Validating Optimizations of Concurrent C/C++ Programs

Soham Chakraborty

IIT Delhi

SAT + SMT, 2019

Concurrent Programming

```
int X = 0;   int Y = 0;

Y = 4;  ‖  if (X)
X = 1;  ‖    r = Y;
```

Concurrent Programming

```
int X = 0;   int Y = 0;

Y = 4;  ‖  if (X)
X = 1;  ‖     r = Y;
```

Race on `X` $\rightsquigarrow$ undefined semantics

$X == 1 \wedge r \neq 4$ is possible

(i.e., the program is wrong)

Concurrent Programming in C11

```
atomic_int X = 0;   int Y = 0;

Y = 4;                   ||  if (atomic_load(&X,
                         ||           mo_acquire))
atomic_store(&X, 1,      ||
     mo_release);        ||     r = Y;
```

Concurrent Programming in C11

```
atomic_int X = 0;   int Y = 0;

Y = 4;                       || if (atomic_load(&X,
                             ||          mo_acquire))
atomic_store(&X, 1,          ||
     mo_release);            ||    r = Y;
```

Concurrent Programming in C11

```
atomic_int X = 0;   int Y = 0;

Y = 4;                         ||  if (atomic_load(&X,
                               ||          mo_acquire))
atomic_store(&X, 1,            ||
     mo_release);              ||      r = Y;
```

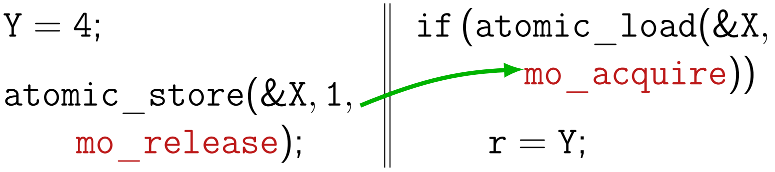
Concurrent Programming in C11

```
atomic_int X = 0;   int Y = 0;
```

Thread 1	Thread 2
`Y = 4;`	`if (atomic_load(&X, mo_acquire))`
`atomic_store(&X, 1, mo_release);`	`r = Y;`

Concurrent Programming in C11

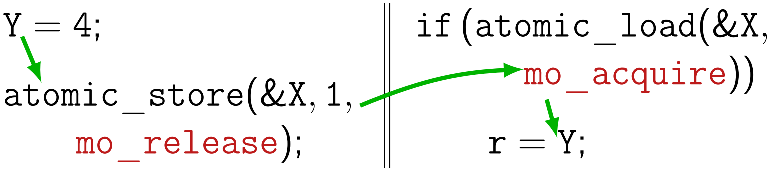
```
atomic_int X = 0;   int Y = 0;
```



```
Y = 4;                    ‖ if (atomic_load(&X,
                          ‖         mo_acquire))
atomic_store(&X, 1,       ‖
     mo_release);         ‖     r = Y;
```

Concurrent Programming in C11

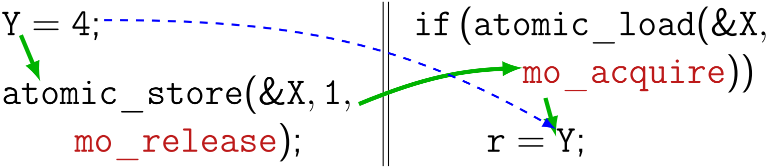
```
atomic_int X = 0;   int Y = 0;
```



```
Y = 4;                          ||  if (atomic_load(&X,
                                ||          mo_acquire))
atomic_store(&X, 1,             ||
    mo_release);                ||      r = Y;
```

Concurrent Programming in C11

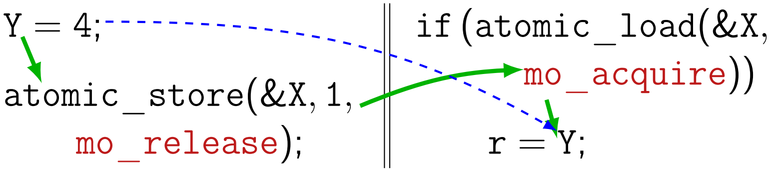
```
atomic_int X = 0;   int Y = 0;
```



```
Y = 4;                          ||  if (atomic_load(&X,
atomic_store(&X, 1,             ||          mo_acquire))
      mo_release);              ||        r = Y;
```

Concurrent Programming in C11

```
atomic_int X = 0;   int Y = 0;
```



```
Y = 4;                   ||  if (atomic_load(&X,
atomic_store(&X, 1,      ||                 mo_acquire))
     mo_release);        ||       r = Y;
```

⇓

$$X = Y = 0;$$

$$\begin{array}{l||l} Y = 4; & \text{if}(X_{\text{acq}}) \\ X_{\text{rel}} = 1; & \quad r = Y; \end{array}$$

An Unsafe Reordering

$$\begin{array}{c} X = Y = 0; \\ \updownarrow \left.\begin{array}{l} Y = 4; \\ X_{\mathrm{rel}} = 1; \end{array}\right\| \begin{array}{l} r = 4; \\ \mathrm{if}(X_{\mathrm{acq}}) \\ \qquad r = Y; \end{array} \end{array} \rightsquigarrow \begin{array}{c} X = Y = 0; \\ \left.\begin{array}{l} X_{\mathrm{rel}} = 1; \\ Y = 4; \end{array}\right\| \begin{array}{l} r = 4; \\ \mathrm{if}(X_{\mathrm{acq}}) \\ \qquad r = Y; \end{array} \end{array}$$

Always returns $r == 4$

May return $r == 0$

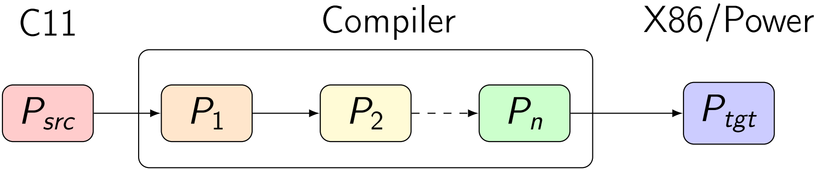
An Unsafe Reordering

$$
\begin{array}{ccc}
X = Y = 0; & \qquad & X = Y = 0; \\
\updownarrow\begin{array}{l} Y = 4; \\ X_{\mathrm{rel}} = 1; \end{array} \;\Bigg\|\; \begin{array}{l} r = 4; \\ \mathrm{if}(X_{\mathrm{acq}}) \\ \qquad r = Y; \end{array} & \rightsquigarrow & \begin{array}{l} X_{\mathrm{rel}} = 1; \\ Y = 4; \end{array} \;\Bigg\|\; \begin{array}{l} r = 4; \\ \mathrm{if}(X_{\mathrm{acq}}) \\ \qquad r = Y; \end{array}
\end{array}
$$

Always returns $r == 4$ May return $r == 0$

Optimizations for sequential programs are NOT always safe for concurrent programs.

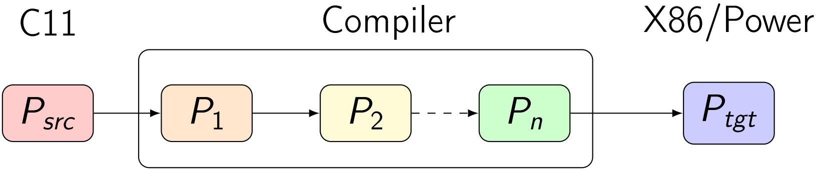
Questions



Q1: Which of the transformations are allowed?

Q2: Does a compiler perform only allowed transformations?

Questions



Q1: Which of the transformations are allowed?

C11 $\rightsquigarrow$ C11 [POPL'15]

- access(X); access(Y); $\rightsquigarrow$ access(Y); access(X);
- access(X); access(X); $\rightsquigarrow$ access(X);

Reordering Transformations

Reordering (a;b $\rightsquigarrow$ b;a)

$\downarrow$ a \ b $\rightarrow$	$\mathrm{R_{na}}$	$\mathrm{W_{na}}$	$\mathrm{R_{acq}}$	$\mathrm{W_{rel}}$	...
$\mathrm{R_{na}}$	✓	✓	✓	×	...
$\mathrm{W_{na}}$	✓	✓	✓	×	...
$\mathrm{R_{acq}}$	×	×	×	×	...
$\mathrm{W_{rel}}$	✓	✓	✓	×	...
...	...	...	...	...	...

$$t = X; s = Y_{\mathrm{acq}}; \rightsquigarrow s = Y_{\mathrm{acq}}; t = X; \quad \checkmark$$

$$s = Y_{\mathrm{acq}}; t = X; \rightsquigarrow t = X; s = Y_{\mathrm{acq}}; \quad \times$$

Elimination Transformations

Read-after-Read(RAR)

$t = X; t' = X; \rightsquigarrow t = X; t' = t;$ ✓

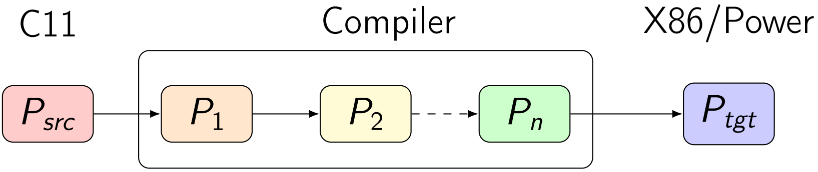
Read-after-Read(RAW)

$X = 1; t = X; \rightsquigarrow X = 1; t = 1;$ ✓

Over-written Writes(OW)

$X = 1; X = 2; \rightsquigarrow X = 2;$ ✓

Questions



Q1: Which of the transformations are allowed?

- C11 $\rightsquigarrow$ C11 [POPL'15]

Q2: Does a compiler perform only allowed transformations?

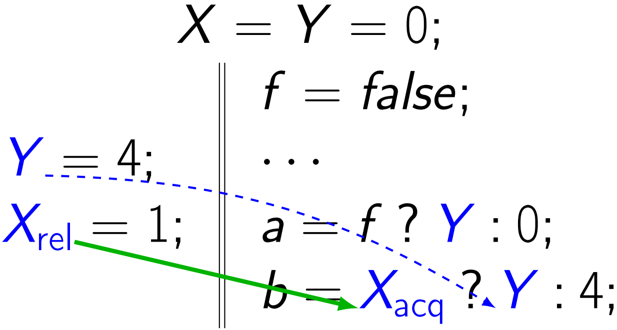
- C11 compilation by LLVM [CGO'16]

Another Example

$$X = Y = 0;$$

$$\begin{array}{r||l} & f = \mathit{false}; \\ Y = 4; & \dots \\ X_{\mathrm{rel}} = 1; & a = f \;?\; Y : 0; \\ & b = X_{\mathrm{acq}} \;?\; Y : 4; \end{array}$$

Another Example



$$X = Y = 0;$$

$$\begin{array}{l||l} & f = \mathit{false}; \\ Y = 4; & \ldots \\ X_{\text{rel}} = 1; & a = f \; ? \; Y : 0; \\ & b = X_{\text{acq}} \; ? \; Y : 4; \end{array}$$

Output: $b == 4$ always

LLVM Compilation Bug #22514

$$\begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ a = f \mathrel{?} Y : 0; \\ b = X_{\mathrm{acq}} \mathrel{?} Y : 4; \end{array} \overset{-O3}{\rightsquigarrow} \begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ s = Y; \\ a = f \mathrel{?} s : 0; \\ t = X_{\mathrm{acq}}; \\ b = t \mathrel{?} s : 4; \end{array}$$

Context:

$$\left[\; \parallel \; \begin{array}{l} Y = 4; \\ X_{\mathrm{rel}} = 1; \end{array} \right]$$

Output $b == 0$ possible in target.

LLVM Compilation Bug in More Detail

$$\begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ a = f \; ? \; Y : 0; \\ b = X_{\mathrm{acq}} \; ? \; Y : 4; \end{array}$$

$$\overset{(1)}{\leadsto}$$

$$\begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ s = Y; \\ a = f \; ? \; s : 0; \\ t = X_{\mathrm{acq}}; \\ r = Y; \\ b = t \; ? \; r : 4; \end{array}$$

$$\overset{(2)}{\leadsto}$$

$$\begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ s = Y; \\ a = f \; ? \; s : 0; \\ t = X_{\mathrm{acq}}; \\ \cancel{r = Y;} \\ b = t \; ? \; s : 4; \end{array}$$

LLVM Compilation Bug in More Detail

$X = Y = 0;$
$f = \mathit{false};$
$\ldots$
$a = f\ ?\ Y : 0;$
$b = X_{\mathrm{acq}}\ ?\ Y : 4;$

$\overset{(1)}{\rightsquigarrow}$

$X = Y = 0;$
$f = \mathit{false};$
$\ldots$
$s = Y;$
$a = f\ ?\ s : 0;$
$t = X_{\mathrm{acq}};$
$r = Y;$
$b = t\ ?\ r : 4;$

$\overset{(2)}{\rightsquigarrow}$

$X = Y = 0;$
$f = \mathit{false};$
$\ldots$
$s = Y;$
$a = f\ ?\ s : 0;$
$t = X_{\mathrm{acq}};$
$\cancel{r = Y;}$
$b = t\ ?\ s : 4;$

LLVM Compilation Bug in More Detail

$$\begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ a = f\ ?\ Y : 0; \\ b = X_{\text{acq}}\ ?\ Y : 4; \end{array} \quad \overset{(1)}{\rightsquigarrow} \quad \begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ s = Y; \\ a = f\ ?\ s : 0; \\ t = X_{\text{acq}}; \\ r = Y; \\ b = t\ ?\ r : 4; \end{array} \quad \overset{(2)}{\rightsquigarrow} \quad \begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ s = Y; \\ a = f\ ?\ s : 0; \\ t = X_{\text{acq}}; \\ \cancel{r = Y;} \\ b = t\ ?\ s : 4; \end{array}$$

C11: (1) **Error**

LLVM Compilation Bug in More Detail

$$\begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ a = f\ ?\ Y : 0; \\ b = X_{\mathrm{acq}}\ ?\ Y : 4; \end{array} \quad \overset{(1)}{\leadsto} \quad \begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ s = Y; \\ a = f\ ?\ s : 0; \\ t = X_{\mathrm{acq}}; \\ r = Y; \\ b = t\ ?\ r : 4; \end{array} \quad \overset{(2)}{\leadsto} \quad \begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \ldots \\ s = Y; \\ a = f\ ?\ s : 0; \\ t = X_{\mathrm{acq}}; \\ \cancel{r = Y;} \\ b = t\ ?\ s : 4; \end{array}$$

C11: (1) **Error** (2) **Correct**

LLVM Compilation Bug in More Detail

$$\begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \dots \\ a = f\ ?\ Y : 0; \\ b = X_{\mathsf{acq}}\ ?\ Y : 4; \end{array} \overset{(1)}{\rightsquigarrow} \begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \dots \\ s = Y; \\ a = f\ ?\ s : 0; \\ t = X_{\mathsf{acq}}; \\ r = Y; \\ b = t\ ?\ r : 4; \end{array} \overset{(2)}{\rightsquigarrow} \begin{array}{l} X = Y = 0; \\ f = \mathit{false}; \\ \dots \\ s = Y; \\ a = f\ ?\ s : 0; \\ t = X_{\mathsf{acq}}; \\ \cancel{r = Y;} \\ b = t\ ?\ s : 4; \end{array}$$

C11: (1) **Error** (2) **Correct**

LLVM: (1) **Correct**

LLVM Compilation Bug in More Detail

$$
\begin{array}{l}
X = Y = 0; \\
f = \mathit{false}; \\
\ldots \\
a = f \mathbin{?} Y : 0; \\
b = X_{\mathrm{acq}} \mathbin{?} Y : 4;
\end{array}
\overset{(1)}{\leadsto}
\begin{array}{l}
X = Y = 0; \\
f = \mathit{false}; \\
\ldots \\
s = Y; \\
a = f \mathbin{?} s : 0; \\
t = X_{\mathrm{acq}}; \\
r = Y; \\
b = t \mathbin{?} r : 4;
\end{array}
\overset{(2)}{\leadsto}
\begin{array}{l}
X = Y = 0; \\
f = \mathit{false}; \\
\ldots \\
s = Y; \\
a = f \mathbin{?} s : 0; \\
t = X_{\mathrm{acq}}; \\
\cancel{r = Y;} \\
b = t \mathbin{?} s : 4;
\end{array}
$$

C11: (1) **Error** (2) **Correct**

LLVM: (1) **Correct** (2) **Error**

Our Approach: LLVM Validation

$$P_{src} \xRightarrow{\text{LLVM}} P_{tgt} \; ? \; \textbf{Correct} : \textbf{Potential Error}$$

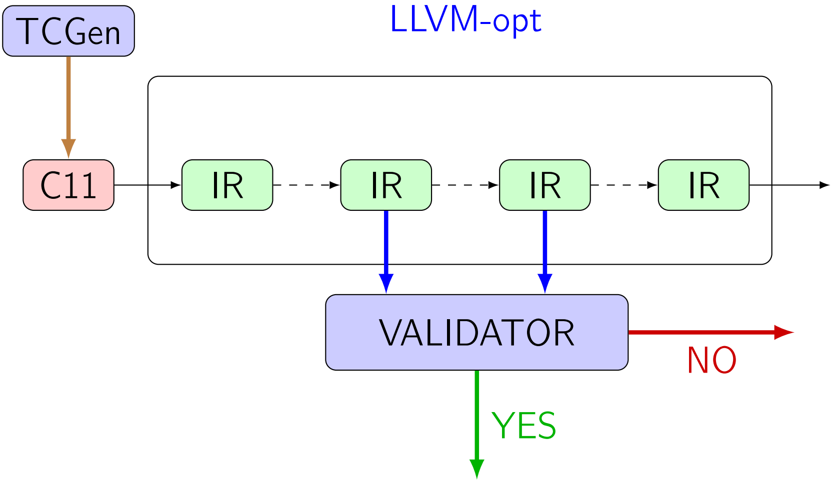
$$\Downarrow$$

$$P_{src} \xRightarrow{(R \cup E)^*} P_{tgt} \; ? \; \textbf{Correct} : \textbf{Potential Error}$$

w.r.t. safe reorderings (R) & eliminations (E):

- For the LLVM model
- For the C11 model

Tool Flow



Exposed concurrency compilation bugs in LLVM 3.6

- Reported and fixed in LLVM 3.7

Validation Schemes

Compiler Independent Matching (CIM)

- Can be used in validating other compilers.

Metadata Based Matching (MD)

- LLVM specific, uses metadata in LLVM.

Compiler Independent Matching Steps

Compiler Independent Matching (CIM)

- Can be used in validating other compilers.

Steps:

- Identify corresponding program paths
- Compute deletability of accesses
- Match access sequences and analyze

Example: Compiler Independent Matching

$$s_1 = X$$

$$s_2 = X$$

$$V = 1$$

$$s_4 = Z_{\text{acq}}$$

$$Y = 1$$

$$Y = 2$$

Example: Compiler Independent Matching

✓ $s_1 = X$

$s_2 = X$

$V = 1$

$s_4 = Z_{\text{acq}}$

$Y = 1$

$Y = 2$

Example: Compiler Independent Matching

✓ $s_1 = X$

✗ $s_2 = X$

$V = 1$

$s_4 = Z_{\mathrm{acq}}$

$Y = 1$

$Y = 2$

Example: Compiler Independent Matching

✓ $s_1 = X$

✗ $s_2 = X$

$V = 1$

✓ $s_4 = Z_{\text{acq}}$

$Y = 1$

$Y = 2$

Example: Compiler Independent Matching

✓ $s_1 = X$

✗ $s_2 = X$

$V = 1$

✓ $s_4 = Z_{\mathrm{acq}}$

$Y = 1$

✓ $Y = 2$

Example: Compiler Independent Matching

✓ $s_1 = X$

✗ $s_2 = X$

$V = 1$

✓ $s_4 = Z_{\mathrm{acq}}$

✗ $Y = 1$

✓ $Y = 2$

Example: Compiler Independent Matching

✓ $s_1 = X$

✗ $s_2 = X$

✓ $V = 1$

✓ $s_4 = Z_{\mathsf{acq}}$

✗ $Y = 1$

✓ $Y = 2$

Example: Compiler Independent Matching

- ✓ $s_1 = X$
- ✗ $s_2 = X$
- ✓ $V = 1$
- ✓ $s_4 = Z_{\text{acq}}$
- ✗ $Y = 1$
- ✓ $Y = 2$

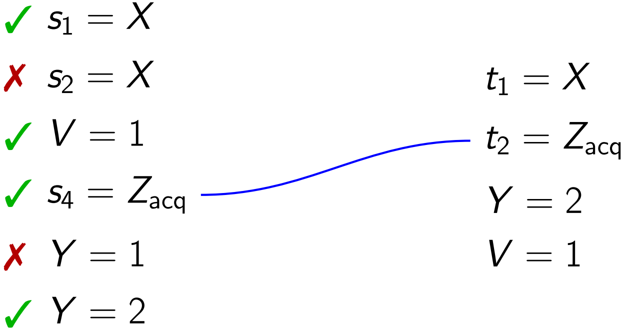
$t_1 = X$

$t_2 = Z_{\text{acq}}$

$Y = 2$

$V = 1$

Example: Compiler Independent Matching



✓ $s_1 = X$

✗ $s_2 = X$

✓ $V = 1$

✓ $s_4 = Z_{\text{acq}}$

✗ $Y = 1$

✓ $Y = 2$

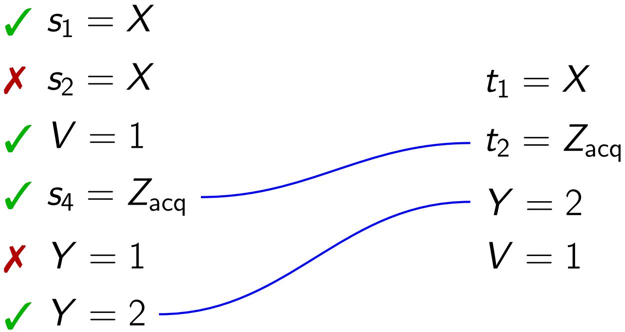
$t_1 = X$

$t_2 = Z_{\text{acq}}$

$Y = 2$

$V = 1$

Example: Compiler Independent Matching



✓ $s_1 = X$

✗ $s_2 = X$

✓ $V = 1$

✓ $s_4 = Z_{\text{acq}}$

✗ $Y = 1$

✓ $Y = 2$

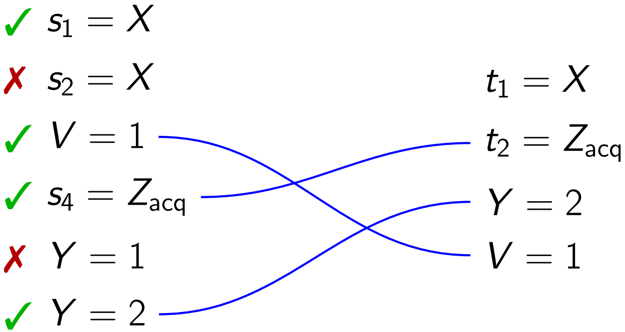
$t_1 = X$

$t_2 = Z_{\text{acq}}$

$Y = 2$

$V = 1$

Example: Compiler Independent Matching



✓ $s_1 = X$

✗ $s_2 = X$

✓ $V = 1$

✓ $s_4 = Z_{\text{acq}}$

✗ $Y = 1$

✓ $Y = 2$

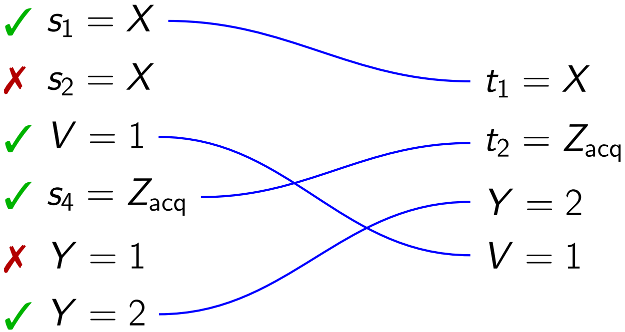
$t_1 = X$

$t_2 = Z_{\text{acq}}$

$Y = 2$

$V = 1$

Example: Compiler Independent Matching



✓ $s_1 = X$

✗ $s_2 = X$

✓ $V = 1$

✓ $s_4 = Z_{\mathrm{acq}}$

✗ $Y = 1$

✓ $Y = 2$

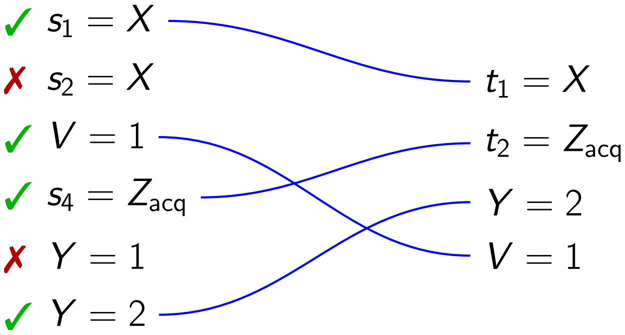
$t_1 = X$

$t_2 = Z_{\mathrm{acq}}$

$Y = 2$

$V = 1$

Example: Compiler Independent Matching



✓ $s_1 = X$

✗ $s_2 = X$

✓ $V = 1$

✓ $s_4 = Z_{\text{acq}}$

✗ $Y = 1$

✓ $Y = 2$

$t_1 = X$

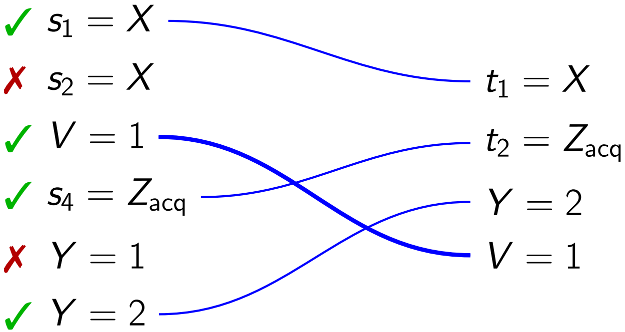
$t_2 = Z_{\text{acq}}$

$Y = 2$

$V = 1$

- Check that unmatched accesses are deletable
- Check that reorderings are allowed

Example: Compiler Independent Matching



✓ $s_1 = X$

✗ $s_2 = X$

✓ $V = 1$

✓ $s_4 = Z_{\mathrm{acq}}$

✗ $Y = 1$

✓ $Y = 2$

$t_1 = X$

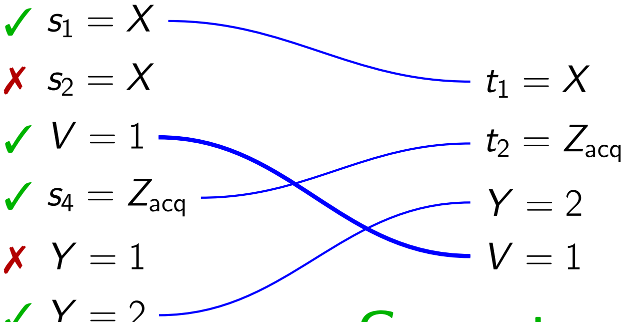
$t_2 = Z_{\mathrm{acq}}$

$Y = 2$

$V = 1$

- Check that unmatched accesses are deletable
- Check that reorderings are allowed

Example: Compiler Independent Matching



✓ $s_1 = X$

✗ $s_2 = X$

✓ $V = 1$

✓ $s_4 = Z_{\mathrm{acq}}$

✗ $Y = 1$

✓ $Y = 2$

$t_1 = X$

$t_2 = Z_{\mathrm{acq}}$

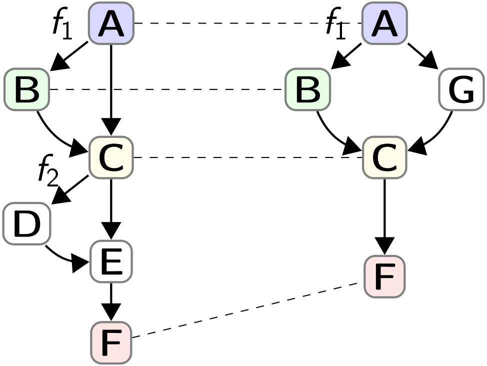
$Y = 2$

$V = 1$

Correct

- Check that unmatched accesses are deletable
- Check that reorderings are allowed

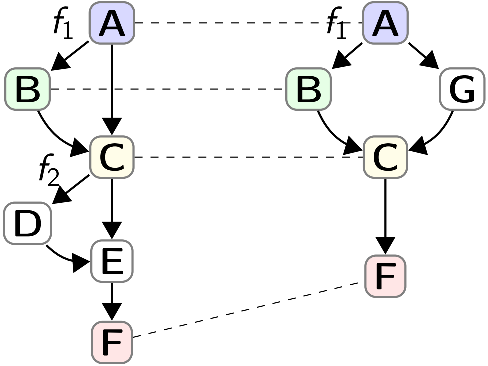
Program with Control Flow



$\{ABCDEF, ABCEF\} \Longrightarrow ABCF, \ \{ACDEF, ACEF\} \Longrightarrow AGCF$

- Use branching conditions to match the paths
  - using Z3 SMT solver
- Match access sequences for each path pair

Program with Loops



- Unroll loops a fixed number of times
- Use branching conditions to match the paths
  - using Z3 SMT solver
- Match access sequences for each path pair

Summary

Compiler optimizations require careful analysis

Reported LLVM concurrency compilation bugs; all were fixed.

Validator: `http://plv.mpi-sws.org/validc/`

Future Work

Integrate with validator for sequential programs.

Future Work

Integrate with validator for sequential programs.

Handle loops effectively.

Future Work

Integrate with validator for sequential programs.

Handle loops effectively.

Handle other language features (e.g. array, pointer)

Future Work

Integrate with validator for sequential programs.

Handle loops effectively.

Handle other language features (e.g. array, pointer)

use SAT/SMT solvers

Future Work

Integrate with validator for sequential programs.

Handle loops effectively.

Handle other language features (e.g. array, pointer)

use SAT/SMT solvers

Thank you !