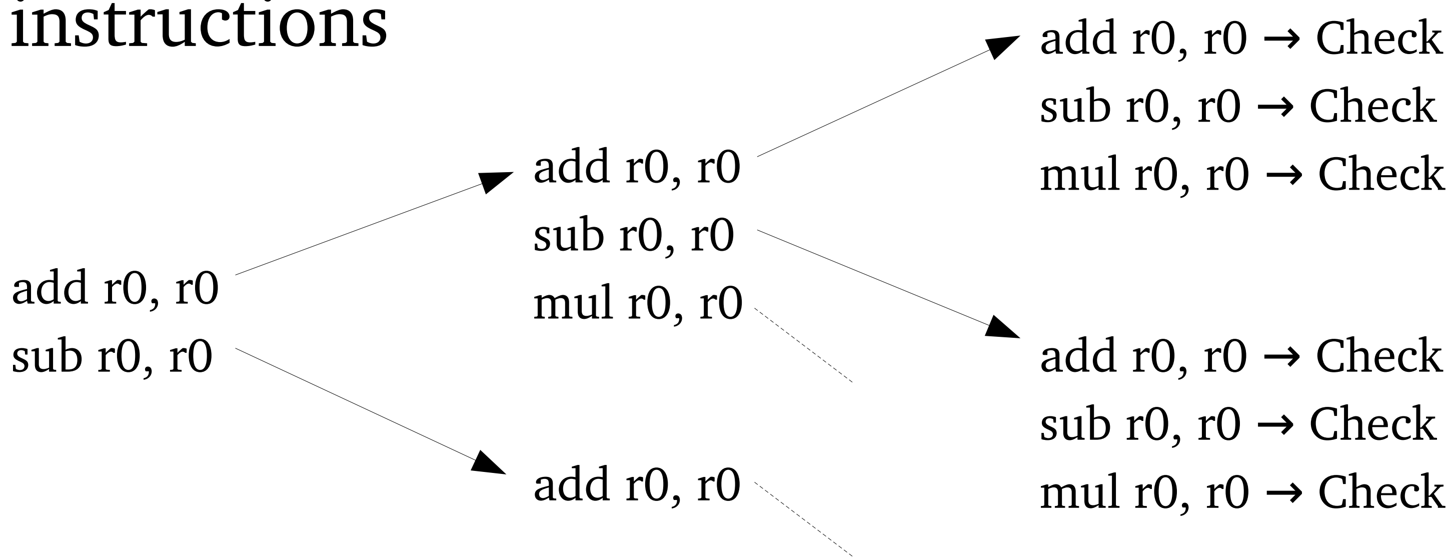


...

How does it work?

Iterative deepening depth first search

3 instructions



...

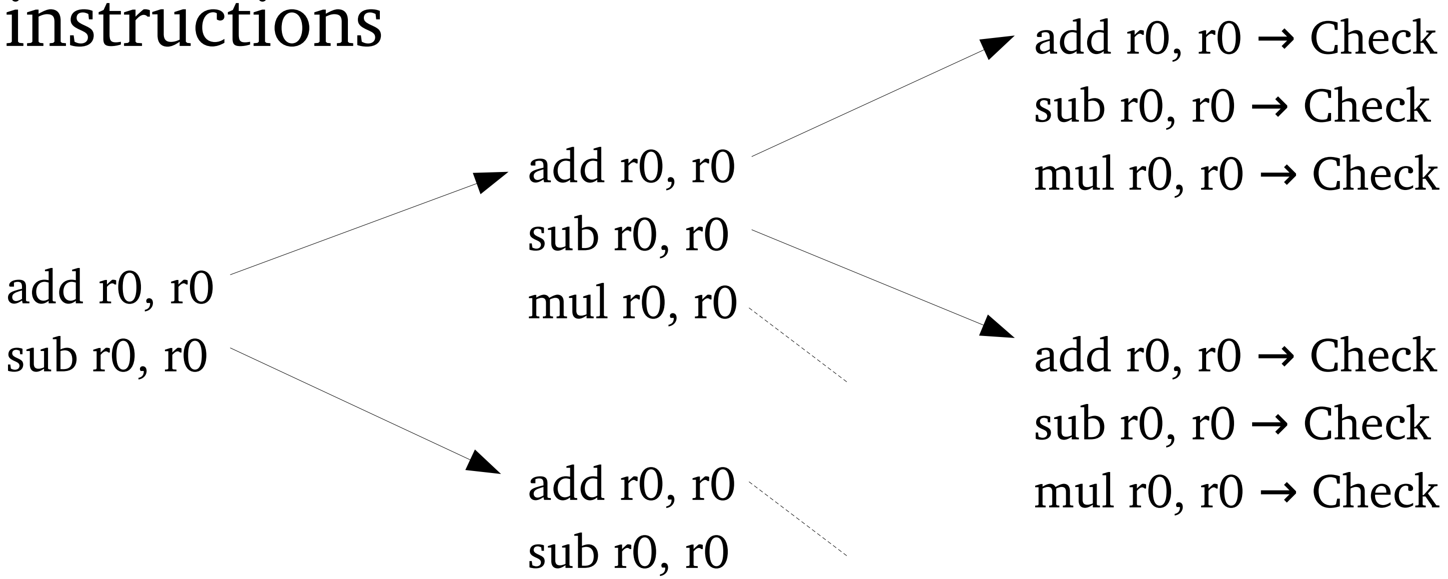
University of BRISTOL

EMBECOSM®

How does it work?

Iterative deepening depth first search

3 instructions



...

How does it work?

Iterative deepening depth first search

3 instructions

add r0, r0
sub r0, r0

add r0, r0
sub r0, r0
mul r0, r0

add r0, r0
sub r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

...

How does it work?

Iterative deepening depth first search

3 instructions

add r0, r0
sub r0, r0

add r0, r0
sub r0, r0
mul r0, r0

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

add r0, r0 → Check
sub r0, r0 → Check
mul r0, r0 → Check

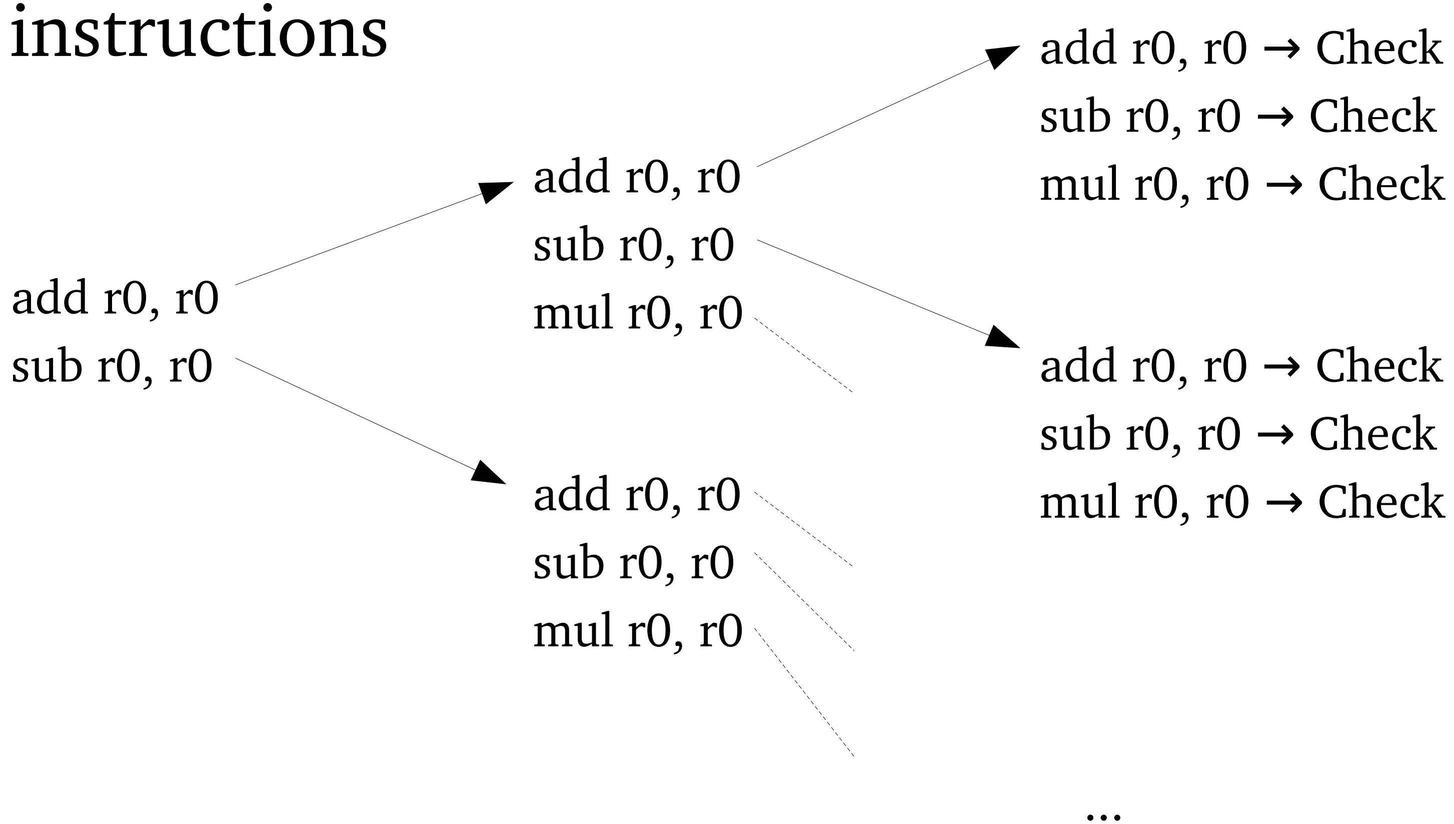
add r0, r0
sub r0, r0
mul r0, r0

...

How does it work?

Iterative deepening depth first search

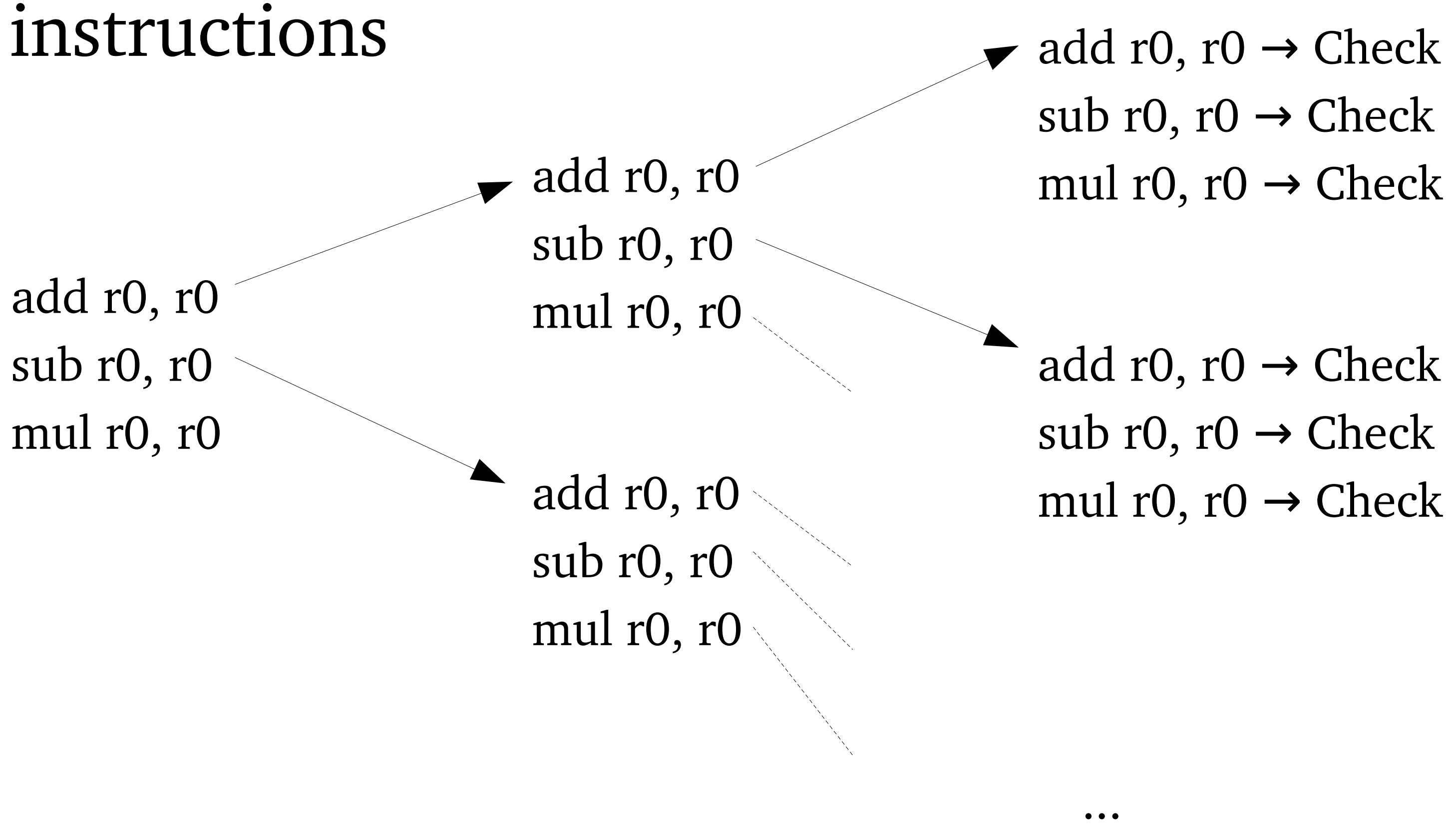
3 instructions



How does it work?

Iterative deepening depth first search

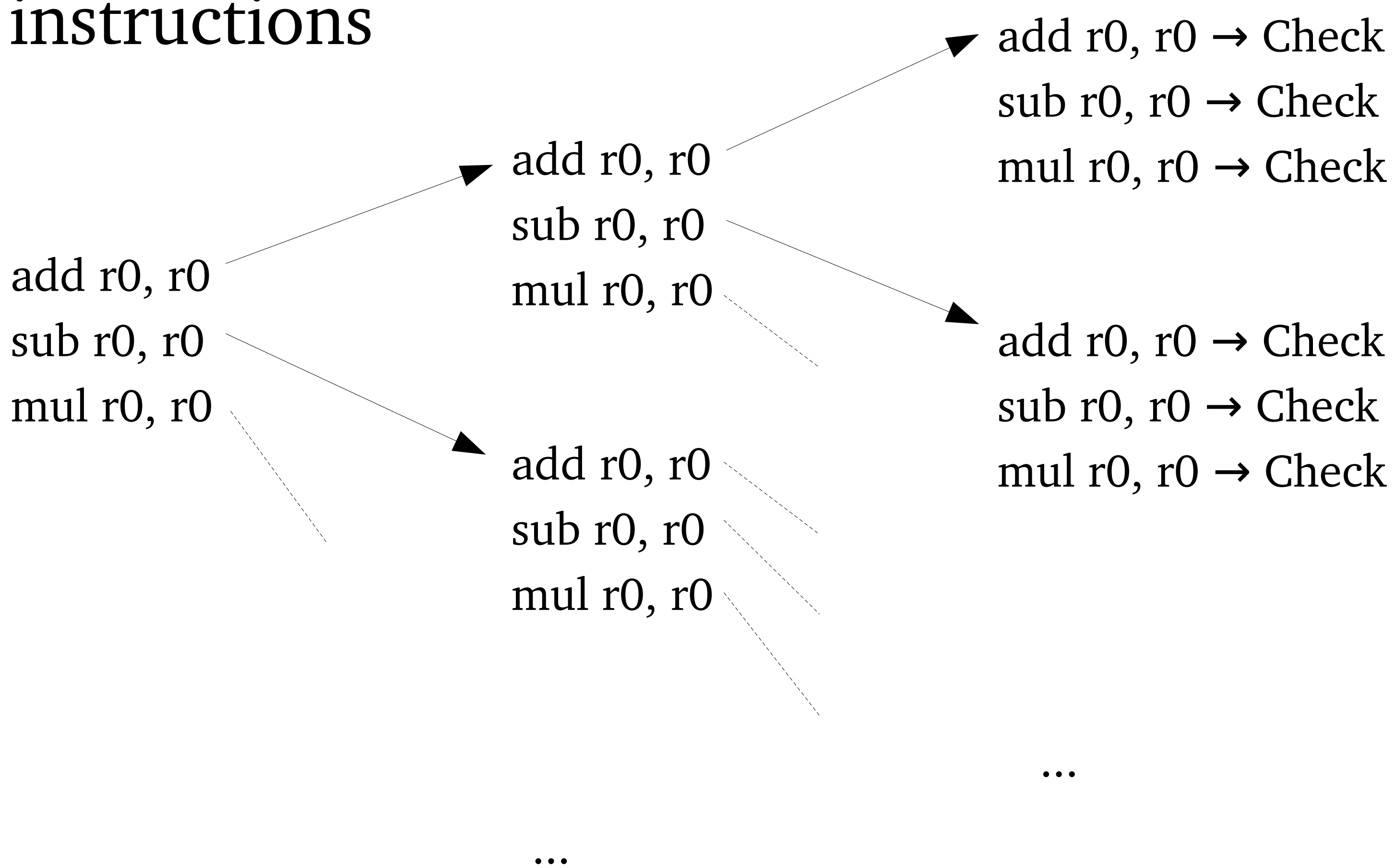
3 instructions



How does it work?

Iterative deepening depth first search

3 instructions



University of BRISTOL
EMBECOSM®

How to add a new goal

University of BRISTOL

EMBECOSM®

How to add a new goal

Goal is GSO's term for a target function to superoptimize

How to add a new goal

Goal is GSO's term for a target function to superoptimize

File: `goal.def`

How to add a new goal

Goal is GSO's term for a target function to superoptimize

File: `goal.def`

Examples:

```
DEF_GOAL (SHIFTL_1, 1, "sll1", { r = v0 << 1; })
DEF_GOAL (SHIFTL_2, 1, "sll2", { r = v0 << 2; })
```

How to add a new goal

Goal is GSO's term for a target function to superoptimize

File: `goal.def`

Examples:

```
DEF_GOAL (SHIFTL_1, 1, "sll1", { r = v0 << 1; })
DEF_GOAL (SHIFTL_2, 1, "sll2", { r = v0 << 2; })
DEF_GOAL (SIGNUM,   1, "signum",
  {
    if(v0 < 0)
      r = -1;
    else if(v0 > 0)
      r = 1;
    else
      r = 0;
  })
```

University of BRISTOL

EMBECOSM®

How to add a new goal

How to add a new goal

```
DEF_GOAL (EQ_MINUS, 3, "eq-",  { r = v2 - (v0 == v1); })
```

How to add a new goal

```
DEF_GOAL (EQ_MINUS, 3, "eq-",  { r = v2 - (v0 == v1); })
```

Format:

DEF_GOAL (*unique id*, *#args*, "*name*", { *C code*; })

r	result
v0	argument 1
v1	argument 2
...	

How to add a new goal

```
DEF_GOAL (EQ_MINUS, 3, "eq-",  { r = v2 - (v0 == v1); })
```

Format:

DEF_GOAL (*unique id*, *#args*, "*name*", { *C code*; })

r	result
v0	argument 1
v1	argument 2
...	

Using the new goal:

$ make all

$ ./superopt-avr -f*name*

University of BRISTOL

EMBECOSM®

How to add a new architecture

How to add a new architecture

Define the implementation in `superopt.h`

- Name, types (bitwidth)

How to add a new architecture

Define the implementation in `superopt.h`

- Name, types (bitwidth)
- “Unusual” instructions

```
#ifndef PERFORM_AND
#define PERFORM_AND(d, co, r1, r2, ci) \
  ((d) = (r1) & (r2), (co) = (ci))
#endif
```

How to add a new architecture

Define the implementation in `superopt.h`

- Name, types (bitwidth)
- "Unusual" instructions

```
#ifndef PERFORM_AND
#define PERFORM_AND(d, co, r1, r2, ci) \
  ((d) = (r1) & (r2), (co) = (ci))
#endif
```

- Also add mnemonic into `insn.def`

```
DEF_INSN (AND,'b',"and")
```

Identifier

Type (binary, unary, etc)

Name

University of BRISTOL

EMBECOSM®

How to add a new architecture

How to add a new architecture

To enable output in native assembler

How to add a new architecture

To enable output in native assembler

Extend `output_assembly`

- Register names
- Disassembly code

How to add a new architecture

To enable output in native assembler

Extend `output_assembly`

- Register names
- Disassembly code

```
case ADD_CO:
  printf("add       %s,%s",NAME(s1),NAME(s2)); break;
case ADD_CIO:
  printf("adc       %s,%s",NAME(s1),NAME(s2)); break;
```

How to add a new architecture

How to add a new architecture

`SYNTH(...)` in `synth.def`

How to add a new architecture

`SYNTH(...)` in `synth.def`

- Where the magic happens

How to add a new architecture

`SYNTH(...)` in `synth.def`

- Where the magic happens
- Loops over all the registers and all the instructions

How to add a new architecture

`SYNTH(...)` in `synth.def`

- Where the magic happens
- Loops over all the registers and all the instructions
- Use the macros for performing the instruction

How to add a new architecture

`SYNTH(...)` in `synth.def`

- Where the magic happens
- Loops over all the registers and all the instructions
- Use the macros for performing the instruction
- Then recurses (next level of iterative deepening)

How to add a new architecture

`SYNTH(...)` in `synth.def`

- Where the magic happens
- Loops over all the registers and all the instructions
- Use the macros for performing the instruction
- Then recurses (next level of iterative deepening)
- Add architecture entries for standard & custom instructions

How to add a new architecture

`SYNTH(...)` in `synth.def`

- Where the magic happens
- Loops over all the registers and all the instructions
- Use the macros for performing the instruction
- Then recurses (next level of iterative deepening)
- Add architecture entries for standard & custom instructions

```
#if SPARC || M88000
    /* sparc:          addx
       m88000:         addu.ci */
    PERFORM_ADD_CI(v, co, r1, r2, ci);
    RECURSE(ADD_CI, s1, s2, prune_hint & ~CY_JUST_SET);
#endif
```

How to add a new architecture

```
#if SPARC || M88000
    /* sparc:              addx
       m88000:             addu.ci */
    PERFORM_ADD_CI(v, co, r1, r2, ci);
    RECURSE(ADD_CI, s1, s2, prune_hint & ~CY_JUST_SET);
#endif
```

How to add a new architecture

```
#if SPARC || M88000
    /* sparc:          addx
       m88000:         addu.ci */
    PERFORM_ADD_CI(v, co, r1, r2, ci);
    RECURSE(ADD_CI, s1, s2, prune_hint & ~CY_JUST_SET);
#endif
```

Identifier
(insn.def)

University of BRISTOL

EMBECOSM®

How to add a new architecture

```
#if SPARC || M88000
    /* sparc:          addx
       m88000:         addu.ci */
    PERFORM_ADD_CI(v, co, r1, r2, ci);
    RECURSE(ADD_CI, s1, s2, prune_hint & ~CY_JUST_SET);
#endif
```

Identifier
(`insn.def`)

Registers
used

How to add a new architecture

```
#if SPARC || M88000
    /* sparc:           addx
       m88000:          addu.ci */
    PERFORM_ADD_CI(v, co, r1, r2, ci);
    RECURSE(ADD_CI, s1, s2, prune_hint & ~CY_JUST_SET);
#endif
```

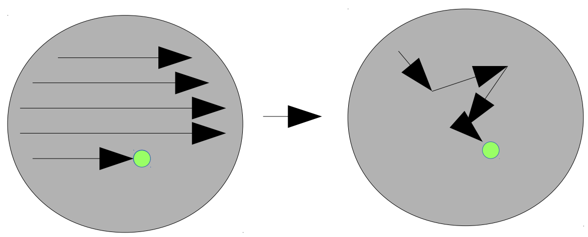
Identifier (`insn.def`)

Registers used

Speed hints. (we didn't just set the carry)

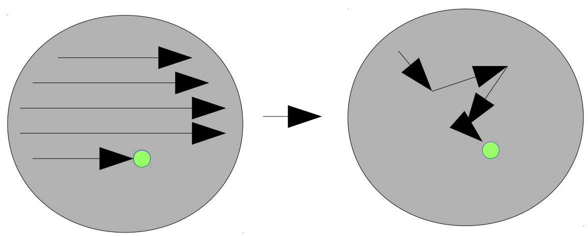
Our plans to extend GSO

Our plans to extend GSO

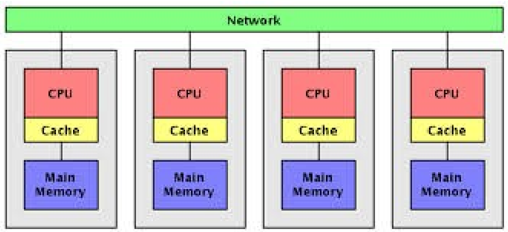


Machine learning

Our plans to extend GSO

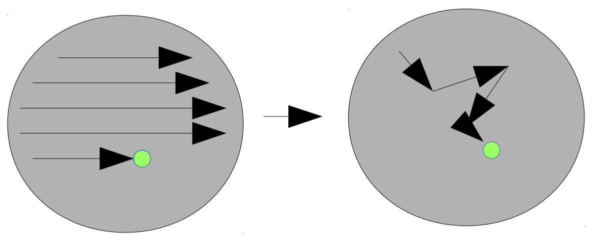


Machine learning

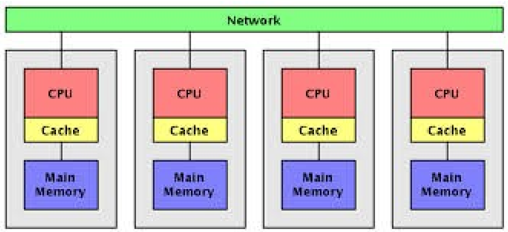


Parallelism

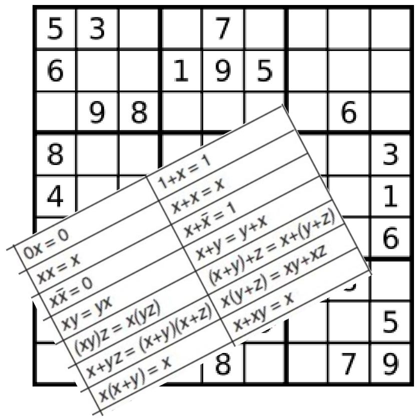
Our plans to extend GSO



Machine learning

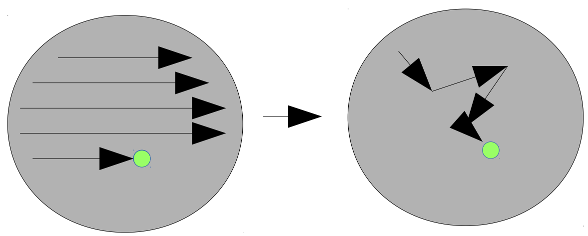


Parallelism

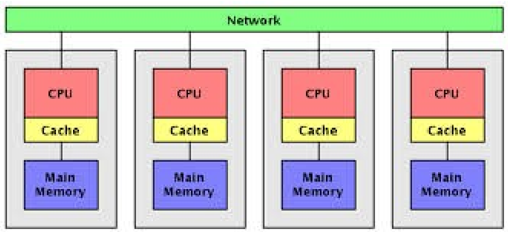


Instruction sequence verification

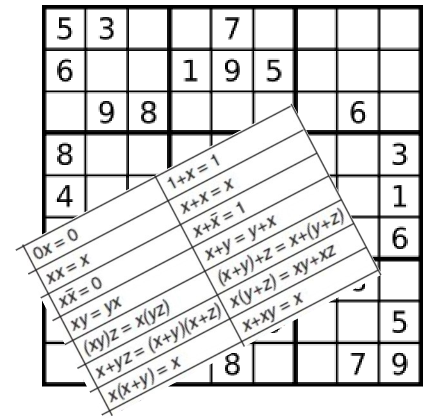
Our plans to extend GSO



Machine learning

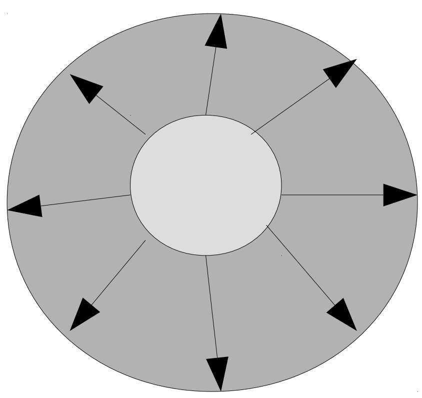


Parallelism

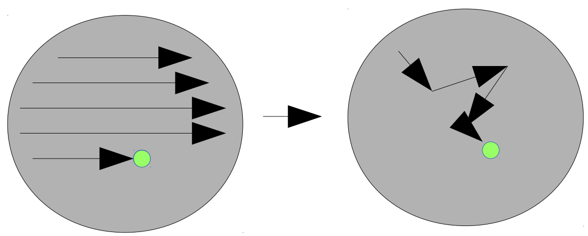


Instruction sequence verification

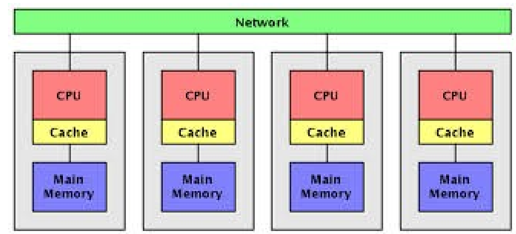
More instruction types



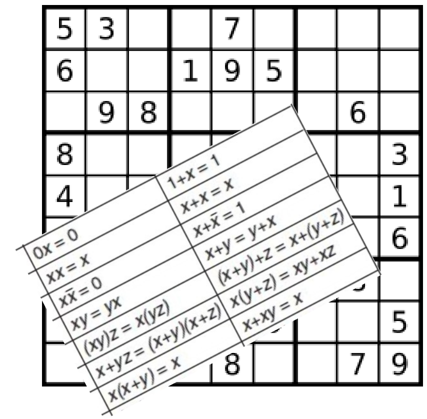
Our plans to extend GSO



Machine learning



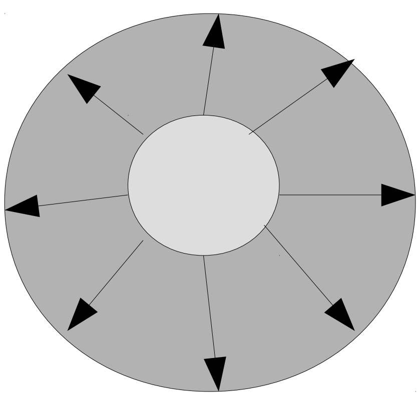
Parallelism



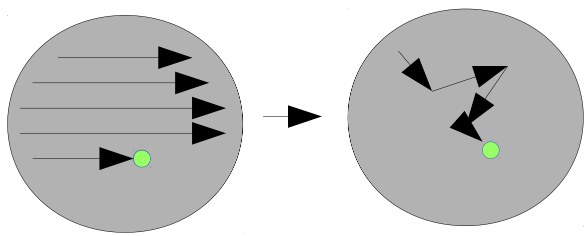
Instruction sequence verification

More instruction types

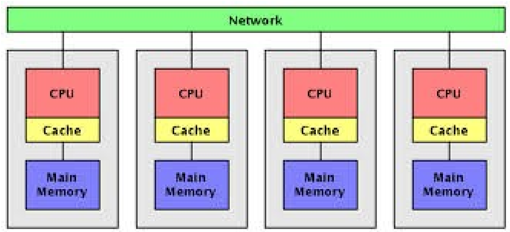
Memory access



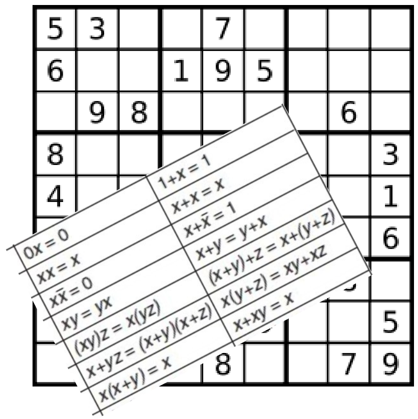
Our plans to extend GSO



Machine learning



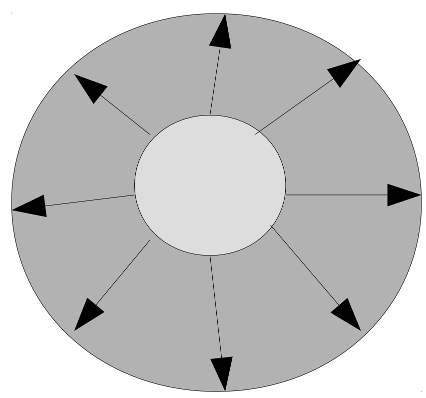
Parallelism



Instruction sequence verification

More instruction types

Memory access

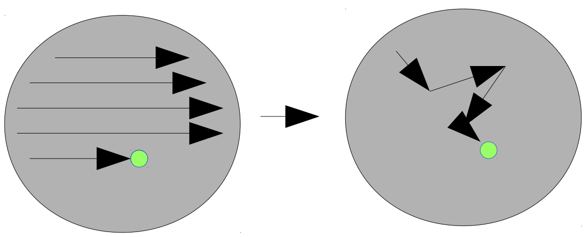


Floating point

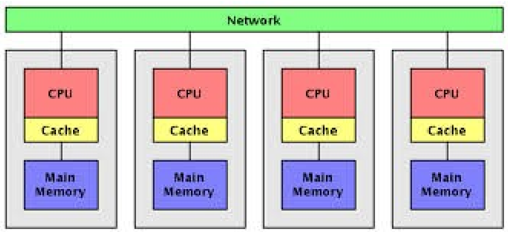
University of BRISTOL

EMBECOSM®

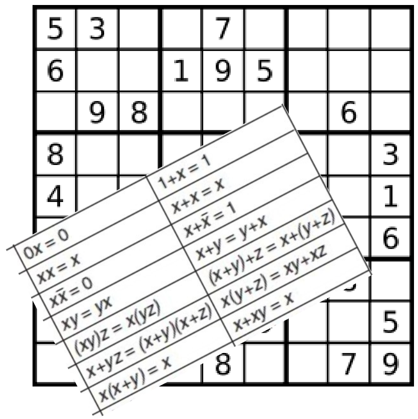
Our plans to extend GSO



Machine learning



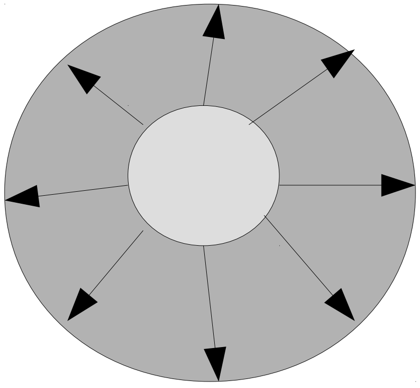
Parallelism



Instruction sequence verification

More instruction types

Memory access



Floating point

Multiple outputs

University of BRISTOL

EMBECOSM®

Thank You

Thank You

Superoptimization works

Thank You

Superoptimization works

New techniques are making it better

Thank You

Superoptimization works

New techniques are making it better

Lots of *free* software and tools

University of BRISTOL

EMBECOSM®

Thank You

Superoptimization works

New techniques are making it better

Lots of *free* software and tools

University of BRISTOL

EMBECOSM®

Innovate UK
Technology Strategy Board

Thank You

Superoptimization works

New techniques are making it better

Lots of *free* software and tools

Try it yourself

github.com/embecosm/gnu-superopt