From Array Domains to Abstract Interpretation Under Store-Buffer-Based Memory Models

Thibault Suzanne [1]

Antoine Miné [2]

9th September 2016

[1] École Normale Supérieure, PSL Research University; CNRS; Inria

[2] Sorbonne Universités, UPMC Univ Paris 6, Laboratoire d'informatique de Paris 6 (LIP6)

Weakly Consistent Behaviour

```
x = 1;  | y = 1;
r0 = y; | r1 = x;
```

The Programmer's Intuition: Sequential Consistency

After execution, `r0 = 1 || r1 = 1`.

On x86

We can observe `r0 = 0 && r1 = 0`.

A More Useful Program: Peterson's Lock Algorithm

```
/* Thread 0 */
flag_0 = true;
// mfence;
turn = true;
mfence;

while (flag1 && turn) { }
critical_section_thread0:
flag_0 = false;
// mfence;
```

```
/* Thread 1 */
flag_1 = true;
// mfence;
turn = false;
mfence;

while (flag_0 && not turn) { }
critical_section_thread1:
flag_1 = false;
// mfence;
```

Presentation of the Problem

- Most concurrent algorithms are designed and proved under sequential consistency.
- We focus on verifying reachability properties on relaxed memory models.
- We will use abstract interpretation and base our method on existing array domains.
- An application is fence removal.

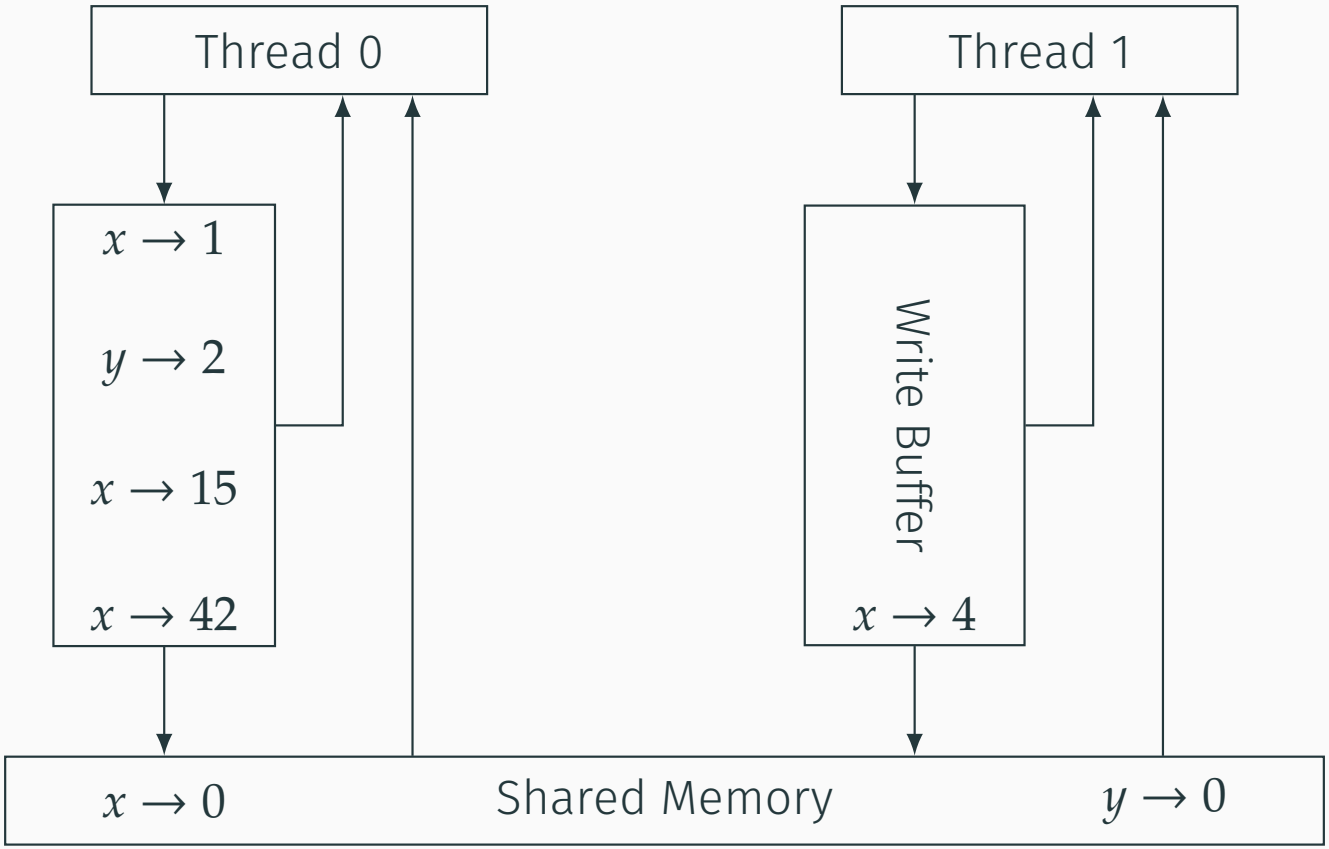
Plan

1. The Relaxed Memory Model
2. The Analysis Method
   - Partitioning
   - Summarisation
   - Numerical Domains
3. Experimentation
4. Future work

The Relaxed Memory Model

Total Store Ordering, the Base Model of x86

- Buffers are totally ordered FIFO queues.
- Buffers entries are flushed non-deterministically.
- Instruction `mfence` flushes the whole buffer of the thread.



Partial Store Ordering

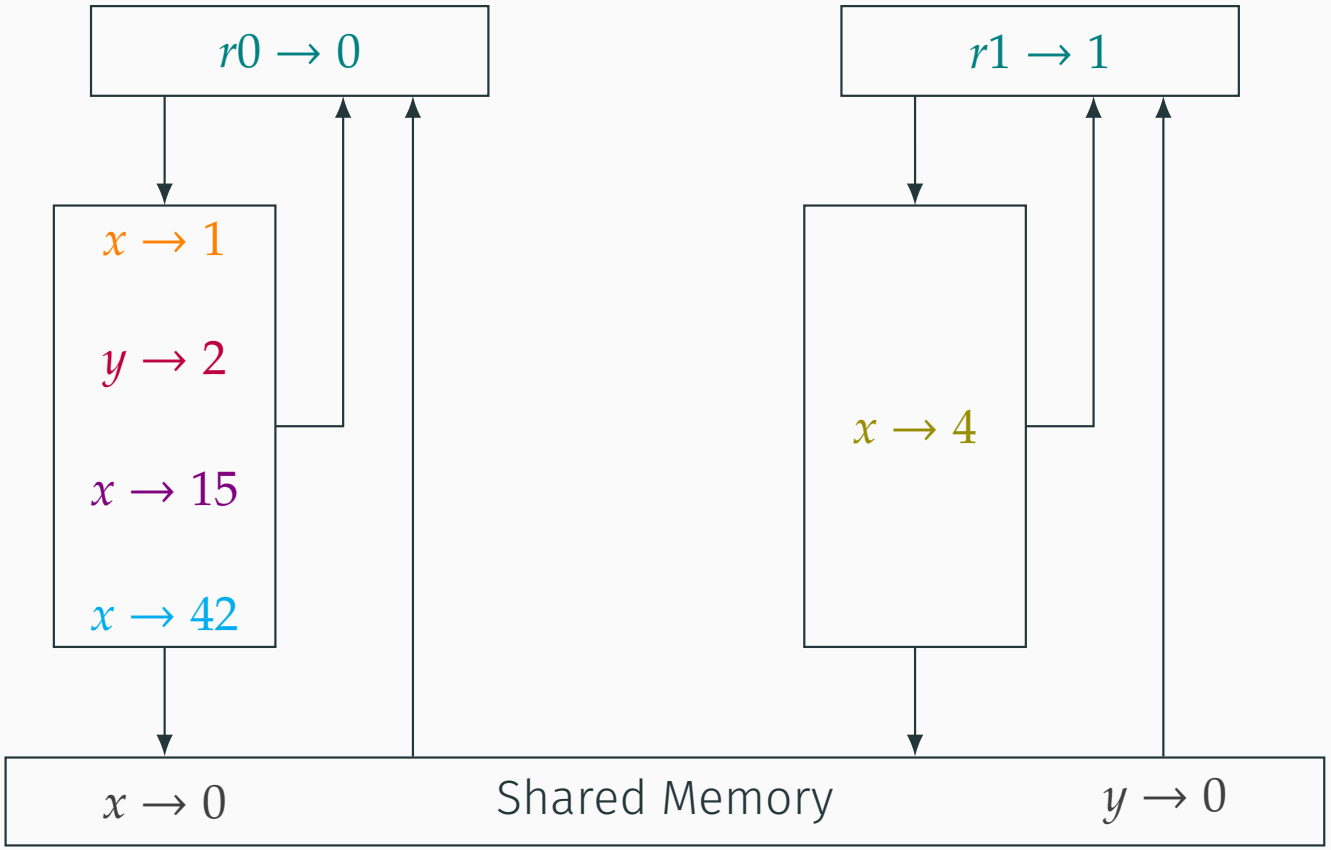
- Only preserves order between same variable entries.
- Equivalently, there is one buffer per variable (per thread).
- Mainly of theoretical interest.

PSO is strictly more relaxed than TSO: a sound analysis in PSO is also sound in TSO.

$\therefore$ PSO can be seen as an abstraction of TSO.

Direct Encoding

We use one pseudo-variable for each buffer entry.



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

$x_2^0 = 15$

$x_3^0 = 42$

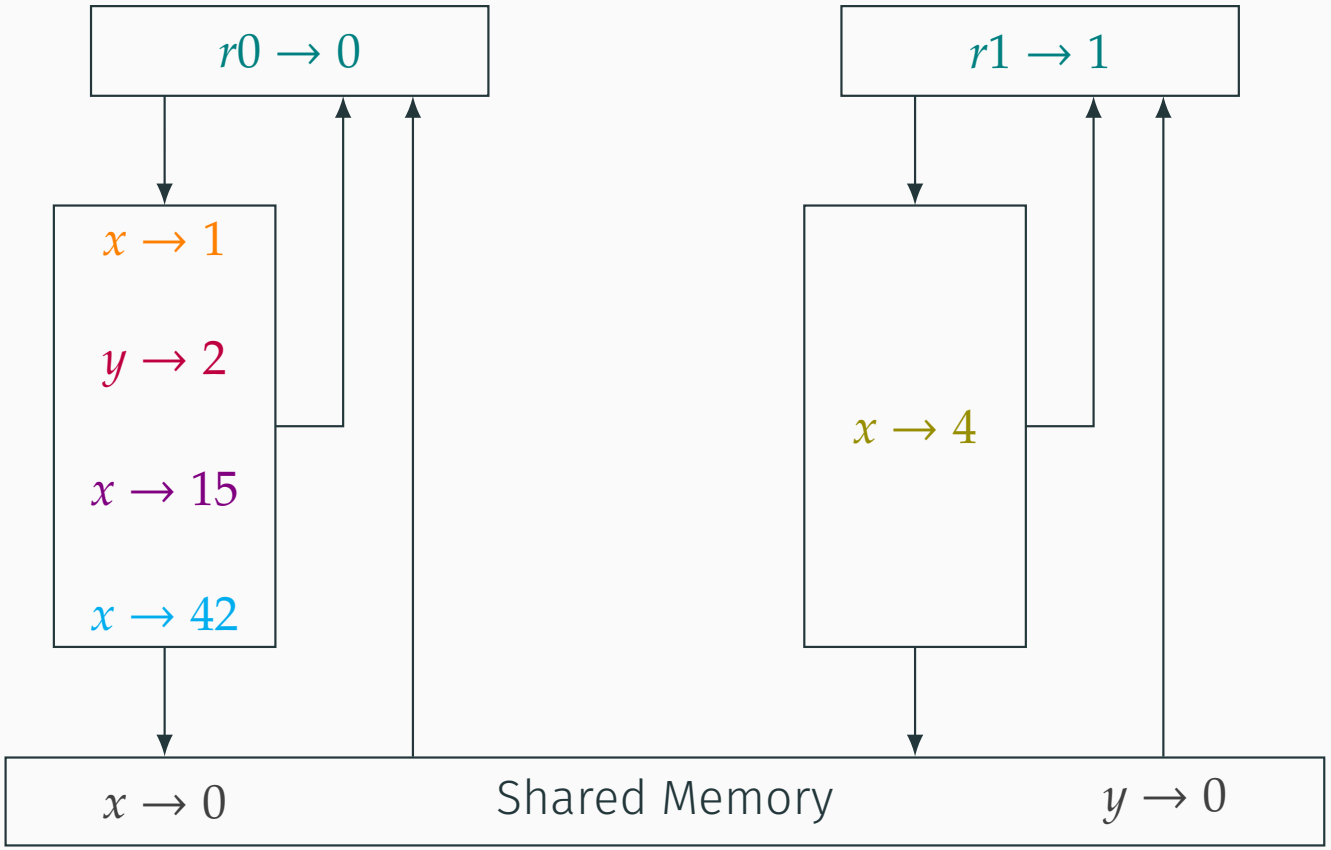
$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Direct Encoding

We use one pseudo-variable for each buffer entry.



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

$x_2^0 = 15$

$x_3^0 = 42$

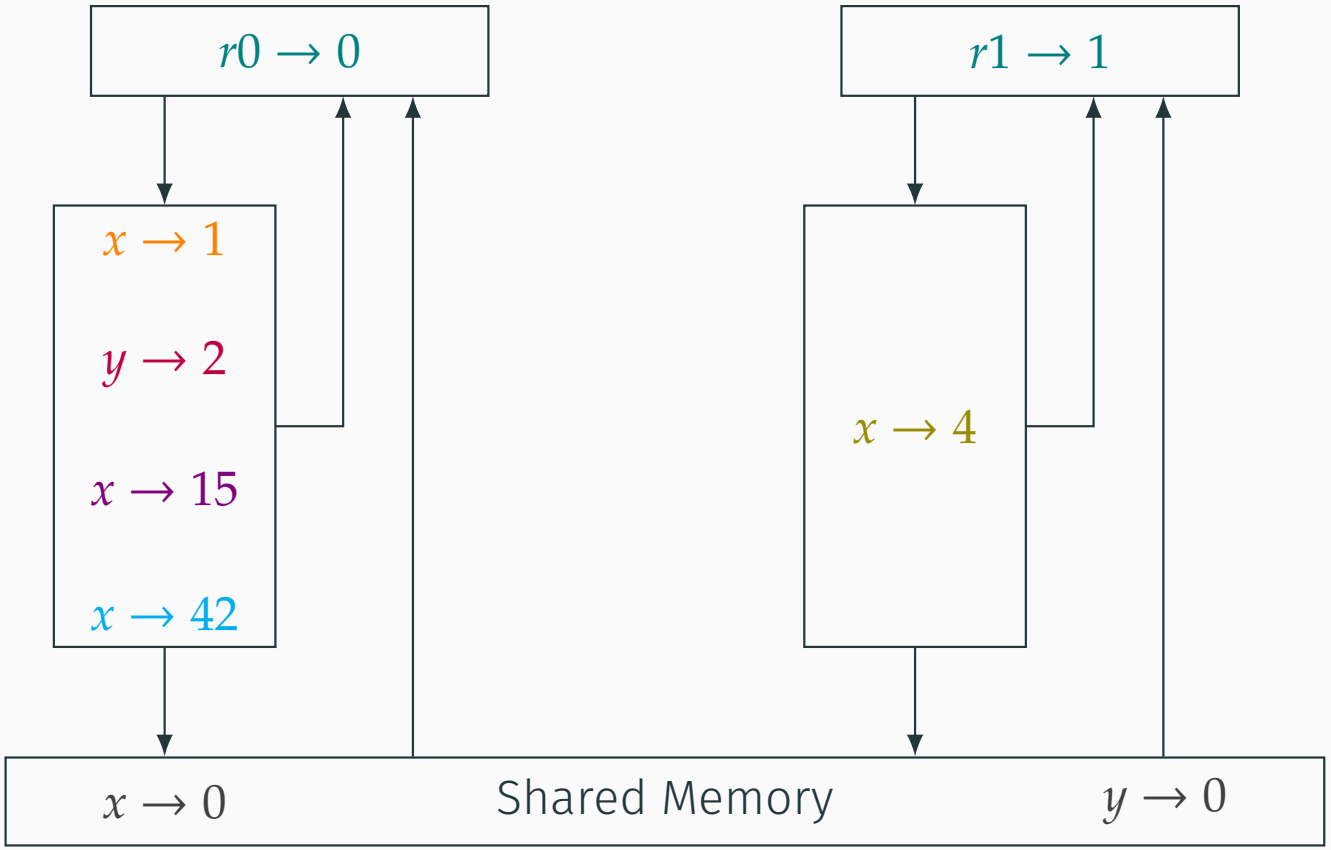
$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Direct Encoding

We use one pseudo-variable for each buffer entry.



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

$x_2^0 = 15$

$x_3^0 = 42$

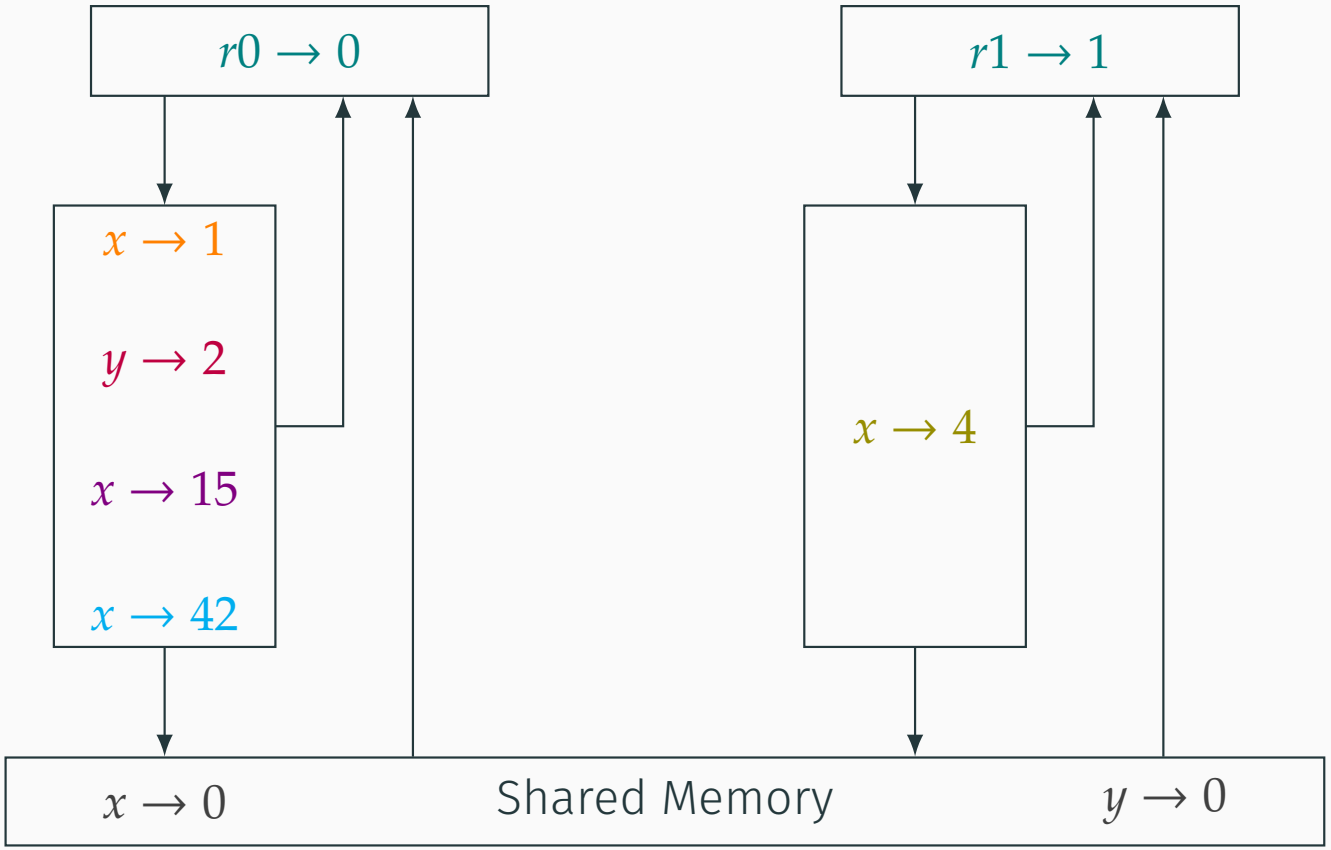
$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Direct Encoding

We use one pseudo-variable for each buffer entry.



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

$x_2^0 = 15$

$x_3^0 = 42$

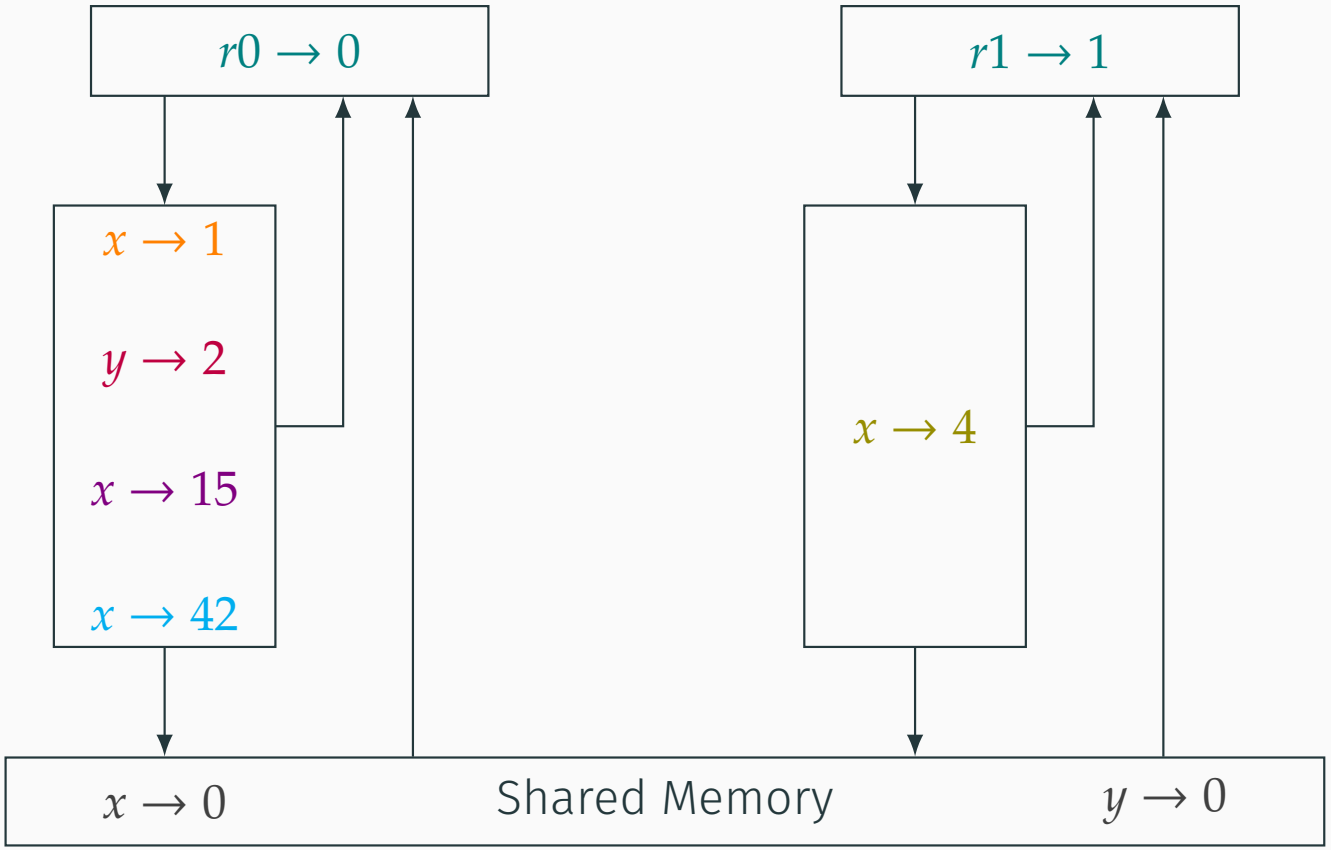
$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Direct Encoding

We use one pseudo-variable for each buffer entry.



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

$x_2^0 = 15$

$x_3^0 = 42$

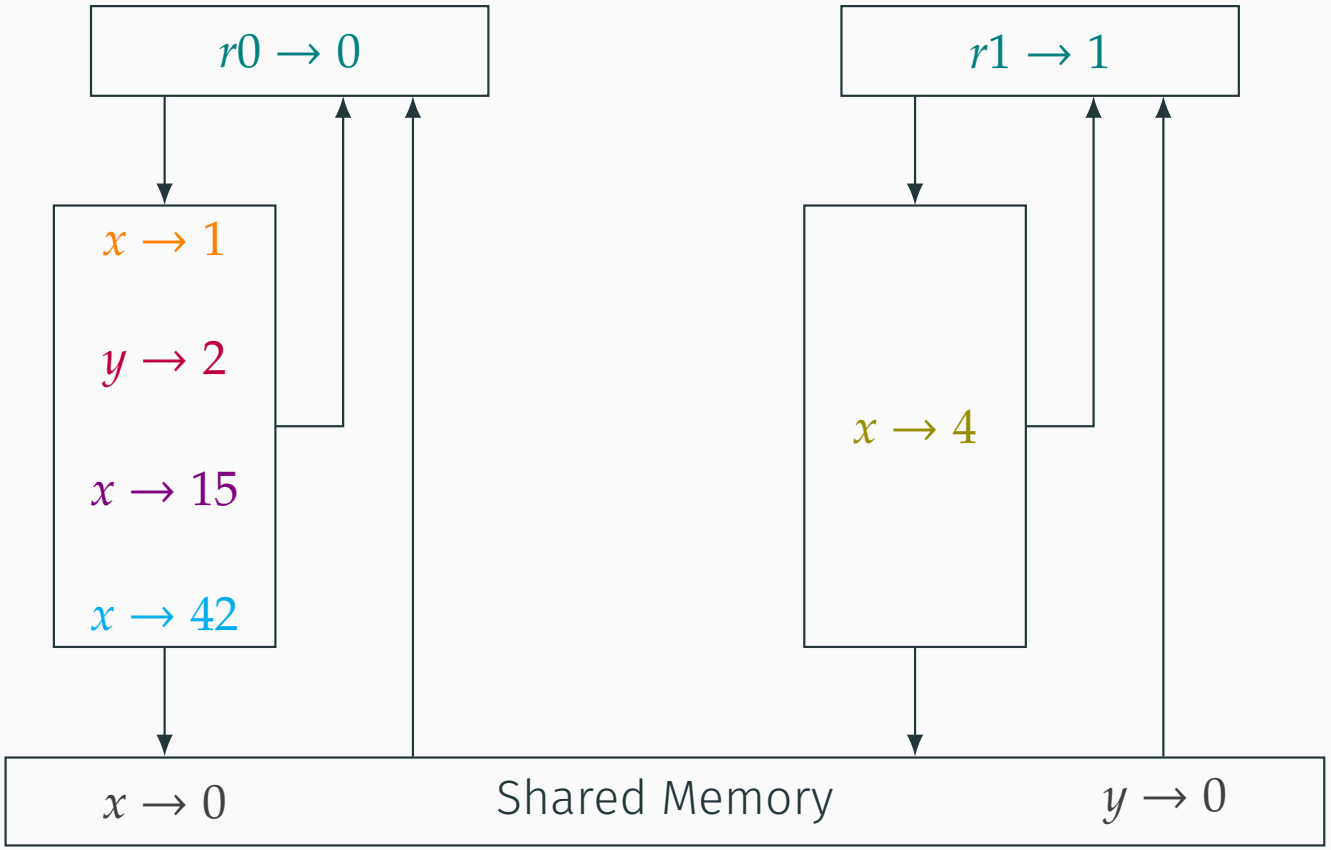
$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Direct Encoding

We use one pseudo-variable for each buffer entry.



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

$x_2^0 = 15$

$x_3^0 = 42$

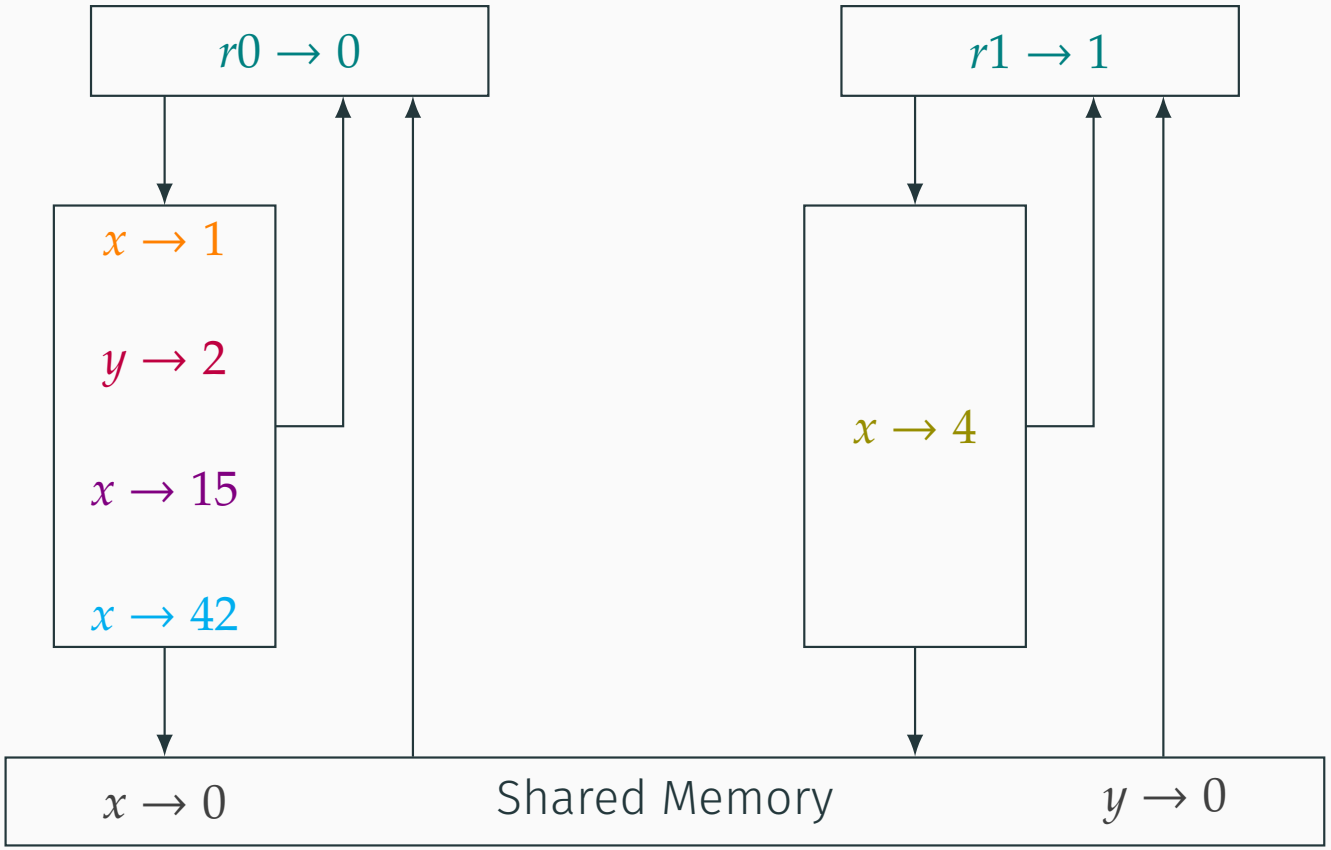
$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Direct Encoding

We use one pseudo-variable for each buffer entry.



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

$x_2^0 = 15$

$x_3^0 = 42$

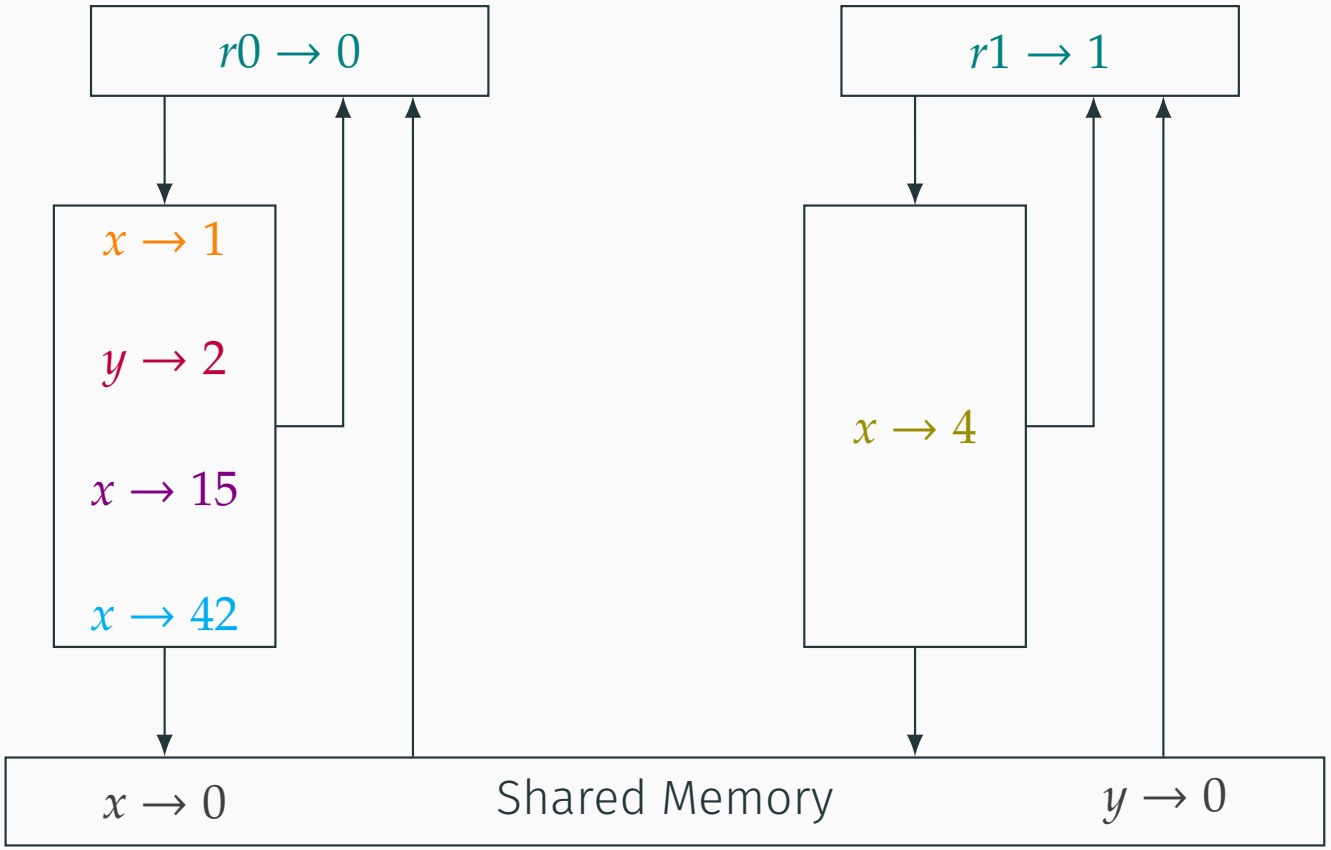
$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Direct Encoding

We use one pseudo-variable for each buffer entry.



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

$x_2^0 = 15$

$x_3^0 = 42$

$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Direct Encoding: Remark

The number of pseudo-variables obtained is :

1. Unbounded.
2. Different between two states.

We need a specific domain that :

1. Is able to abstract unbounded states.
2. Will allow us to use usual abstractions (like numerical domains) that operate on sets composed of states with one same dimension.

The Analysis Method

The Interleaving Control Graph

- We build the product control flow graph representing all possible interleavings.
- Each edge has a self loop `while(random) { flush oldest };` to represent the non-determinism of flushes.
- The analysis now becomes a usual fixpoint computation.

The Interleaving Control Graph

- We build the product control flow graph representing all possible interleavings.
- Each edge has a self loop `while(random) { flush oldest };` to represent the non-determinism of flushes.
- The analysis now becomes a usual fixpoint computation.

The Interleaving Control Graph

- We build the product control flow graph representing all possible interleavings.
- Each edge has a self loop `while(random) { flush oldest };` to represent the non-determinism of flushes.
- The analysis now becomes a usual fixpoint computation.

General Idea

Buffers are the hard(est) part of the analysis:

- They have an unbounded size.
- This size can change in a dynamic and undeterministic way on virtually each execution step.

We propose to adapt array abstractions to efficiently represent buffers.

The Analysis Method

Partitioning

Partitioning Criterion

Observation

The behaviour of program operations is sensibly different whether the buffers are empty or not.

- A state has one buffer per variable per thread.
- We can abstract the length of these buffers: either it is 0, 1 or $1+$ (more than 1).
- We regroup together the states that have the same buffer abstract lengths.

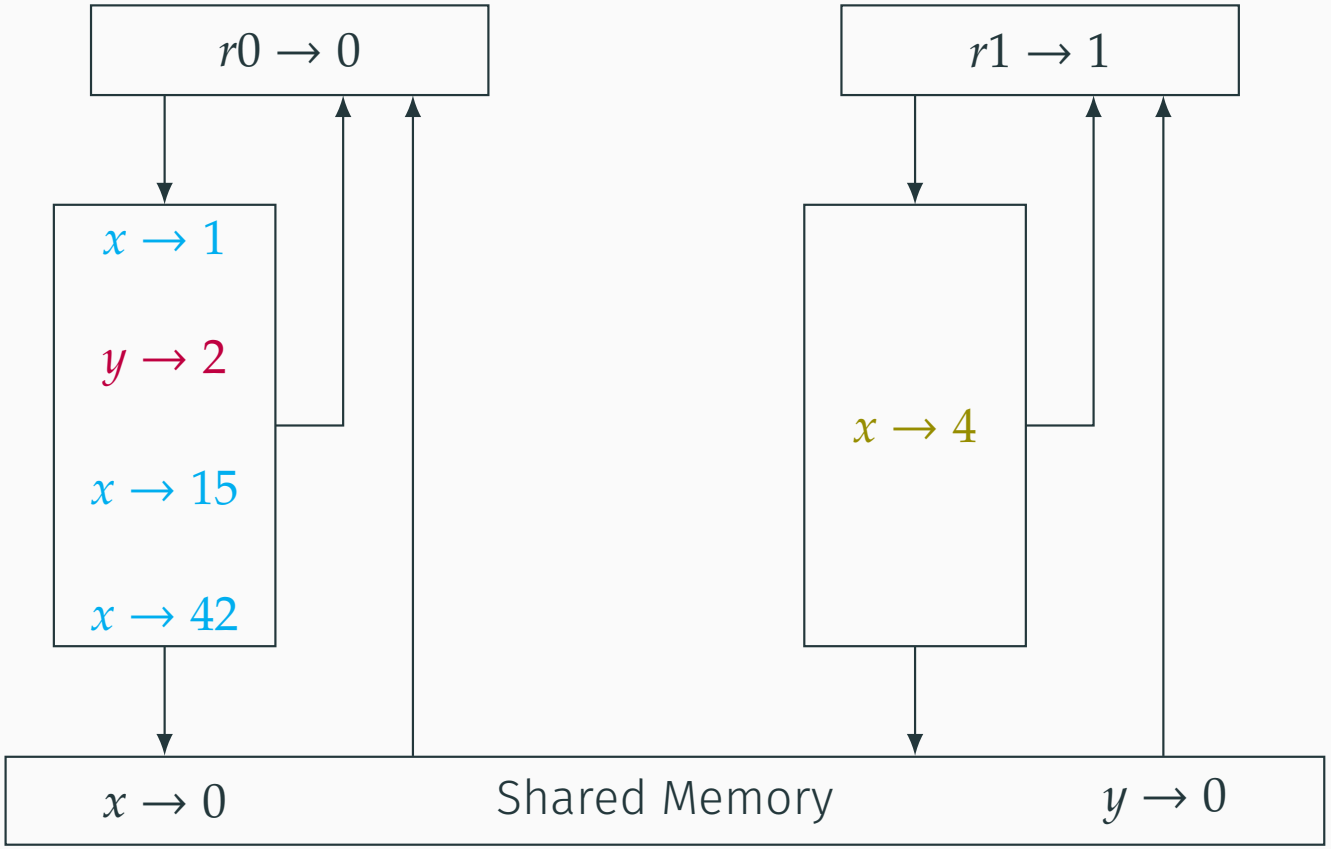
Partitioning Criterion

Observation

The behaviour of program operations is sensibly different whether the buffers are empty or not.

- A state has one buffer per variable per thread.
- We can abstract the length of these buffers: either it is 0, 1 or $1+$ (more than 1).
- We regroup together the states that have the same buffer abstract lengths.

Abstract Buffer Sizes: an Example



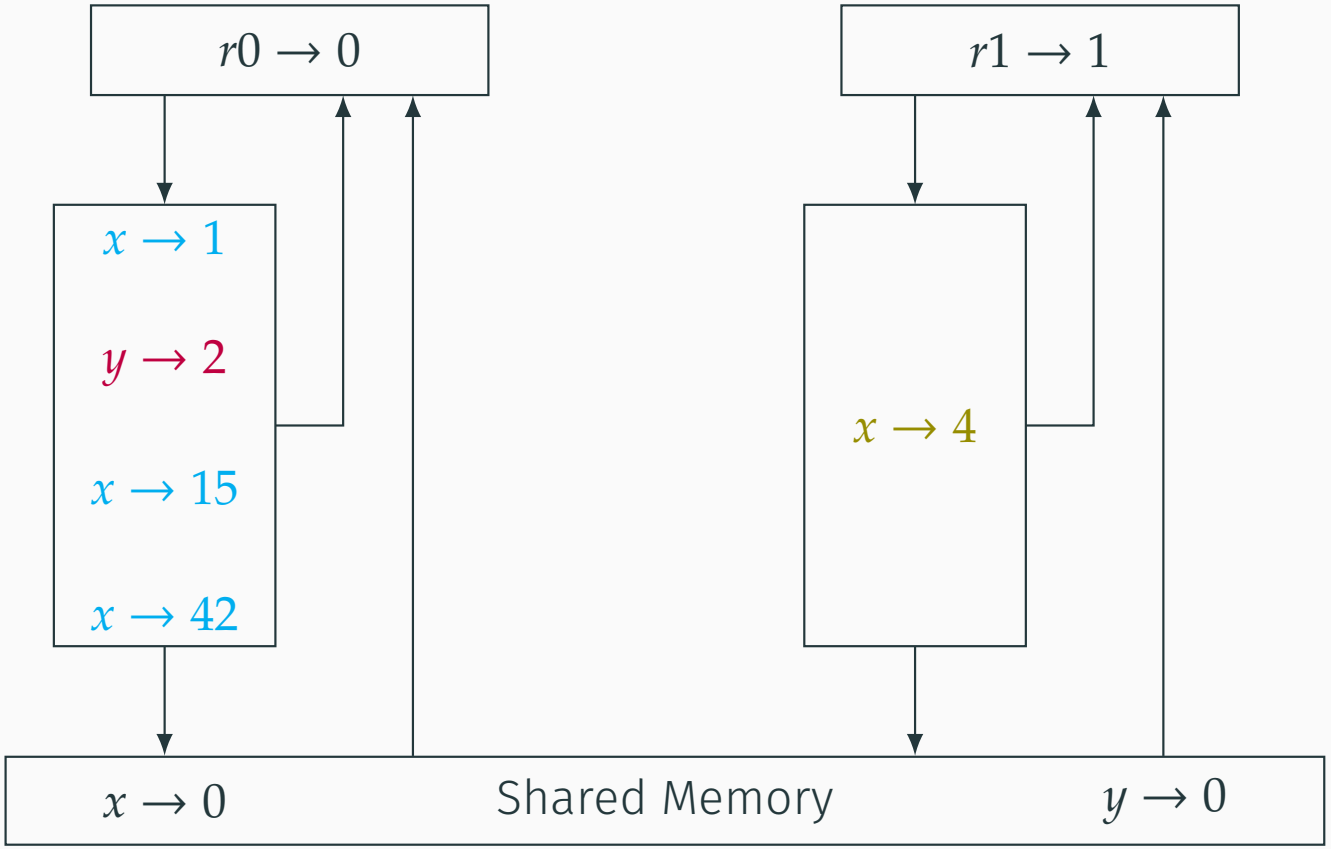
$(x, 0) \mapsto 1+$

$(y, 0) \mapsto 1$

$(x, 1) \mapsto 1$

$(y, 1) \mapsto 0$

Abstract Buffer Sizes: an Example



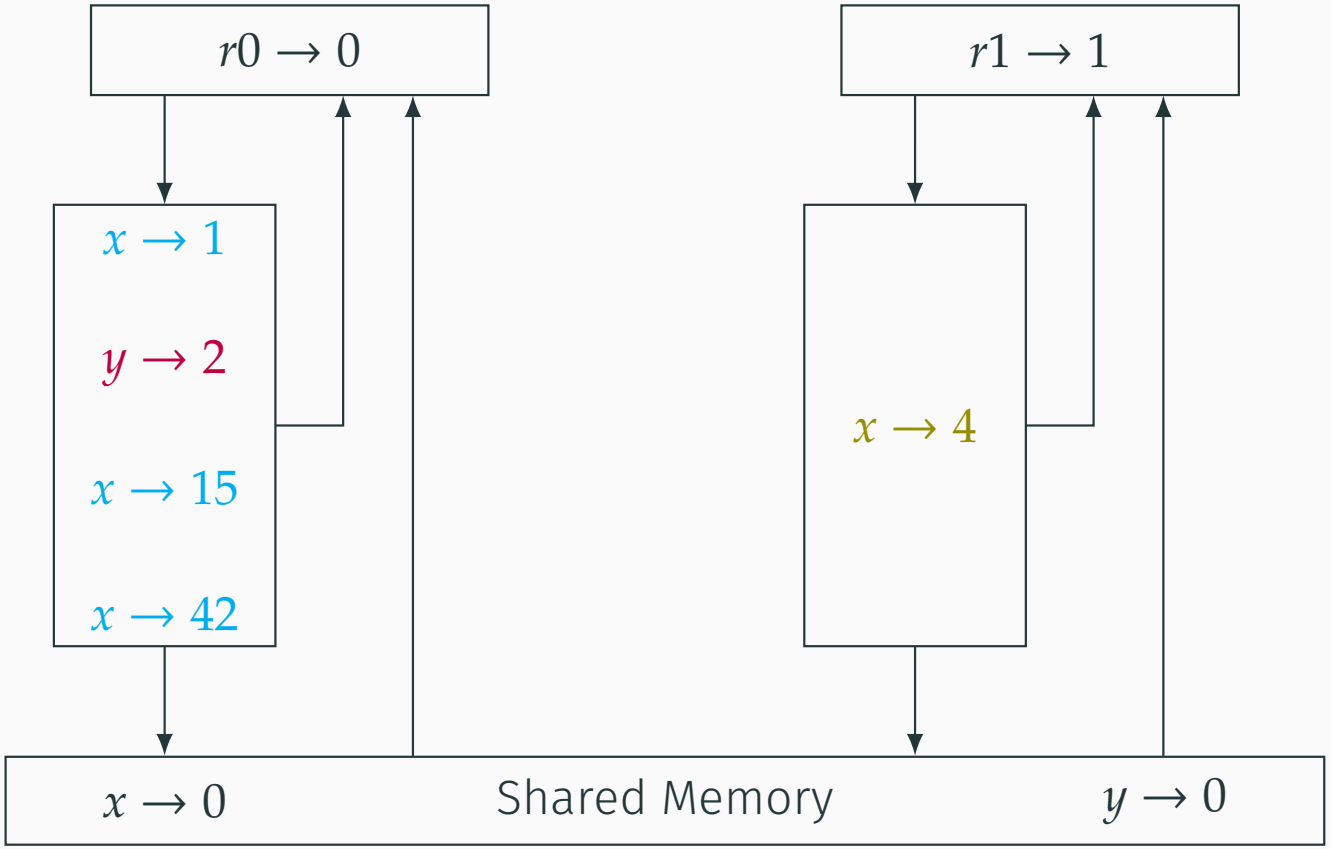
$(x, 0) \mapsto 1+$

$(y, 0) \mapsto 1$

$(x, 1) \mapsto 1$

$(y, 1) \mapsto 0$

Abstract Buffer Sizes: an Example



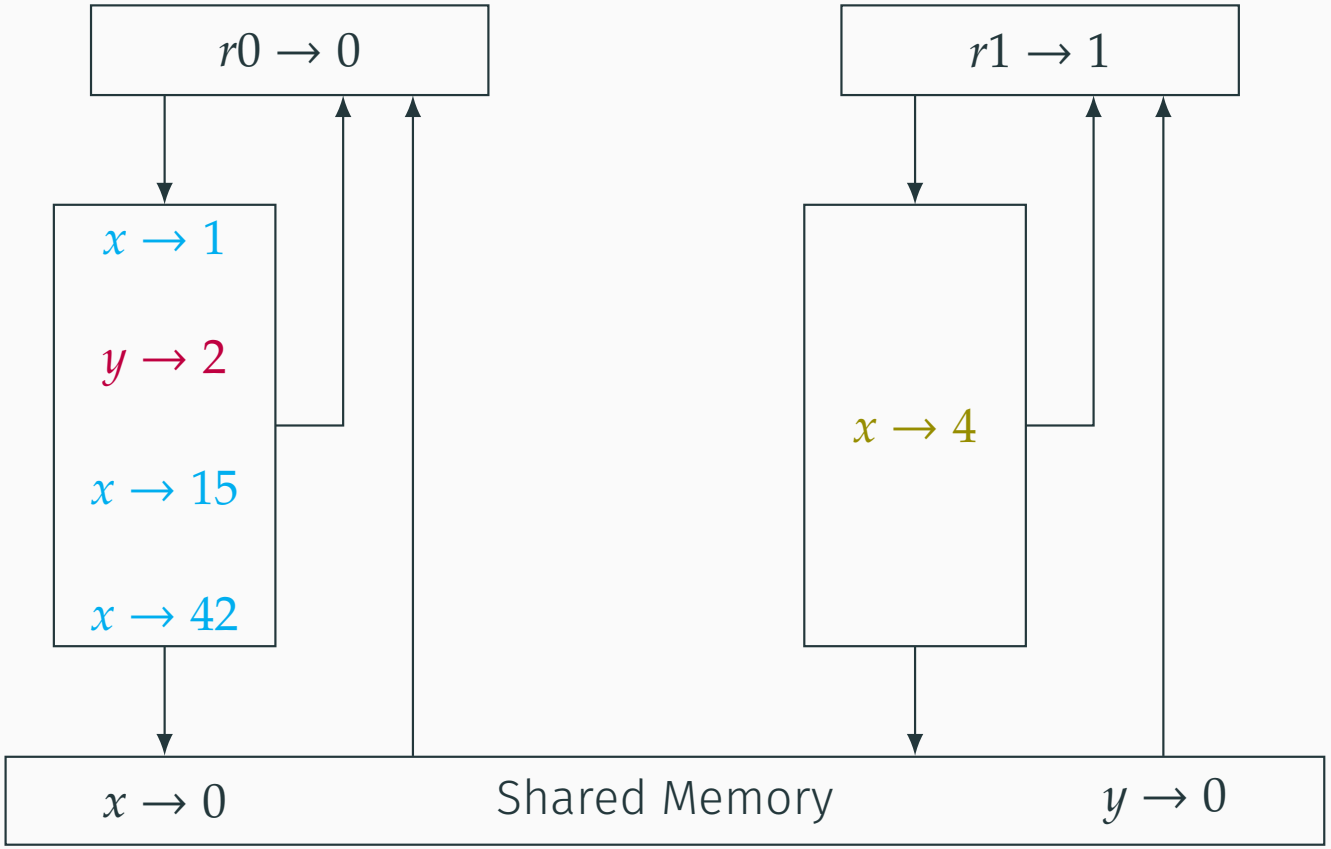
$(x, 0) \mapsto 1+$

$(y, 0) \mapsto 1$

$(x, 1) \mapsto 1$

$(y, 1) \mapsto 0$

Abstract Buffer Sizes: an Example



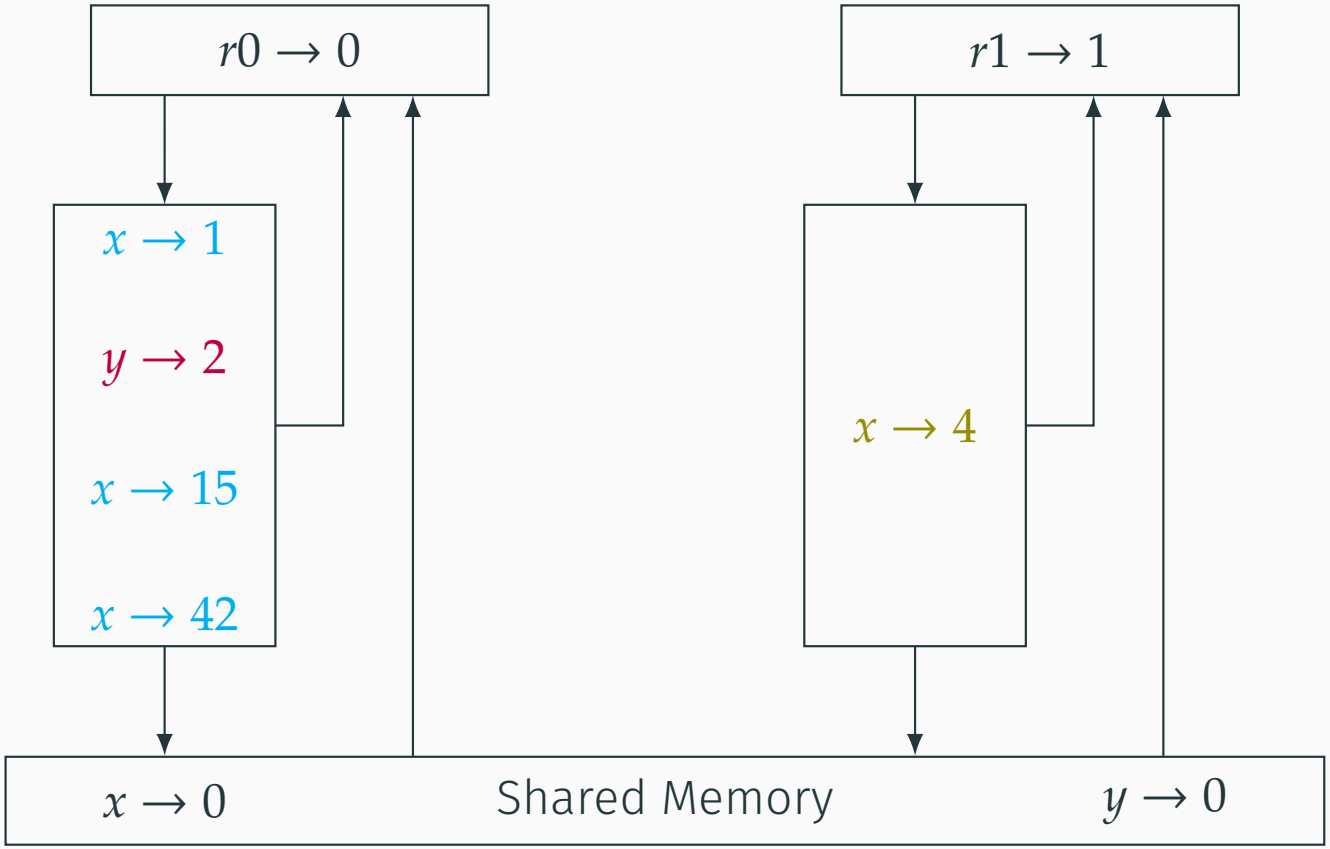
$(x, 0) \mapsto 1+$

$(y, 0) \mapsto 1$

$(x, 1) \mapsto 1$

$(y, 1) \mapsto 0$

Abstract Buffer Sizes: an Example



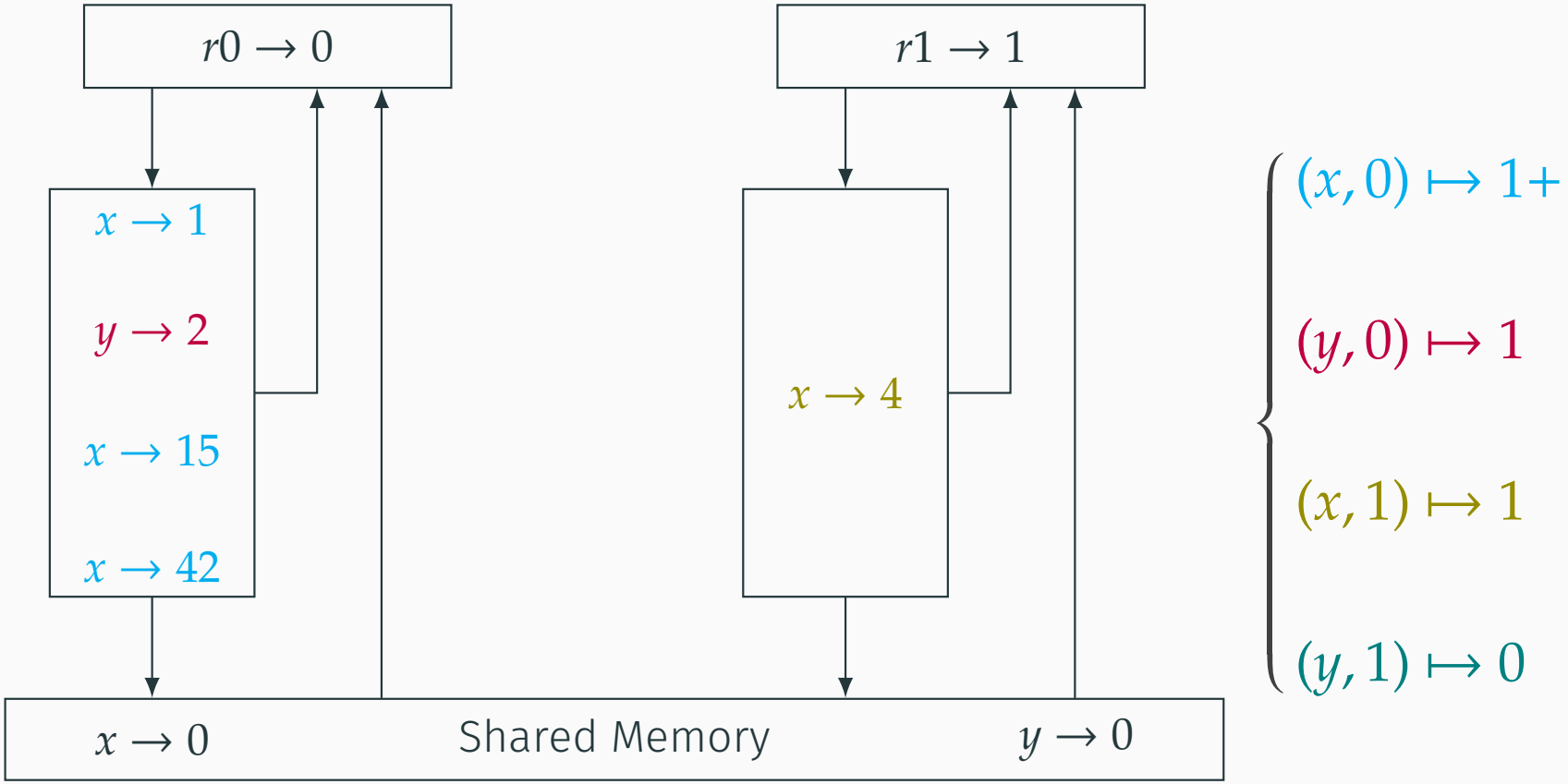
$(x, 0) \mapsto 1+$

$(y, 0) \mapsto 1$

$(x, 1) \mapsto 1$

$(y, 1) \mapsto 0$

Abstract Buffer Sizes: an Example



The Analysis Method

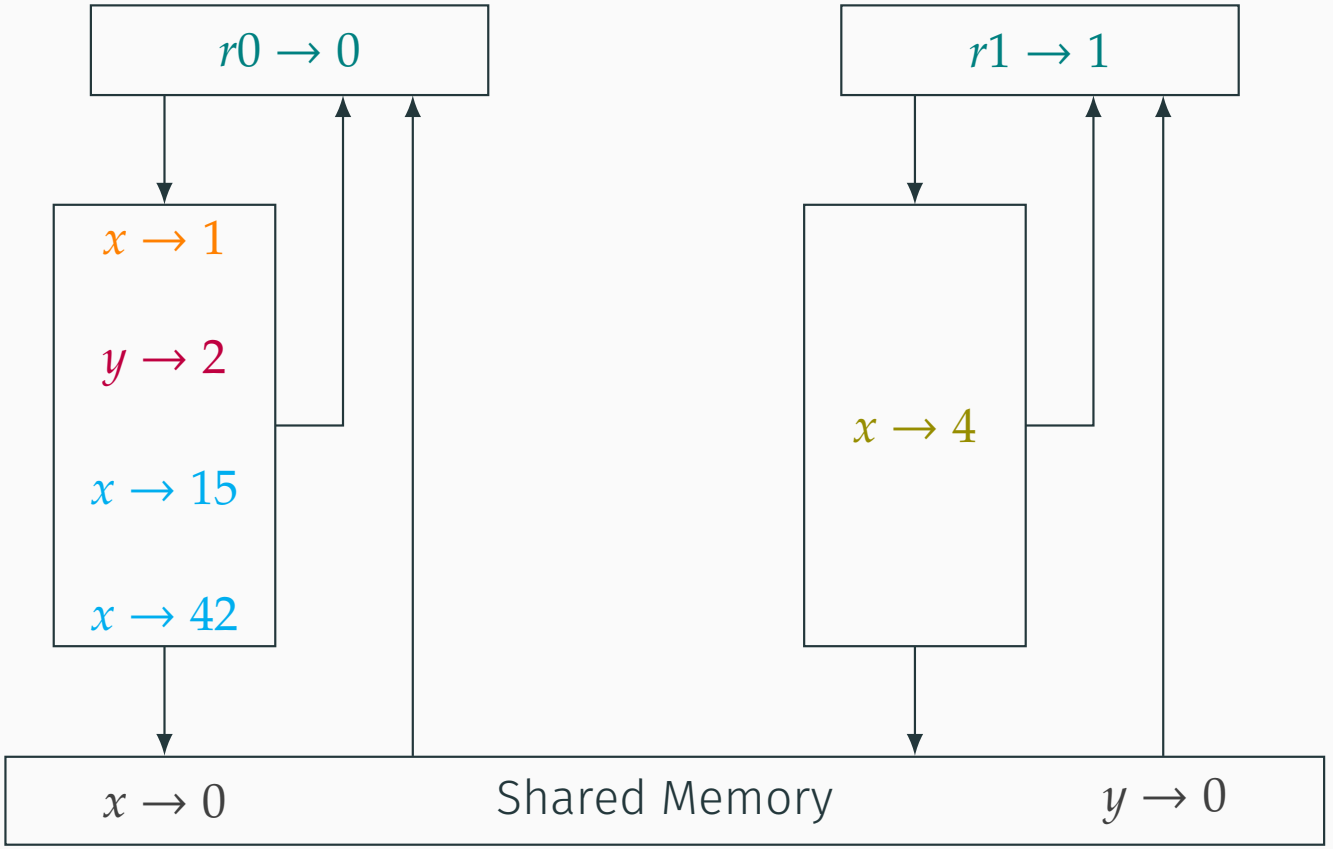
Summarisation

Summarising the Buffers

In each state where they are defined ($1+$ in the partition), we regroup the variables $x_2^T, ..., x_\infty^T$ into a single summarised variable x_{bot}^T.

x_1^T is kept separated: otherwise, reading from the buffer would be very imprecise.

Summarisation: an Example



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

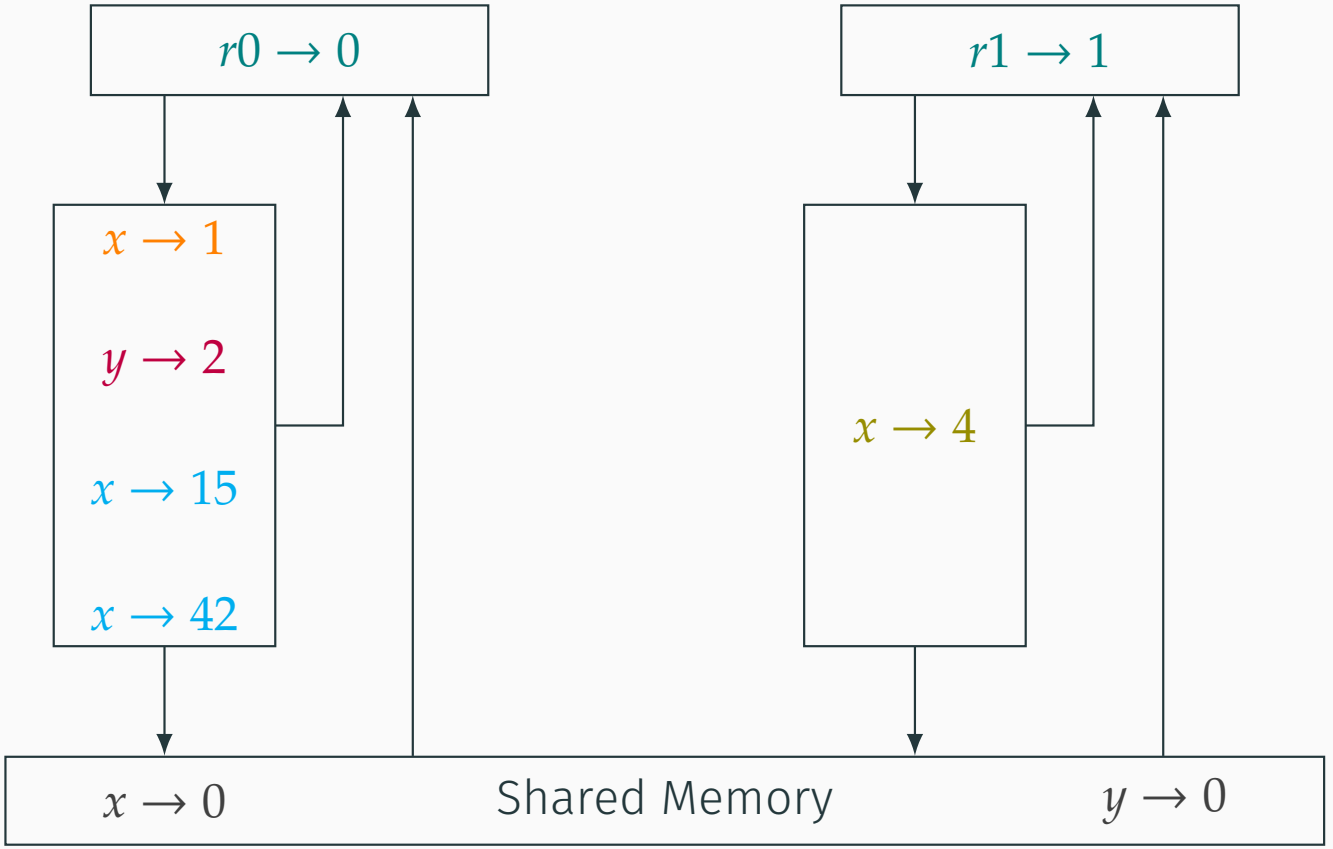
$x_{bot}^0 \in \{15; 42\}$

$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Summarisation: an Example



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

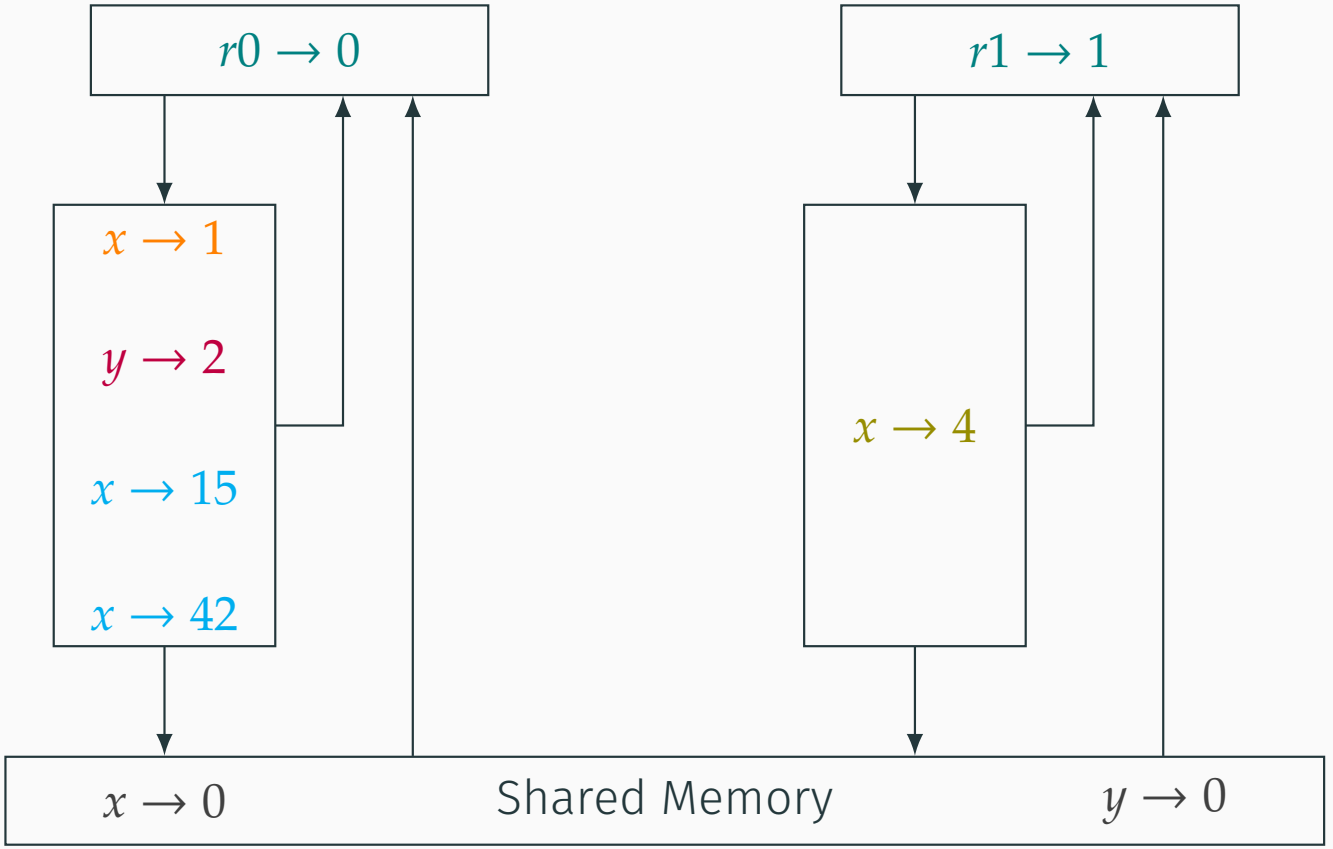
$x_{bot}^0 \in \{15; 42\}$

$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Summarisation: an Example



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

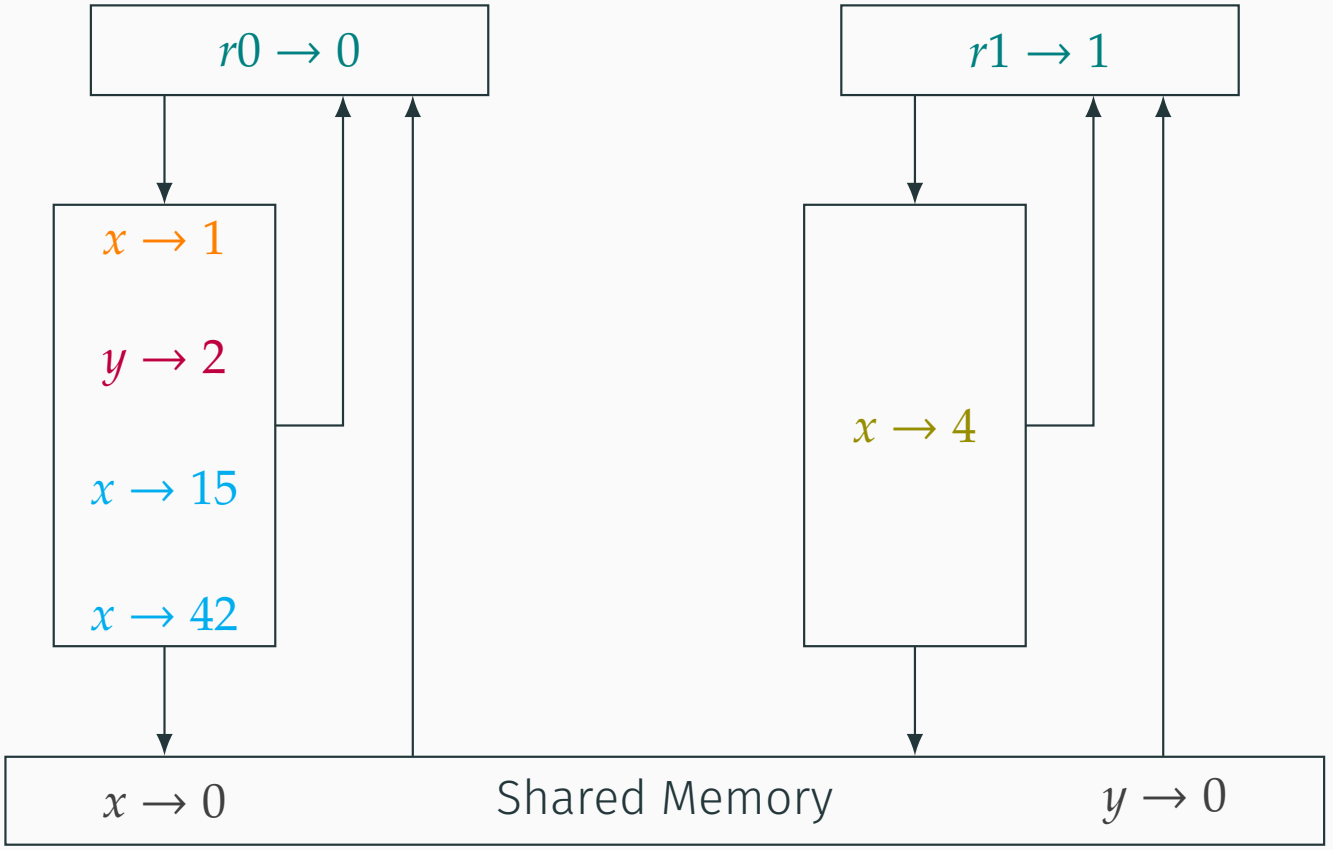
$x_{bot}^0 \in \{15; 42\}$

$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Summarisation: an Example



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

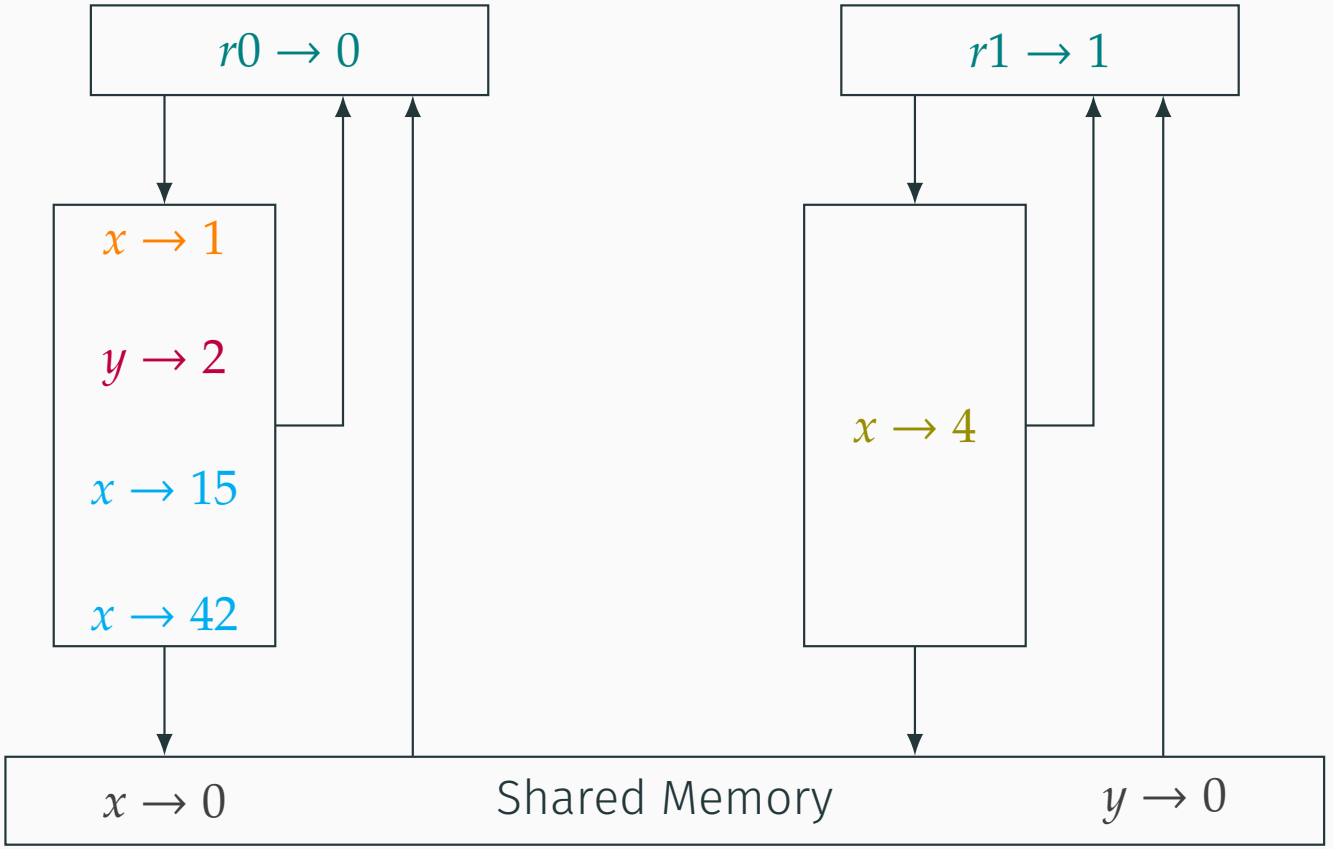
$x_{bot}^0 \in \{15; 42\}$

$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Summarisation: an Example



$r0 = 0$

$r1 = 1$

$x_1^0 = 1$

$y_1^0 = 2$

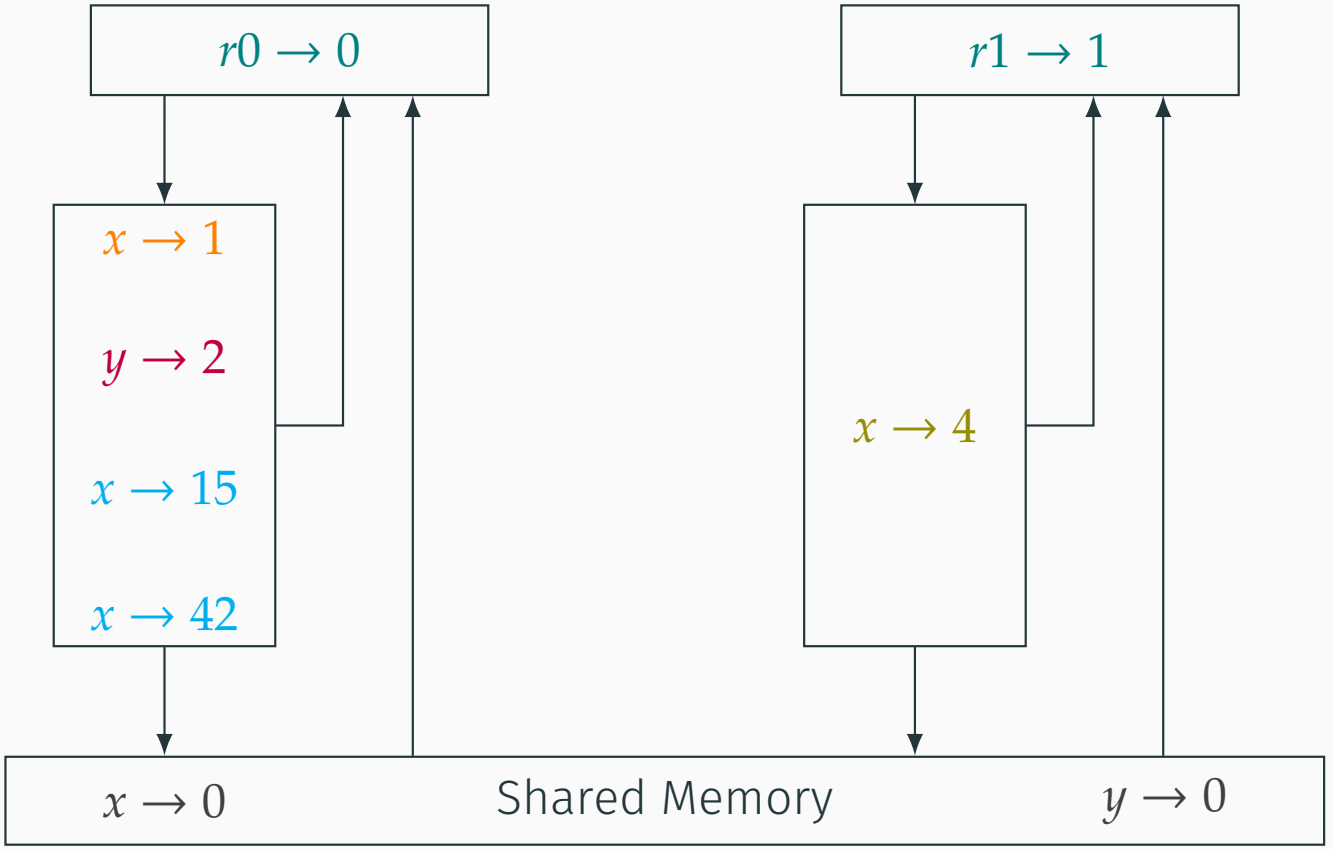
$x_{bot}^0 \in \{15; 42\}$

$x_1^1 = 4$

$x^{mem} = 0$

$y^{mem} = 1$

Summarisation: an Example



$$r0 = 0$$

$$r1 = 1$$

$$x_1^0 = 1$$

$$y_1^0 = 2$$

$$x_{bot}^0 \in \{15; 42\}$$

$$x_1^1 = 4$$

$$x^{mem} = 0$$

$$y^{mem} = 1$$

The Analysis Method

Numerical Domains

Using Numerical Domains for Abstraction

In each partition, all the states have the same buffer lengths. Therefore the same summarised variables are defined.

Then we can use numerical domains (octagons, polyhedra...) to abstract each partition.

Experimentation

Results

Algorithm	Fences	Time	Fences*	Time*
Abp	0	0.3	0	6
Bakery	-	-	4	3429
Concloop	2	0.19	2	6
Dekker	4	23	4	121
Kessel	4	4	4	6
Loop2 TLM	0	4.3	2	36
Peterson	4	1.53	4	20
Queue	0	0.15	1	1

* Dan, Meshman, Vechev, Yahav. Effective abstractions for verification under relaxed memory models. *VMCAI 2014*.

Future work

Improving Performances

- There can be a lot of non-empty partitions. Practically, we found their number to be usually quite low.
  Solution: an abstraction that merges partitions.
- Each numerical domain has a high dimension, which is $O(nb\_var \times nb\_threads)$.
  Solution: packing.
- The control flow graph has an exploding size.
  Solution: thread-modular analysis.

Order-Preserving Abstractions

- Our abstraction loses order information between two different variables.
- Summarisation makes it non-trivial to keep it.
  Eg.: consider this concrete buffer:

$$x \to 1 \bullet x \to 2 \bullet y \to 3 \bullet x \to 4 \bullet y \to 5$$

- Idea: summarised order information.

$$x_0 < y_0 \wedge x_{bot} < y_{bot}$$

  The flush operation cannot flush x anymore.

Order Matters: Back to Peterson

The first (commented) fence is needed in PSO but not in TSO.

```
/* Thread 0 */
flag_0 = true;
// mfence;
turn = true;
mfence;

while (flag1 && turn) { }
critical_section_thread0:
flag_0 = false;
// mfence;
```

```
/* Thread 1 */
flag_1 = true;
// mfence;
turn = false;
mfence;

while (flag_0 && not turn) { }
critical_section_thread1:
flag_1 = false;
// mfence;
```

Non-Uniform Abstractions

In general, we would like to use a wide range of table abstractions, including non-uniform ones.

They could be used to encode some specific properties of the program, like “the buffer is sorted”, which can be hard to do with a program transformation approach.

Related Work

Analysis Under Weak Memory Models

- Kuperstein, Vechev, Yahav. Partial-coherence abstractions for relaxed memory models. *ACM SIGPLAN Notices 2011.*
- Dan, Meshman, Vechev, Yahav. Effective abstractions for verification under relaxed memory models. *VMCAI 2014.*
- Alglave, Kroening, Nimal, Tautschnig. Software verification for weak memory via program transformation. *ESOP 2013.*

Table Abstractions

- Gopan, DiMaio, Dor, Reps, Sagiv. Numeric domains with summarized dimensions. *TACAS 2004.*
- Cousot, Cousot, Logozzo. A parametric segmentation functor for fully automatic and scalable array content analysis. *ACM SIGPLAN Notices 2011.*

Conclusion

- Our general framework adapts array abstractions to represent buffers.
- With summarisation, we get a computable sound analysis for TSO, which also happens to be sound for PSO.
- This analysis is both precise and efficient compared to the state of the art.

Thanks for your attention !