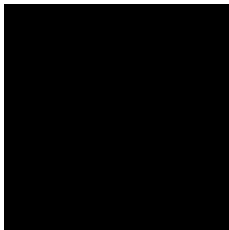
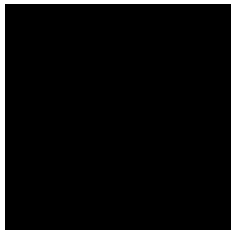


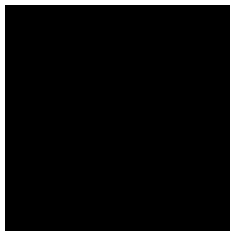
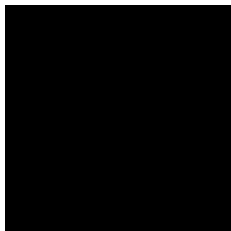
Tamarin: Concolic Disequivalence for MIPS

Abel Nieto

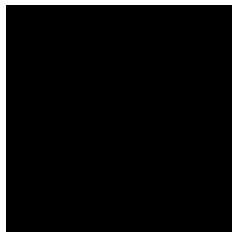
A Tale of Two Boxes



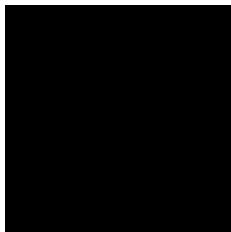
A Tale of Two Boxes



A Tale of Two Boxes

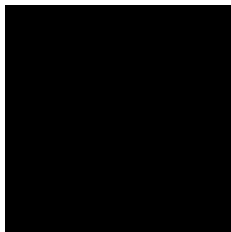


nom nom

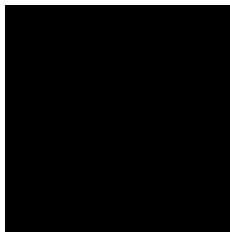


nom nom

A Tale of Two Boxes

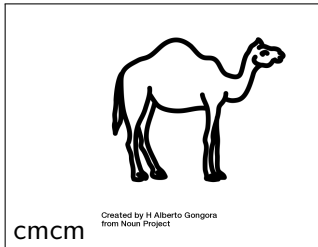


nom nom

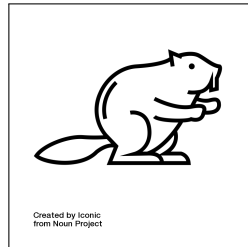


nom nom

A Tale of Two Boxes



≠



The Problem

Given MIPS program P_1 and P_2 , when are they equivalent?

The Problem

Attempt 1: two programs are equivalent if they give the same output (resp.) for all inputs.

The Problem

Attempt 1: two programs are equivalent if they give the same output (resp.) for all inputs.

What's an input?

The Problem

Attempt 1: two programs are equivalent if they give the same output (resp.) for all inputs.

What's an input? Register \$1 and \$2.

The Problem

Attempt 1: two programs are equivalent if they give the same output (resp.) for all inputs.

What's an input? Register \$1 and \$2.

What's an output?

The Problem

Attempt 1: two programs are equivalent if they give the same output (resp.) for all inputs.

What's an input? Register \$1 and \$2.

What's an output? Register \$3.

The Problem

Attempt 1: two programs are equivalent if they give the same output (resp.) for all inputs.

What's an input? Register \$1 and \$2.

What's an output? Register \$3.

Don't care about (most) CPU interrupts/IO.

The Problem

Attempt 1: two programs are equivalent if they give the same output (resp.) for all inputs.

The Problem

Attempt 1: two programs are equivalent if they give the same output (resp.) for all inputs.

Problem: undecidable via Rice's theorem.

The Problem

Attempt 2: two programs are S -equivalent if they cannot be told apart after S steps.

The Problem

Attempt 2: two programs are S -equivalent if they cannot be told apart after S steps.

S -equivalent (e.g. for $S = 10$), but not equivalent:

```
# P_1  
add $3, $1, $2
```

```
# P_2  
    add $4, $0, 1 # counter  
    add $5, $0, 42 # upper bound  
loop:  
    slt $6, $4, $5  
    beq $6, $0, end  
    add $4, $4, 1  
    beq $0, $0, loop  
end:  
    add $3, $1, $1
```

The Problem

Attempt 2: two programs are S -equivalent if they **cannot be told apart** after S steps.

| | | |
|-------|-------|------------|
| R_1 | R_2 | S -equiv |
|-------|-------|------------|

The Problem

Attempt 2: two programs are S -equivalent if they **cannot be told apart** after S steps.

| R_1 | R_2 | S -equiv |
|-------|-------|------------|
| v | v | yes |

The Problem

Attempt 2: two programs are S -equivalent if they **cannot be told apart** after S steps.

| R_1 | R_2 | S -equiv |
|-------|------------|------------|
| v | v | yes |
| v | $w \neq v$ | no |

The Problem

Attempt 2: two programs are S -equivalent if they **cannot be told apart** after S steps.

| R_1 | R_2 | S -equiv |
|-------|------------|------------|
| v | v | yes |
| v | $w \neq v$ | no |
| v | error | no |

The Problem

Attempt 2: two programs are S -equivalent if they **cannot be told apart** after S steps.

| R_1 | R_2 | S -equiv |
|-------|------------|------------|
| v | v | yes |
| v | $w \neq v$ | no |
| v | error | no |
| error | error | yes |

The Problem

Attempt 2: two programs are S -equivalent if they **cannot be told apart** after S steps.

| R_1 | R_2 | S -equiv |
|-----------------|------------|------------|
| v | v | yes |
| v | $w \neq v$ | no |
| v | error | no |
| error | error | yes |
| non-termination | ??? | yes |
| ... | | |

The Problem

Lemma

Equivalence implies S -equivalence.

The Problem

Lemma

Equivalence implies S -equivalence.

Corollary (Soundness)

If two programs are not S -equivalent (for any S), then they are not equivalent.

The Problem

Attempt 2: two programs are S -equivalent if they **cannot be told apart** after S steps.

Which inputs?

The Problem

Attempt 2: two programs are S -equivalent if they **cannot be told apart** after S steps.

Which inputs?

Try some inputs by hand: low coverage, fast (unit tests)

The Problem

Attempt 2: two programs are S -equivalent if they **cannot be told apart** after S steps.

Which inputs?

Try some inputs by hand: low coverage, fast (unit tests)

Try all 2^{64} values of \$1 and \$2: high coverage, slow (but decidable)

The Problem

Attempt 2: two programs are S -equivalent if they **cannot be told apart** after S steps.

Which inputs?

Try some inputs by hand: low coverage, fast (unit tests)

Try all 2^{64} values of \$1 and \$2: high coverage, slow (but decidable)

Tamarin: use concolic execution: higher coverage(?), not too slow(?)

Idea

Alternating concolic execution

```
# P_1
  bne $1, 42, end
  add $3, $3, $0
end:
  add $3, $1, $2
```

```
# P_2
  add $3, $1, $2
  bne $2, 100, end
  add $3, $3, $2
end:
```

Idea

Alternating concolic execution

```
# P_1
  bne $1, 42, end
  add $3, $3, $0
end:
  add $3, $1, $2
```

```
# P_2
  add $3, $1, $2
  bne $2, 100, end
  add $3, $3, $2
end:
```

| Run | Driver | Verifier | \$1 | \$2 | Path | R_D | R_V |
|-----|--------|----------|-----|-----|------|-------|-------|
|-----|--------|----------|-----|-----|------|-------|-------|

Idea

Alternating concolic execution

```
# P_1
  bne $1, 42, end
  add $3, $3, $0
end:
  add $3, $1, $2
```

```
# P_2
  add $3, $1, $2
  bne $2, 100, end
  add $3, $3, $2
end:
```

| Run | Driver | Verifier | \$1 | \$2 | Path | R_D | R_V |
|-----|--------|----------|-----|-----|---------------|-------|-------|
| 1 | P_1 | P_2 | 1 | 1 | $\$1 \neq 42$ | 2 | 2 |

Idea

Alternating concolic execution

```
# P_1
  bne $1, 42, end
  add $3, $3, $0
end:
  add $3, $1, $2
```

```
# P_2
  add $3, $1, $2
  bne $2, 100, end
  add $3, $3, $2
end:
```

| Run | Driver | Verifier | \$1 | \$2 | Path | R_D | R_V |
|-----|--------|----------|-----|-----|----------------|-------|-------|
| 1 | P_1 | P_2 | 1 | 1 | $\$1 \neq 42$ | 2 | 2 |
| 2 | P_2 | P_1 | 1 | 1 | $\$2 \neq 100$ | 2 | 2 |

Idea

Alternating concolic execution

```
# P_1
  bne $1, 42, end
  add $3, $3, $0
end:
  add $3, $1, $2
```

```
# P_2
  add $3, $1, $2
  bne $2, 100, end
  add $3, $3, $2
end:
```

| Run | Driver | Verifier | \$1 | \$2 | Path | R_D | R_V |
|-----|--------|----------|-----|-----|----------------|-------|-------|
| 1 | P_1 | P_2 | 1 | 1 | $\$1 \neq 42$ | 2 | 2 |
| 2 | P_2 | P_1 | 1 | 1 | $\$2 \neq 100$ | 2 | 2 |
| 3 | P_1 | P_2 | 42 | 1 | $\$1 = 42$ | 2 | 2 |

Idea

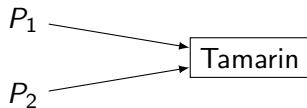
Alternating concolic execution

```
# P_1
  bne $1, 42, end
  add $3, $3, $0
end:
  add $3, $1, $2
```

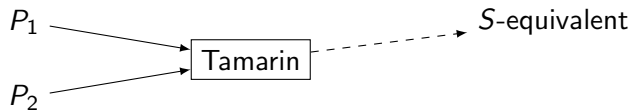
```
# P_2
  add $3, $1, $2
  bne $2, 100, end
  add $3, $3, $2
end:
```

| Run | Driver | Verifier | \$1 | \$2 | Path | R_D | R_V |
|-----|--------|----------|-----|-----|----------------|-------|-------|
| 1 | P_1 | P_2 | 1 | 1 | $\$1 \neq 42$ | 2 | 2 |
| 2 | P_2 | P_1 | 1 | 1 | $\$2 \neq 100$ | 2 | 2 |
| 3 | P_1 | P_2 | 42 | 1 | $\$1 = 42$ | 2 | 2 |
| 4 | P_2 | P_1 | 1 | 100 | $\$2 = 100$ | 201 | 2 |

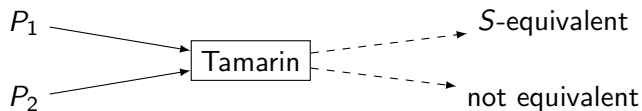
Tamarin (Overview)



Tamarin (Overview)



Tamarin (Overview)



Tamarin (Overview)

Concolic

Tamarin (Overview)

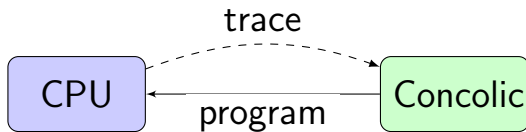
CPU

Concolic

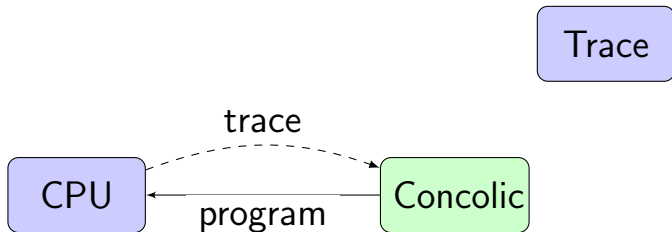
Tamarin (Overview)



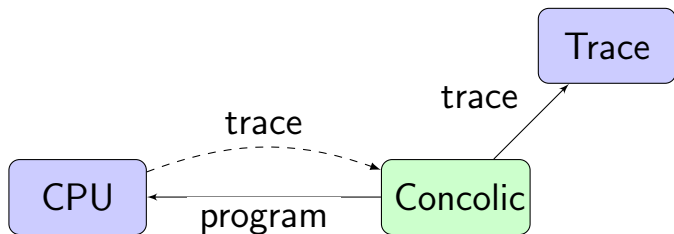
Tamarin (Overview)



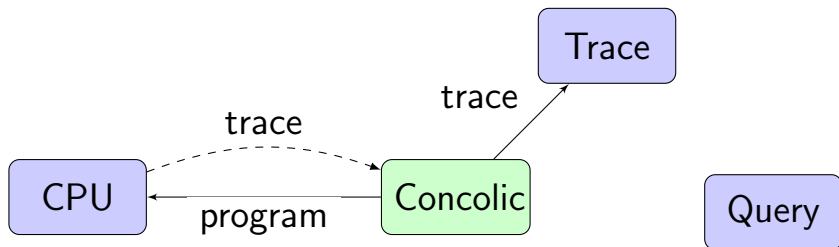
Tamarin (Overview)



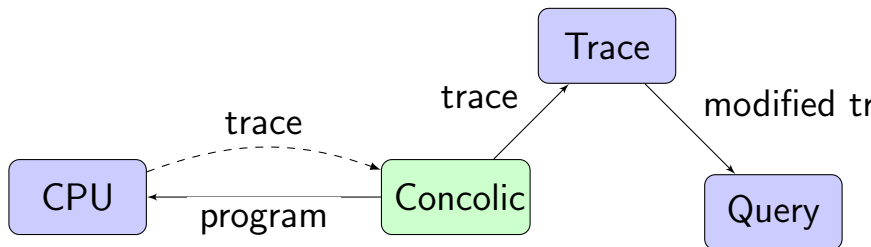
Tamarin (Overview)



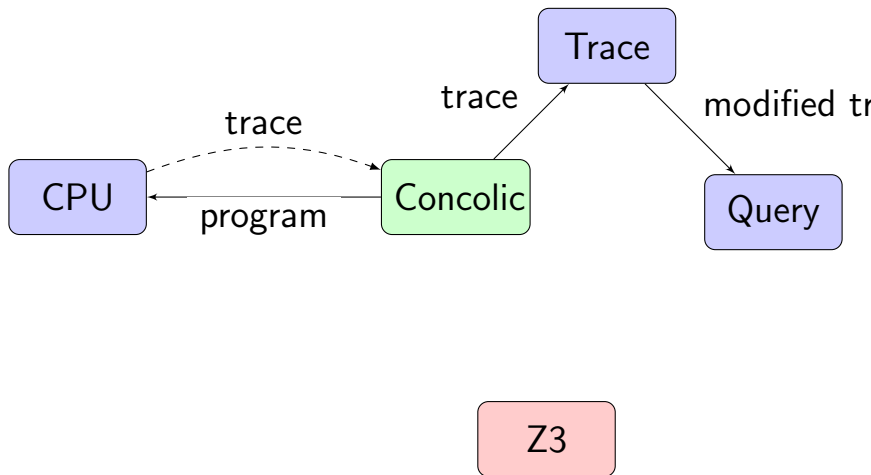
Tamarin (Overview)



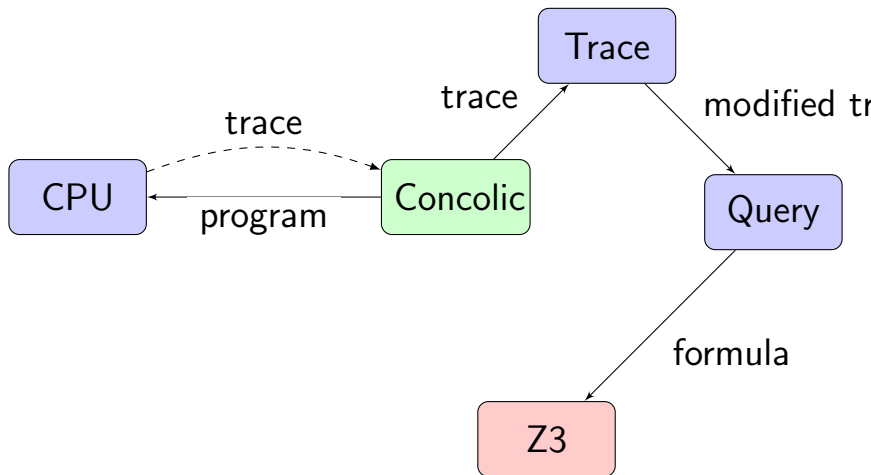
Tamarin (Overview)



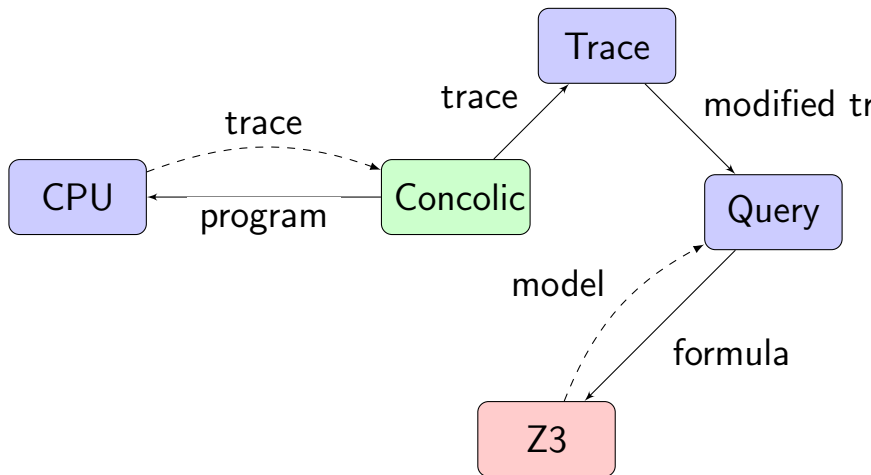
Tamarin (Overview)



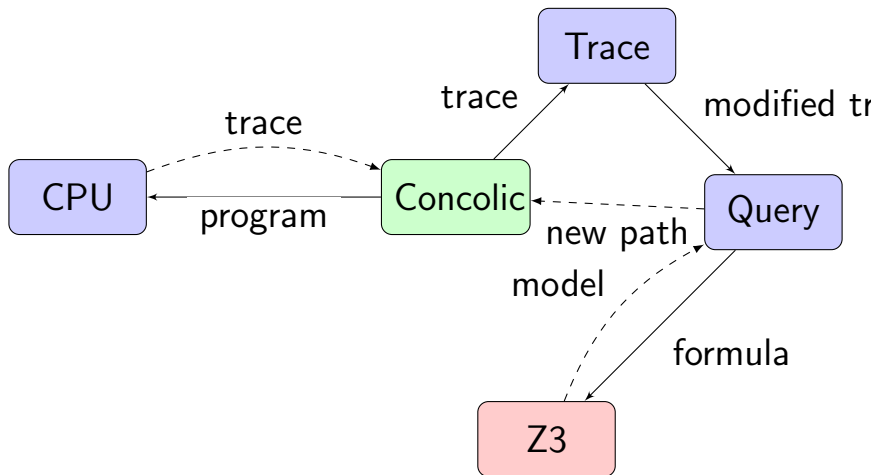
Tamarin (Overview)



Tamarin (Overview)

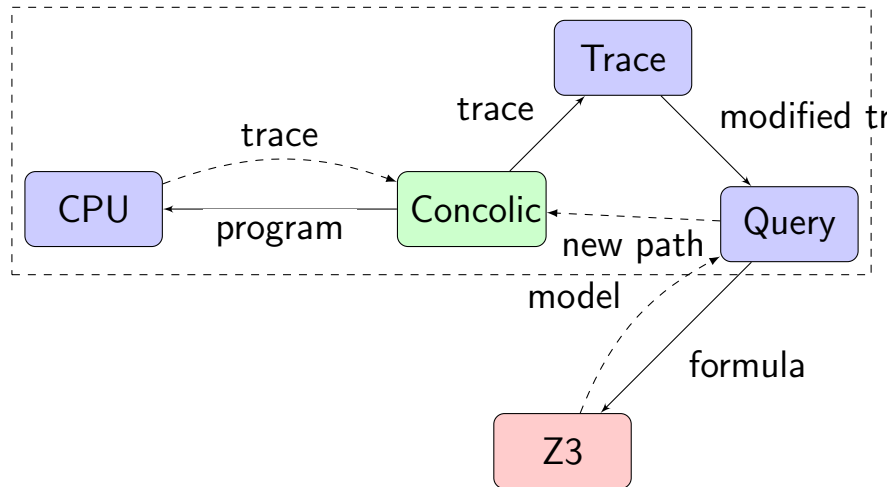


Tamarin (Overview)



Tamarin (Overview)

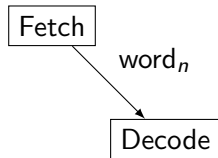
Tamarin



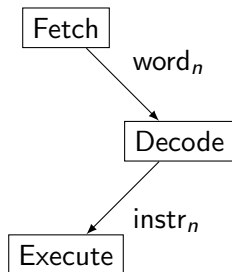
CPU

Fetch

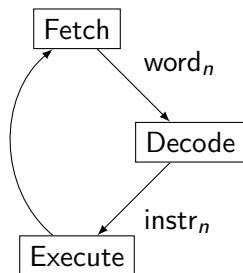
CPU



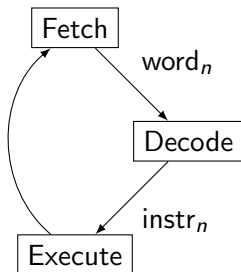
CPU



CPU



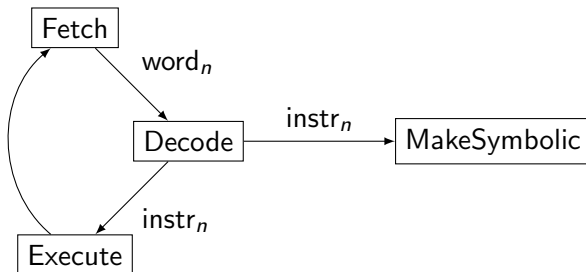
CPU



MakeSymbolic

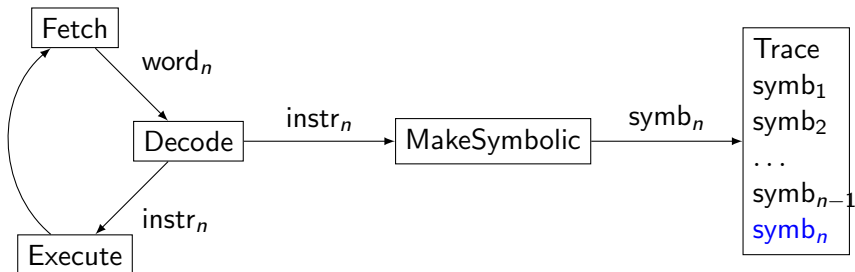
Trace
 symb_1
 symb_2
...
 symb_{n-1}
 symb_n

CPU



Trace
symb₁
symb₂
...
symb _{$n-1$}
symb _{n}

CPU



CPU (MakeSymbolic)

| | |
|-------------|----------|
| Instruction | Symbolic |
|-------------|----------|

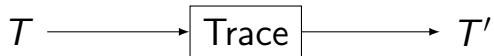
CPU (MakeSymbolic)

| Instruction | Symbolic |
|-------------------|----------------------------|
| add \$3, \$1, \$2 | $r_3 \leftarrow r_1 + r_2$ |

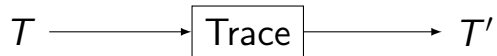
CPU (MakeSymbolic)

| Instruction | Symbolic |
|----------------------------------|--------------------------------------|
| <code>add \$3, \$1, \$2</code> | $r_3 \leftarrow r_1 + r_2$ |
| <code>beq \$1, \$2, label</code> | $r_1 = r_2 \text{ or } r_1 \neq r_2$ |
| <code>add \$3, \$pc, \$0</code> | $r_3 \leftarrow 0x8BADF00D$ |
| <code>lis \$3; 42</code> | $r_3 \leftarrow 42$ |

Trace

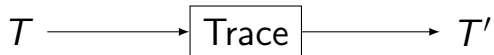


Trace



Seq[Trace => Trace]

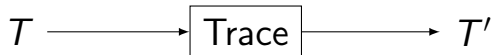
Trace



Seq[Trace => Trace]

- ▶ Desugar

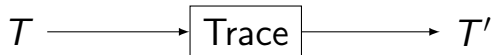
Trace



Seq[Trace => Trace]

- ▶ Desugar
- ▶ Simplify

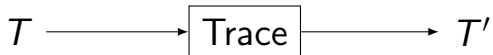
Trace



Seq[Trace => Trace]

- ▶ Desugar
- ▶ Simplify
- ▶ Trim

Trace



Seq[Trace => Trace]

- ▶ Desugar
- ▶ Simplify
- ▶ Trim
- ▶ SSA convert

Trace

Desugar

Trace

Desugar

`Mult(s, t) \rightarrow Mult64(tmp, s, t); Low32(lo, tmp); High32(hi, tmp)`

Trace

Desugar

`Mult(s, t) \rightarrow Mult64(tmp, s, t); Low32(lo, tmp); High32(hi, tmp)`

Simplify

Trace

Desugar

`Mult(s, t) \rightarrow Mult64(tmp, s, t); Low32(lo, tmp); High32(hi, tmp)`

Simplify

`beq $0, $0, label \rightarrow \emptyset`

Trace

Desugar

`Mult(s, t) \rightarrow Mult64(tmp, s, t); Low32(lo, tmp); High32(hi, tmp)`

Simplify

`beq $0, $0, label \rightarrow \emptyset`

Trim

Trace

Desugar

`Mult(s, t) \rightarrow Mult64(tmp, s, t); Low32(lo, tmp); High32(hi, tmp)`

Simplify

`beq $0, $0, label \rightarrow \emptyset`

Trim

limit trace to D path conditions

Trace

Desugar

`Mult(s, t) \rightarrow Mult64(tmp, s, t); Low32(lo, tmp); High32(hi, tmp)`

Simplify

`beq $0, $0, label $\rightarrow \emptyset$`

Trim

limit trace to D path conditions

SSA convert

Trace (SSA convert)

| Trace | Incorrect | Correct |
|-------|-----------|---------|
|-------|-----------|---------|

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-----------|---------|
| Add(\$3, \$1, \$2) | | |

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|---------|
| Add(\$3, \$1, \$2) | $z = x + y$ | |

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | | |

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | $x = x - y$ | |

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | $x = x - y$ | $x_2 = x_1 - y_1$ |

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | $x = x - y$ | $x_2 = x_1 - y_1$ |
| Add(\$2, \$1, \$2) | | |

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | $x = x - y$ | $x_2 = x_1 - y_1$ |
| Add(\$2, \$1, \$2) | $y = x + y$ | |

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | $x = x - y$ | $x_2 = x_1 - y_1$ |
| Add(\$2, \$1, \$2) | $y = x + y$ | $y_2 = x_2 + y_1$ |

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | $x = x - y$ | $x_2 = x_1 - y_1$ |
| Add(\$2, \$1, \$2) | $y = x + y$ | $y_2 = x_2 + y_1$ |

,

$$\$3 = 9 \wedge \$1 = 3$$

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | $x = x - y$ | $x_2 = x_1 - y_1$ |
| Add(\$2, \$1, \$2) | $y = x + y$ | $y_2 = x_2 + y_1$ |

,

$$\$3 = 9 \wedge \$1 = 3$$

$$z = 9 \wedge x = 3$$

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | $x = x - y$ | $x_2 = x_1 - y_1$ |
| Add(\$2, \$1, \$2) | $y = x + y$ | $y_2 = x_2 + y_1$ |

,

$$\$3 = 9 \wedge \$1 = 3$$

$$z = 9 \wedge x = 3 \text{ (no sol)}$$

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | $x = x - y$ | $x_2 = x_1 - y_1$ |
| Add(\$2, \$1, \$2) | $y = x + y$ | $y_2 = x_2 + y_1$ |

,

$$\$3 = 9 \wedge \$1 = 3$$

$$z = 9 \wedge x = 3 \text{ (no sol)}$$

$$z_1 = 9 \wedge x_2 = 3$$

Trace (SSA convert)

| Trace | Incorrect | Correct |
|--------------------|-------------|-------------------|
| Add(\$3, \$1, \$2) | $z = x + y$ | $z_1 = x_1 + y_1$ |
| Sub(\$1, \$1, \$2) | $x = x - y$ | $x_2 = x_1 - y_1$ |
| Add(\$2, \$1, \$2) | $y = x + y$ | $y_2 = x_2 + y_1$ |

,

$$\$3 = 9 \wedge \$1 = 3$$

$$z = 9 \wedge x = 3 \text{ (no sol)}$$

$$z_1 = 9 \wedge x_2 = 3 \text{ (} x_1 = 6 \wedge y_1 = 3 \text{)}$$