

MemSAT

**checking axiomatic
specifications of
memory models**

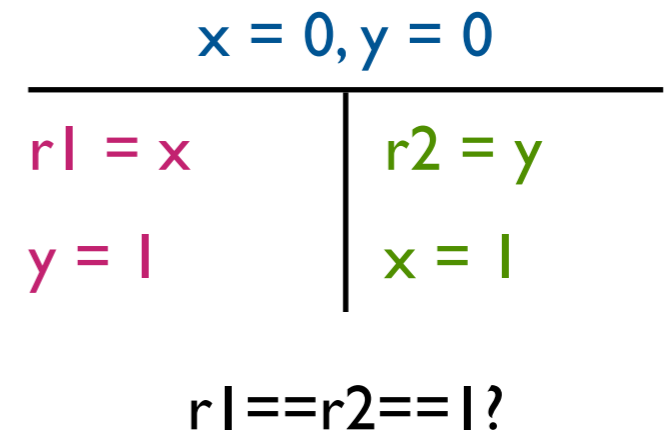
Emina Torlak · Mandana Vaziri · Julian Dolby

MIT SAT/SMT Summer School · June 16, 2011

Introduction

memory model

- ▶ contract between programmer and programming environment
- ▶ specifies which writes can be seen by a read



Introduction

memory model

- ▶ contract between programmer and programming environment
- ▶ specifies which writes can be seen by a read
- ▶ described (in)formally by a set of **axioms** and **litmus tests**

$x = 0, y = 0$

$r1 = x$	$r2 = y$
$y = 1$	$x = 1$

$r1 == r2 == 1?$

Introduction

memory model

- ▶ contract between programmer and programming environment
- ▶ specifies which writes can be seen by a read
- ▶ described (in)formally by a set of **axioms** and **litmus tests**
- ▶ **hard to design and reason about**

On Validity of Program Transformations in the Java Memory Model 45

Definition 2. An execution E is a tuple $E = (A, P, \leq_{po}, \leq_{so}, W, V)$, where $A \subseteq \mathcal{A}$ is a set of actions; P is a program, represented as a thread-indexed set of memory traces; the partial order $\leq_{po} \subseteq A \times A$ is the program order, which is a union of total orders $\leq_{po}^i \subseteq A \times A$ is the synchronisation order.

Definition 7. We say that a program P is sequentially consistent if

A JMM Definitions

The following definitions are used in the definition of execution. The starting point is the definition of execution names, m for synchronisation locations, in example 1. The starting point is the definition of execution names, m for synchronisation locations, in example 1. The starting point is the definition of execution names, m for synchronisation locations, in example 1.

Definition 1. An abstract type A is a set of values. For each thread, we will assume reasonable properties of execution. We assume reasonable properties of execution. We assume reasonable properties of execution.

We denote the associated operations by $Rd_v(v)$, $Wr_v(v)$, U , L , S , F , U , L , S , F . The JMM also defines the operations by $Rd_v(v)$, $Wr_v(v)$, U , L , S , F . The JMM also defines the operations by $Rd_v(v)$, $Wr_v(v)$, U , L , S , F .

well-formed execution and for each $i > 0$

1. $C_i \subseteq A_i$.

2. For all reads r in C_i , $W(r) \subseteq C_i$.

3. $V|_{C_i} = V|_{C_{i-1}}$.

4. $W|_{C_{i-1}} = W|_{C_i}$.

Definition 6. We say that program P is well-formed if sequential validity of trace t in P implies:

1. any prefix t' of t is sequentially valid (prefix closedness).
2. if t is a read with value v , then the trace obtained from t by replacing the last action by v' is also sequentially valid in P (independence).

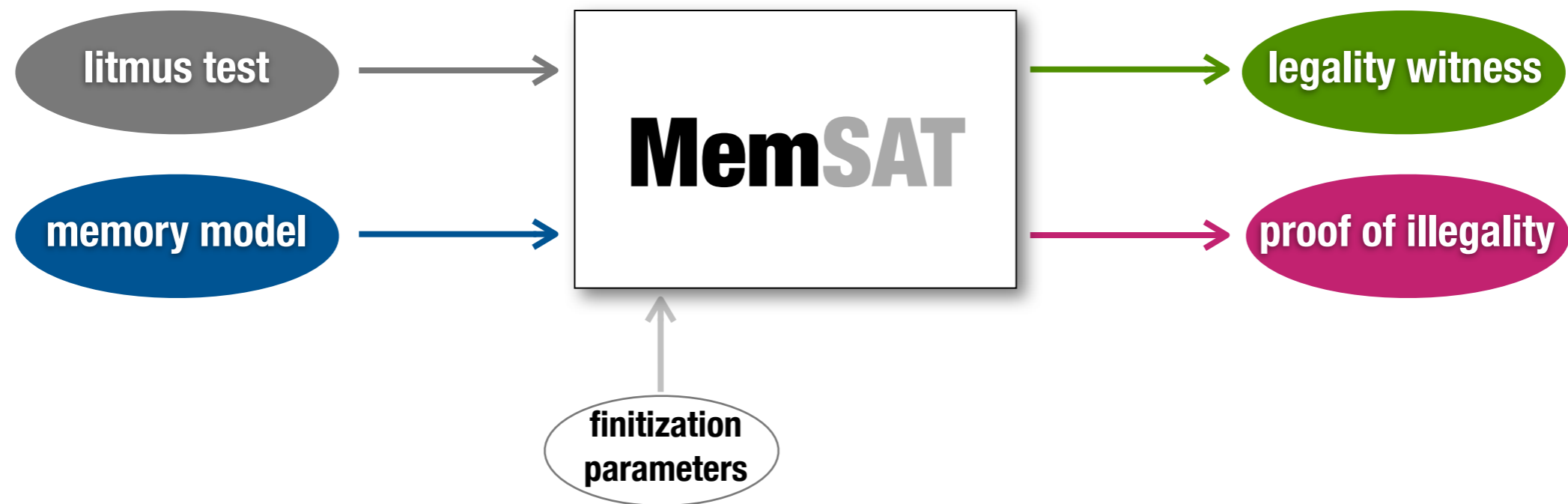
$x = 0, y = 0$

$r1 = x$	$r2 = y$
$y = 1$	$x = 1$

$r1 == r2 == 1?$

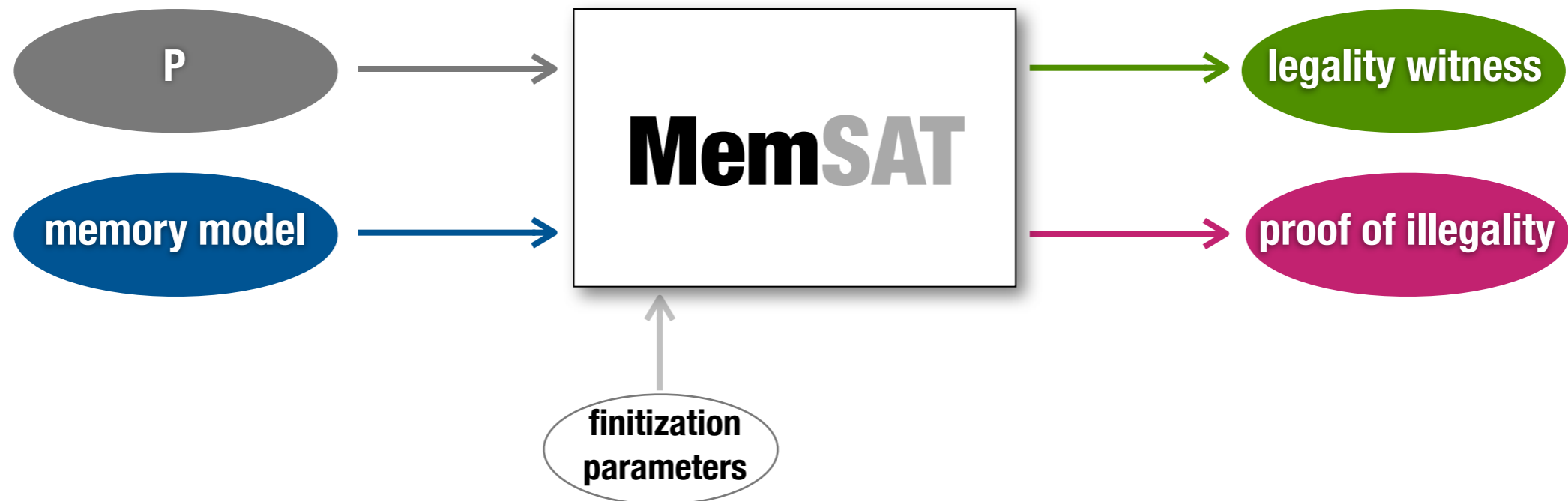


MemSAT overview



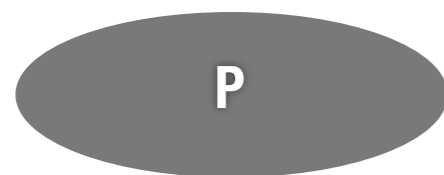
MemSAT overview

annotated java
program with one
or more assertions

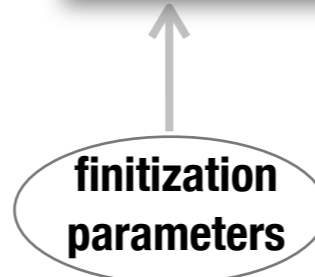


MemSAT overview

annotated java
program with one
or more assertions



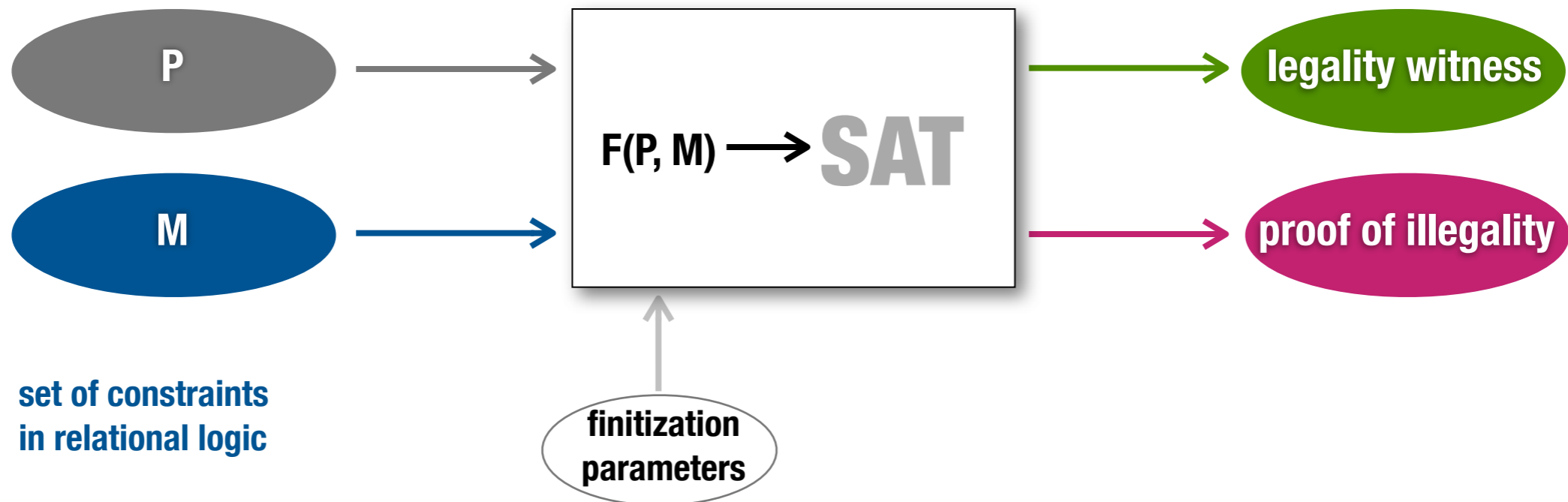
set of constraints
in relational logic



MemSAT overview

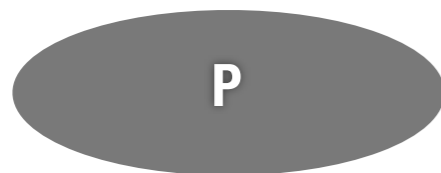
annotated java
program with one
or more assertions

- translate P to relational logic
- combine result with M



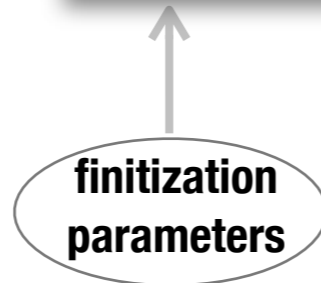
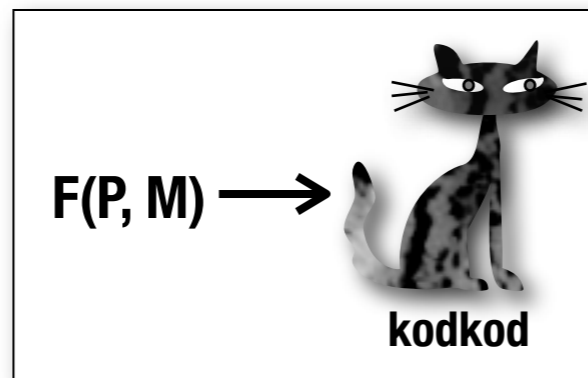
MemSAT overview

annotated java
program with one
or more assertions



set of constraints
in relational logic

- translate P to relational logic
- combine result with M
- solve combined constraints



legality witness



proof of illegality

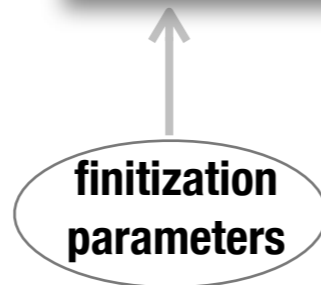
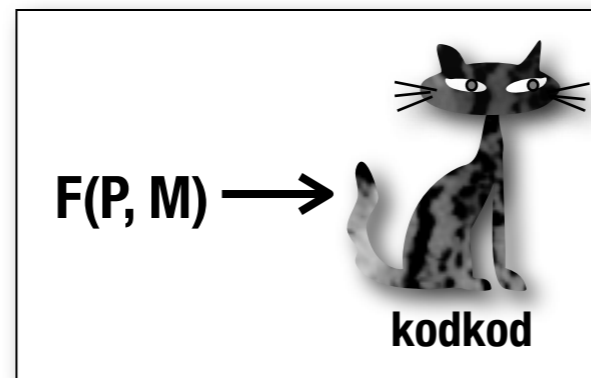
MemSAT overview

annotated java
program with one
or more assertions



set of constraints
in relational logic

- ▶ translate P to relational logic
- ▶ combine result with M
- ▶ solve combined constraints



model (solution) of
the legality formula



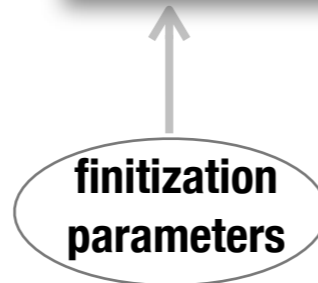
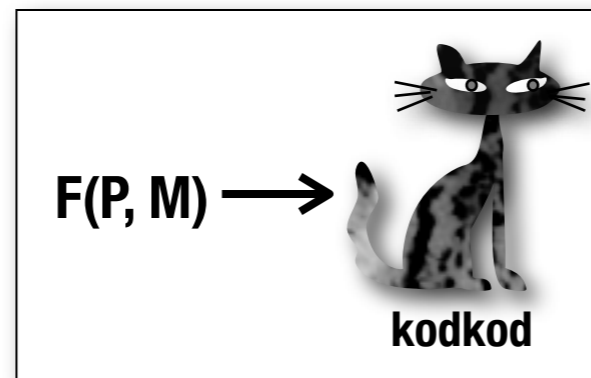
MemSAT overview

annotated java
program with one
or more assertions



set of constraints
in relational logic

- translate P to relational logic
- combine result with M
- solve combined constraints



model (solution) of
the legality formula



minimal unsatisfiable
core of the legality formula

Specifying a litmus test

$x = 0, y = 0$	
$r1 = x$	$r2 = y$
$y = 1$	$x = 1$

$r1 == r2 == 1?$

- ▶ control flow
- ▶ synchronize
- ▶ method calls
- ▶ field and array accesses
- ▶ assertions

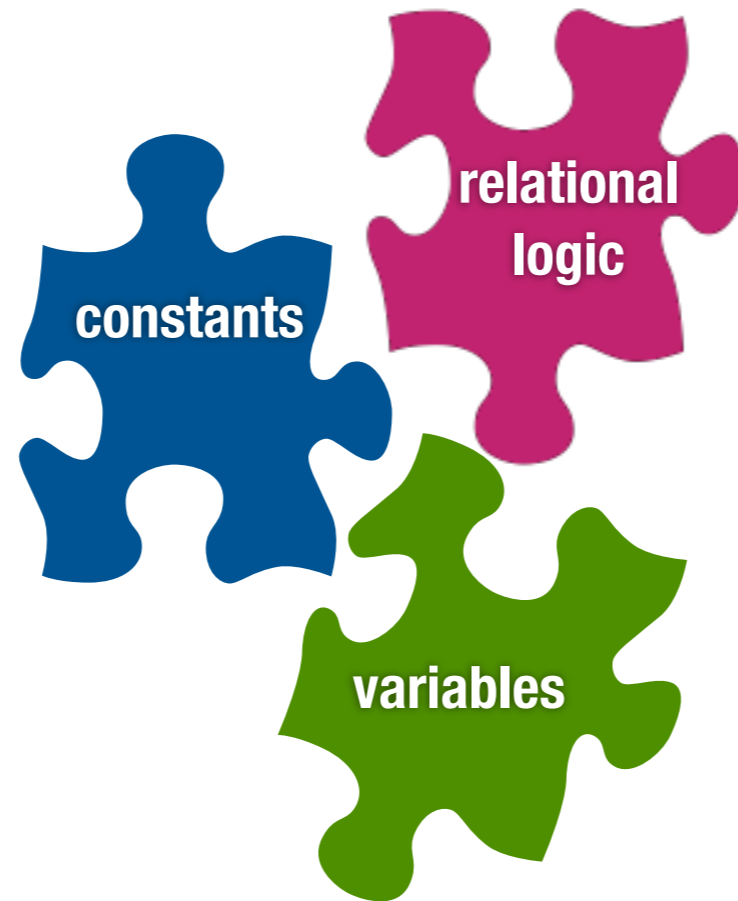


```
public class Test0 {
    static int x = 0;
    static int y = 0;

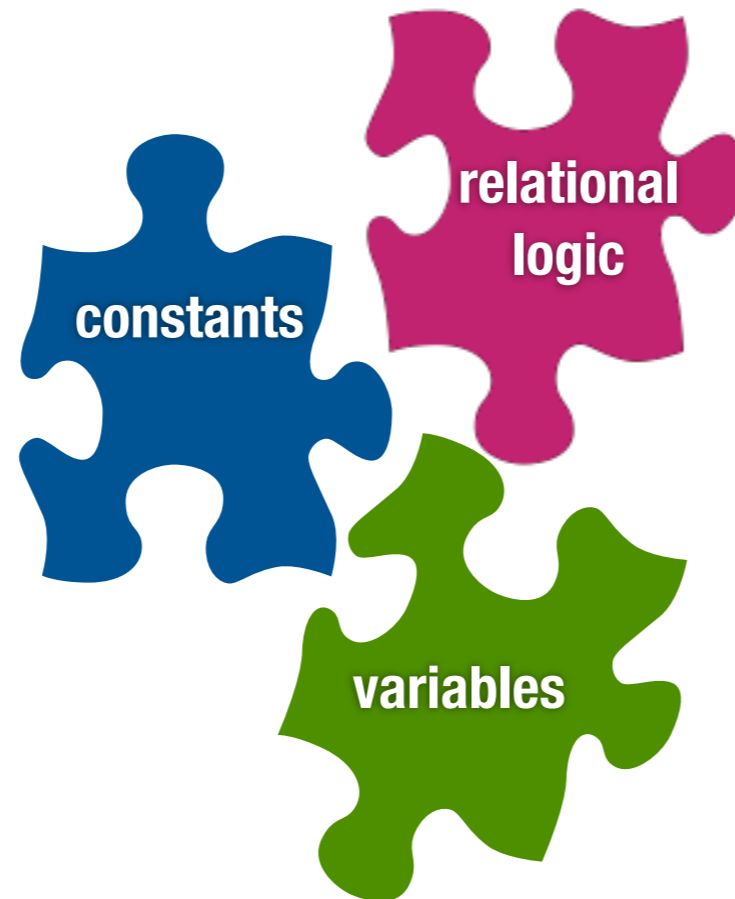
    @thread
    public static void thread1() {
        final int r1 = x;
        y = 1;
        assert r1 == 1;
    }

    @thread
    public static void thread2() {
        final int r2 = y;
        x = 1;
        assert r2 == 1;
    }
}
```

Specifying a memory model



Specifying a memory model

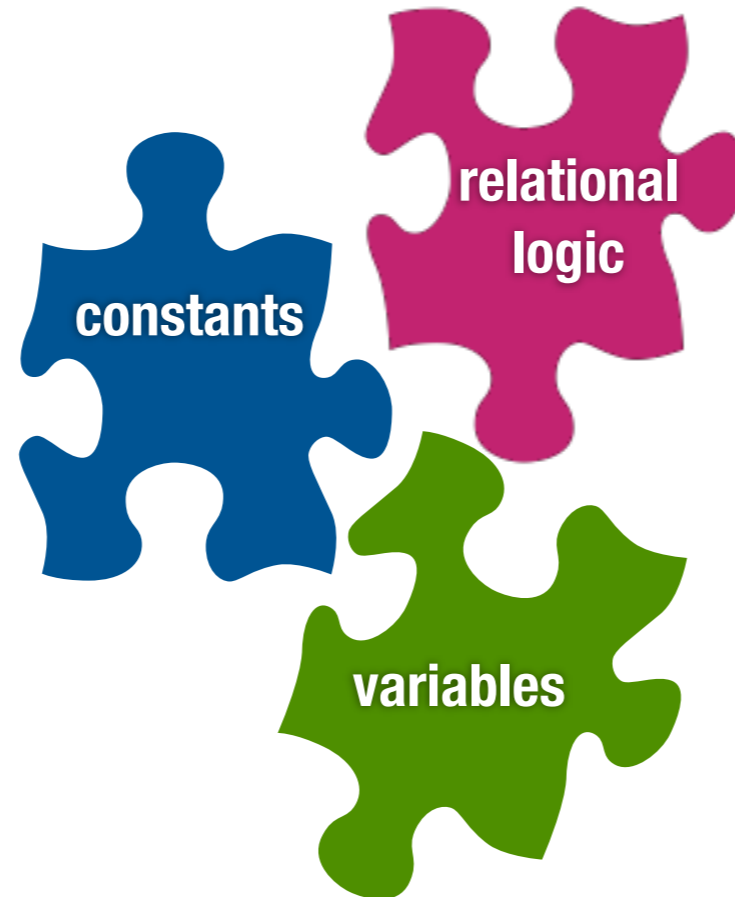


first order logic ($\forall, \exists, \wedge, \vee, \neg$)
relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)
bitvector arithmetic ($+, -, *, /$)

Specifying a memory model

relational constants capture static properties of a program

- ▶ **co**, control flow
- ▶ **to**, thread order



first order logic ($\forall, \exists, \wedge, \vee, \neg$)

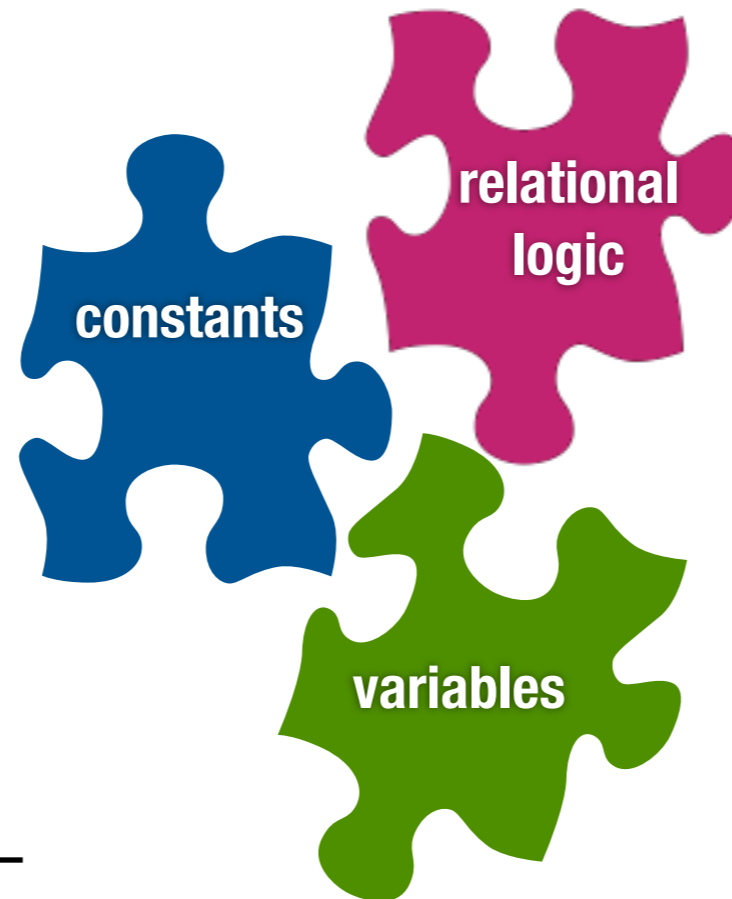
relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)

bitvector arithmetic ($+, -, *, /$)

Specifying a memory model

relational constants capture static properties of a program

- co, control flow
- to, thread order



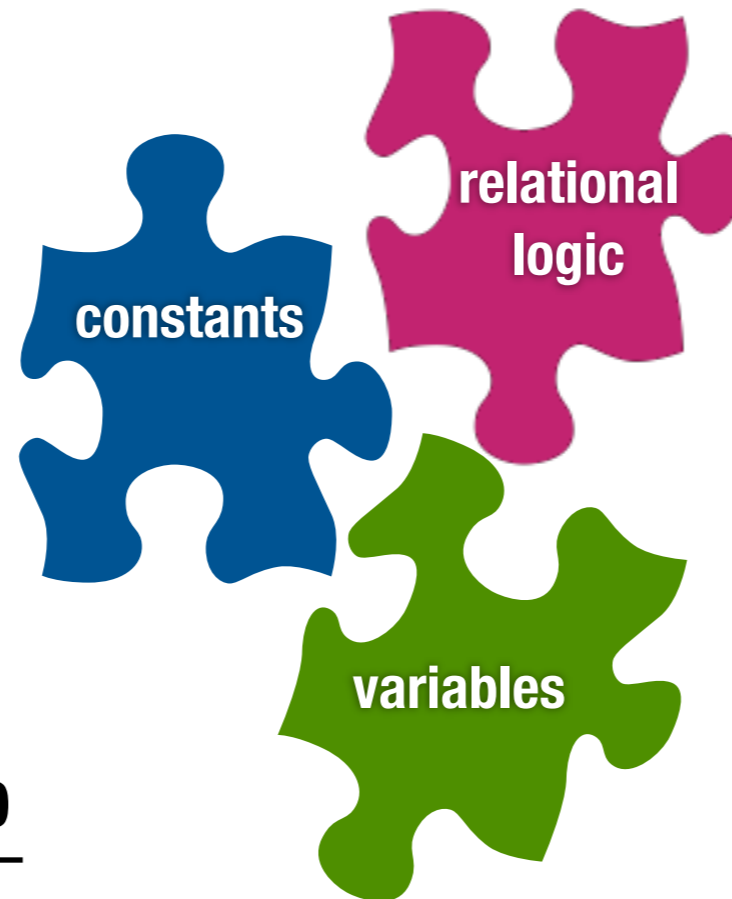
first order logic ($\forall, \exists, \wedge, \vee, \neg$)
relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)
bitvector arithmetic ($+, -, *, /$)

$x = 0, y = 0$	
$r1 = x$	$r2 = y$
$y = 1$	$x = 1$

Specifying a memory model

relational constants capture static properties of a program

- co, control flow
- to, thread order



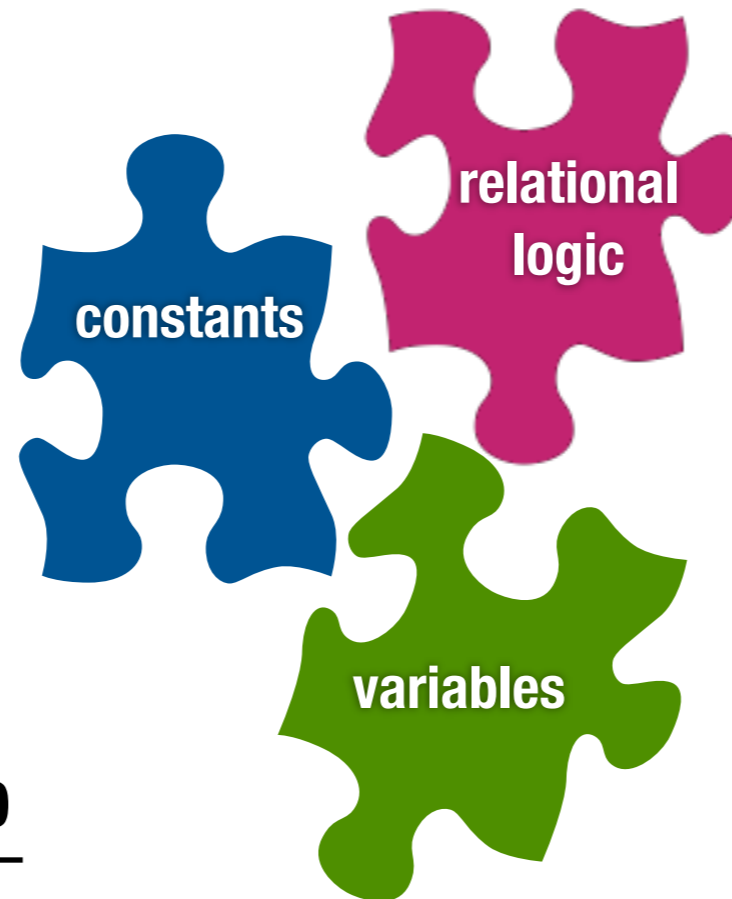
first order logic ($\forall, \exists, \wedge, \vee, \neg$)
relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)
bitvector arithmetic ($+, -, *, /$)

$x = 0, y = 0$		t0
$r1 = x$	$r2 = y$	
$y = 1$	t1	$x = 1$
		t2

Specifying a memory model

relational constants capture static properties of a program

- **co**, control flow
- **to** = { $\langle t0, t1 \rangle$, $\langle t0, t2 \rangle$ }



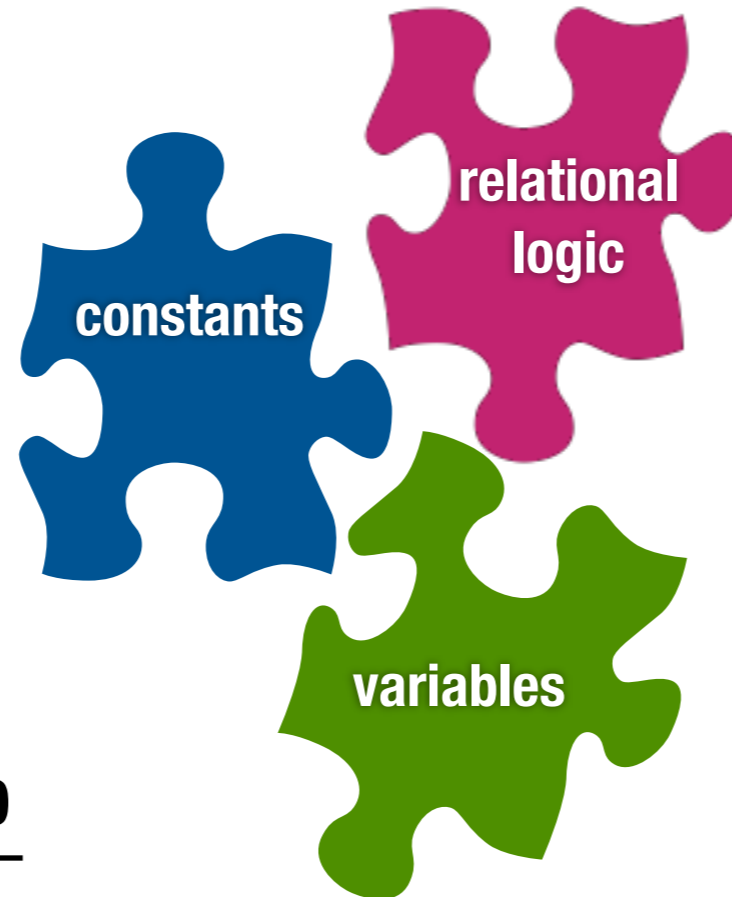
first order logic ($\forall, \exists, \wedge, \vee, \neg$)
relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)
bitvector arithmetic ($+, -, *, /$)

$x = 0, y = 0$		t0
$r1 = x$	$r2 = y$	
$y = 1$	$x = 1$	
t1		t2

Specifying a memory model

relational constants capture static properties of a program

- **co**, control flow
- **to** = { $\langle t0, t1 \rangle$, $\langle t0, t2 \rangle$ }



first order logic ($\forall, \exists, \wedge, \vee, \neg$)

relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)

bitvector arithmetic ($+, -, *, /$)

$x = 0, y = 0$		t0
$r1 = x$		$r2 = y$
$y = 1$	t1	$x = 1$
		t2

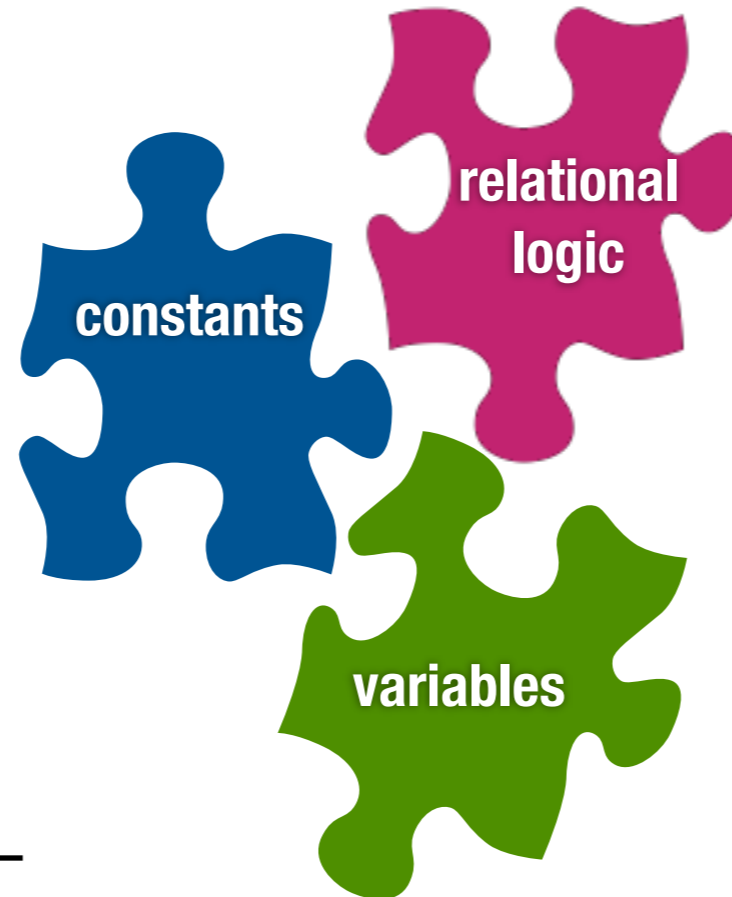
relational variables capture runtime properties of a program

- **A**, set of all executed actions
- **W**, maps reads to seen writes
- **V**, maps writes to written values
- **l**, maps reads/writes to locations
- **m**, maps locks/unlocks to monitors

Specifying a memory model

relational constants capture static properties of a program

- ▶ **co**, control flow
- ▶ **to** = { $\langle t0, t1 \rangle$, $\langle t0, t2 \rangle$ }



first order logic ($\forall, \exists, \wedge, \vee, \neg$)

relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)

bitvector arithmetic ($+, -, *, /$)

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

$r1 = x$		$r2 = y$	
$y = 1$	t1	$x = 1$	t2

relational variables capture runtime properties of a program

- ▶ **A**, set of all executed actions
- ▶ **W**, maps reads to seen writes
- ▶ **V**, maps writes to written values
- ▶ **l**, maps reads/writes to locations
- ▶ **m**, maps locks/unlocks to monitors

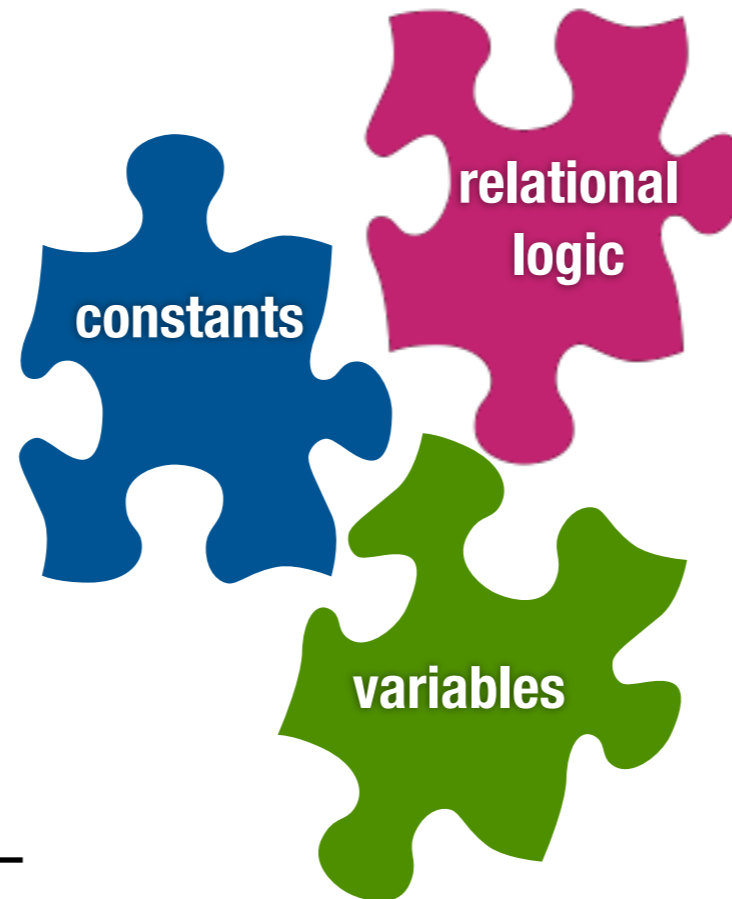
Specifying a memory model

relational constants capture static properties of a program

- ▶ **co**, control flow
- ▶ **to** = { $\langle t0, t1 \rangle, \langle t0, t2 \rangle$ }

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

a10: start		
a11: read(x, 0)	$r2 = y$	
a12: write(y, 1)	$x = 1$	
a13: end		t2



first order logic ($\forall, \exists, \wedge, \vee, \neg$)

relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)

bitvector arithmetic ($+, -, *, /$)

relational variables capture runtime properties of a program

- ▶ **A**, set of all executed actions
- ▶ **W**, maps reads to seen writes
- ▶ **V**, maps writes to written values
- ▶ **l**, maps reads/writes to locations
- ▶ **m**, maps locks/unlocks to monitors

Specifying a memory model

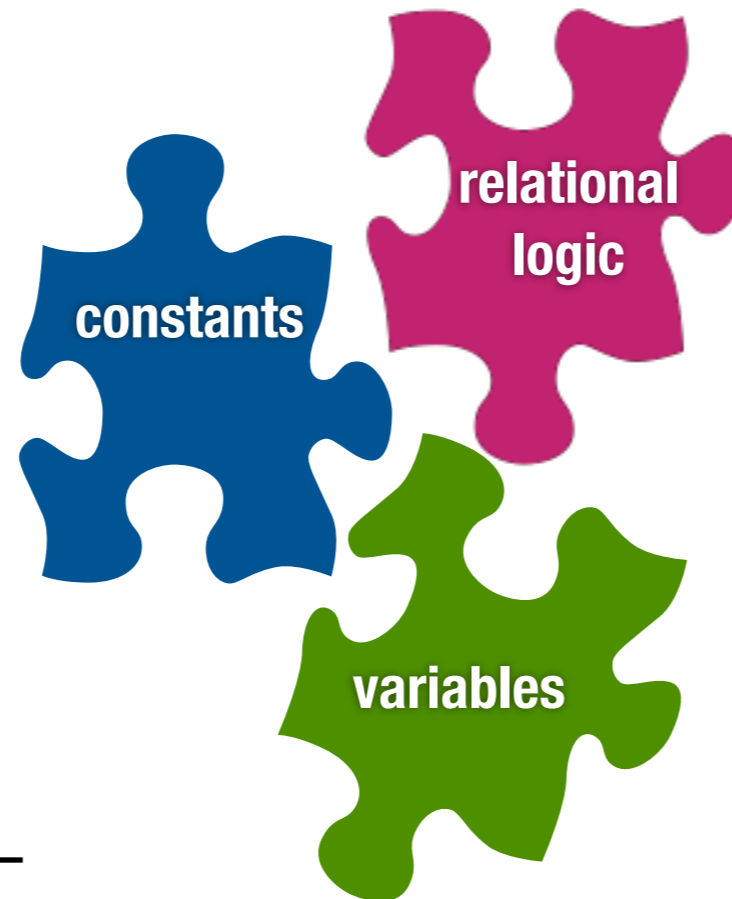
relational constants capture static properties of a program

- ▶ **co**, control flow
- ▶ **to** = { $\langle t0, t1 \rangle$, $\langle t0, t2 \rangle$ }

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

a10: start
a11: read(x, 0)
a12: write(y, 1)
a13: end

a20: start
a21: read(y, 1)
a22: write(x, 1)
a23: end



first order logic ($\forall, \exists, \wedge, \vee, \neg$)

relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)

bitvector arithmetic ($+, -, *, /$)

relational variables capture runtime properties of a program

- ▶ **A**, set of all executed actions
- ▶ **W**, maps reads to seen writes
- ▶ **V**, maps writes to written values
- ▶ **l**, maps reads/writes to locations
- ▶ **m**, maps locks/unlocks to monitors

Specifying a memory model

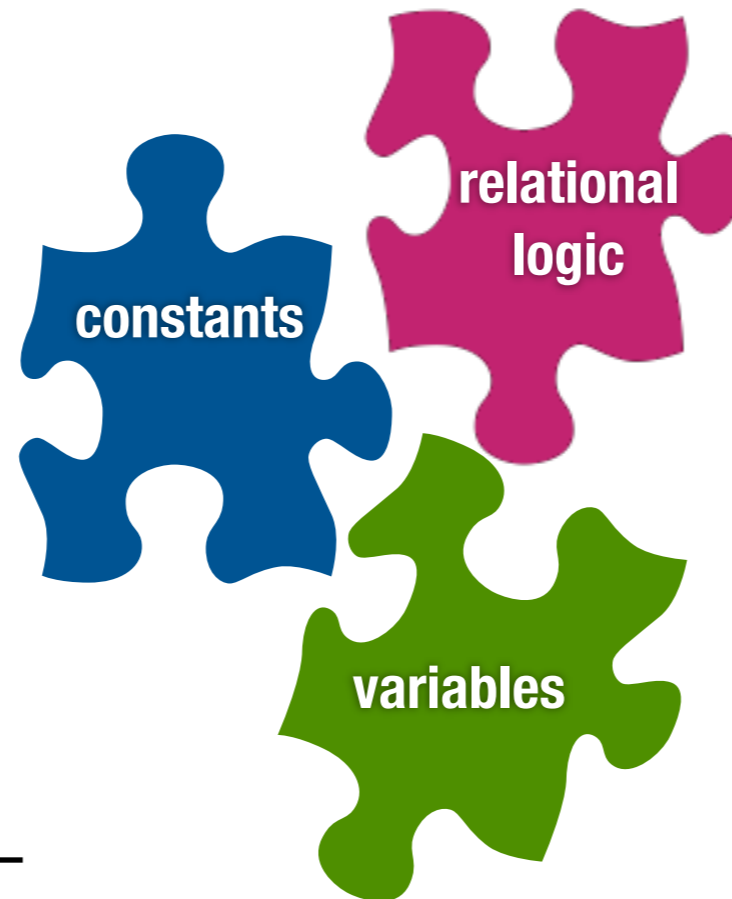
relational constants capture static properties of a program

- ▶ **co**, control flow
- ▶ **to** = { $\langle t0, t1 \rangle$, $\langle t0, t2 \rangle$ }

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

a10: start
a11: read(x, 0)
a12: write(y, 1)
a13: end

a20: start
a21: read(y, 1)
a22: write(x, 1)
a23: end



first order logic ($\forall, \exists, \wedge, \vee, \neg$)

relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)

bitvector arithmetic ($+, -, *, /$)

relational variables capture runtime properties of a program

- ▶ **A** = { $\langle a00 \rangle$, $\langle a01 \rangle$, ..., $\langle a23 \rangle$ }
- ▶ **W**, maps reads to seen writes
- ▶ **V**, maps writes to written values
- ▶ **l**, maps reads/writes to locations
- ▶ **m**, maps locks/unlocks to monitors

Specifying a memory model

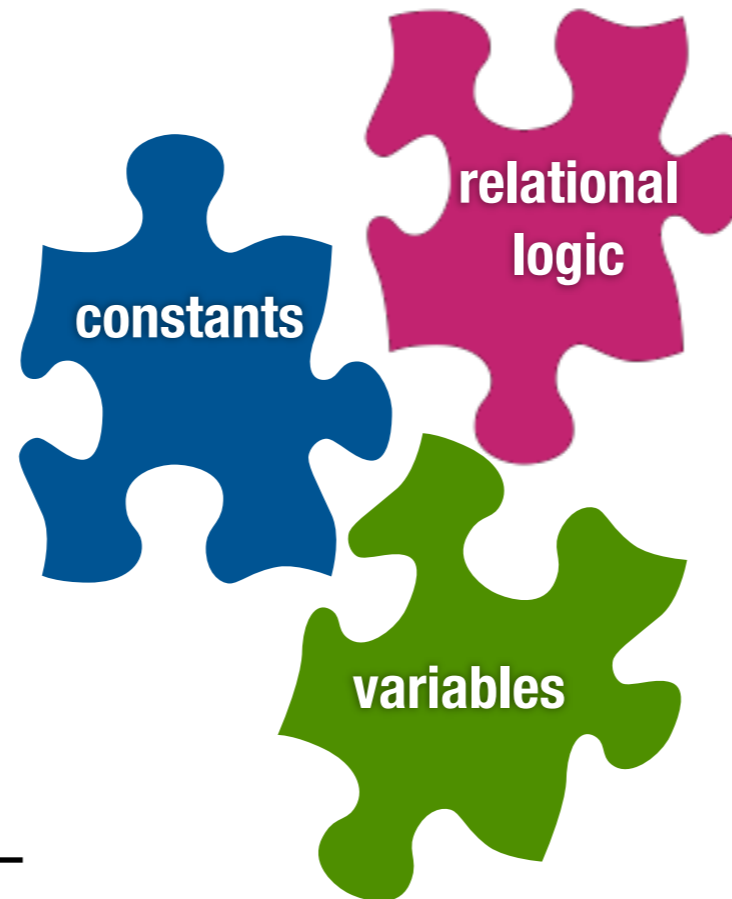
relational constants capture static properties of a program

- ▶ **co**, control flow
- ▶ **to** = { $\langle t0, t1 \rangle$, $\langle t0, t2 \rangle$ }

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

a10: start
a11: read(x, 0)
a12: write(y, 1)
a13: end

a20: start
a21: read(y, 1)
a22: write(x, 1)
a23: end



first order logic ($\forall, \exists, \wedge, \vee, \neg$)

relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)

bitvector arithmetic ($+, -, *, /$)

relational variables capture runtime properties of a program

- ▶ **A** = { $\langle a00 \rangle$, $\langle a01 \rangle$, ..., $\langle a23 \rangle$ }
- ▶ **W** = { $\langle a11, a01 \rangle$, $\langle a21, a12 \rangle$ }
- ▶ **V**, maps writes to written values
- ▶ **l**, maps reads/writes to locations
- ▶ **m**, maps locks/unlocks to monitors

Specifying a memory model

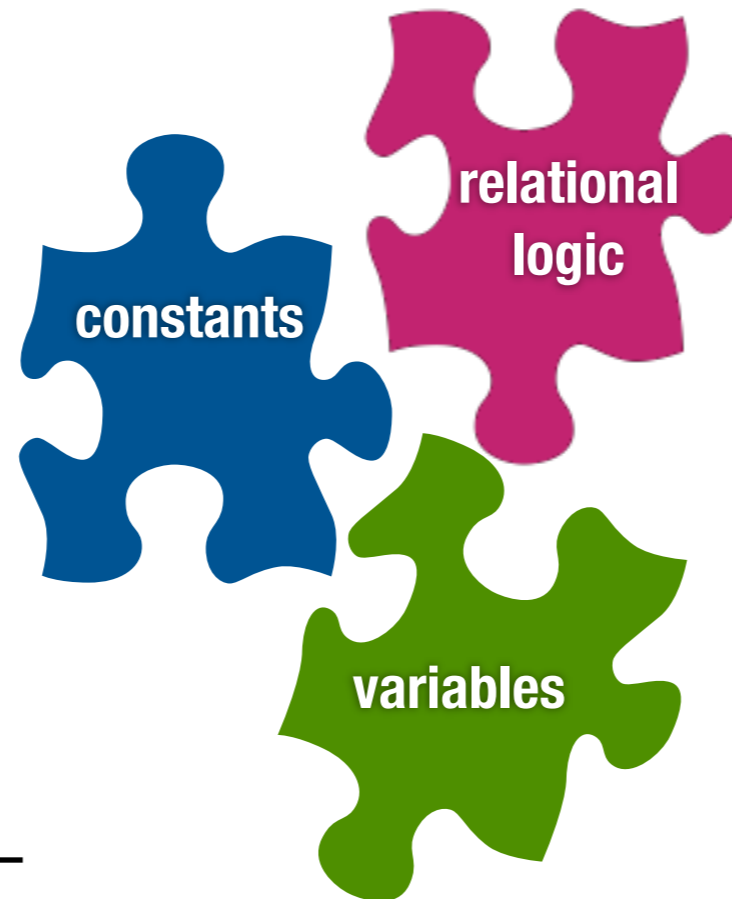
relational constants capture static properties of a program

- ▶ **co**, control flow
- ▶ **to** = { $\langle t0, t1 \rangle$, $\langle t0, t2 \rangle$ }

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

a10: start
a11: read(x, 0)
a12: write(y, 1)
a13: end

a20: start
a21: read(y, 1)
a22: write(x, 1)
a23: end



first order logic ($\forall, \exists, \wedge, \vee, \neg$)

relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)

bitvector arithmetic ($+, -, *, /$)

relational variables capture runtime properties of a program

- ▶ **A** = { $\langle a00 \rangle$, $\langle a01 \rangle$, ..., $\langle a23 \rangle$ }
- ▶ **W** = { $\langle a11, a01 \rangle$, $\langle a21, a12 \rangle$ }
- ▶ **V** = { $\langle a01, 0 \rangle$, $\langle a02, 0 \rangle$, $\langle a12, 1 \rangle$, $\langle a22, 1 \rangle$ }
- ▶ **l**, maps reads/writes to locations
- ▶ **m**, maps locks/unlocks to monitors

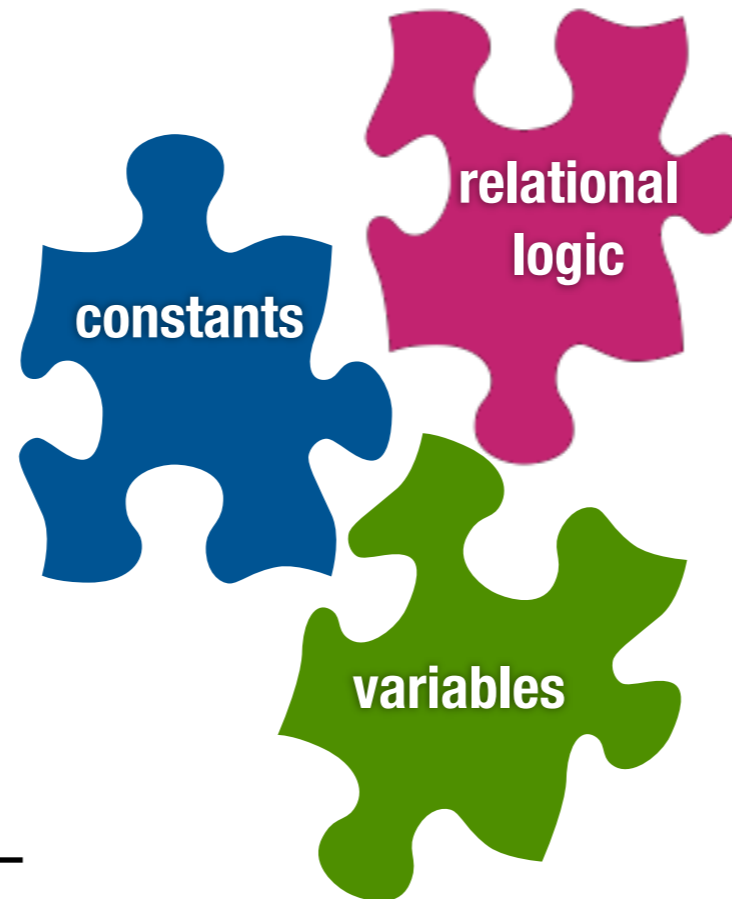
Specifying a memory model

relational constants capture static properties of a program

- ▶ **co**, control flow
- ▶ **to** = { $\langle t0, t1 \rangle$, $\langle t0, t2 \rangle$ }

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

a10: start a11: read(x, 0) a12: write(y, 1) a13: end	a20: start a21: read(y, 1) a22: write(x, 1) a23: end
---	---



first order logic ($\forall, \exists, \wedge, \vee, \neg$)

relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)

bitvector arithmetic ($+, -, *, /$)

relational variables capture runtime properties of a program

- ▶ **A** = { $\langle a00 \rangle, \langle a01 \rangle, \dots, \langle a23 \rangle$ }
- ▶ **W** = { $\langle a11, a01 \rangle, \langle a21, a12 \rangle$ }
- ▶ **V** = { $\langle a01, 0 \rangle, \langle a02, 0 \rangle, \langle a12, 1 \rangle, \langle a22, 1 \rangle$ }
- ▶ **I** = { $\langle a01, x \rangle, \langle a02, y \rangle, \dots, \langle a22, x \rangle$ }
- ▶ **m**, maps locks/unlocks to monitors

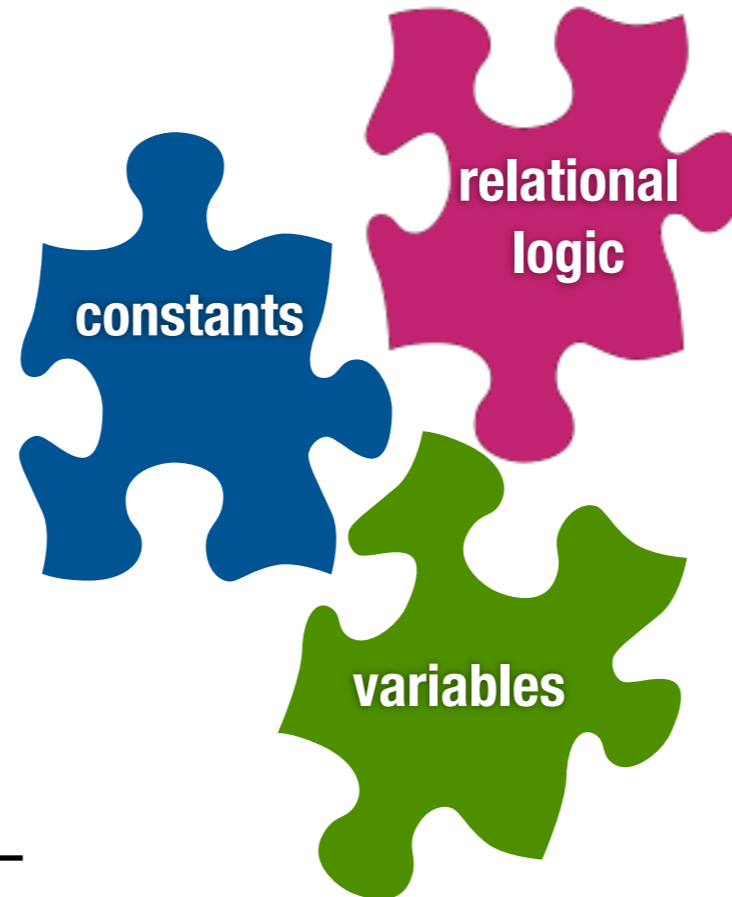
Specifying a memory model

relational constants capture static properties of a program

- ▶ **co**, control flow
- ▶ **to** = { $\langle t0, t1 \rangle$, $\langle t0, t2 \rangle$ }

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

a10: start a11: read(x, 0) a12: write(y, 1) a13: end	a20: start a21: read(y, 1) a22: write(x, 1) a23: end
---	---



first order logic ($\forall, \exists, \wedge, \vee, \neg$)

relational algebra ($\cdot, \cup, \cap, /, \times, \subseteq$)

bitvector arithmetic ($+, -, *, /$)

relational variables capture runtime properties of a program

- ▶ **A** = { $\langle a00 \rangle, \langle a01 \rangle, \dots, \langle a23 \rangle$ }
- ▶ **W** = { $\langle a11, a01 \rangle, \langle a21, a12 \rangle$ }
- ▶ **V** = { $\langle a01, 0 \rangle, \langle a02, 0 \rangle, \langle a12, 1 \rangle, \langle a22, 1 \rangle$ }
- ▶ **I** = { $\langle a01, x \rangle, \langle a02, y \rangle, \dots, \langle a22, x \rangle$ }
- ▶ **m** = { }

Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. **Execution order is total,**
2. **antisymmetric, and**
3. **transitive.**
4. **It respects the control flow and**
5. **thread order.**
6. **Reads cannot see out of order writes.**
7. **No write interferes between a read and the write seen by that read.**

Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$
2. **antisymmetric, and**
3. **transitive.**
4. **It respects the control flow and**
5. **thread order.**
6. **Reads cannot see out of order writes.**
7. **No write interferes between a read and the write seen by that read.**

Execution order is total,

Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. **transitive.**
4. **It respects the control flow and**
5. **thread order.**
6. **Reads cannot see out of order writes.**
7. **No write interferes between a read and the write seen by that read.**

**Execution order is total,
antisymmetric, and**

Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. **It respects the control flow and**
5. **thread order.**
6. **Reads cannot see out of order writes.**
7. **No write interferes between a read and the write seen by that read.**

Execution order is total, antisymmetric, and transitive.

Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. **thread order.**
6. **Reads cannot see out of order writes.**
7. **No write interferes between a read and the write seen by that read.**

Execution order is total, antisymmetric, and transitive.

It respects the control flow and

Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \wedge \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. Reads cannot see out of order writes.
7. No write interferes between a read and the write seen by that read.

Execution order is total, antisymmetric, and transitive.

It respects the control flow and thread order.

Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \wedge \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$
7. **No write interferes between a read and the write seen by that read.**

Execution order is total, antisymmetric, and transitive.

It respects the control flow and thread order.

Reads cannot see out of order writes.

Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \wedge \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$
7. $\forall k: A \cap \text{Read}, j: A \cap \text{Write} \mid$
 $\neg (l[k] = l[j] \wedge \text{ord}[W[k], j] \wedge \text{ord}[j, k])$

Execution order is total,
antisymmetric, and
transitive.

It respects the control flow and
thread order.

Reads cannot see out of order writes.

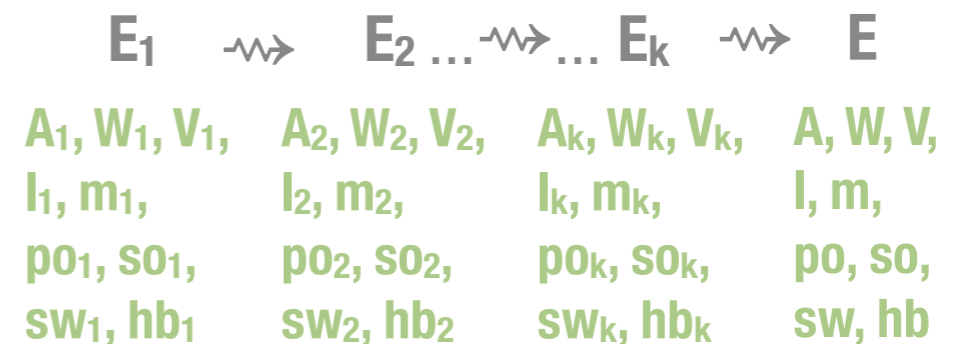
No write interferes between a read and the
write seen by that read.

Example: Java memory model

committing semantics

an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. $\forall i: [1..k] \mid C_i \subseteq A_i$
2. $\forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hb}_i[W[r], r]) \wedge \neg \text{hb}_i[r, W[r]]$
3. $\forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V$
4. $\forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W$
5. $\forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_i[W_i[r], r]$
6. $\forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1}$
7. $\forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \wedge \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1}$



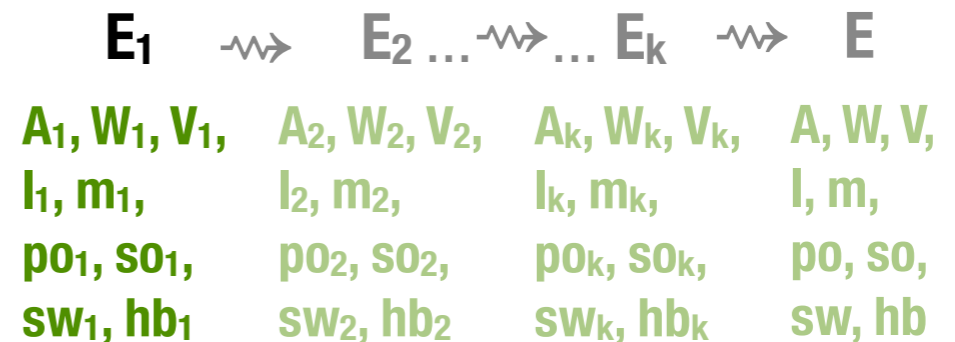
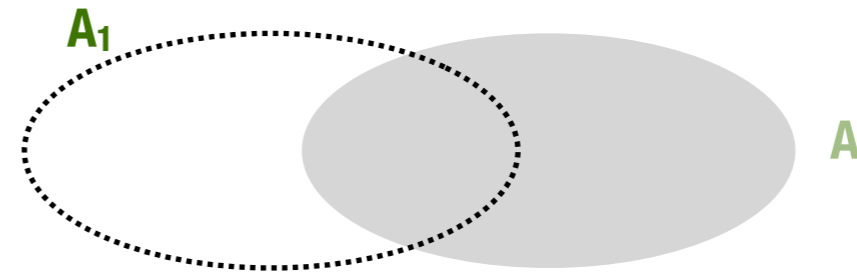
Example: Java memory model

committing semantics

an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. $\forall i: [1..k] \mid C_i \subseteq A_i$
2. $\forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hb}_i[W[r], r]) \wedge \neg \text{hb}_i[r, W[r]]$
3. $\forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V$
4. $\forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W$
5. $\forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_i[W_i[r], r]$
6. $\forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1}$
7. $\forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \wedge \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1}$

initial execution: reads can only see writes that happen-before them



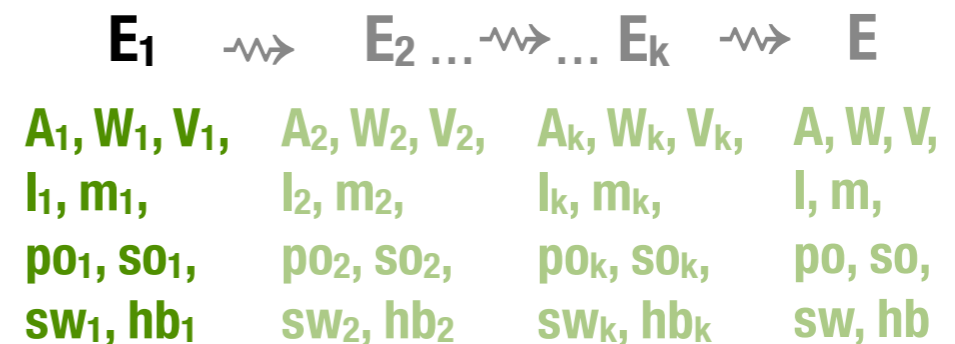
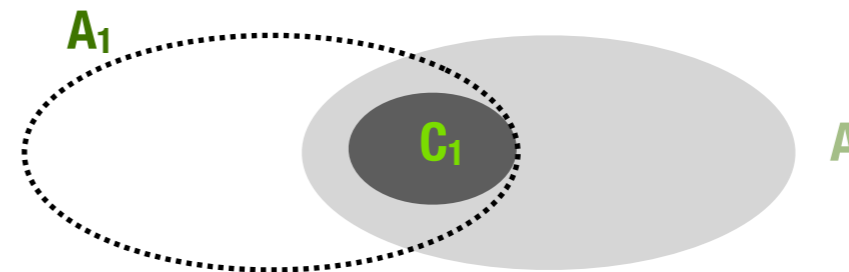
Example: Java memory model

committing semantics

an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. $\forall i: [1..k] \mid C_i \subseteq A_i$
2. $\forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hb}_i[W[r], r]) \wedge \neg \text{hb}_i[r, W[r]]$
3. $\forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V$
4. $\forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W$
5. $\forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_i[W_i[r], r]$
6. $\forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1}$
7. $\forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \wedge \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1}$

initial execution: reads can only see writes that happen-before them



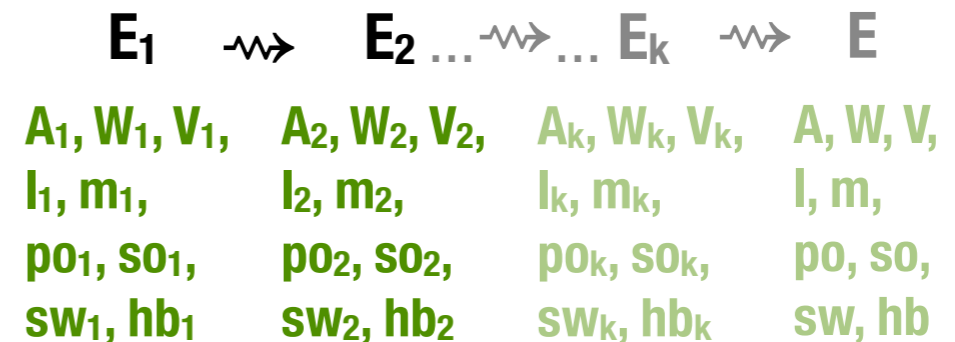
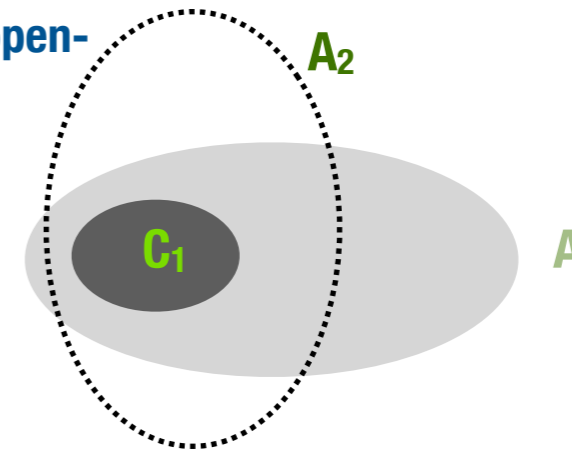
Example: Java memory model

committing semantics

an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. $\forall i: [1..k] \mid C_i \subseteq A_i$
2. $\forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hb}_i[W[r], r]) \wedge \neg \text{hb}_i[r, W[r]]$
3. $\forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V$
4. $\forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W$
5. $\forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_i[W_i[r], r]$
6. $\forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1}$
7. $\forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \wedge \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1}$

i^{th} execution: committed reads can see committed writes; other reads must see writes that happen-before them



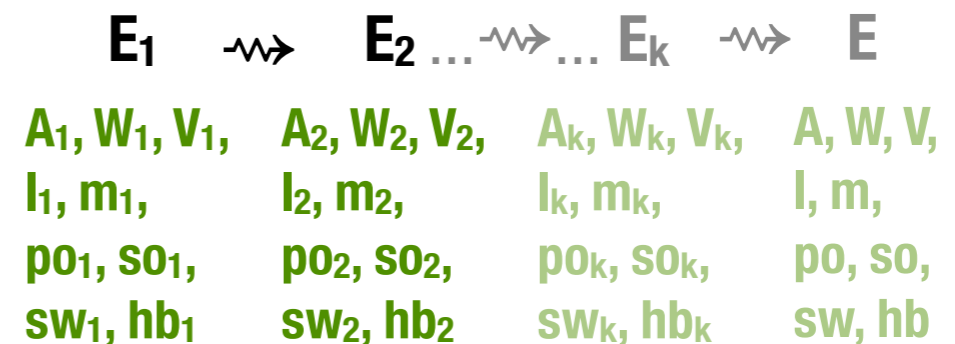
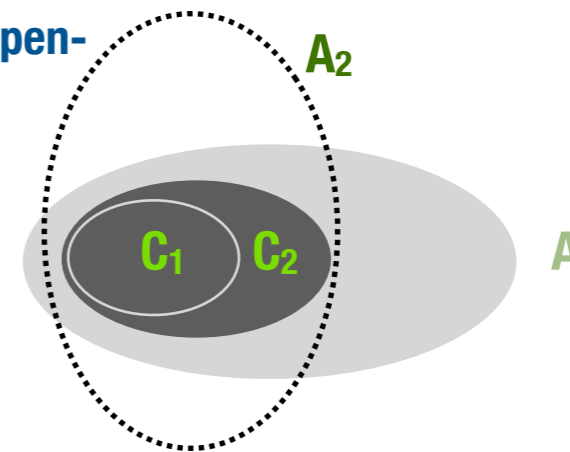
Example: Java memory model

committing semantics

an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. $\forall i: [1..k] \mid C_i \subseteq A_i$
2. $\forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hb}_i[W[r], r]) \wedge \neg \text{hb}_i[r, W[r]]$
3. $\forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V$
4. $\forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W$
5. $\forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_i[W_i[r], r]$
6. $\forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1}$
7. $\forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \wedge \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1}$

i^{th} execution: committed reads can see committed writes; other reads must see writes that happen-before them



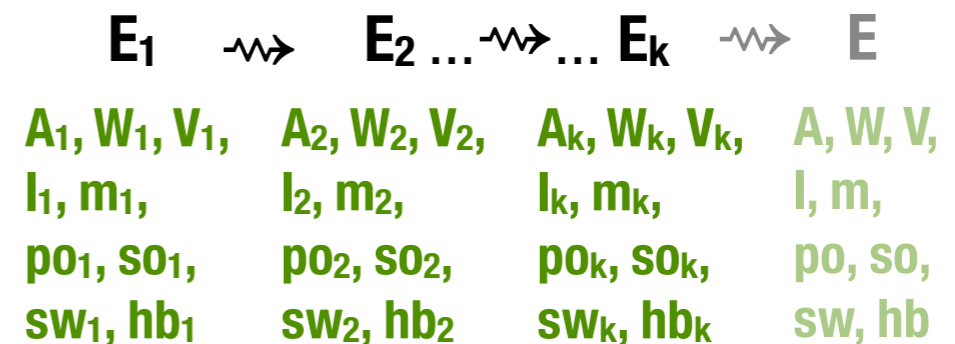
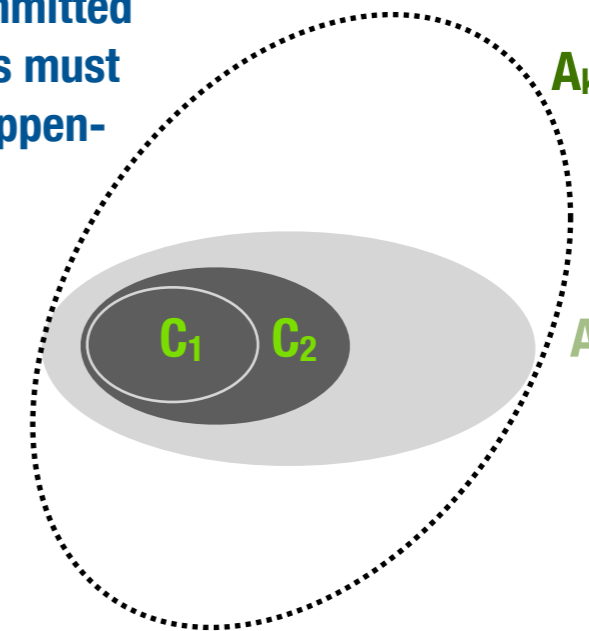
Example: Java memory model

committing semantics

an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. $\forall i: [1..k] \mid C_i \subseteq A_i$
2. $\forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hb}_i[W[r], r]) \wedge \neg \text{hb}_i[r, W[r]]$
3. $\forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V$
4. $\forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W$
5. $\forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_i[W_i[r], r]$
6. $\forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1}$
7. $\forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \wedge \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1}$

i^{th} execution: committed reads can see committed writes; other reads must see writes that happen-before them



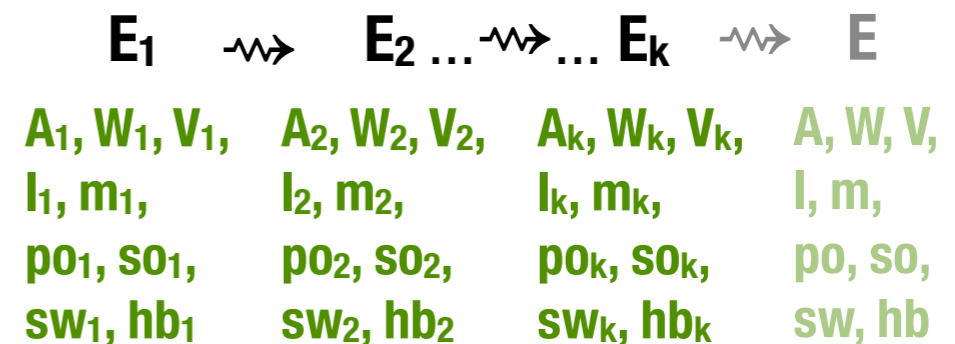
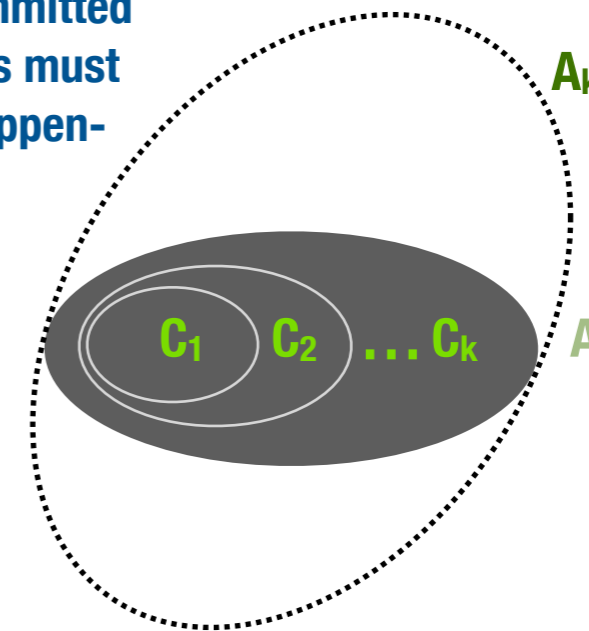
Example: Java memory model

committing semantics

an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. $\forall i: [1..k] \mid C_i \subseteq A_i$
2. $\forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hb}_i[W[r], r]) \wedge \neg \text{hb}_i[r, W[r]]$
3. $\forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V$
4. $\forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W$
5. $\forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_i[W_i[r], r]$
6. $\forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1}$
7. $\forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \wedge \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1}$

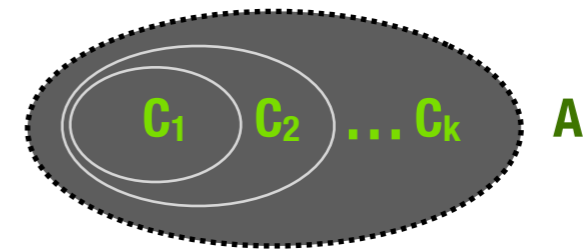
i^{th} execution: committed reads can see committed writes; other reads must see writes that happen-before them



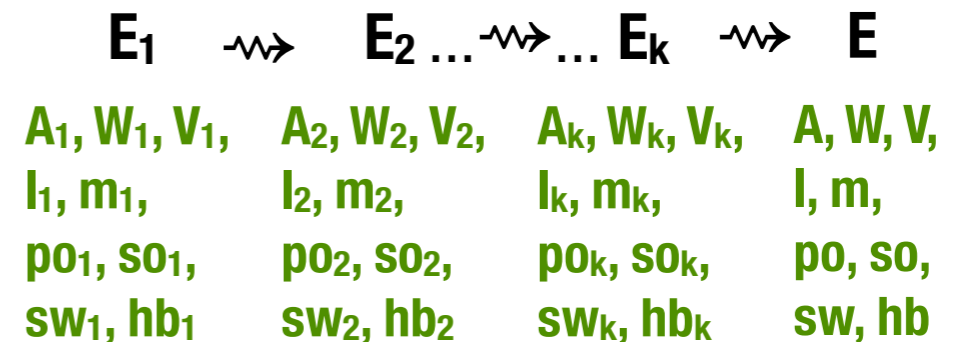
Example: Java memory model

committing semantics

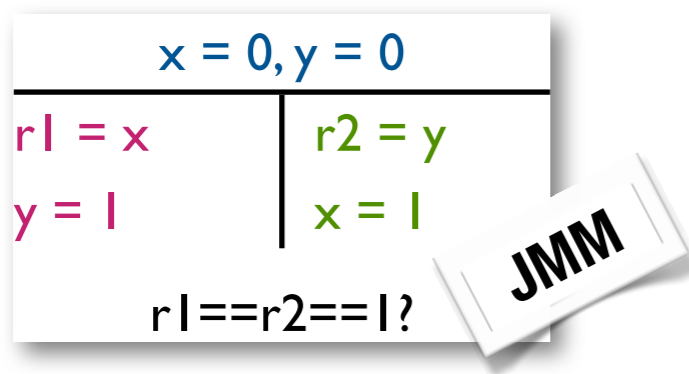
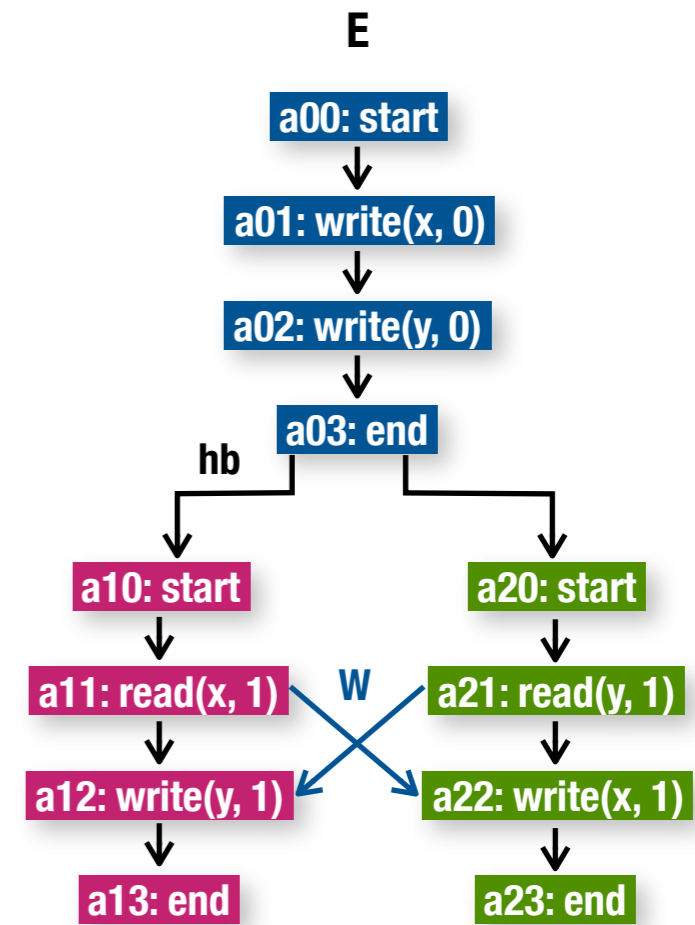
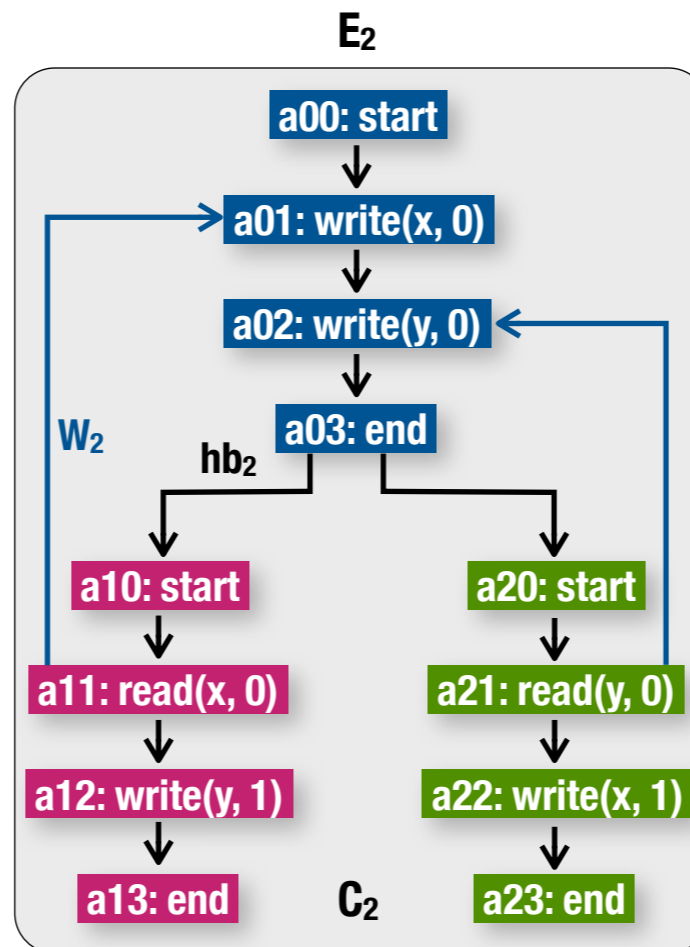
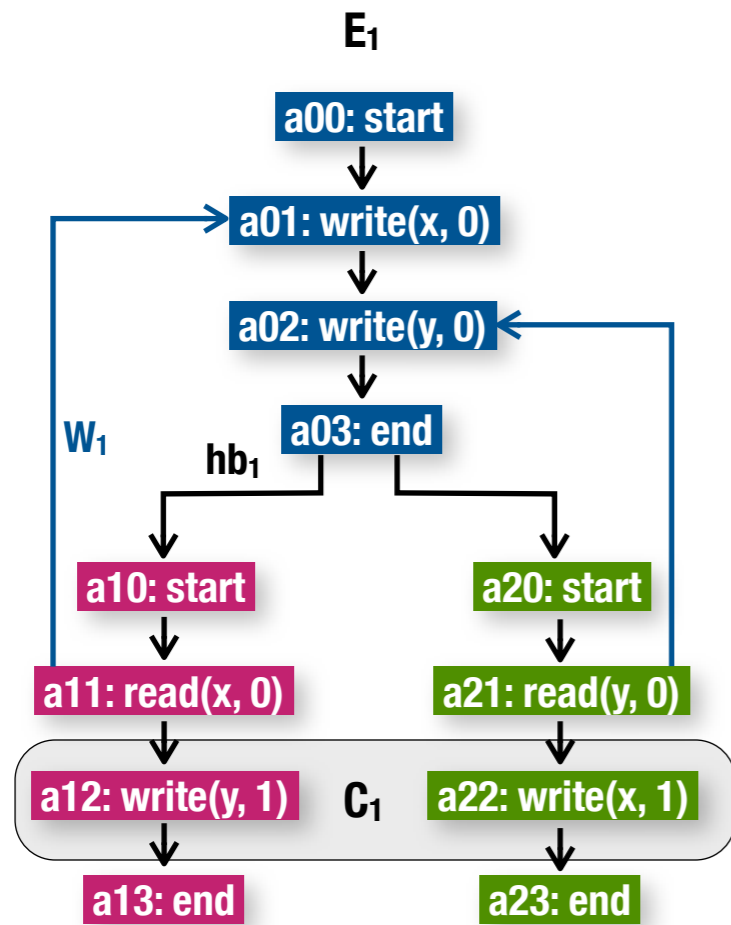
an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions



1. $\forall i: [1..k] \mid C_i \subseteq A_i$
2. $\forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hb}_i[W[r], r]) \wedge \neg \text{hb}_i[r, W[r]]$
3. $\forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V$
4. $\forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W$
5. $\forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_i[W_i[r], r]$
6. $\forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1}$
7. $\forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \wedge \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1}$



Witness of legality (model)



witness: an execution of the program that satisfies both the assertions and the memory model constraints.

Proof of illegality (minimal core)

$x = 0, y = 0$	
$r1 = x$	$r2 = y$
$y = 1$	$x = 1$
$r1 == 1 \ \&\& \ r2 == 1?$	

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$ **SC**
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \wedge \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$
7. $\forall k: A \cap \text{Read}, j: A \cap \text{Write} \mid$
 $\neg (l[k] = l[j] \wedge \text{ord}[W[k], j] \wedge \text{ord}[j, k])$

$$V[a_{01}] = 0$$

$$V[a_{02}] = 0$$

$$V[W[a_{11}]] = 1$$

$$V[W[a_{21}]] = 1$$

$$\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$$

$$\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$$

$$\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$$

$$\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$$

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed

Proof of illegality (minimal core)

$x = 0, y = 0$	
$r1 = x$	$r2 = y$
$y = 1$	$x = 1$
$r1 == 1 \ \&\& \ r2 == 1?$	

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$ **SC**
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \wedge \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$
7. $\forall k: A \cap \text{Read}, j: A \cap \text{Write} \mid$
 $\neg (l[k] = l[j] \wedge \text{ord}[W[k], j] \wedge \text{ord}[j, k])$

$$V[a_{01}] = 0$$

$$V[a_{02}] = 0$$

a_{ij} represents the action (if any) performed by the j^{th} statement of the i^{th} thread

$$V[W[a_{11}]] = 1$$

$$V[W[a_{21}]] = 1$$

$$\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$$

$$\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$$

$$\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$$

$$\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$$

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed

Proof of illegality (minimal core)

$x = 0, y = 0$	
$r1 = x$	$r2 = y$
$y = 1$	$x = 1$
$r1 == 1 \ \&\& \ r2 == 1?$	

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$ **SC**
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \wedge \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$
7. $\forall k: A \cap \text{Read}, j: A \cap \text{Write} \mid$
 $\neg (l[k] = l[j] \wedge \text{ord}[W[k], j] \wedge \text{ord}[j, k])$

$$V[a_{01}] = 0$$

$$V[a_{02}] = 0$$

$$V[W[a_{11}]] = 1$$

$$V[W[a_{21}]] = 1$$

$$\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$$

$$\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$$

$$\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$$

$$\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$$

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed

Proof of illegality (minimal core)

$x = 0, y = 0$	
$r1 = x$	$r2 = y$
$y = 1$	$x = 1$
$r1 == 1 \ \&\& \ r2 == 1?$	

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$ **SC**
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \wedge \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$
7. $\forall k: A \cap \text{Read}, j: A \cap \text{Write} \mid$
 $\neg (l[k] = l[j] \wedge \text{ord}[W[k], j] \wedge \text{ord}[j, k])$

$$V[a_{01}] = 0$$

$$V[a_{02}] = 0$$

$$V[W[a_{11}]] = 1$$

$$V[W[a_{21}]] = 1$$

$$\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$$

$$\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$$

$$\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$$

$$\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$$

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed

Proof of illegality (minimal core)

$x = 1, y = 0$	
$r1 = x$	$r2 = y$
$y = 1$	$x = 1$
$r1 == 1 \ \&\& \ r2 == 1?$	

a01: write(x, 1)
a02: write(y, 0)
a11: read(x, 1)
a12: write(y, 1)
a21: read(y, 1)
a22: write(x, 1)

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$ **SC**
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \wedge \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$
7. $\forall k: A \cap \text{Read}, j: A \cap \text{Write} \mid$
 $\neg (l[k] = l[j] \wedge \text{ord}[W[k], j] \wedge \text{ord}[j, k])$



$V[a_{01}] = 0$
 $V[a_{02}] = 0$

$V[W[a_{11}]] = 1$
 $V[W[a_{21}]] = 1$

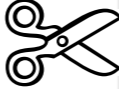
- $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$
- $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
- $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
- $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed

Proof of illegality (minimal core)

$x = 0, y = 0$		$a01: \text{write}(x, 0)$ $a02: \text{write}(y, 0)$ $a11: \text{read}(x, 0)$ $a12: \text{write}(y, 1)$ $a21: \text{read}(y, 1)$ $a22: \text{write}(x, 1)$
$r1 = x$	$r2 = y$	
$y = 1$	$x = 1$	
$r2 == 1?$		

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$ **SC**
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \wedge \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$
7. $\forall k: A \cap \text{Read}, j: A \cap \text{Write} \mid$
 $\neg (l[k] = l[j] \wedge \text{ord}[W[k], j] \wedge \text{ord}[j, k])$

- $V[a_{01}] = 0$
 $V[a_{02}] = 0$
 $V[W[a_{11}]] = 1$
 $V[W[a_{21}]] = 1$
- $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$
 $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
 $\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
 $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed

Proof of illegality (minimal core)

$x = 0, y = 0$	
$r1 = x$	$r2 = y$
$y = 1$	$x = 1$
$r1 == 1 \ \&\& \ r2 == 1?$	

a01: write(x, 0)
a02: write(y, 0)
a12: write(y, 1)
a21: read(y, 1)
a22: write(x, 1)
a11: read(x, 1)

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$ **SC**
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \wedge \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$
7. $\forall k: A \cap \text{Read}, j: A \cap \text{Write} \mid$
 $\neg (l[k] = l[j] \wedge \text{ord}[W[k], j] \wedge \text{ord}[j, k])$

$V[a_{01}] = 0$

$V[a_{02}] = 0$

$V[W[a_{11}]] = 1$

$V[W[a_{21}]] = 1$

$\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \vee \text{ord}[j, i]$

$\forall i, j, k: A \mid (\text{ord}[i, j] \wedge \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$

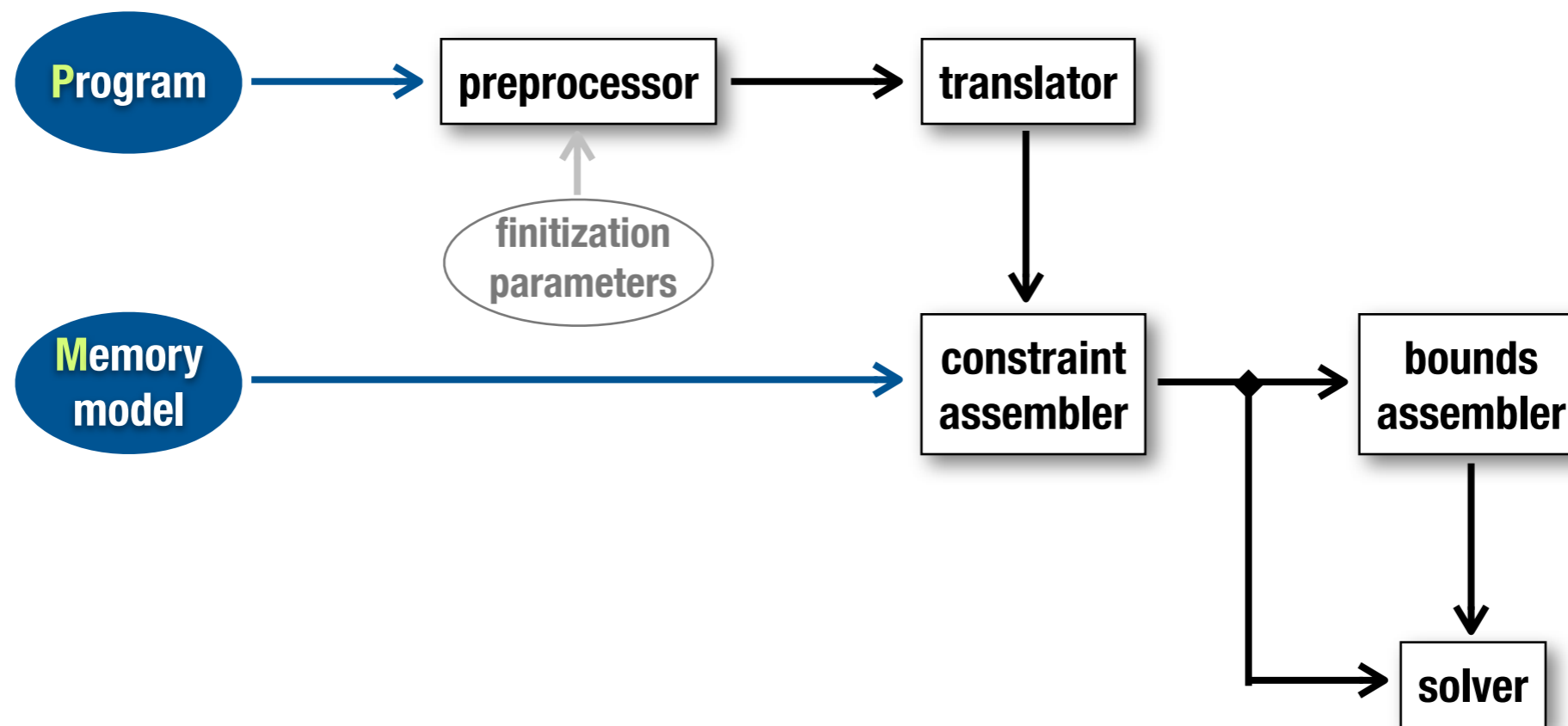


$\forall i, j: A \mid (t[i] = t[j] \wedge \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$

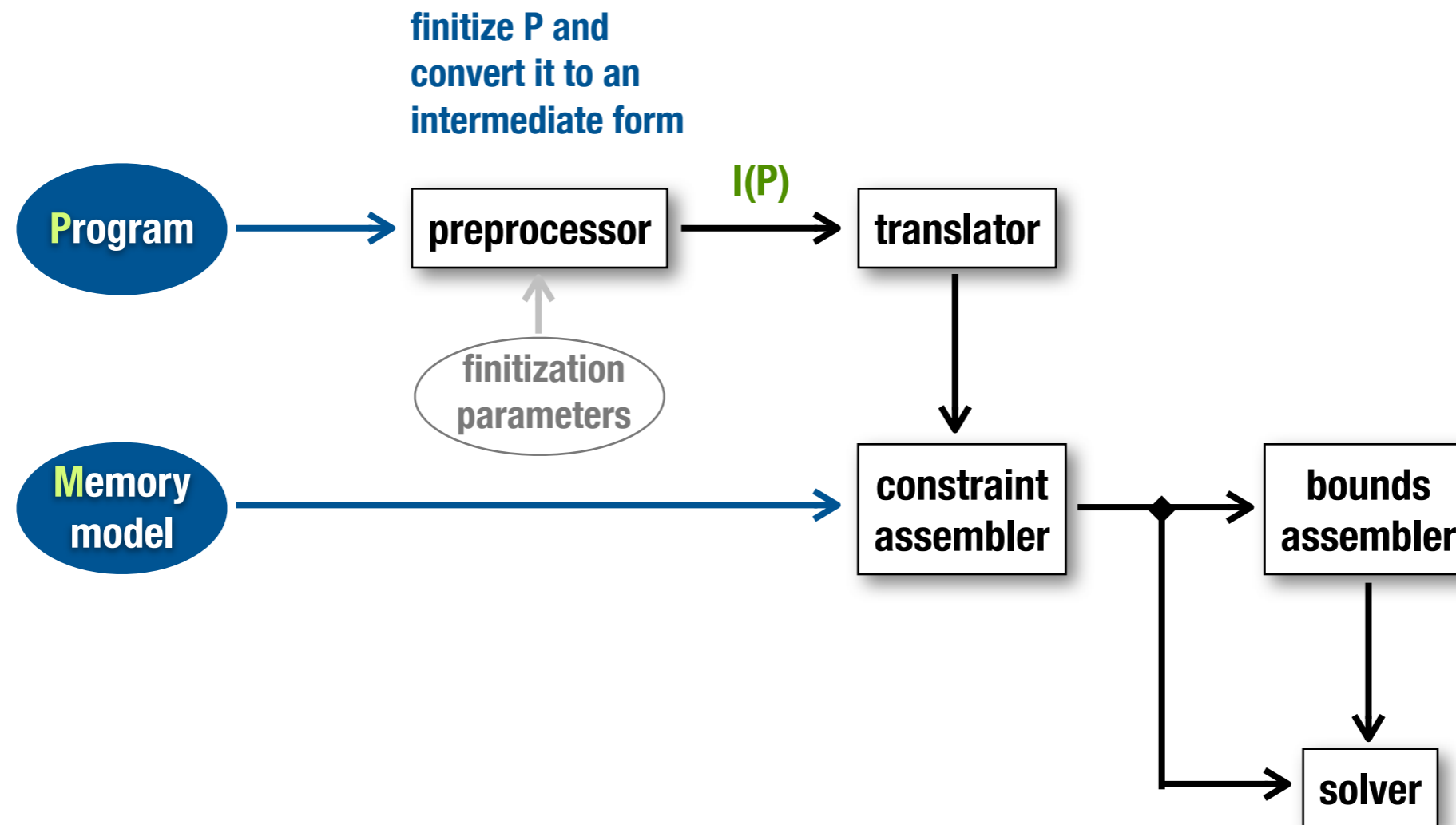
$\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed

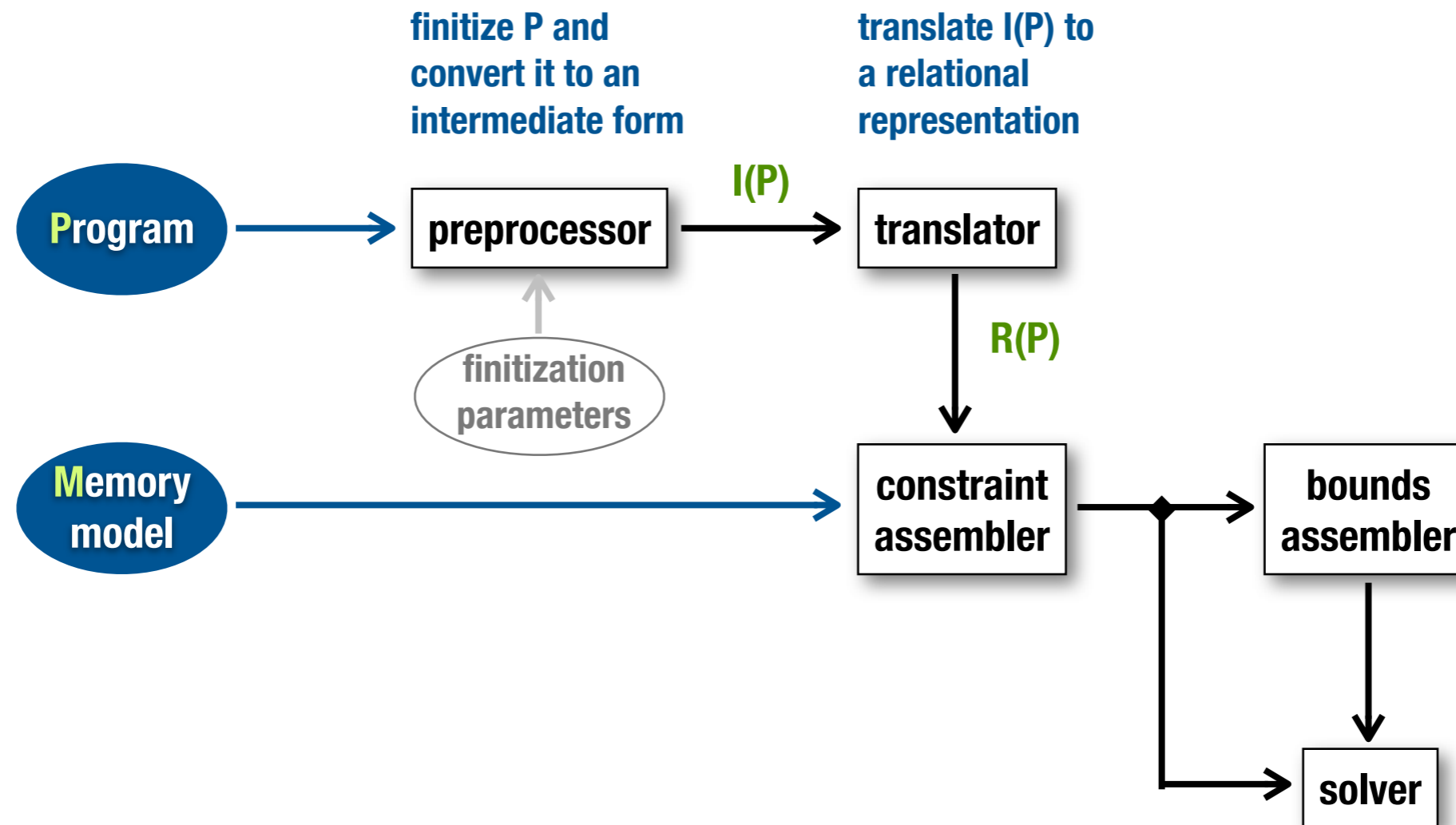
Approach



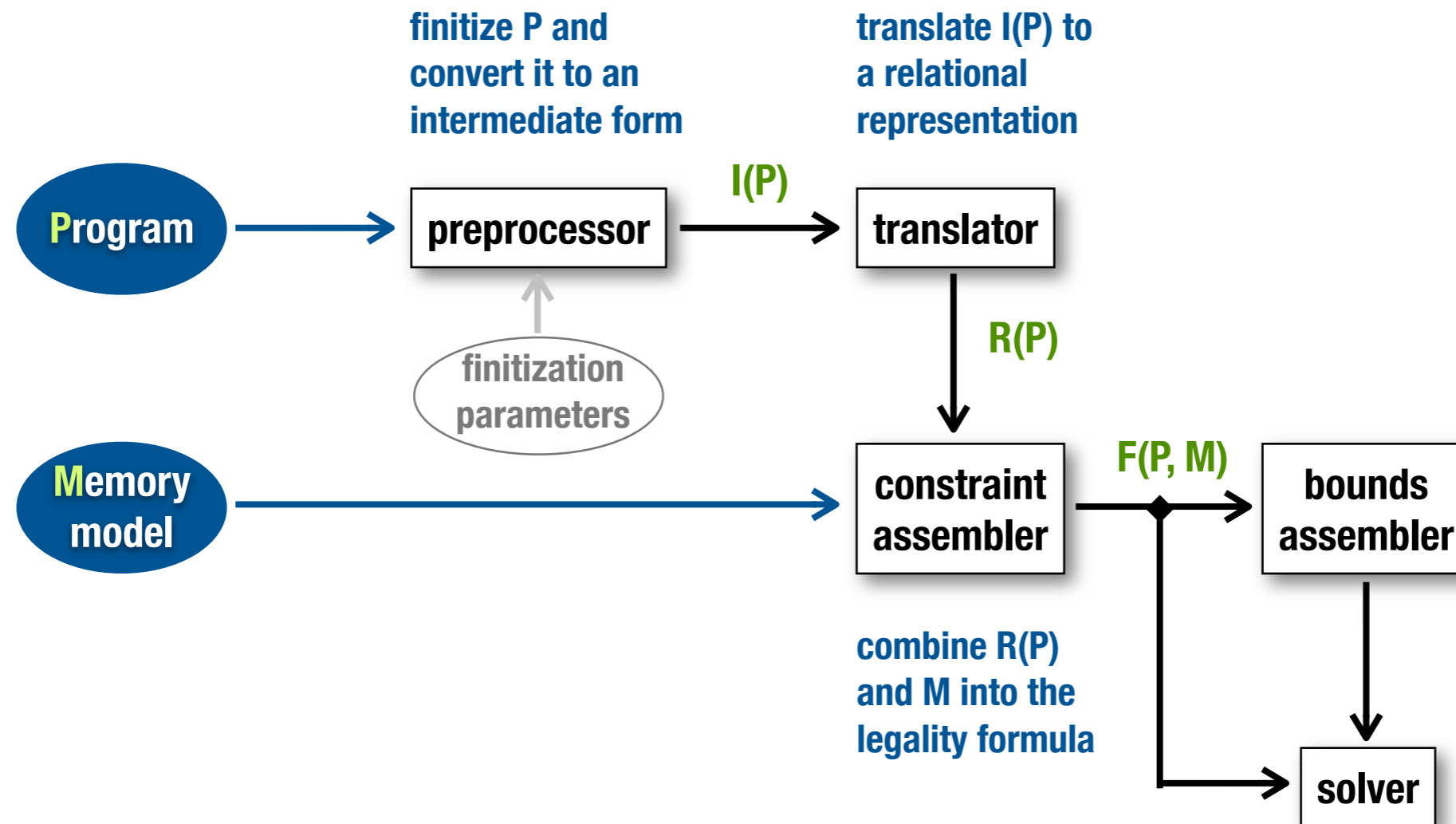
Approach



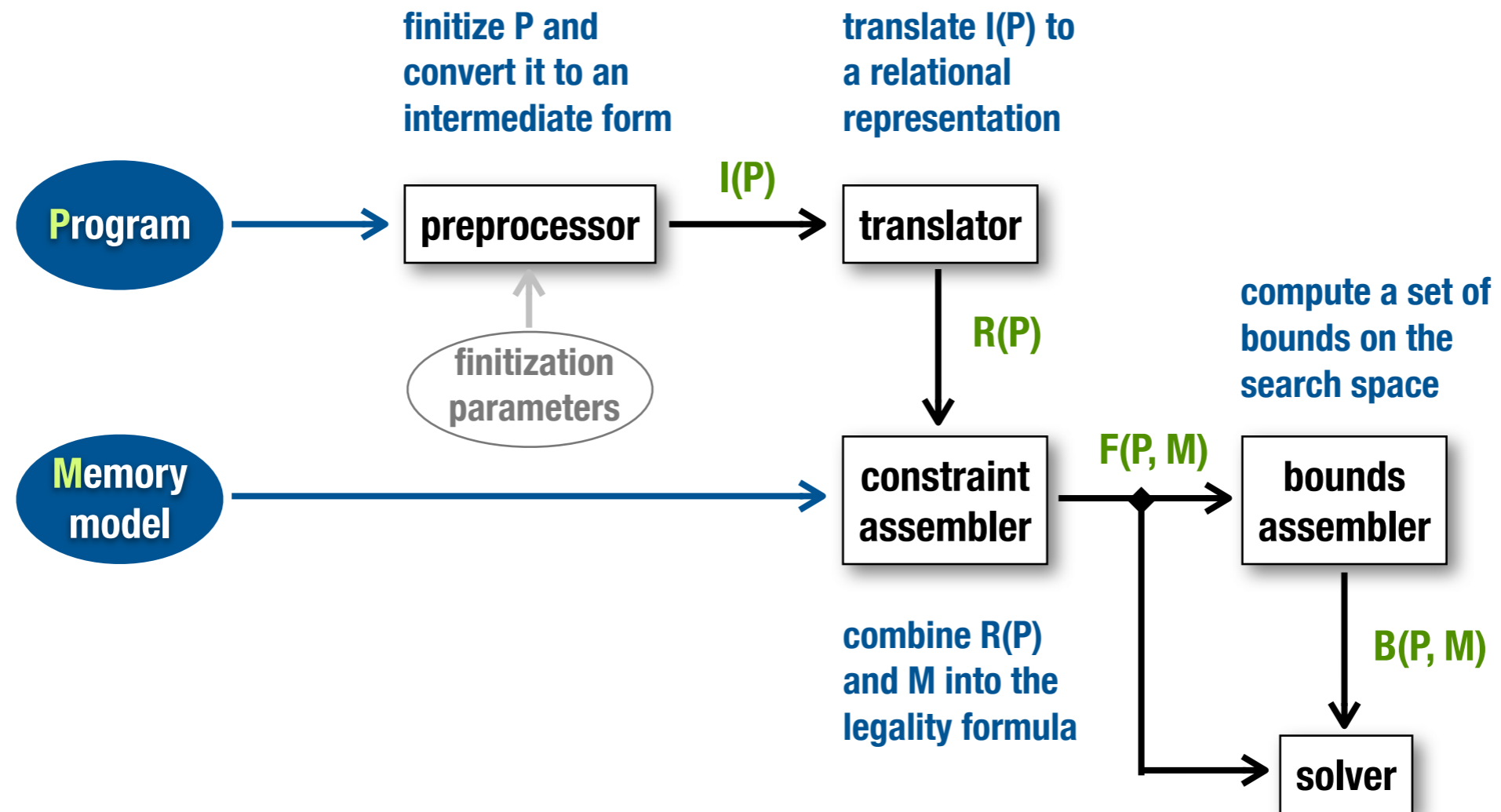
Approach



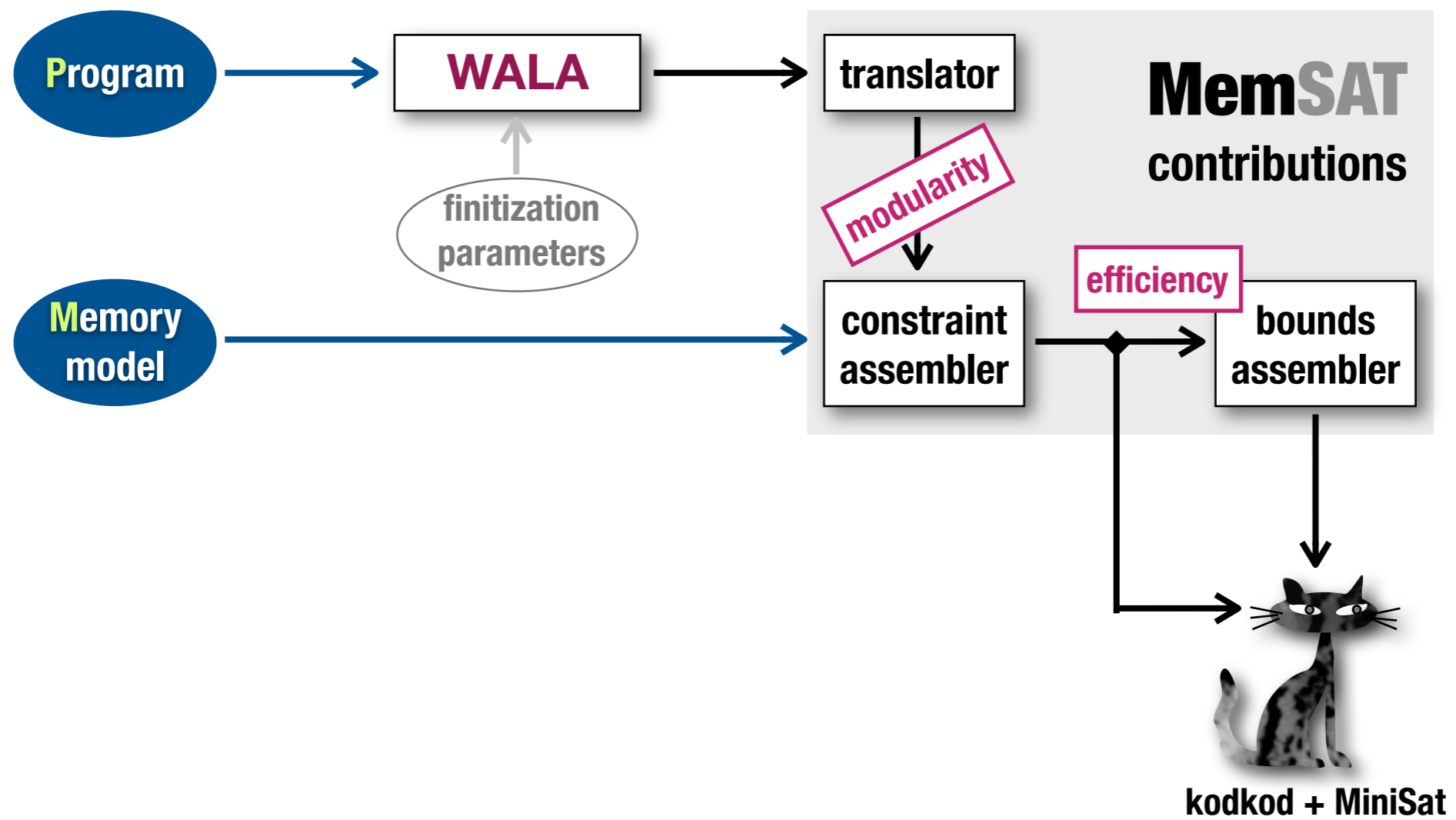
Approach



Approach



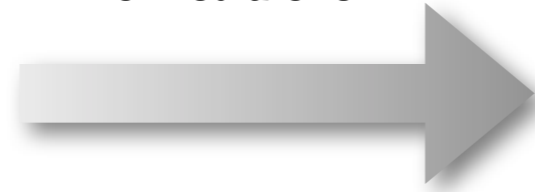
Approach



Preprocessing

```
public class Test1 {  
    static int x = 0;  
    static int y = 0;  
  
    @thread  
    public static void thread1() {  
        final int r1 = x;  
        if (r1 != 0)  
            y = r1;  
        else  
            y = 1;  
        assert r1==1;  
    }  
  
    @thread  
    public static void thread2() {  
        final int r2 = y;  
        x = 1;  
        assert r2==1;  
    }  
}
```

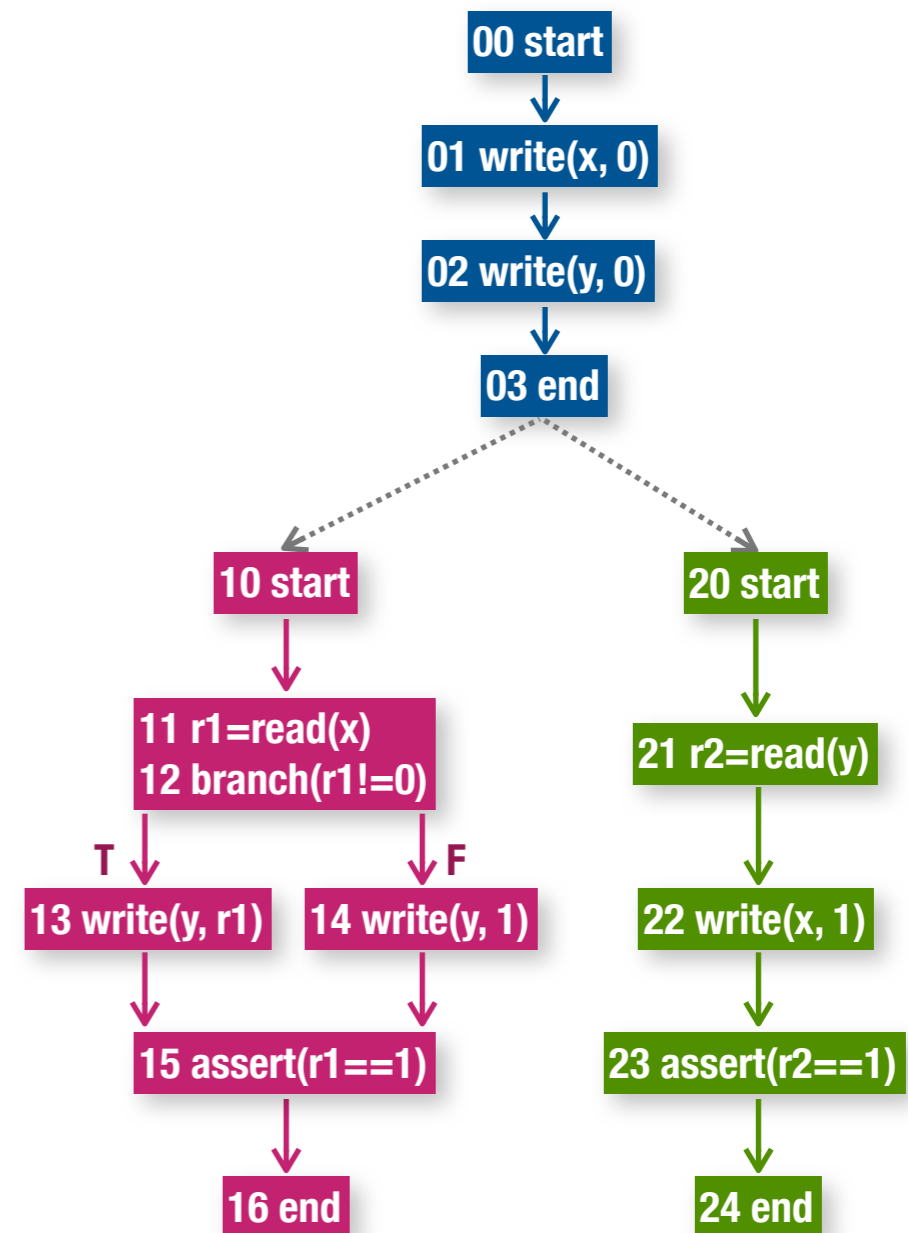
finitize P and
convert it to an
intermediate form



I(P)

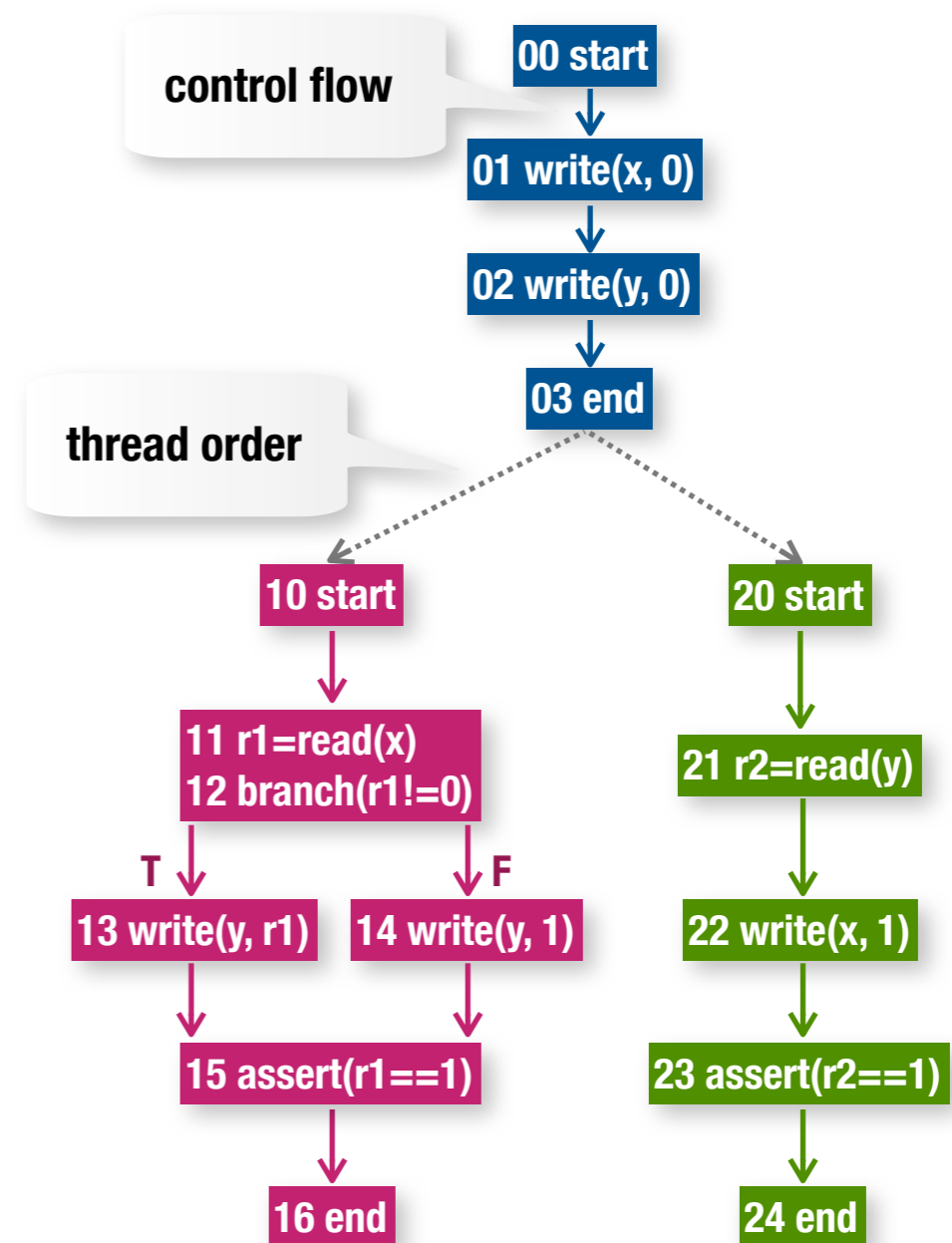
Preprocessing

```
public class Test1 {  
    static int x = 0;  
    static int y = 0;  
  
    @thread  
    public static void thread1() {  
        final int r1 = x;  
        if (r1 != 0)  
            y = r1;  
        else  
            y = 1;  
        assert r1==1;  
    }  
  
    @thread  
    public static void thread2() {  
        final int r2 = y;  
        x = 1;  
        assert r2==1;  
    }  
}
```



Preprocessing

```
public class Test1 {  
    static int x = 0;  
    static int y = 0;  
  
    @thread  
    public static void thread1() {  
        final int r1 = x;  
        if (r1 != 0)  
            y = r1;  
        else  
            y = 1;  
        assert r1==1;  
    }  
  
    @thread  
    public static void thread2() {  
        final int r2 = y;  
        x = 1;  
        assert r2==1;  
    }  
}
```



Preprocessing

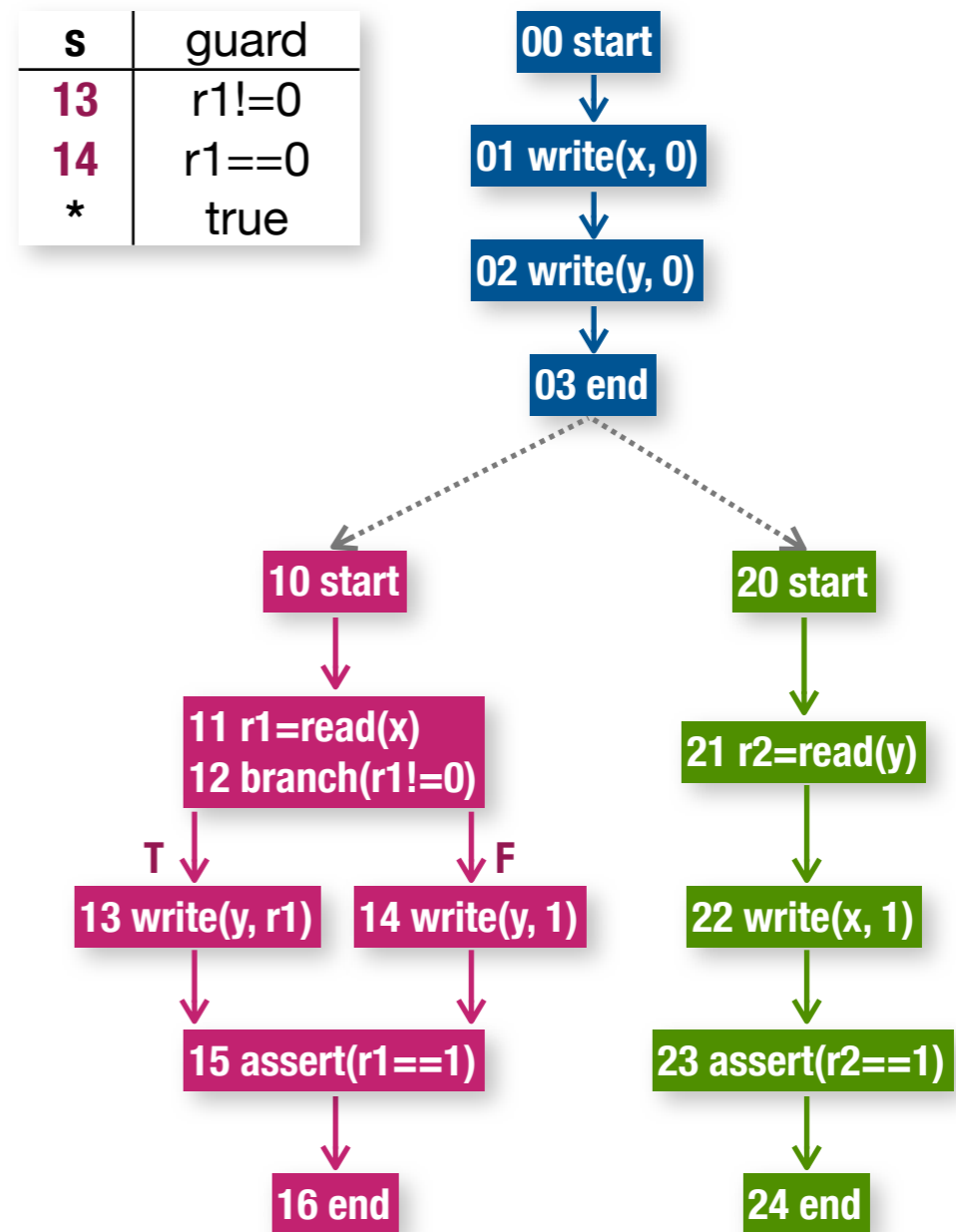
```

public class Test1 {
    static int x = 0;
    static int y = 0;

    @thread
    public static void thread1() {
        final int r1 = x;
        if (r1 != 0)
            y = r1;
        else
            y = 1;
        assert r1==1;
    }

    @thread
    public static void thread2() {
        final int r2 = y;
        x = 1;
        assert r2==1;
    }
}

```



Preprocessing

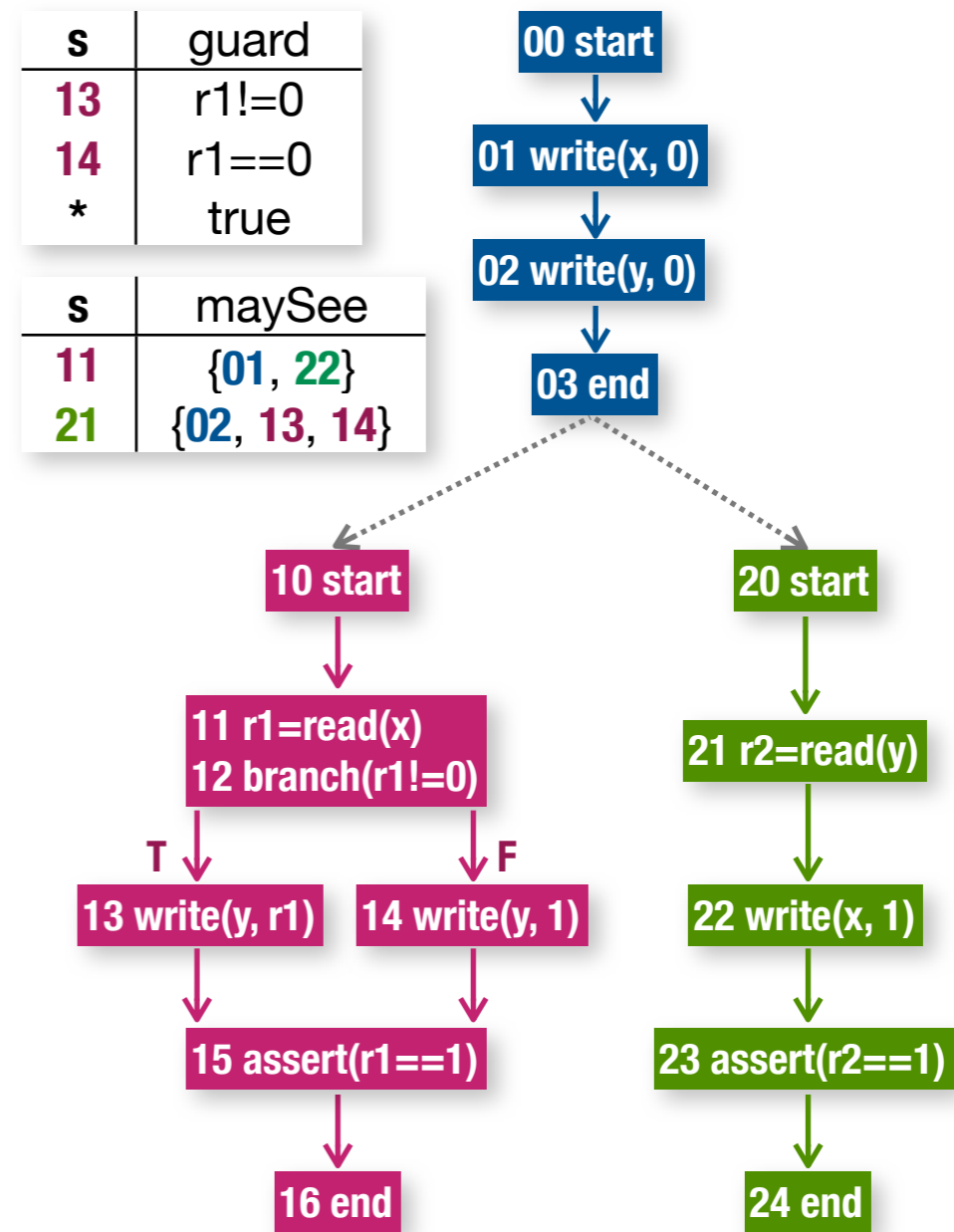
```

public class Test1 {
    static int x = 0;
    static int y = 0;

    @thread
    public static void thread1() {
        final int r1 = x;
        if (r1 != 0)
            y = r1;
        else
            y = 1;
        assert r1==1;
    }

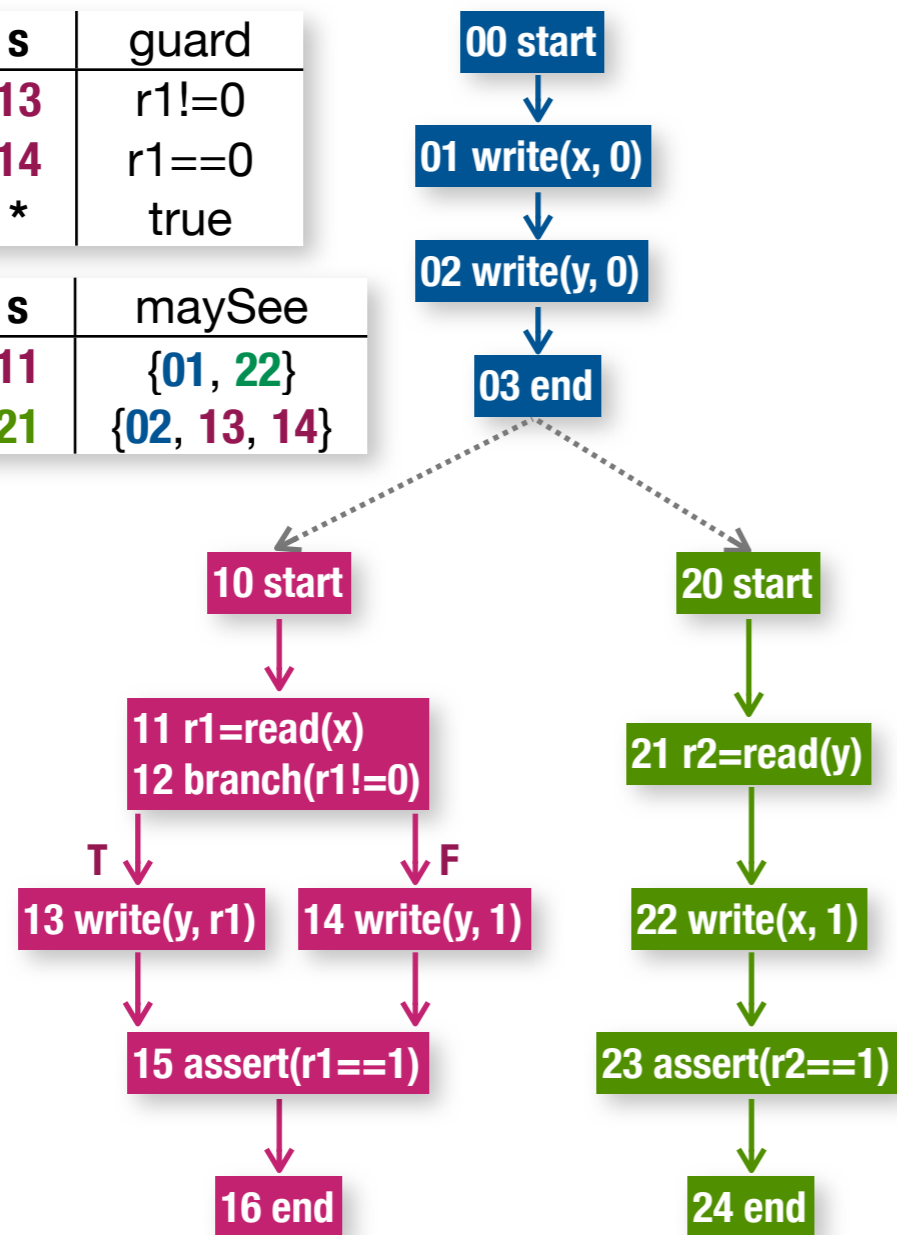
    @thread
    public static void thread2() {
        final int r2 = y;
        x = 1;
        assert r2==1;
    }
}

```

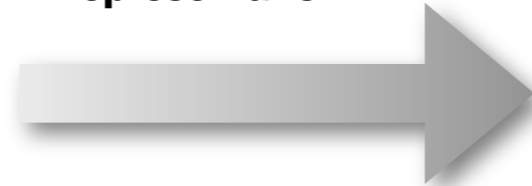


Translation

s	guard
13	r1!=0
14	r1==0
*	true
s	maySee
11	{ 01 , 22 }
21	{ 02 , 13 , 14 }

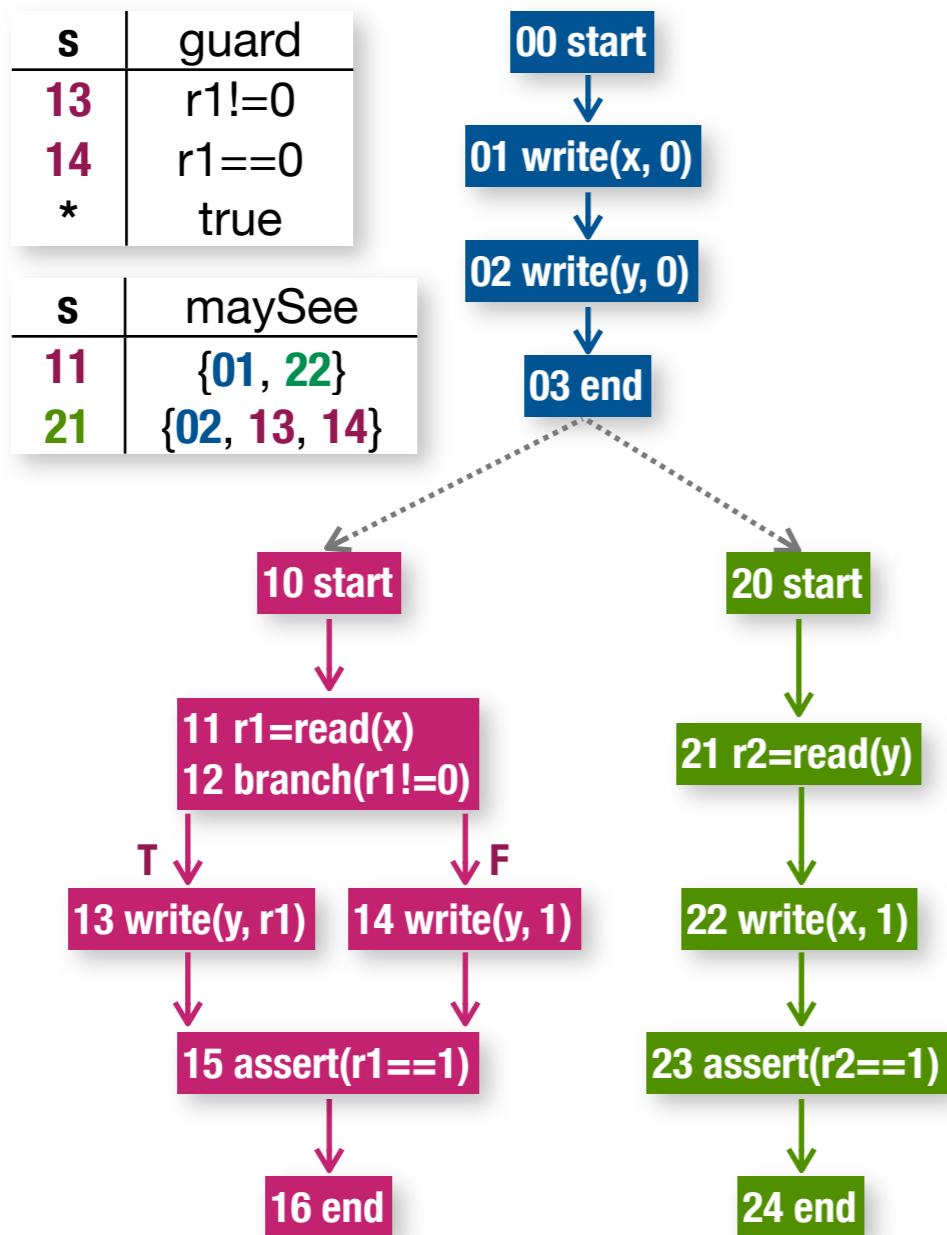


translate I(P) to a relational representation



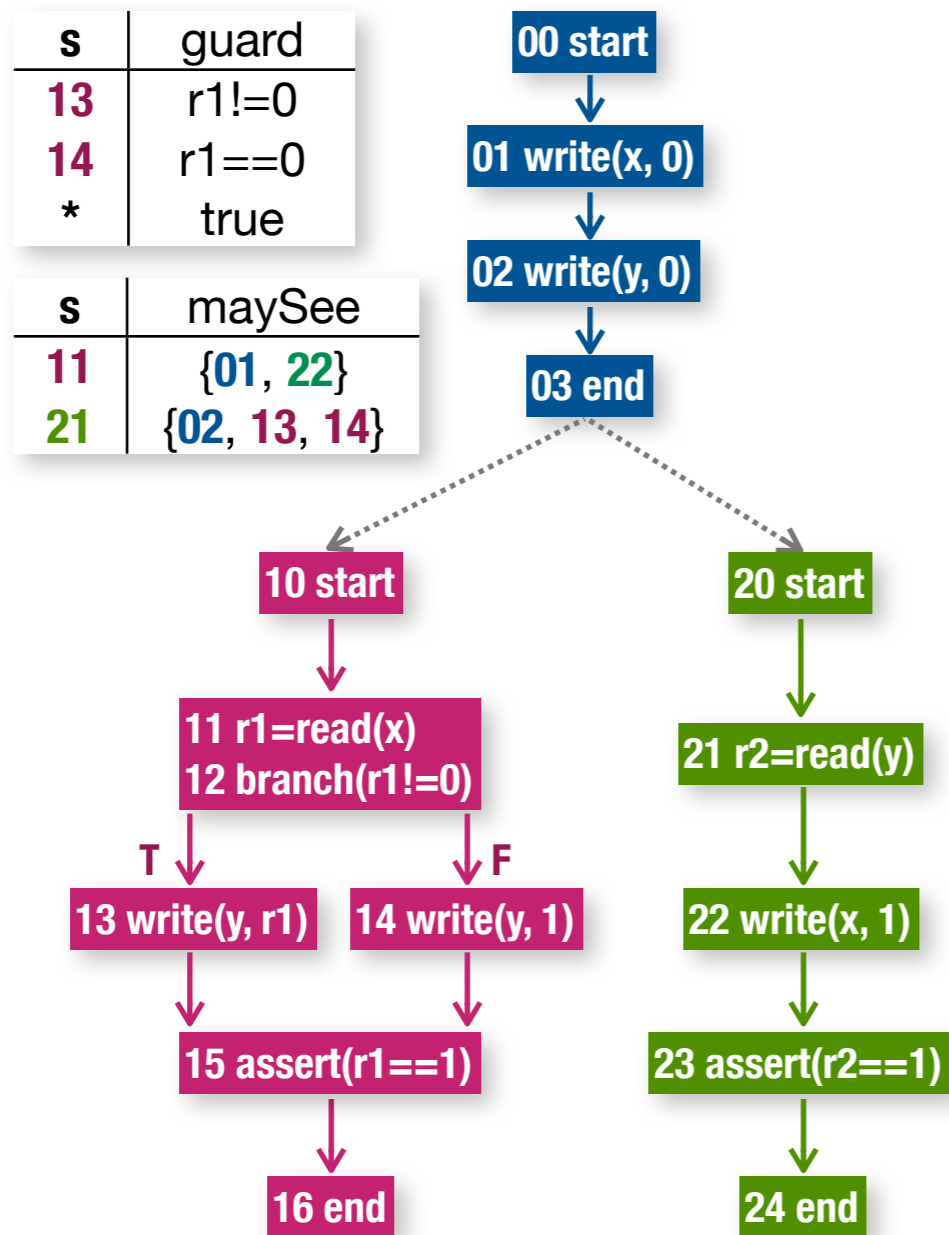
R(P)

Translation



s	Loc	Val	Guard
00			
01			
02			
03			
10			
11			
12			
13			
14			
15			
20			
21			
22			
23			
24			

Translation

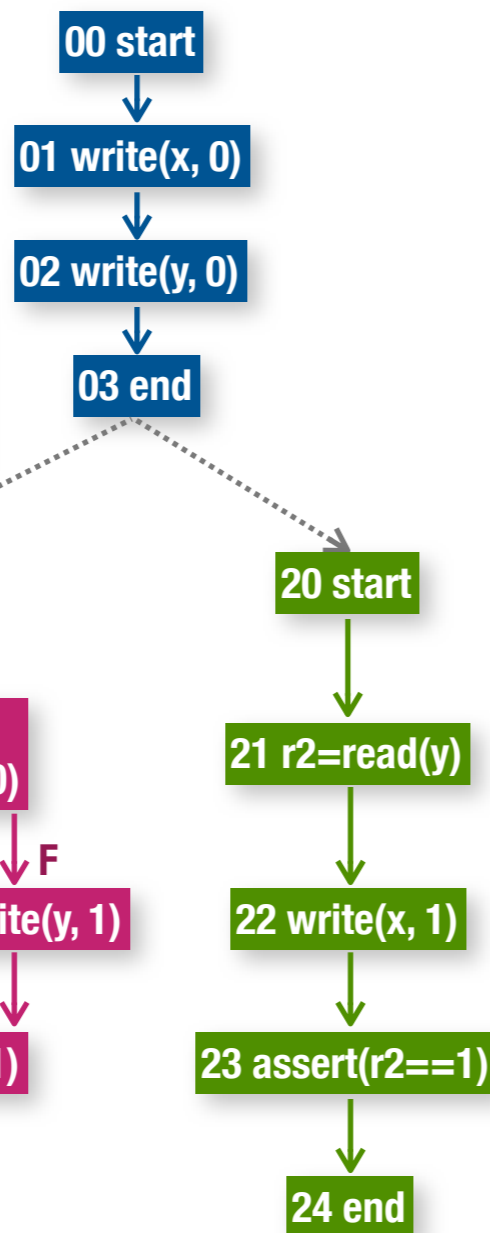


maps reads, writes, locks, and unlocks to relations representing locations that are accessed

s	Loc	Val	Guard
00			
01	x		
02	y		
03			
10			
11	x		
12			
13	y		
14	y		
15			
16			
20			
21	y		
22	x		
23			
24			

Translation

s	guard
13	r1!=0
14	r1==0
*	true
s	maySee
11	{ 01 , 22 }
21	{ 02 , 13 , 14 }

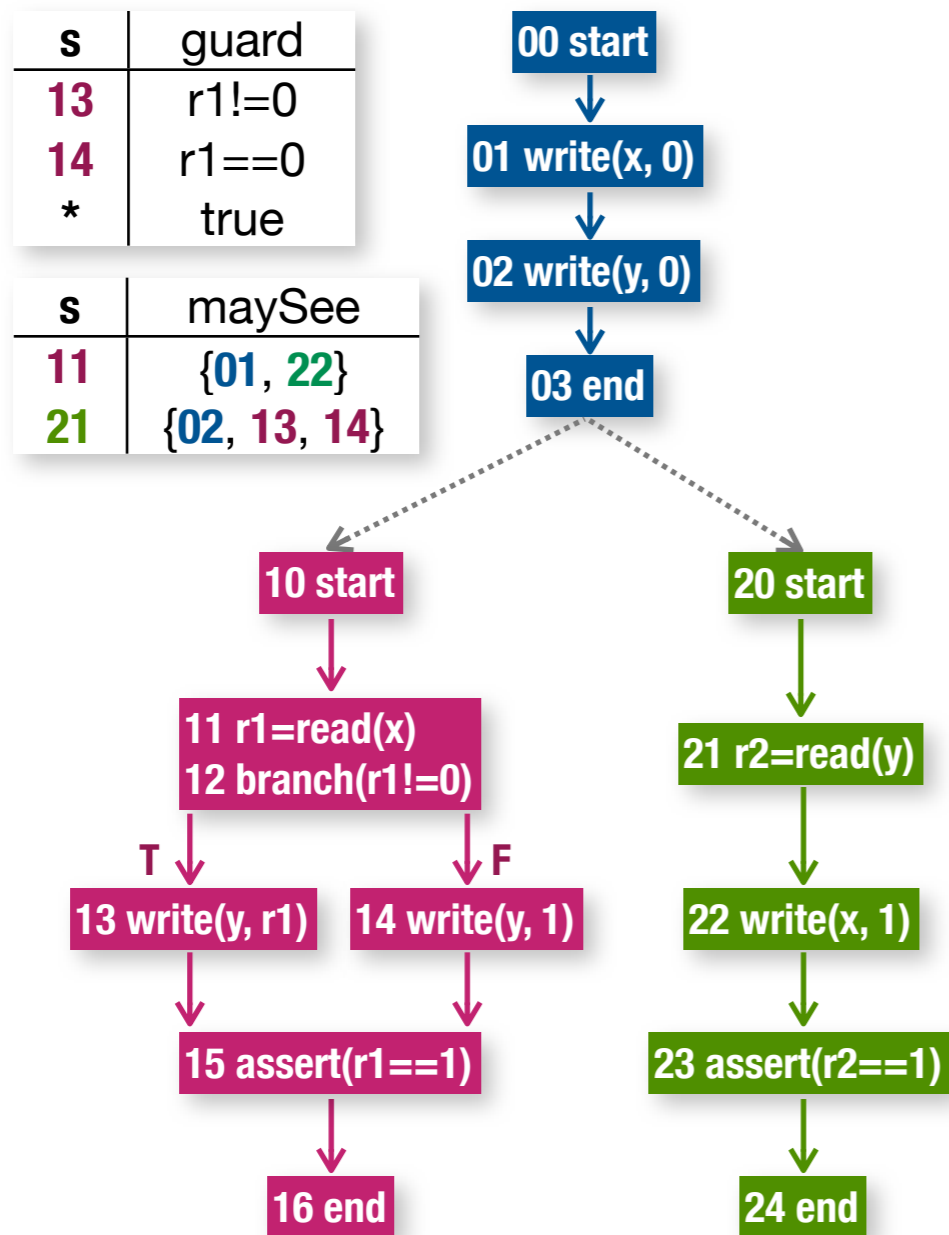


maps reads, writes, locks, and unlocks to relations representing locations that are accessed

relational constants that represent fields: $x = \{<x>\}$ and $y = \{<y>\}$

s	Loc	Val	Guard
00			
01	x		
02	y		
03			
10			
11	x		
12			
13	y		
14	y		
15			
20			
21	y		
22	x		
23			
24			

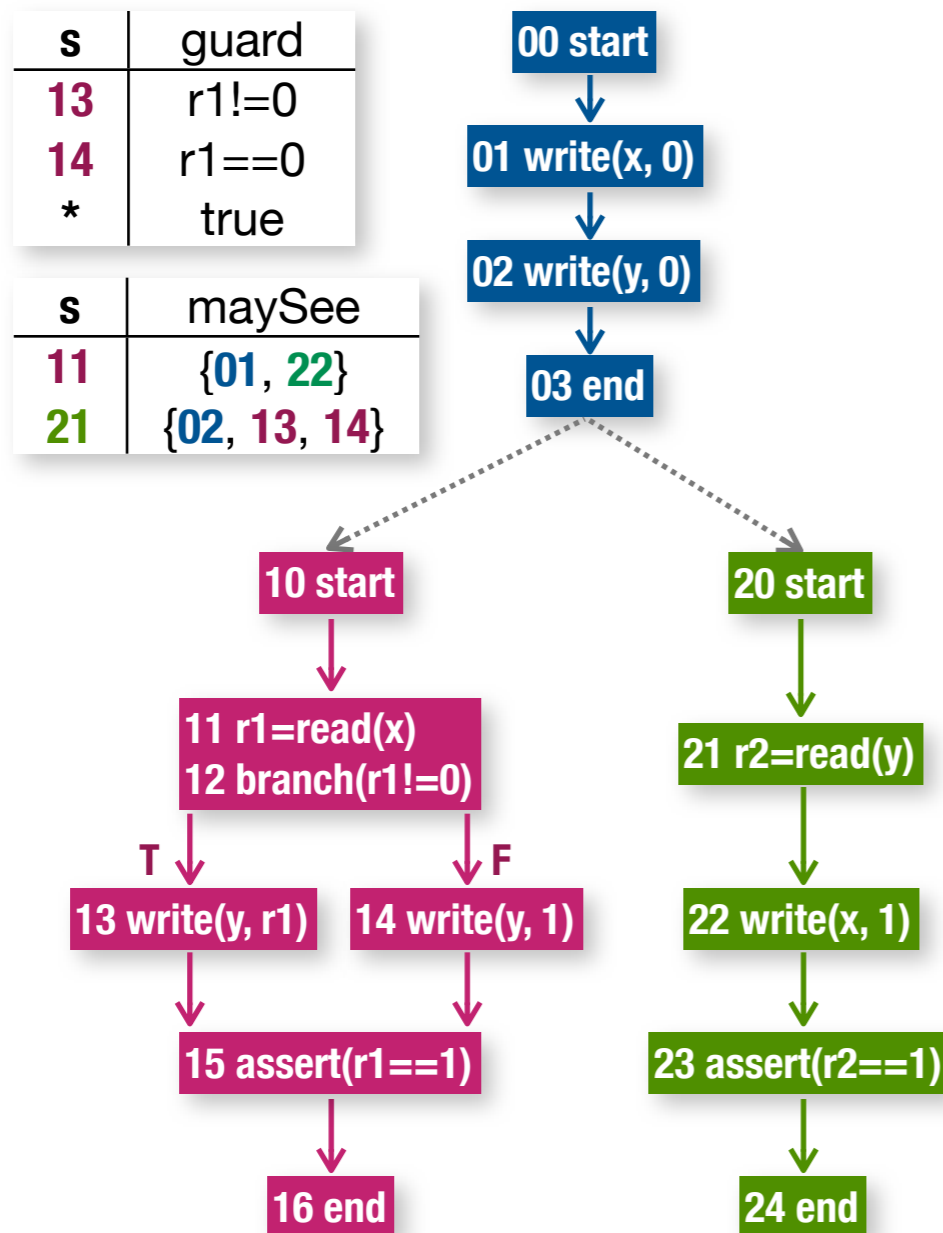
Translation



maps writes and asserts to relational encodings of the values written or asserted

s	Loc	Val	Guard
00			
01	x	Bits(0)	
02	y	Bits(0)	
03			
10			
11	x		
12			
13	y	<i>r1</i>	
14	y	Bits(1)	
15		<i>r1</i> =Bits(1)	
16			
20			
21	y		
22	x	Bits(1)	
23		<i>r2</i> =Bits(1)	
24			

Translation



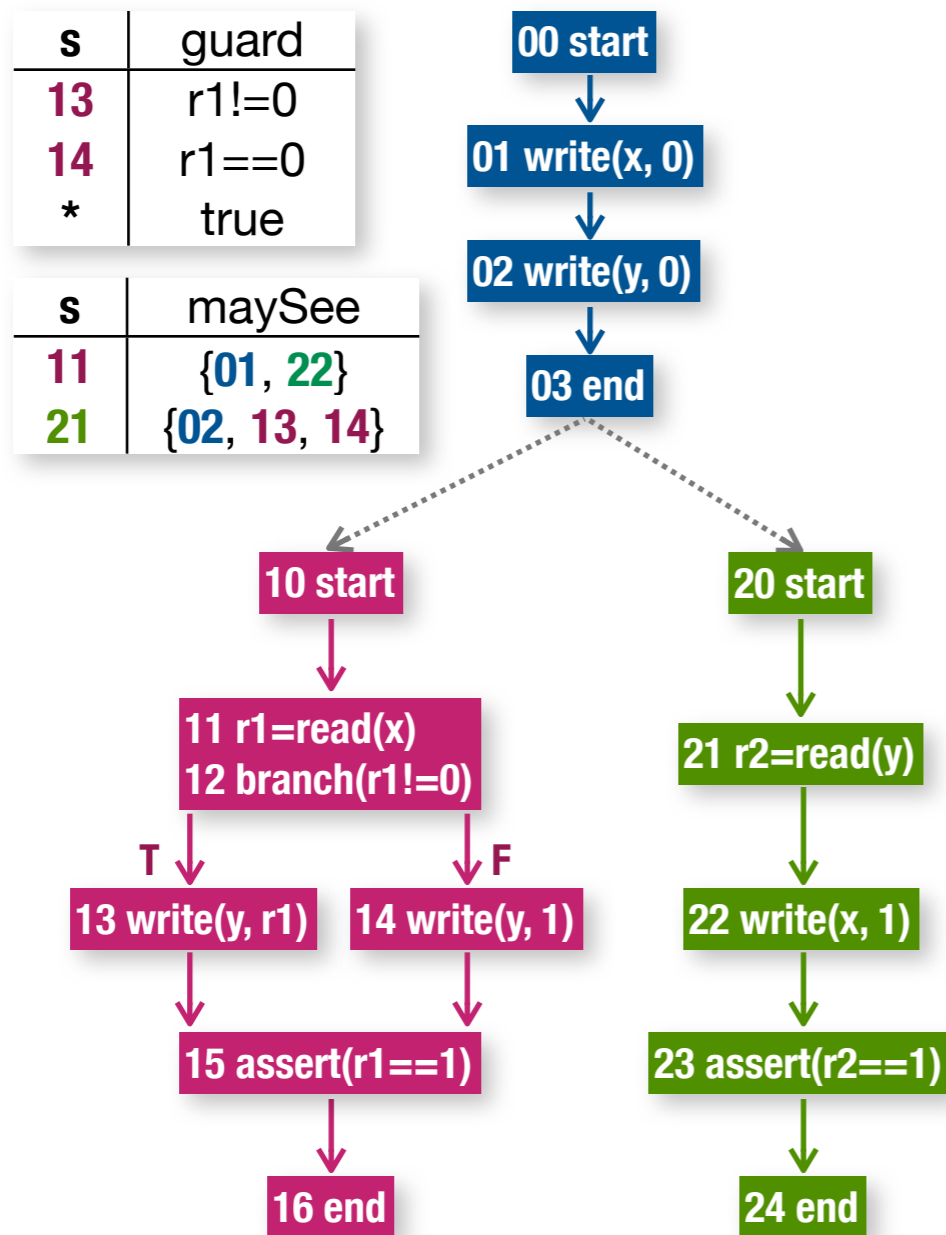
s	guard
13	r1!=0
14	r1==0
*	true
s	maySee
11	{01, 22}
21	{02, 13, 14}

maps writes and asserts to relational encodings of the values written or asserted

s	Loc	Val	Guard
00			
01	x	Bits(0)	
02	y	Bits(0)	
03			
15		<i>r1</i> Bits(1)	
16		<i>r1</i> =Bits(1)	
20			
21	y		
22	x	Bits(1)	
23		<i>r2</i> =Bits(1)	
24			

relational variable that acts as a placeholder for the value read into r1

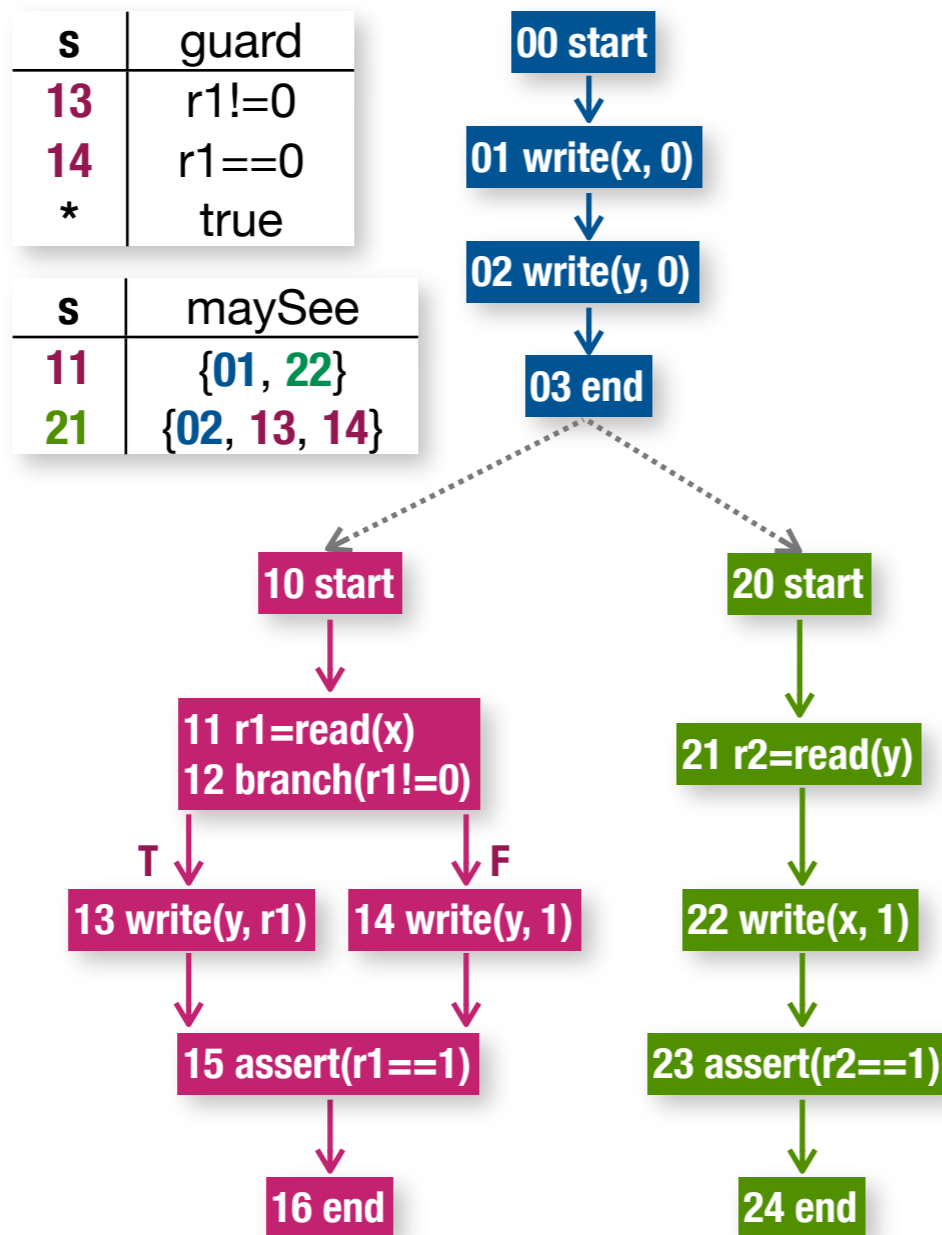
Translation



maps statements to formulas that encode their guards

s	Loc	Val	Guard
00			T
01	x	Bits(0)	T
02	y	Bits(0)	T
03			T
10			T
11	x		T
12			T
13	y	<i>r1</i>	<i>r1</i> ≠Bits(0)
14	y	Bits(1)	<i>r1</i> =Bits(0)
15		<i>r1</i> =Bits(1)	T
16			T
20			T
21	y		T
22	x	Bits(1)	T
23		<i>r2</i> =Bits(1)	T
24			T

Translation



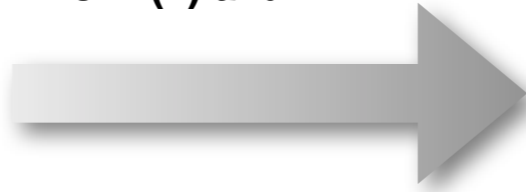
s	Loc	Val	Guard
00			T
01	x	Bits(0)	T
02	y	Bits(0)	T
03			T
10			T
11	x		T
12			T
13	y	<i>r1</i>	$r1 \neq \text{Bits}(0)$
14	y	Bits(1)	$r1 = \text{Bits}(0)$
15		$r1 = \text{Bits}(1)$	T
16			T
20			T
21	y		T
22	x	Bits(1)	T
23		$r2 = \text{Bits}(1)$	T
24			T

Constraint assembly

s	guard			
13	r1!=0			
14		s	Loc	Val
*		00		T
		01	x	Bits(0)
s		02	y	Bits(0)
11		03		T
21		10		T
		11	x	T
		12		T
		13	y	r1
		14	y	Bits(1)
		15		r1=Bits(1)
		16		T
		20		T
13		21	y	T
		22	x	Bits(1)
		23		r2=Bits(1)
		24		T

Annotations: "00 start" points to the first row. "16 end" points to row 16. "24 end" points to row 24.

construct the
legality formula
for R(P) and M



F(P, M)

Constraint assembly

s	guard			
13	r1!=0			
14		s	Loc	Val
*		00		T
		01	x	Bits(0)
s		02	y	Bits(0)
11		03		T
21		10		T
		11	x	T
		12		T
		13	y	r1
		14	y	Bits(1)
		15		r1=Bits(1)
		16		T
		20		T
13		21	y	T
		22	x	Bits(1)
		23		r2=Bits(1)
		24		T

Annotations: "00 start" points to row 14; "16 end" points to row 16; "24 end" points to row 24.

$F(R(P), E) \wedge$

$F_{\alpha}(R(P), E) \wedge$

$\wedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$

$M(E, E_1, \dots, E_k)$

Constraint assembly

s	guard			
13	r1!=0			
14		s	Loc	Val
*		00		T
		01	x	Bits(0)
s		02	y	Bits(0)
11		03		T
21		10		T
		11	x	T
		12		T
		13	y	r1
		14	y	Bits(1)
		15		r1=Bits(1)
		16		T
		20		T
13		21	y	T
		22	x	Bits(1)
		23		r2=Bits(1)
		24		T

Annotations: "00 start" points to row 14; "16 end" points to row 16; "24 end" points to row 24.

The witness execution E respects the sequential semantics of P

$$F(R(P), E) \wedge$$

$$F_{\alpha}(R(P), E) \wedge$$

$$\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$$

$$M(E, E_1, \dots, E_k)$$

Constraint assembly

s	guard			
13	r1!=0			
14		s	Loc	Val
*		00		T
		01	x	Bits(0)
s		02	y	Bits(0)
11		03		T
21		10		T
		11	x	T
		12		T
		13	y	r1
		14	y	Bits(1)
		15		r1=Bits(1)
		16		T
		20		T
13		21	y	T
		22	x	Bits(1)
		23		r2=Bits(1)
		24		T

Annotations: "00 start" points to row 00; "16 end" points to row 16; "24 end" points to row 24.

E executes and satisfies the assertions in P

The witness execution E respects the sequential semantics of P

$$\begin{aligned}
 & \mathbf{F(R(P), E)} \wedge \\
 & \mathbf{F_\alpha(R(P), E)} \wedge \\
 & \wedge_{1 \leq i \leq k} \mathbf{F(R(P), E_i)} \wedge \\
 & \mathbf{M(E, E_1, \dots, E_k)}
 \end{aligned}$$

Constraint assembly

s	guard			
13	r1!=0			00 start
s	Loc	Val	Guard	
*	00		T	
	01	x	Bits(0)	T
s	02	y	Bits(0)	T
11	03		T	
21	10		T	
	11	x	T	
	12		T	
	13	y	<i>r1</i>	<i>r1</i> ≠Bits(0)
	14	y	Bits(1)	<i>r1</i> =Bits(0)
	15		<i>r1</i> =Bits(1)	T
	16		T	
	20		T	
13	21	y	T	
	22	x	Bits(1)	T
	23		<i>r2</i> =Bits(1)	T
	24		T	

16 end

24 end

The witness execution E respects the sequential semantics of P

E executes and satisfies the assertions in P

Each speculative execution E_i respects the sequential semantics of P

$$F(R(P), E) \wedge$$

$$F_{\alpha}(R(P), E) \wedge$$

$$\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$$

$$M(E, E_1, \dots, E_k)$$

Constraint assembly

s	guard			
13	r1!=0			
14		s	Loc	Val
*		00		T
		01	x	Bits(0)
s		02	y	Bits(0)
11		03		T
21		10		T
		11	x	T
		12		T
		13	y	r1
		14	y	Bits(1)
		15		r1=Bits(1)
		16		T
		20		T
13		21	y	T
		22	x	Bits(1)
		23		r2=Bits(1)
		24		T

Annotations: "00 start" points to row 00; "16 end" points to row 16; "24 end" points to row 24.

The witness execution E respects the sequential semantics of P

E executes and satisfies the assertions in P

Each speculative execution E_i respects the sequential semantics of P

$$F(R(P), E) \wedge$$

$$F_{\alpha}(R(P), E) \wedge$$

$$\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$$

$$M(E, E_1, \dots, E_k)$$

E and all E_i respect the memory model constraints

Constraint assembly

s	guard			
13	r1!=0			
14		s	Loc	Val
*		00		T
		01	x	Bits(0)
s		02	y	Bits(0)
11		03		T
21		10		T
		11	x	T
		12		T
		13	y	r1
		14	y	Bits(1)
		15		r1=Bits(1)
		16		T
		20		T
13		21	y	T
		22	x	Bits(1)
		23		r2=Bits(1)
		24		T

Annotations: "00 start" points to the first row. "16 end" points to the row with s=16. "24 end" points to the row with s=24.

$F(R(P), E) \wedge$

$F_{\alpha}(R(P), E) \wedge$

$\wedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$

$M(E, E_1, \dots, E_k)$

Constraint assembly: $F_\alpha(R(P), E)$

s	Loc	Val	Guard
00			T
01	x	Bits(0)	T
02	y	Bits(0)	T
03			T
10			T
11	x		T
12			T
13	y	r1	r1 ≠Bits(0)
14	y	Bits(1)	r1 =Bits(0)
15		r1 =Bits(1)	T
16			T
20			T
21	y		T
22	x	Bits(1)	T
23		r2 =Bits(1)	T
24			T

- A* set of all executed actions
- W* maps reads to seen writes
- V* maps writes to written values
- l* maps writes to written values
- m* maps locks/unlocks to monitors

$F(R(P), E)$

$F_\alpha(R(P), E) \wedge$

$\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$

$M(E, E_1, \dots, E_k)$

Constraint assembly: $F_{\alpha}(R(P), E)$

- A set of all executed actions
- W maps reads to seen writes
- V maps writes to written values
- I maps writes to written values
- m maps locks/unlocks to monitors

s	Loc	Val	Guard	
00 start			T	a_{00}
01 write(x, 0)	x	Bits(0)	T	a_{01}
02 write(y, 0)	y	Bits(0)	T	a_{02}
03 end			T	a_{03}
10 start			T	a_{10}
11 r1=read(x)	x		T	a_{11}
12 branch(r1!=0)			T	
13 write(y, r1)	y	r1	r1 ≠Bits(0)	a_{13}
14 write(y, 1)	y	Bits(1)	r1 =Bits(0)	a_{14}
15 assert(r1==1)		r1 =Bits(1)	T	
16 end			T	a_{16}
20 start			T	a_{20}
21 r2=read(y)	y		T	a_{21}
22 write(x, 1)	x	Bits(1)	T	a_{22}
23 assert(r2==1)		r2 =Bits(1)	T	
24 end			T	a_{24}

relational variable a_{ij} represents the action performed if E executes the statement ij

$F(R(P), E)$

$F_{\alpha}(R(P), E) \wedge$

$\wedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$

$M(E, E_1, \dots, E_k)$

Constraint assembly: $F_\alpha(R(P), E)$

- A set of all executed actions
- W maps reads to seen writes
- V maps writes to written values
- l maps writes to written values
- m maps locks/unlocks to monitors

	s	Loc	Val	Guard	
00 start				T	a_{00}
01 write(x, 0)		x	Bits(0)	T	a_{01}
02 write(y, 0)		y	Bits(0)	T	a_{02}
03 end				T	a_{03}
10 start				T	a_{10}
11 r1=read(x)		x		T	a_{11}
12 branch(r1!=0)				T	
13 write(y, r1)		y	r1	r1 ≠Bits(0)	a_{13}
14 write(y, 1)		y	Bits(1)	r1 =Bits(0)	a_{14}
15 assert(r1==1)			r1 =Bits(1)	T	
16 end				T	a_{16}
20 start				T	a_{20}
21 r2=read(y)		y		T	a_{21}
22 write(x, 1)		x	Bits(1)	T	a_{22}
23 assert(r2==1)			r2 =Bits(1)	T	
24 end				T	a_{24}

$V[W[a_{11}]]$

$V[W[a_{21}]]$

$F(R(P), E)$

$F_\alpha(R(P), E) \wedge$

$\wedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$

$M(E, E_1, \dots, E_k)$

Constraint assembly: $F_{\alpha}(R(P), E)$

	s	Loc	Val	Guard	
00 start				T	a_{00}
01 write(x, 0)		x	Bits(0)	T	a_{01}
02 write(y, 0)		y	Bits(0)	T	a_{02}
03 end				T	a_{03}
10 start				T	a_{10}
11 r1=read(x)		x		T	a_{11}
12 branch(r1!=0)				T	
13 write(y, r1)		y	$r1$	$r1 \neq \text{Bits}(0)$	a_{13}
14 write(y, 1)		y	Bits(1)	$r1 = \text{Bits}(0)$	a_{14}
15 assert(r1==1)			$r1 = \text{Bits}(1)$	T	
16 end				T	a_{16}
20 start				T	a_{20}
21 r2=read(y)		y		T	a_{21}
22 write(x, 1)		x	Bits(1)	T	a_{22}
23 assert(r2==1)			$r2 = \text{Bits}(1)$	T	
24 end				T	a_{24}

Note: A pink callout bubble labeled $V[W[a_{11}]]$ points to the state transition between lines 11 and 12. A green callout bubble labeled $V[W[a_{21}]]$ points to the state transition between lines 21 and 22.

$$\begin{aligned}
 &F(R(P), E) \wedge \\
 &V[W[a_{11}]] = \text{Bits}(1) \wedge \\
 &V[W[a_{21}]] = \text{Bits}(1) \wedge \\
 &\wedge_{1 \leq i \leq k} F(R(P), E_i) \wedge \\
 &M(E, E_1, \dots, E_k)
 \end{aligned}$$

Constraint assembly: $F(R(P), E)$

- A set of all executed actions
- W maps reads to seen writes
- V maps writes to written values
- l maps writes to written values
- m maps locks/unlocks to monitors

	s	Loc	Val	Guard	
00 start				T	a_{00}
01 write(x, 0)		x	Bits(0)	T	a_{01}
02 write(y, 0)		y	Bits(0)	T	a_{02}
03 end				T	a_{03}
10 start				T	a_{10}
11 r1=read(x)		x		T	a_{11}
12 branch(r1!=0)				T	
13 write(y, r1)		y	r1	r1 ≠Bits(0)	a_{13}
14 write(y, 1)		y	Bits(1)	r1 =Bits(0)	a_{14}
15 assert(r1==1)			r1 =Bits(1)	T	
16 end				T	a_{16}
20 start				T	a_{20}
21 r2=read(y)		y		T	a_{21}
22 write(x, 1)		x	Bits(1)	T	a_{22}
23 assert(r2==1)			r2 =Bits(1)	T	
24 end				T	a_{24}

$V[W[a_{11}]]$

$V[W[a_{21}]]$

$F(R(P), E) \wedge$

$V[W[a_{11}]] = \text{Bits}(1) \wedge$

$V[W[a_{21}]] = \text{Bits}(1) \wedge$

$\wedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$

$M(E, E_1, \dots, E_k)$

Constraint assembly: $F(R(P), E)$

- A set of all executed actions
- W maps reads to seen writes
- V maps writes to written values
- l maps writes to written values
- m maps locks/unlocks to monitors

	s	Loc	Val	Guard	
00 start				T	a_{00}
01 write(x, 0)		x	Bits(0)	T	a_{01}
02 write(y, 0)		y	Bits(0)	T	a_{02}
03 end				T	a_{03}
10 start				T	a_{10}
11 r1=read(x)		x		T	a_{11}
12 branch(r1!=0)				T	
13 write(y, r1)		y	r1	r1 ≠Bits(0)	a_{13}
14 write(y, 1)		y	Bits(1)	r1 =Bits(0)	a_{14}
15 assert(r1==1)			r1 =Bits(1)	T	
16 end				T	a_{16}
20 start				T	a_{20}
21 r2=read(y)		y		T	a_{21}
22 write(x, 1)		x	Bits(1)	T	a_{22}
23 assert(r2==1)			r2 =Bits(1)	T	
24 end				T	a_{24}

$V[W[a_{11}]]$

$V[W[a_{21}]]$

$$\bigwedge_{s \in P} F(s, R(P), E) \wedge$$

$$A = a_{00} \cup \dots \cup a_{24} \wedge$$

$$V[W[a_{11}]] = \text{Bits}(1) \wedge$$

$$V[W[a_{21}]] = \text{Bits}(1) \wedge$$

$$\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$$

$$M(E, E_1, \dots, E_k)$$

Constraint assembly: $F(R(P), E)$

A set of all executed actions
W maps reads to seen writes
V maps writes to written values
l maps writes to written values
m maps locks/unlocks to monitors

s	Loc	Val	Guard	
00 start			T	a_{00}
01 write(x, 0)	x	Bits(0)	T	a_{01}
02 write(y, 0)	y	Bits(0)	T	a_{02}
03 end			T	a_{03}
10 start			T	a_{10}
11 r1=read(x)	x		T	a_{11}
12 branch(r1!=0)			T	
13 write(y, r1)	y	r1	r1 ≠Bits(0)	a_{13}
14 write(y, 1)	y	Bits(1)	r1 =Bits(0)	a_{14}
15 assert(r1==1)		r1 =Bits(1)	T	
16 end			T	a_{16}
20 start			T	a_{20}
21 r2=read(y)	y		T	a_{21}
22 write(x, 1)	x	Bits(1)	T	a_{22}
23 assert(r2==1)		r2 =Bits(1)	T	
24 end			T	a_{24}

$V[W[a_{11}]]$

$V[W[a_{21}]]$

- ▶ 0 or 1 action performed
- ▶ action performed iff the guard is true
- ▶ no other statement performs the same action
- ▶ action location is valid
- ▶ action value is valid

$$\bigwedge_{s \in P} F(s, R(P), E) \wedge \\
 A = a_{00} \cup \dots \cup a_{24} \wedge \\
 V[W[a_{11}]] = \text{Bits}(1) \wedge \\
 V[W[a_{21}]] = \text{Bits}(1) \wedge \\
 \bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge \\
 M(E, E_1, \dots, E_k)$$

Constraint assembly: $F(R(P), E)$

A set of all executed actions
W maps reads to seen writes
V maps writes to written values
l maps writes to written values
m maps locks/unlocks to monitors

s	Loc	Val	Guard	
00 start			T	a_{00}
01 write(x, 0)	x	Bits(0)	T	a_{01}
02 write(y, 0)	y	Bits(0)	T	a_{02}
03 end			T	a_{03}
10 start			T	a_{10}
11 r1=read(x)	x		T	a_{11}
12 branch(r1!=0)			T	
13 write(y, r1)	y	r1	$r1 \neq \text{Bits}(0)$	a_{13}
14 write(y, 1)	y	Bits(1)	$r1 = \text{Bits}(0)$	a_{14}
15 assert(r1==1)		$r1 = \text{Bits}(1)$	T	
16 end			T	a_{16}
20 start			T	a_{20}
21 r2=read(y)	y		T	a_{21}
22 write(x, 1)	x	Bits(1)	T	a_{22}
23 assert(r2==1)		$r2 = \text{Bits}(1)$	T	
24 end			T	a_{24}

$V[W[a_{11}]]$

$V[W[a_{21}]]$

- ▶ 0 or 1 action performed
- ▶ action performed iff the guard is true
- ▶ no other statement performs the same action
- ▶ action location is valid
- ▶ action value is valid

$$\bigwedge_{s \in P} F(s, R(P), E) \wedge \\
 A = a_{00} \cup \dots \cup a_{24} \wedge \\
 V[W[a_{11}]] = \text{Bits}(1) \wedge \\
 V[W[a_{21}]] = \text{Bits}(1) \wedge \\
 \bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge \\
 M(E, E_1, \dots, E_k)$$

Constraint assembly: $F(R(P), E)$

A set of all executed actions
W maps reads to seen writes
V maps writes to written values
l maps writes to written values
m maps locks/unlocks to monitors

s	Loc	Val	Guard	
00 start			T	a_{00}
01 write(x, 0)	x	Bits(0)	T	a_{01}
02 write(y, 0)	y	Bits(0)	T	a_{02}
03 end			T	a_{03}
10 start			T	a_{10}
11 r1=read(x)	x		T	a_{11}
12 branch(r1!=0)			T	
13 write(y, r1)	y	r1	$r1 \neq \text{Bits}(0)$	a_{13}
14 write(y, 1)	y	Bits(1)	$r1 = \text{Bits}(0)$	a_{14}
15 assert(r1==1)		$r1 = \text{Bits}(1)$	T	
16 end			T	a_{16}
20 start			T	a_{20}
21 r2=read(y)	y		T	a_{21}
22 write(x, 1)	x	Bits(1)	T	a_{22}
23 assert(r2==1)		$r2 = \text{Bits}(1)$	T	
24 end			T	a_{24}

$V[W[a_{11}]]$

$V[W[a_{21}]]$

- $|a_{13}| \leq 1 \wedge$
- action performed iff the guard is true
- no other statement performs the same action
- action location is valid
- action value is valid

$$\bigwedge_{s \in P} F(s, R(P), E) \wedge \\
 A = a_{00} \cup \dots \cup a_{24} \wedge \\
 V[W[a_{11}]] = \text{Bits}(1) \wedge \\
 V[W[a_{21}]] = \text{Bits}(1) \wedge \\
 \bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge \\
 M(E, E_1, \dots, E_k)$$

Constraint assembly: $F(R(P), E)$

A set of all executed actions
W maps reads to seen writes
V maps writes to written values
l maps writes to written values
m maps locks/unlocks to monitors

	s	Loc	Val	Guard	
00 start				T	a_{00}
01 write(x, 0)		x	Bits(0)	T	a_{01}
02 write(y, 0)		y	Bits(0)	T	a_{02}
03 end				T	a_{03}
10 start				T	a_{10}
11 r1=read(x)		x		T	a_{11}
12 branch(r1!=0)				T	
13 write(y, r1)		y	r1	$r1 \neq \text{Bits}(0)$	a_{13}
14 write(y, 1)		y	Bits(1)	$r1 = \text{Bits}(0)$	a_{14}
15 assert(r1==1)			$r1 = \text{Bits}(1)$	T	
16 end				T	a_{16}
20 start				T	a_{20}
21 r2=read(y)		y		T	a_{21}
22 write(x, 1)		x	Bits(1)	T	a_{22}
23 assert(r2==1)			$r2 = \text{Bits}(1)$	T	
24 end				T	a_{24}

$V[W[a_{11}]]$

$V[W[a_{21}]]$

$|a_{13}| \leq 1 \wedge$
 $(|a_{13}| = 1 \Leftrightarrow$
 $V[W[a_{11}]] \neq \text{Bits}(0)) \wedge$
 ▶ no other statement
 performs the same action
 ▶ action location is valid
 ▶ action value is valid

$\bigwedge_{s \in P} F(s, R(P), E) \wedge$
 $A = a_{00} \cup \dots \cup a_{24} \wedge$
 $V[W[a_{11}]] = \text{Bits}(1) \wedge$
 $V[W[a_{21}]] = \text{Bits}(1) \wedge$
 $\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$
 $M(E, E_1, \dots, E_k)$

Constraint assembly: $F(R(P), E)$

A set of all executed actions
W maps reads to seen writes
V maps writes to written values
l maps writes to written values
m maps locks/unlocks to monitors

	s	Loc	Val	Guard	
00 start				T	a_{00}
01 write(x, 0)		x	Bits(0)	T	a_{01}
02 write(y, 0)		y	Bits(0)	T	a_{02}
03 end				T	a_{03}
10 start				T	a_{10}
11 r1=read(x)		x		T	a_{11}
12 branch(r1!=0)				T	
13 write(y, r1)		y	r1	$r1 \neq \text{Bits}(0)$	a_{13}
14 write(y, 1)		y	Bits(1)	$r1 = \text{Bits}(0)$	a_{14}
15 assert(r1==1)			$r1 = \text{Bits}(1)$	T	
16 end				T	a_{16}
20 start				T	a_{20}
21 r2=read(y)		y		T	a_{21}
22 write(x, 1)		x	Bits(1)	T	a_{22}
23 assert(r2==1)			$r2 = \text{Bits}(1)$	T	
24 end				T	a_{24}

$V[W[a_{11}]]$

$V[W[a_{21}]]$

$|a_{13}| \leq 1 \wedge$
 $(|a_{13}| = 1 \Leftrightarrow$
 $V[W[a_{11}]] \neq \text{Bits}(0)) \wedge$
 $(a_{13} \cap a_{00}) = \emptyset \wedge \dots \wedge$
 $(a_{13} \cap a_{24}) = \emptyset \wedge$
 ▶ action location is valid
 ▶ action value is valid

$\bigwedge_{s \in P} F(s, R(P), E) \wedge$
 $A = a_{00} \cup \dots \cup a_{24} \wedge$
 $V[W[a_{11}]] = \text{Bits}(1) \wedge$
 $V[W[a_{21}]] = \text{Bits}(1) \wedge$
 $\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$
 $M(E, E_1, \dots, E_k)$

Constraint assembly: $F(R(P), E)$

- A set of all executed actions
- W maps reads to seen writes
- V maps writes to written values
- l maps writes to written values
- m maps locks/unlocks to monitors

	s	Loc	Val	Guard	
00 start				T	a_{00}
01 write(x, 0)		x	Bits(0)	T	a_{01}
02 write(y, 0)		y	Bits(0)	T	a_{02}
03 end				T	a_{03}
10 start				T	a_{10}
11 r1=read(x)		x		T	a_{11}
12 branch(r1!=0)				T	
13 write(y, r1)		y	r1	$r1 \neq \text{Bits}(0)$	a_{13}
14 write(y, 1)		y	Bits(1)	$r1 = \text{Bits}(0)$	a_{14}
15 assert(r1==1)			$r1 = \text{Bits}(1)$	T	
16 end				T	a_{16}
20 start				T	a_{20}
21 r2=read(y)		y		T	a_{21}
22 write(x, 1)		x	Bits(1)	T	a_{22}
23 assert(r2==1)			$r2 = \text{Bits}(1)$	T	
24 end				T	a_{24}

$V[W[a_{11}]]$

$V[W[a_{21}]]$

$|a_{13}| \leq 1 \wedge$
 $(|a_{13}| = 1 \Leftrightarrow$
 $V[W[a_{11}]] \neq \text{Bits}(0)) \wedge$
 $(a_{13} \cap a_{00}) = \emptyset \wedge \dots \wedge$
 $(a_{13} \cap a_{24}) = \emptyset \wedge$
 $l[a_{13}] = y \wedge$
 action value is valid

$\bigwedge_{s \in P} F(s, R(P), E) \wedge$
 $A = a_{00} \cup \dots \cup a_{24} \wedge$
 $V[W[a_{11}]] = \text{Bits}(1) \wedge$
 $V[W[a_{21}]] = \text{Bits}(1) \wedge$
 $\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$
 $M(E, E_1, \dots, E_k)$

Constraint assembly: $F(R(P), E)$

- A set of all executed actions
- W maps reads to seen writes
- V maps writes to written values
- l maps writes to written values
- m maps locks/unlocks to monitors

	s	Loc	Val	Guard	
00 start				T	a_{00}
01 write(x, 0)		x	Bits(0)	T	a_{01}
02 write(y, 0)		y	Bits(0)	T	a_{02}
03 end				T	a_{03}
10 start				T	a_{10}
11 r1=read(x)		x		T	a_{11}
12 branch(r1!=0)				T	
13 write(y, r1)		y	r1	$r1 \neq \text{Bits}(0)$	a_{13}
14 write(y, 1)		y	Bits(1)	$r1 = \text{Bits}(0)$	a_{14}
15 assert(r1==1)			$r1 = \text{Bits}(1)$	T	
16 end				T	a_{16}
20 start				T	a_{20}
21 r2=read(y)		y		T	a_{21}
22 write(x, 1)		x	Bits(1)	T	a_{22}
23 assert(r2==1)			$r2 = \text{Bits}(1)$	T	
24 end				T	a_{24}

$V[W[a_{11}]]$

$V[W[a_{21}]]$

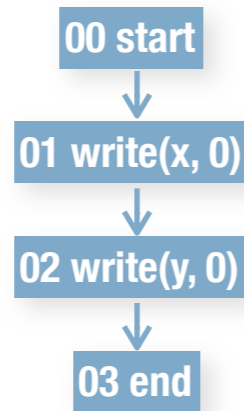
$|a_{13}| \leq 1 \wedge$
 $(|a_{13}| = 1 \Leftrightarrow$
 $V[W[a_{11}]] \neq \text{Bits}(0)) \wedge$
 $(a_{13} \cap a_{00}) = \emptyset \wedge \dots \wedge$
 $(a_{13} \cap a_{24}) = \emptyset \wedge$
 $l[a_{13}] = y \wedge$
 $V[a_{13}] = V[W[a_{11}]]$

$\bigwedge_{s \in P} F(s, R(P), E) \wedge$
 $A = a_{00} \cup \dots \cup a_{24} \wedge$
 $V[W[a_{11}]] = \text{Bits}(1) \wedge$
 $V[W[a_{21}]] = \text{Bits}(1) \wedge$
 $\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge$
 $M(E, E_1, \dots, E_k)$

Bounds assembly

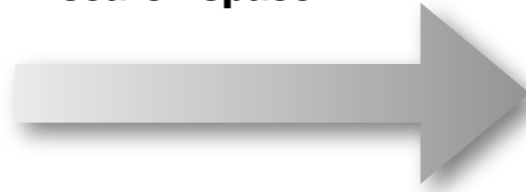
s	guard
13	r1!=0
14	r1==0
*	true

s	maySee
11	{01, 22}
21	{02, 13, 14}

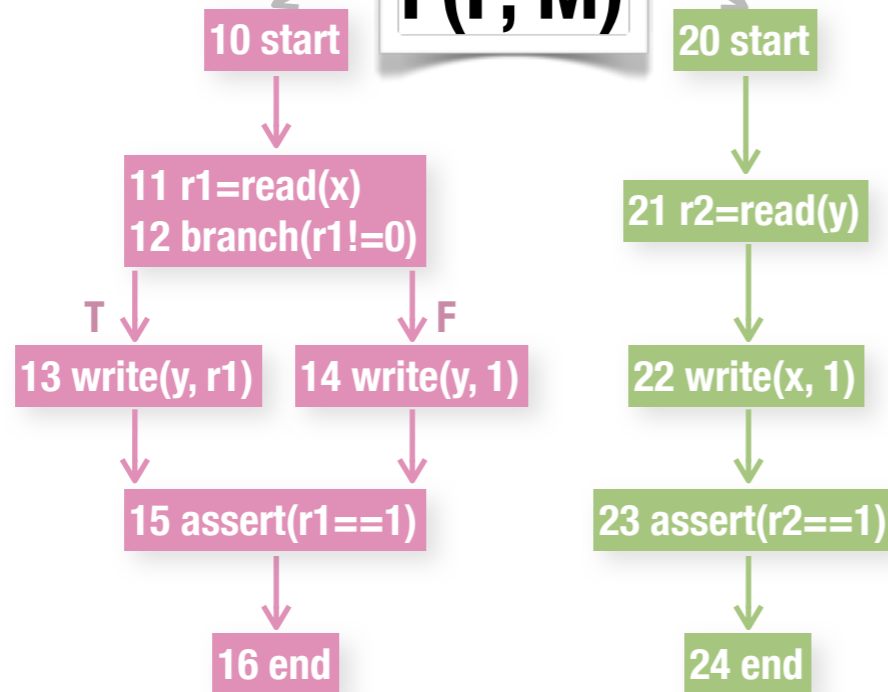


F(P, M)

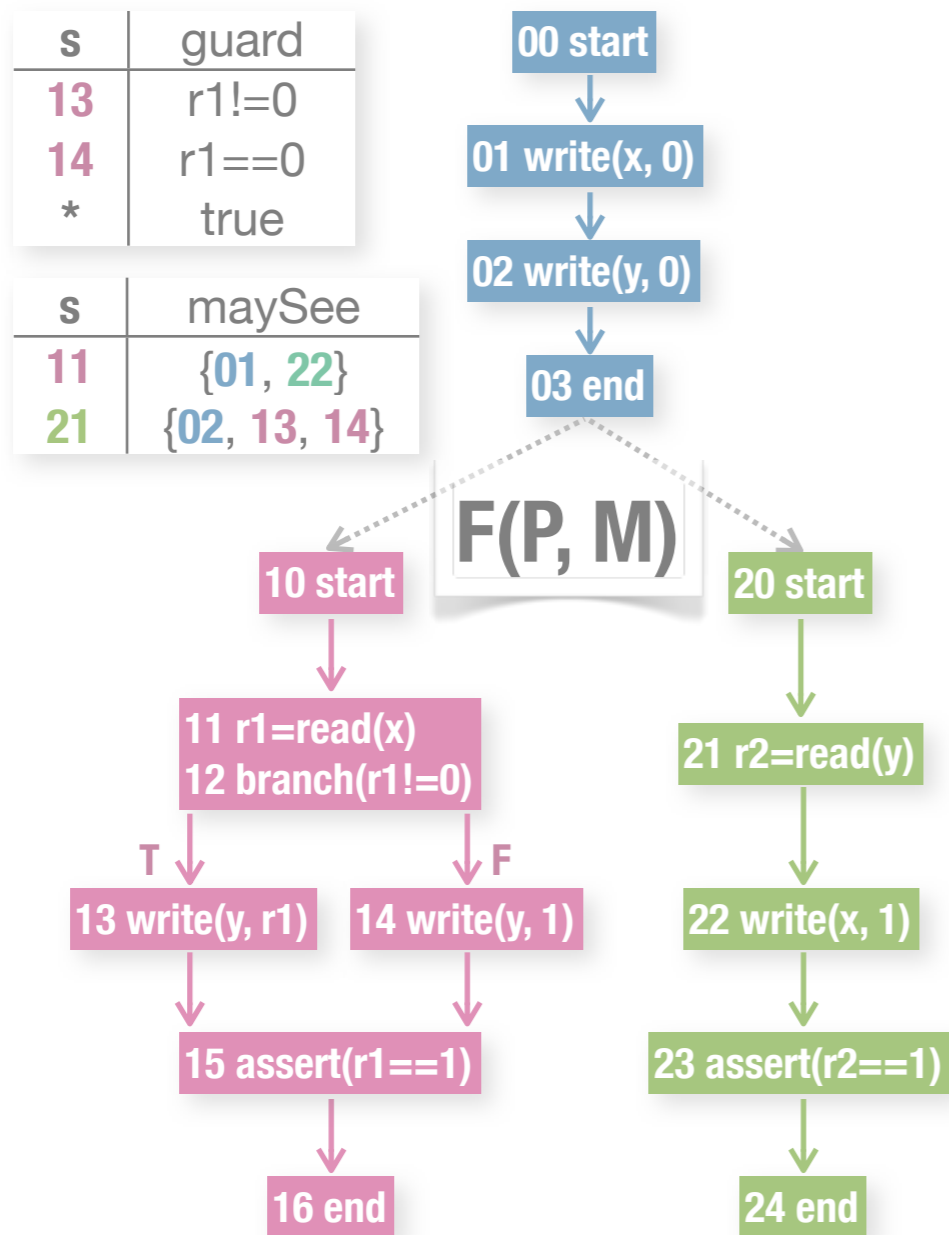
compute a set of bounds on the search space



B(P, M)



Bounds assembly



-8, 1, 2, 4, x, y,
 a00, a01, a02, a03,
 a10, a11, a13, a16,
 a20, a21, a22, a24

$\{\dots\} \subseteq A \subseteq \{\dots\}$

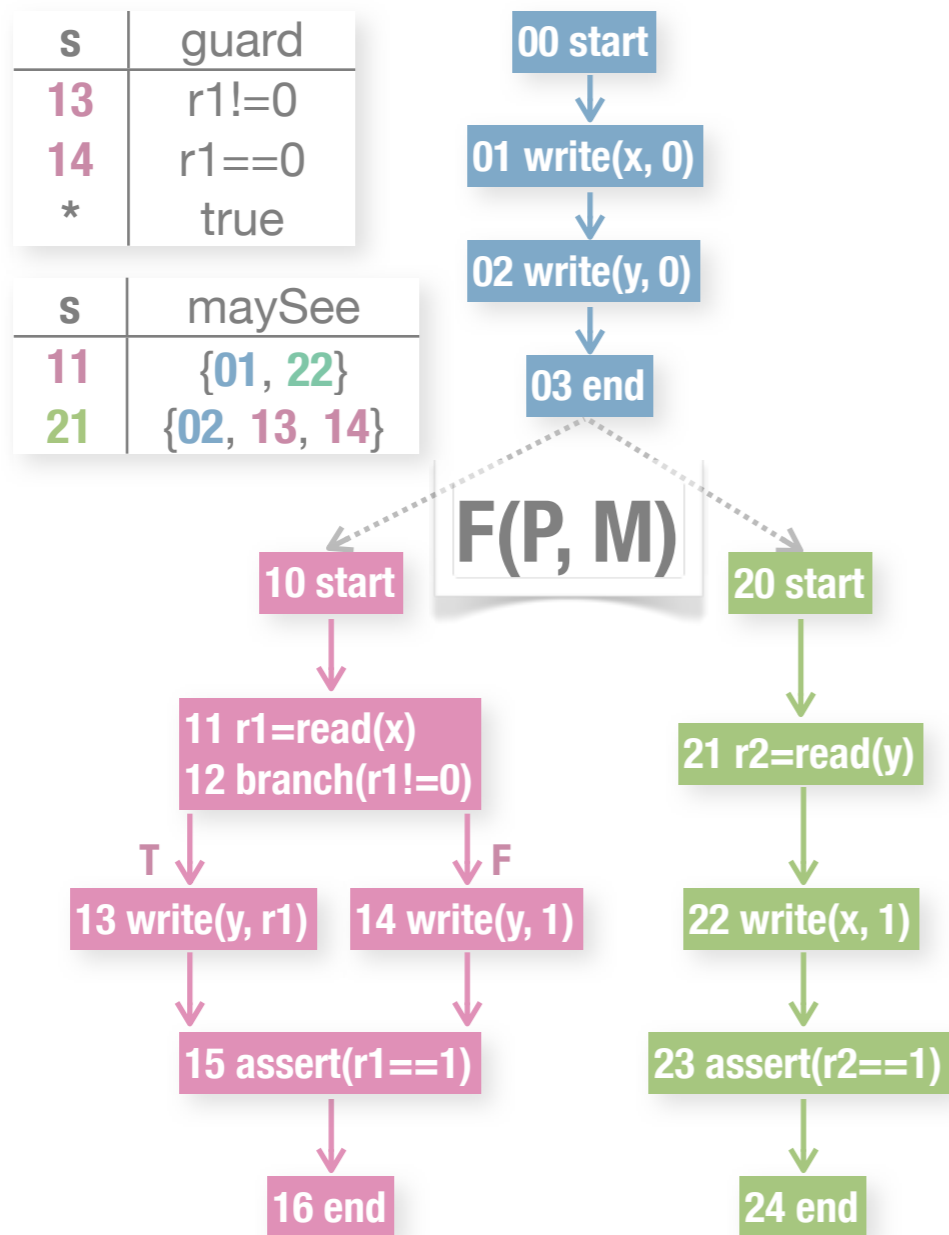
$\{\dots\} \subseteq V \subseteq \{\dots\}$

$\{\dots\} \subseteq W \subseteq \{\dots\}$

$\{\dots\} \subseteq I \subseteq \{\dots\}$

$\{\dots\} \subseteq m \subseteq \{\dots\}$

Bounds assembly: universe



finite universe of symbolic values from which the model, if any, is drawn

-8, 1, 2, 4, x, y,
a00, a01, a02, a03,
a10, a11, a13, a16,
a20, a21, a22, a24

$\{\dots\} \subseteq A \subseteq \{\dots\}$

$\{\dots\} \subseteq V \subseteq \{\dots\}$

$\{\dots\} \subseteq W \subseteq \{\dots\}$

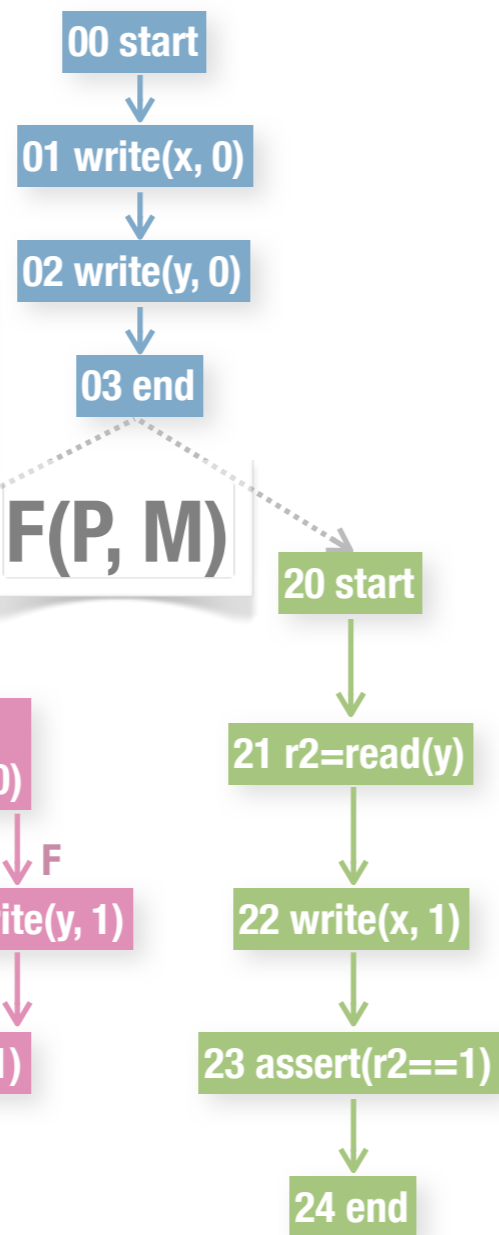
$\{\dots\} \subseteq I \subseteq \{\dots\}$

$\{\dots\} \subseteq m \subseteq \{\dots\}$

Bounds assembly: universe

s	guard
13	r1!=0
14	r1==0
*	true

s	maySee
11	{01, 22}
21	{02, 13, 14}



finite universe of symbolic values from which the model, if any, is drawn

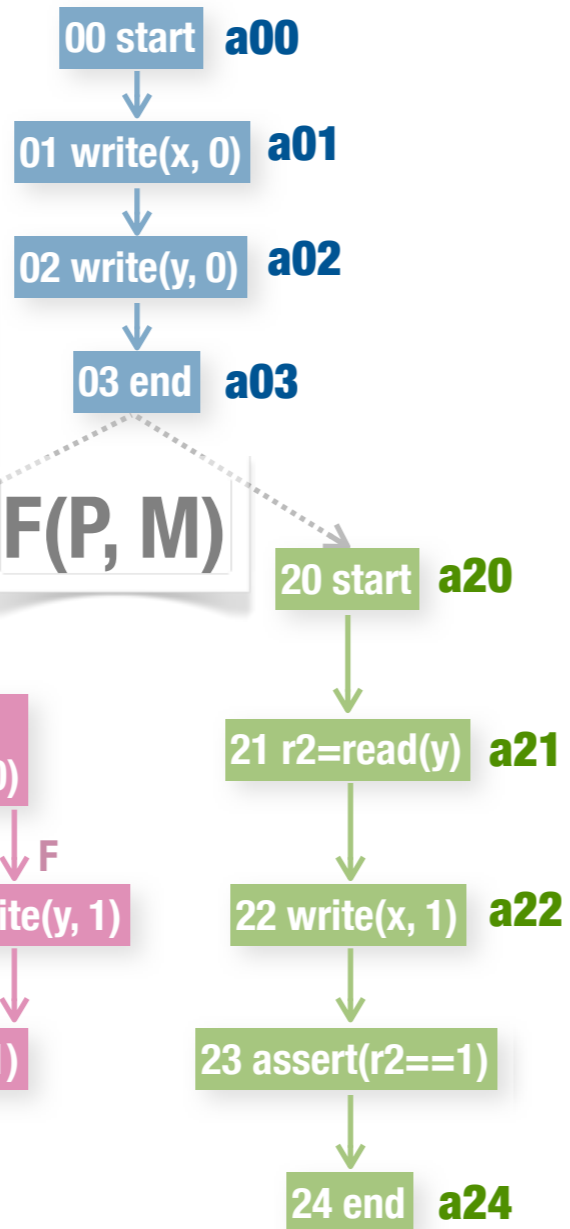
primitives fields

-8, 1, 2, 4, x, y, a00, a01, a02, a03, a10, a11, a13, a16, a20, a21, a22, a24

- {...} ⊆ A ⊆ {...}
- {...} ⊆ V ⊆ {...}
- {...} ⊆ W ⊆ {...}
- {...} ⊆ / ⊆ {...}
- {...} ⊆ m ⊆ {...}

Bounds assembly: universe

s	guard
13	r1!=0
14	r1==0
*	true
s	maySee
11	{01, 22}
21	{02, 13, 14}



F(P, M)

finite universe of symbolic values from which the model, if any, is drawn

primitives fields

-8, 1, 2, 4, x, y, a00, a01, a02, a03, a10, a11, a13, a16, a20, a21, a22, a24

actions

$$\{\dots\} \subseteq A \subseteq \{\dots\}$$

$$\{\dots\} \subseteq V \subseteq \{\dots\}$$

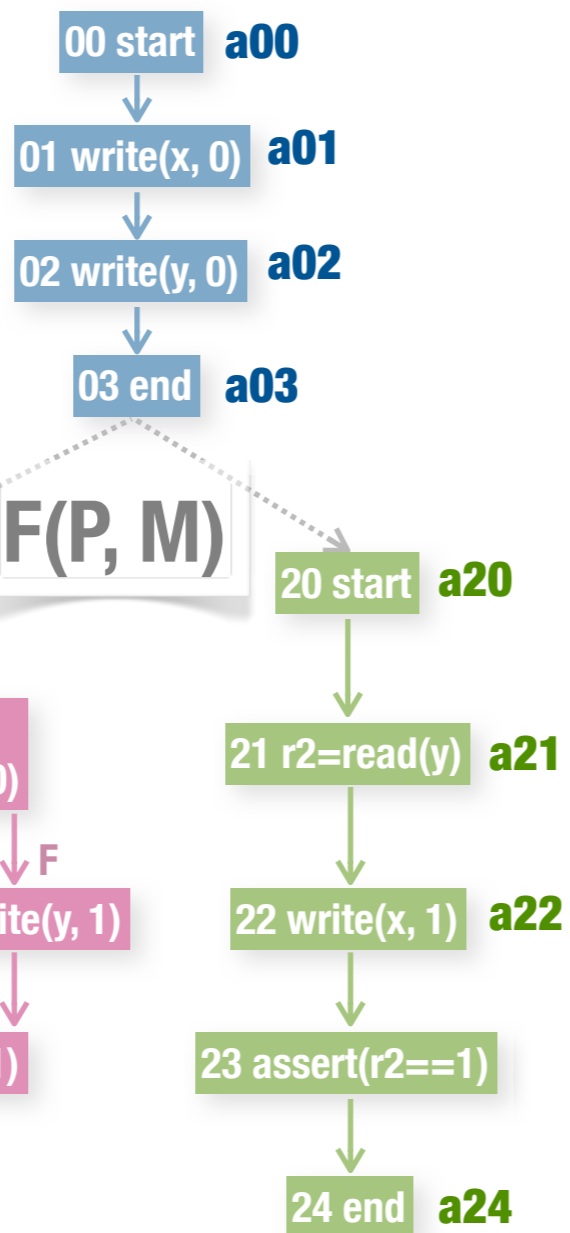
$$\{\dots\} \subseteq W \subseteq \{\dots\}$$

$$\{\dots\} \subseteq I \subseteq \{\dots\}$$

$$\{\dots\} \subseteq m \subseteq \{\dots\}$$

Bounds assembly: lower/upper bounds

s	guard
13	r1!=0
14	r1==0
*	true
s	maySee
11	{01, 22}
21	{02, 13, 14}



-8, 1, 2, 4, x, y,
a00, a01, a02, a03,
a10, a11, a13, a16,
a20, a21, a22, a24

$$\{\dots\} \subseteq A \subseteq \{\dots\}$$

$$\{\dots\} \subseteq V \subseteq \{\dots\}$$

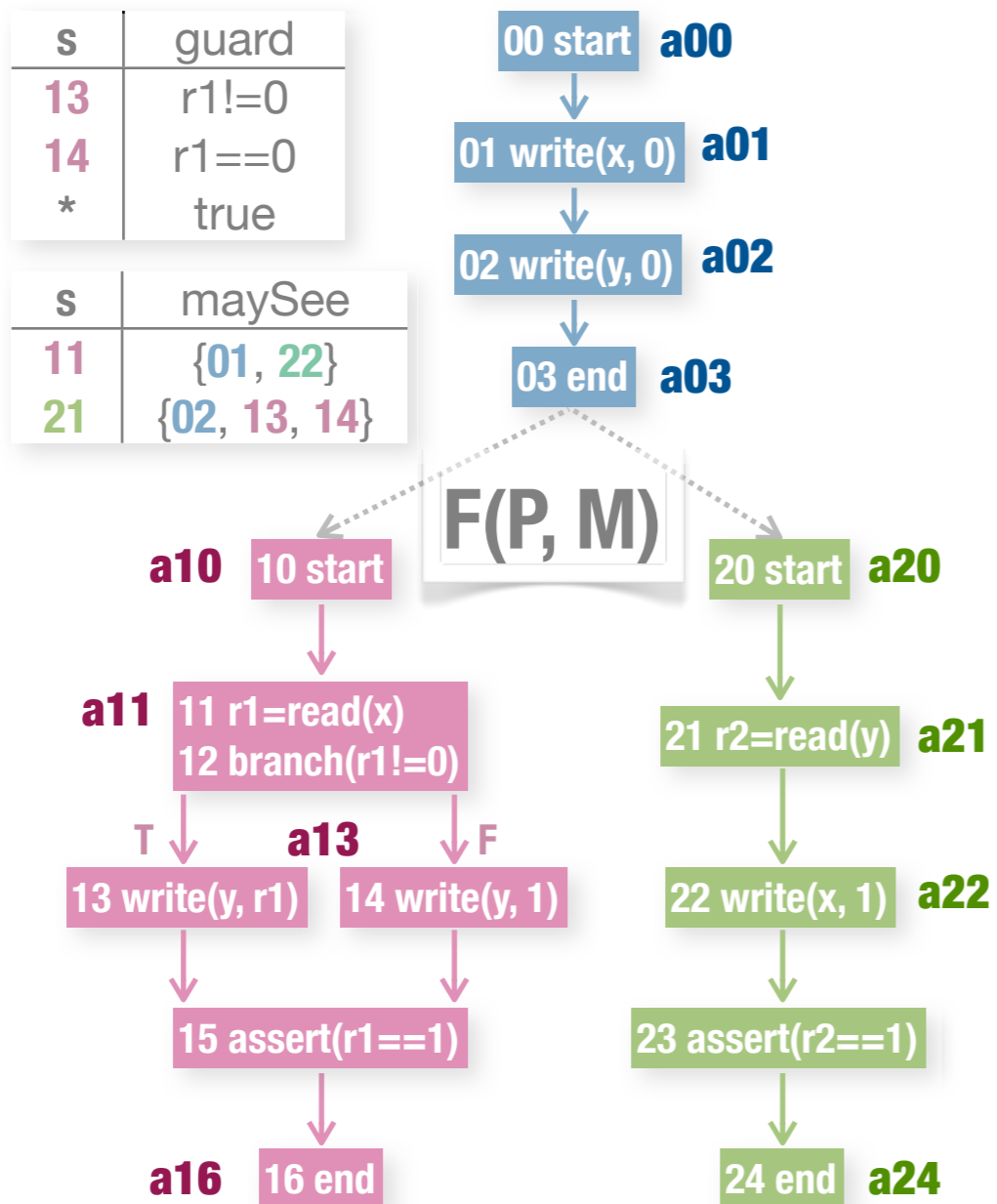
$$\{\dots\} \subseteq W \subseteq \{\dots\}$$

$$\{\dots\} \subseteq I \subseteq \{\dots\}$$

$$\{\dots\} \subseteq m \subseteq \{\dots\}$$

upper and lower bound on the value of each relation that appears in F(P, M)

Bounds assembly: lower/upper bounds



-8, 1, 2, 4, x, y,
a00, a01, a02, a03,
a10, a11, a13, a16,
a20, a21, a22, a24

$\{\dots\} \subseteq A \subseteq \{\dots\}$

$\{\dots\} \subseteq V \subseteq \{\dots\}$

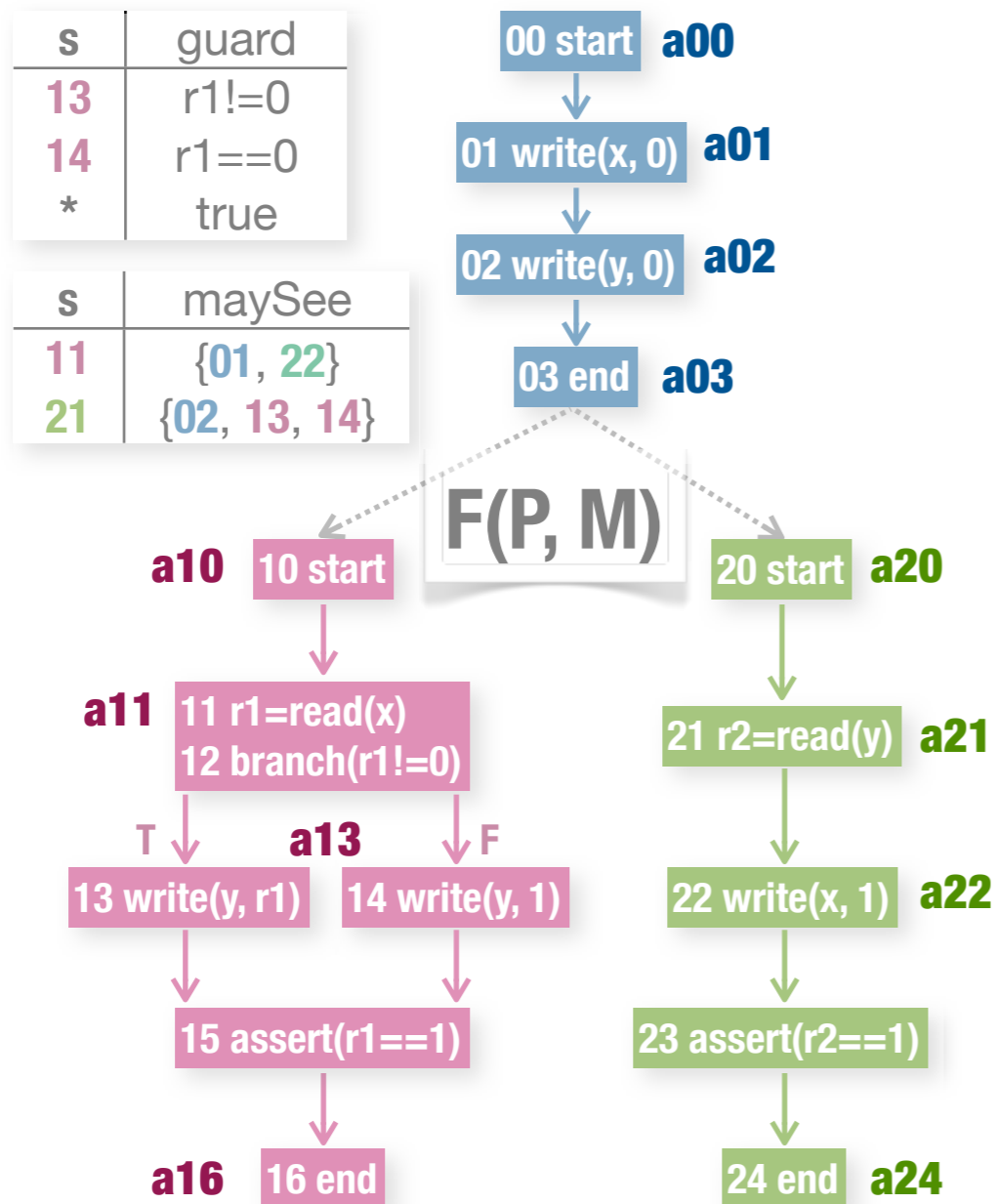
$\{\dots\} \subseteq W \subseteq \{\dots\}$

$\{\dots\} \subseteq / \subseteq \{\dots\}$

$\{\dots\} \subseteq m \subseteq \{\dots\}$

upper and lower bound on the value of each relation that appears in F(P, M)

Bounds assembly: lower/upper bounds



-8, 1, 2, 4, x, y,
a00, a01, a02, a03,
a10, a11, a13, a16,
a20, a21, a22, a24

$$\{ \dots \} \subseteq A \subseteq \left\{ \begin{array}{l} \langle a00 \rangle, \langle a01 \rangle, \langle a02 \rangle, \\ \langle a03 \rangle, \langle a10 \rangle, \langle a11 \rangle, \\ \langle a13 \rangle, \langle a16 \rangle, \langle a20 \rangle, \\ \langle a21 \rangle, \langle a22 \rangle, \langle a24 \rangle \end{array} \right\}$$

$$\{ \dots \} \subseteq V \subseteq \{ \dots \}$$

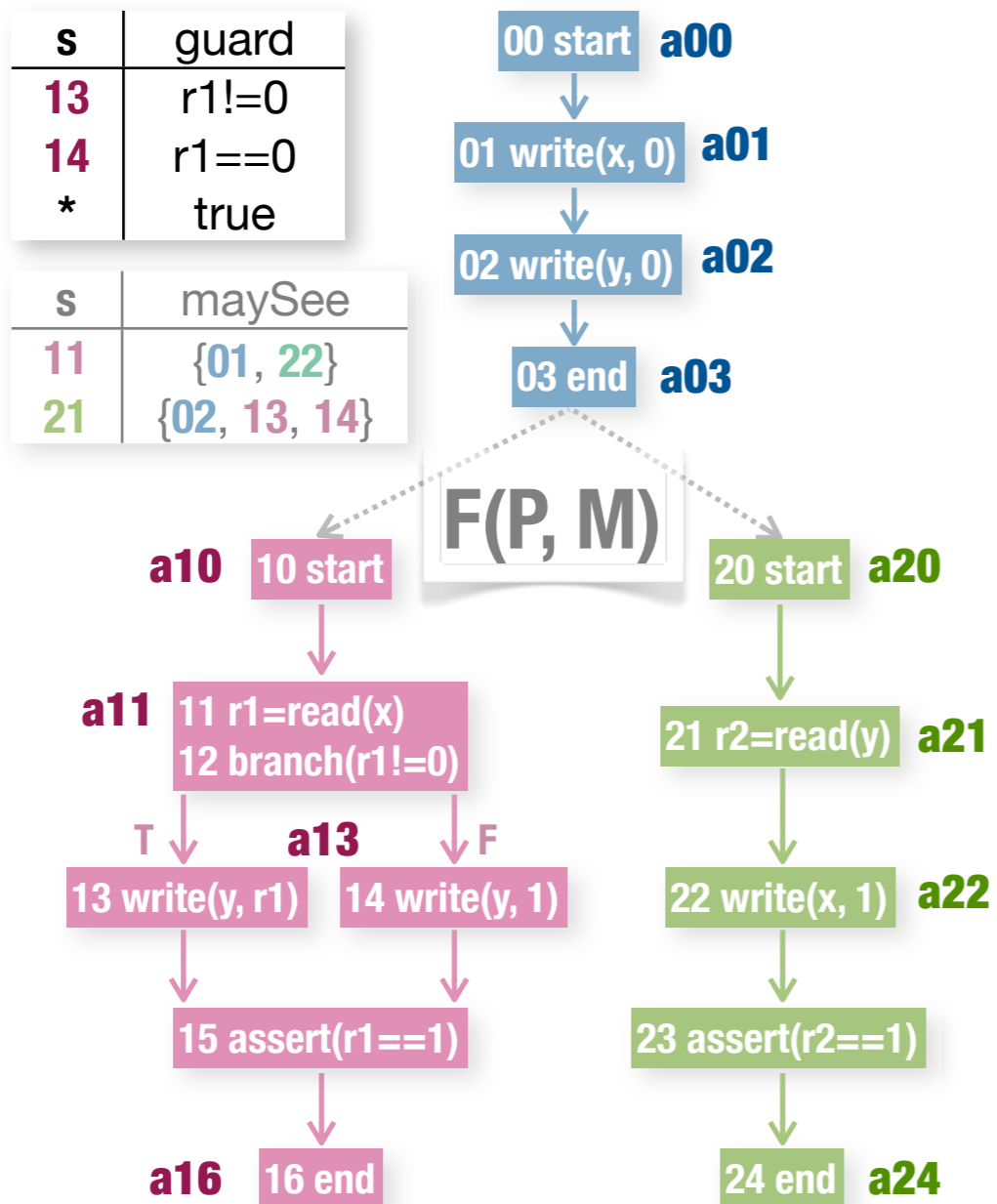
$$\{ \dots \} \subseteq W \subseteq \{ \dots \}$$

$$\{ \dots \} \subseteq / \subseteq \{ \dots \}$$

$$\{ \dots \} \subseteq m \subseteq \{ \dots \}$$

upper and lower bound on the value of each relation that appears in F(P, M)

Bounds assembly: lower/upper bounds



-8, 1, 2, 4, x, y,
a00, a01, a02, a03,
a10, a11, a13, a16,
a20, a21, a22, a24

$$\left\{ \begin{array}{l} \langle a00 \rangle, \langle a01 \rangle, \langle a02 \rangle, \\ \langle a03 \rangle, \langle a10 \rangle, \langle a11 \rangle, \\ \langle a16 \rangle, \langle a20 \rangle, \\ \langle a21 \rangle, \langle a22 \rangle, \langle a24 \rangle \end{array} \right\} \subseteq A \subseteq \left\{ \begin{array}{l} \langle a00 \rangle, \langle a01 \rangle, \langle a02 \rangle, \\ \langle a03 \rangle, \langle a10 \rangle, \langle a11 \rangle, \\ \langle a13 \rangle, \langle a16 \rangle, \langle a20 \rangle, \\ \langle a21 \rangle, \langle a22 \rangle, \langle a24 \rangle \end{array} \right\}$$

$$\{ \dots \} \subseteq V \subseteq \{ \dots \}$$

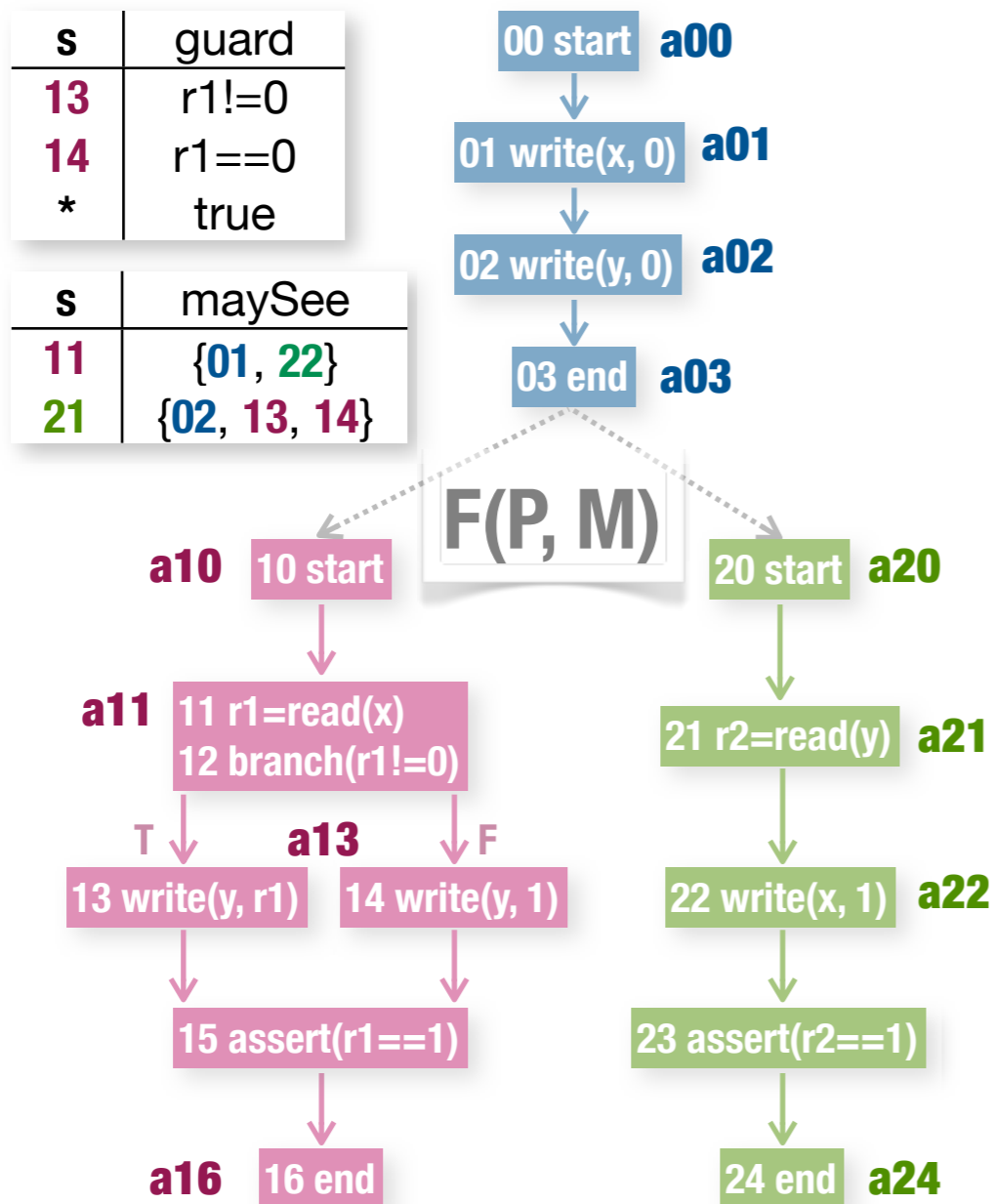
upper and lower bound on the value of each relation that appears in F(P, M)

$$\{ \dots \} \subseteq W \subseteq \{ \dots \}$$

$$\{ \dots \} \subseteq I \subseteq \{ \dots \}$$

$$\{ \dots \} \subseteq m \subseteq \{ \dots \}$$

Bounds assembly: lower/upper bounds



-8, 1, 2, 4, x, y,
a00, a01, a02, a03,
a10, a11, a13, a16,
a20, a21, a22, a24

$$\left\{ \begin{array}{l} \langle a00 \rangle, \langle a01 \rangle, \langle a02 \rangle, \\ \langle a03 \rangle, \langle a10 \rangle, \langle a11 \rangle, \\ \langle a16 \rangle, \langle a20 \rangle, \\ \langle a21 \rangle, \langle a22 \rangle, \langle a24 \rangle \end{array} \right\} \subseteq A \subseteq \left\{ \begin{array}{l} \langle a00 \rangle, \langle a01 \rangle, \langle a02 \rangle, \\ \langle a03 \rangle, \langle a10 \rangle, \langle a11 \rangle, \\ \langle a13 \rangle, \langle a16 \rangle, \langle a20 \rangle, \\ \langle a21 \rangle, \langle a22 \rangle, \langle a24 \rangle \end{array} \right\}$$

$$\{ \dots \} \subseteq V \subseteq \{ \dots \}$$

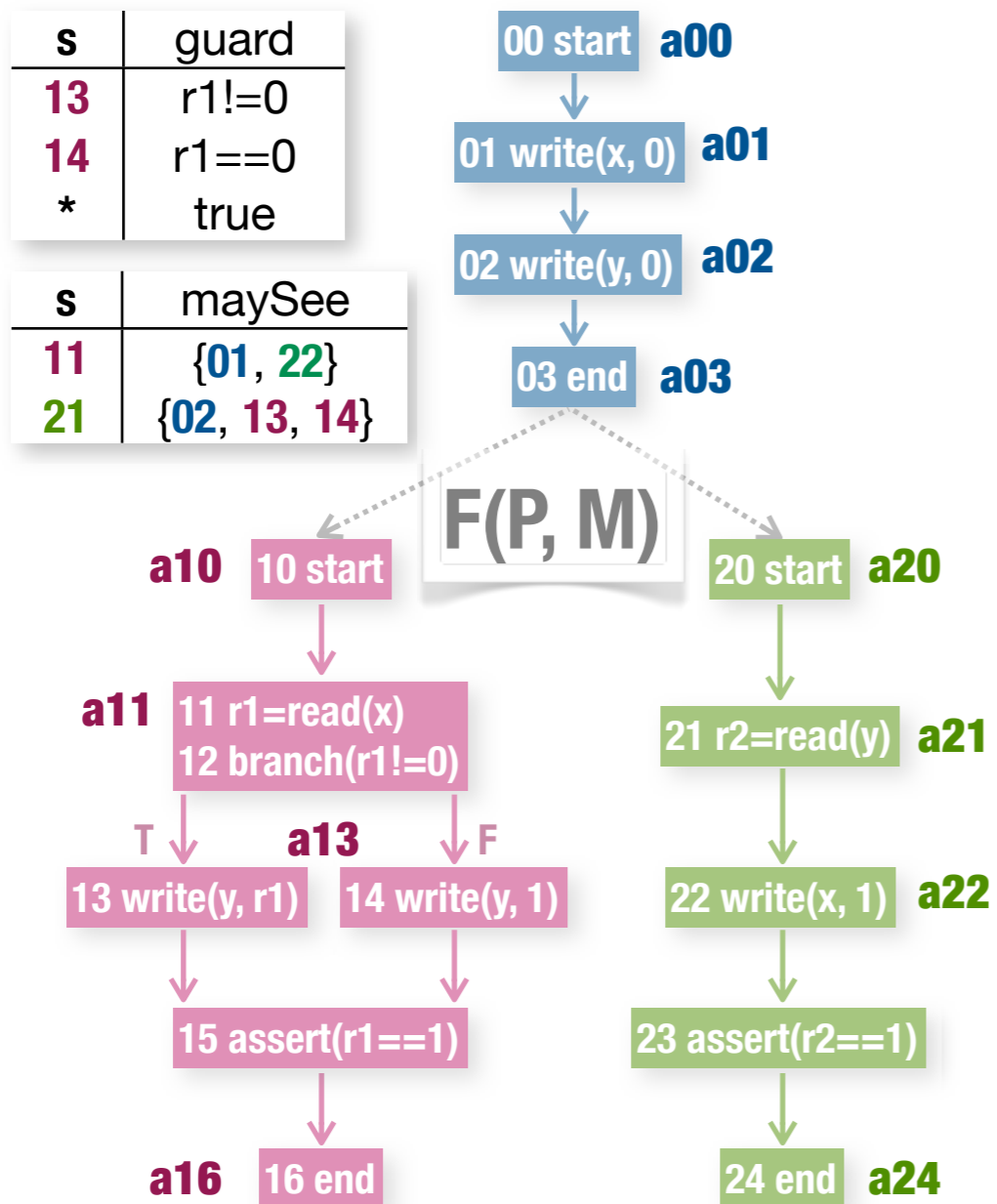
upper and lower bound on the value of each relation that appears in F(P, M)

$$\{ \dots \} \subseteq W \subseteq \left\{ \begin{array}{l} \langle a11, a01 \rangle, \\ \langle a11, a22 \rangle, \\ \langle a21, a02 \rangle, \\ \langle a21, a13 \rangle \end{array} \right\}$$

$$\{ \dots \} \subseteq I \subseteq \{ \dots \}$$

$$\{ \dots \} \subseteq m \subseteq \{ \dots \}$$

Bounds assembly: lower/upper bounds



-8, 1, 2, 4, x, y,
 a00, a01, a02, a03,
 a10, a11, a13, a16,
 a20, a21, a22, a24

$$\left\{ \begin{array}{l} \langle a00 \rangle, \langle a01 \rangle, \langle a02 \rangle, \\ \langle a03 \rangle, \langle a10 \rangle, \langle a11 \rangle, \\ \langle a16 \rangle, \langle a20 \rangle, \\ \langle a21 \rangle, \langle a22 \rangle, \langle a24 \rangle \end{array} \right\} \subseteq A \subseteq \left\{ \begin{array}{l} \langle a00 \rangle, \langle a01 \rangle, \langle a02 \rangle, \\ \langle a03 \rangle, \langle a10 \rangle, \langle a11 \rangle, \\ \langle a13 \rangle, \langle a16 \rangle, \langle a20 \rangle, \\ \langle a21 \rangle, \langle a22 \rangle, \langle a24 \rangle \end{array} \right\}$$

$$\{ \dots \} \subseteq V \subseteq \{ \dots \}$$

upper and lower bound on the value of each relation that appears in F(P, M)

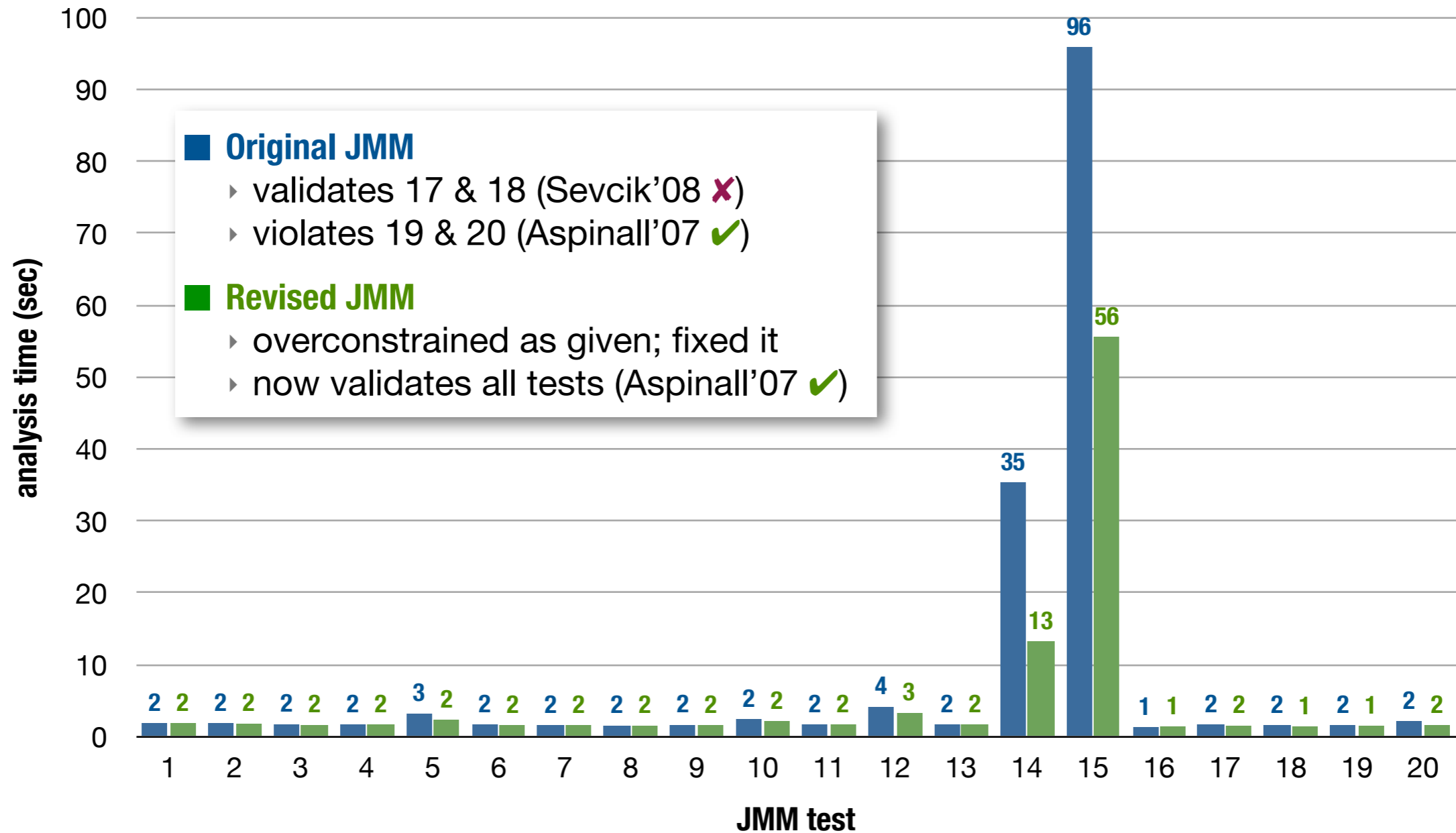
$$\{ \quad \} \subseteq W \subseteq \left\{ \begin{array}{l} \langle a11, a01 \rangle, \\ \langle a11, a22 \rangle, \\ \langle a21, a02 \rangle, \\ \langle a21, a13 \rangle \end{array} \right\}$$

$$\{ \dots \} \subseteq / \subseteq \{ \dots \}$$

$$\{ \dots \} \subseteq m \subseteq \{ \dots \}$$

Results (highlights)

MemSAT performance on JMM causality tests



Conclusion

Practical checker for axiomatic specifications of memory models

- ▶ first tool to directly handle the current JMM
- ▶ first tool to provide minimal cores

Prior work (highlights)

- ▶ CheckFence hardcodes the memory model
- ▶ Nemos accepts simple axiomatic specs but no cores
- ▶ JMM checkers (e.g. OpMM) use operational approximations

Future work

- ▶ extend MemSAT to handle hardware memory models

MemSAT