Propositional Logic V

Zhaoguo Wang

Adapted From:

NYU G22.2390-001, Lecture 3, 5
Stanford CS 357, saturn
教材《数理逻辑与集合论》1.4.2, 2.2.3, 2.5, 2.7~2.10

A Quick Recap – Last Lecture

- Normal Form
- Basic DPLL Algorithm

Outline – This Lecture

- Program Analysis with SAT solvers
- Deduction
- Others

Exercise

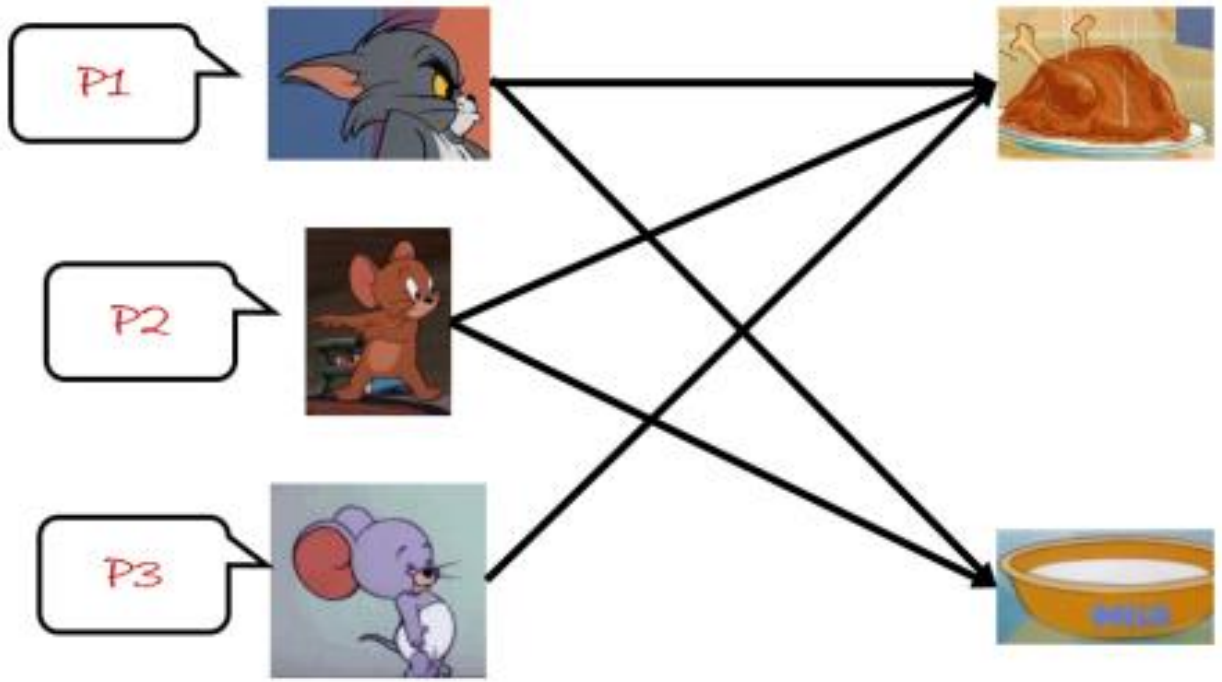
$$1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 2,\ \ 1 \vee 4$$

$$
\begin{array}{rcl}
\emptyset \parallel & 1\vee\overline{2},\ \overline{1}\vee\overline{2},\ 2\vee3,\ \overline{3}\vee2,\ 1\vee4 & \Longrightarrow \quad \text{(PureLiteral)} \\
4 \parallel & 1\vee\overline{2},\ \overline{1}\vee\overline{2},\ 2\vee3,\ \overline{3}\vee2,\ 1\vee4 & \Longrightarrow \quad \text{(Decide)} \\
4\ 1^{\mathsf{d}} \parallel & 1\vee\overline{2},\ \overline{1}\vee\overline{2},\ 2\vee3,\ \overline{3}\vee2,\ 1\vee4 & \Longrightarrow \quad \text{(UnitProp)} \\
4\ 1^{\mathsf{d}}\ \overline{2} \parallel & 1\vee\overline{2},\ \overline{1}\vee\overline{2},\ 2\vee3,\ \overline{3}\vee2,\ 1\vee4 & \Longrightarrow \quad \text{(UnitProp)} \\
4\ 1^{\mathsf{d}}\ \overline{2}\ 3 \parallel & 1\vee\overline{2},\ \overline{1}\vee\overline{2},\ 2\vee3,\ \overline{3}\vee2,\ 1\vee4 & \Longrightarrow \quad \text{(Backtrack)} \\
4\ \overline{1} \parallel & 1\vee\overline{2},\ \overline{1}\vee\overline{2},\ 2\vee3,\ \overline{3}\vee2,\ 1\vee4 & \Longrightarrow \quad \text{(UnitProp)} \\
4\ \overline{1}\ \overline{2}\ \overline{3} \parallel & 1\vee\overline{2},\ \overline{1}\vee\overline{2},\ 2\vee3,\ \overline{3}\vee2,\ 1\vee4 & \Longrightarrow \quad \text{(Fail)} \\
\mathit{fail} &&
\end{array}
$$

GSAT vs. DPLL

- GSAT is sound
  - If GSAT gives an assignment, it must satisfy the formula
- GSAT is not complete
  - If the formula is satisfiable, sometimes GSAT cannot give an assignment
- You cannot use GSAT to generate all satisfying assignments
- For a while, GSAT was beating DPLL in SAT contests, but now the DPLL people are tuning up their heuristics and doing better

UNSAT



$$P3 \wedge \neg(P1 \wedge P2) \wedge \neg(P2 \wedge P3) \wedge \neg(P1 \wedge P3) \wedge \neg(\neg P1 \wedge \neg P2)$$

$$= P3 \wedge (\neg P1 \vee \neg P2) \wedge (\neg P2 \vee \neg P3) \wedge (\neg P1 \vee \neg P3) \wedge (P1 \vee P2)$$

UNSAT

$$\emptyset \parallel \boxed{3,}\ \overline{1} \vee \overline{2},\ \overline{2} \vee \overline{3},\ \overline{1} \vee \overline{3},\ 1 \vee 2 \implies (UnitProp)$$

$$3 \parallel 3,\ \overline{1} \vee \overline{2},\ \boxed{\overline{2} \vee \overline{3}}\ \overline{1} \vee \overline{3},\ 1 \vee 2 \implies (UnitProp)$$

$$3\ \overline{2} \parallel 3,\ \overline{1} \vee \overline{2},\ \overline{2} \vee \overline{3},\ \boxed{\overline{1} \vee \overline{3}}\ 1 \vee 2 \implies (UnitProp)$$

$$3\ \overline{2}\ \overline{1} \parallel 3,\ \overline{1} \vee \overline{2},\ \overline{2} \vee \overline{3},\ \overline{1} \vee \overline{3},\ \boxed{1 \vee 2} \implies (Fail)$$

$$fail$$

UNSAT

$$\emptyset \parallel \boxed{3,}\ \overline{1} \vee \overline{2},\ \overline{2} \vee \overline{3},\ \overline{1} \vee \overline{3},\ 1 \vee 2 \implies (UnitProp)$$

$$3 \parallel 3,\ \overline{1} \vee \overline{2},\ \boxed{\overline{2} \vee \overline{3}}\ \overline{1} \vee \overline{3},\ 1 \vee 2 \implies (UnitProp)$$

$$3\ \overline{2} \parallel 3,\ \overline{1} \vee \overline{2},\ \overline{2} \vee \overline{3},\ \boxed{\overline{1} \vee \overline{3}}\ 1 \vee 2 \implies (UnitProp)$$

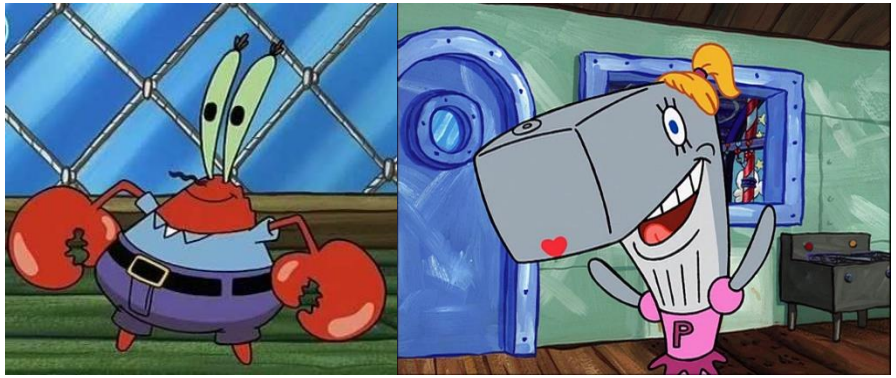
$$3\ \overline{2}\ \overline{1} \parallel 3,\ \overline{1} \vee \overline{2},\ \overline{2} \vee \overline{3},\ \overline{1} \vee \overline{3},\ \boxed{1 \vee 2} \implies (Fail)$$

$fail$

Can we use GSAT to prove the formula is unsatisfiable?

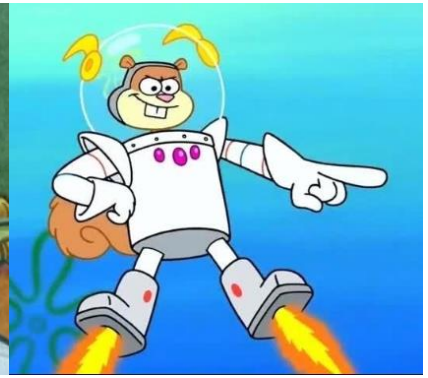
All in one

Inviting one of every group to have a meeting



However, there are some requirements

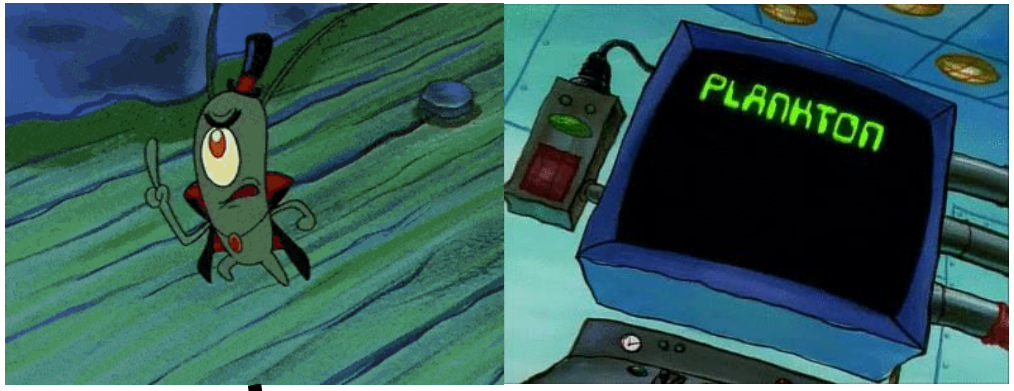
They cannot attend together.

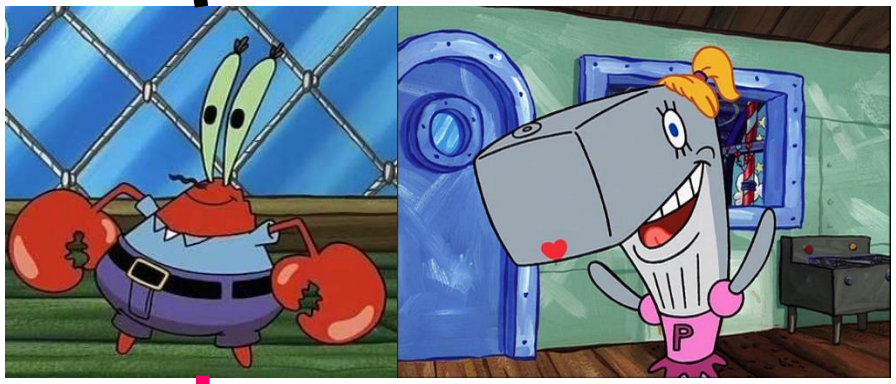


However, there are some requirements

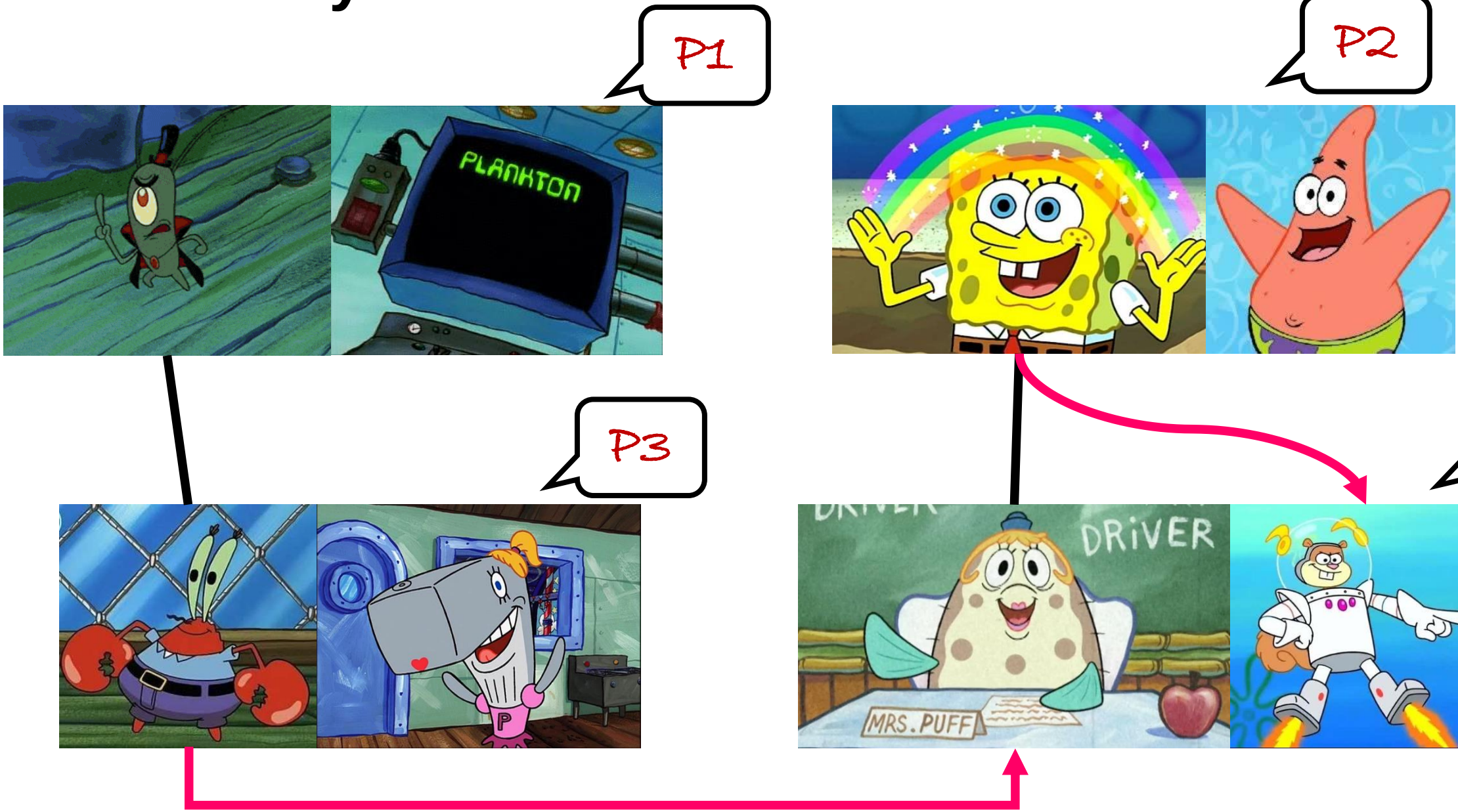
If Sandy attends, SpongeBob must attend.

Who should you invite?

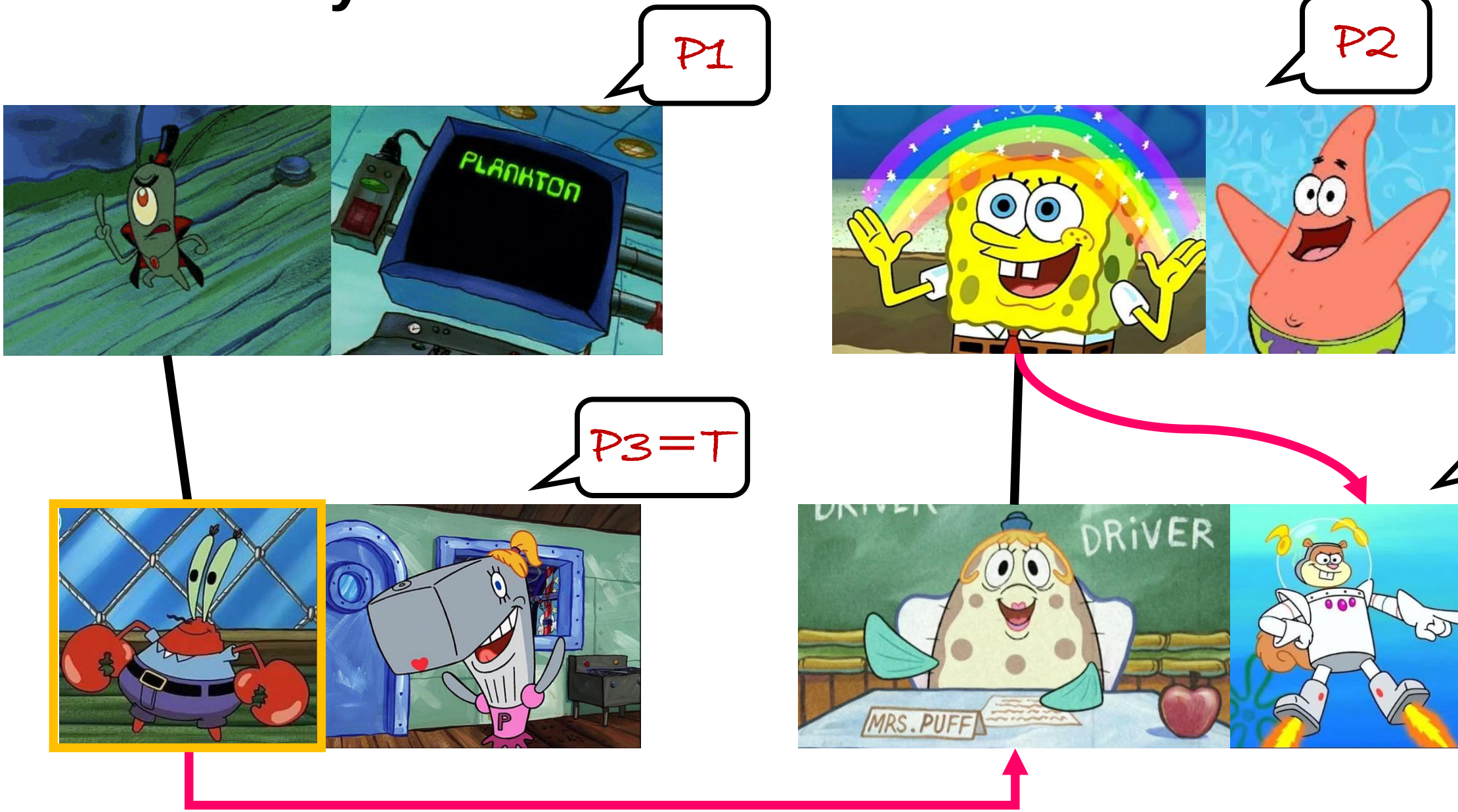




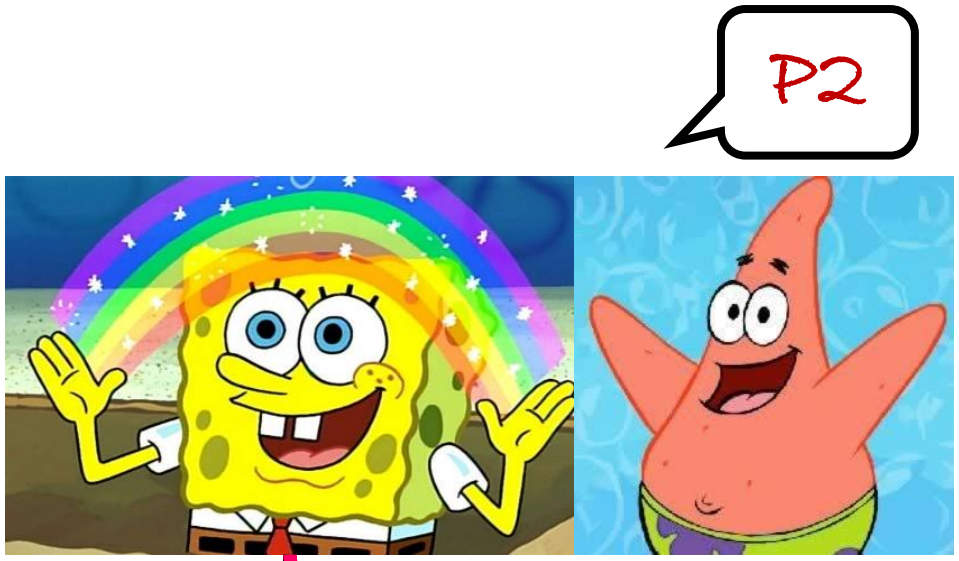
Who should you invite?

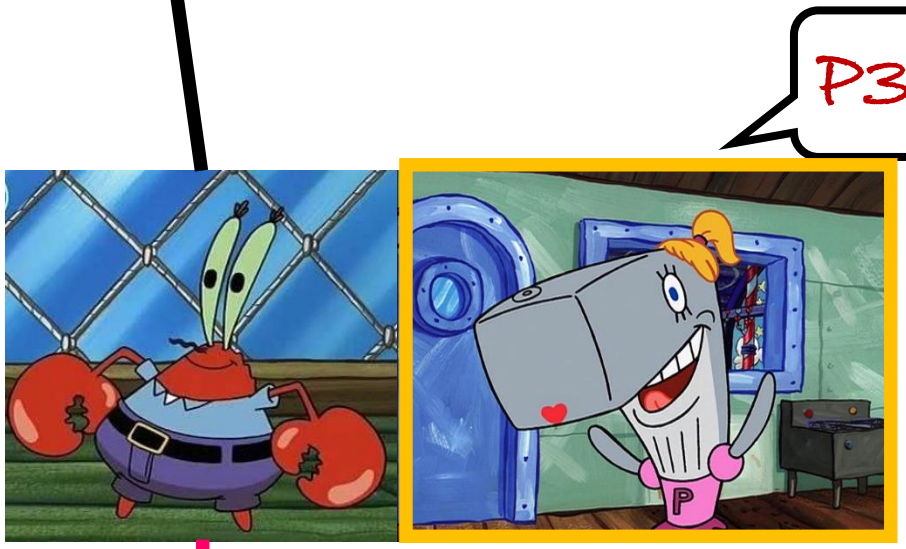


Who should you invite?



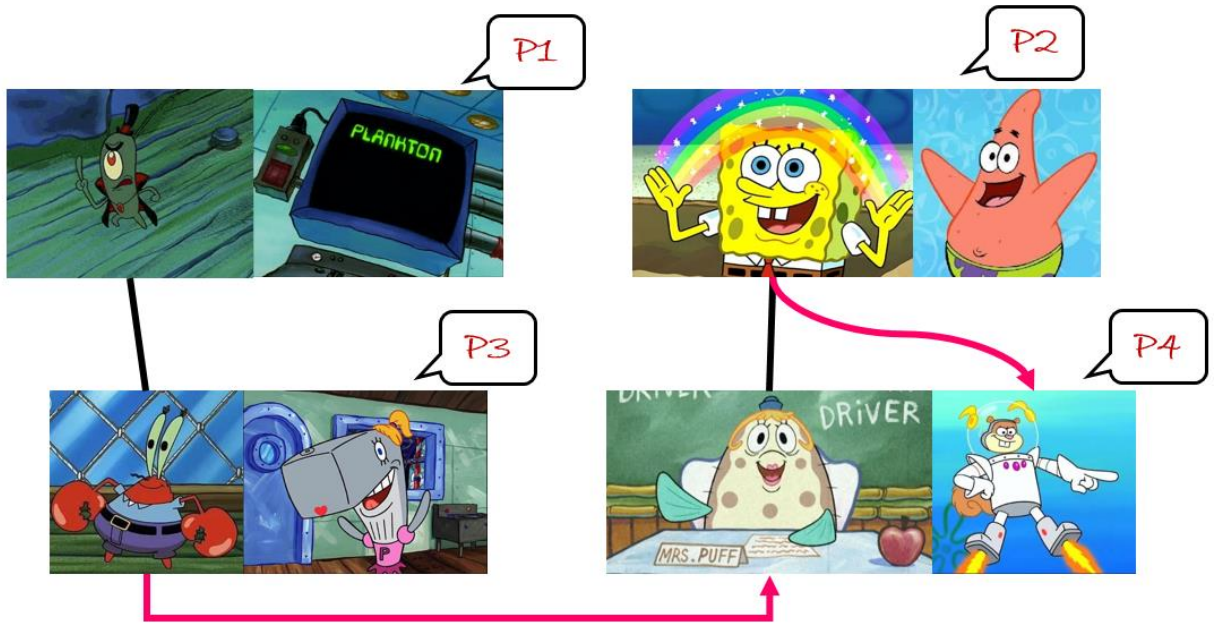
Who should you invite?





Formalize the problem

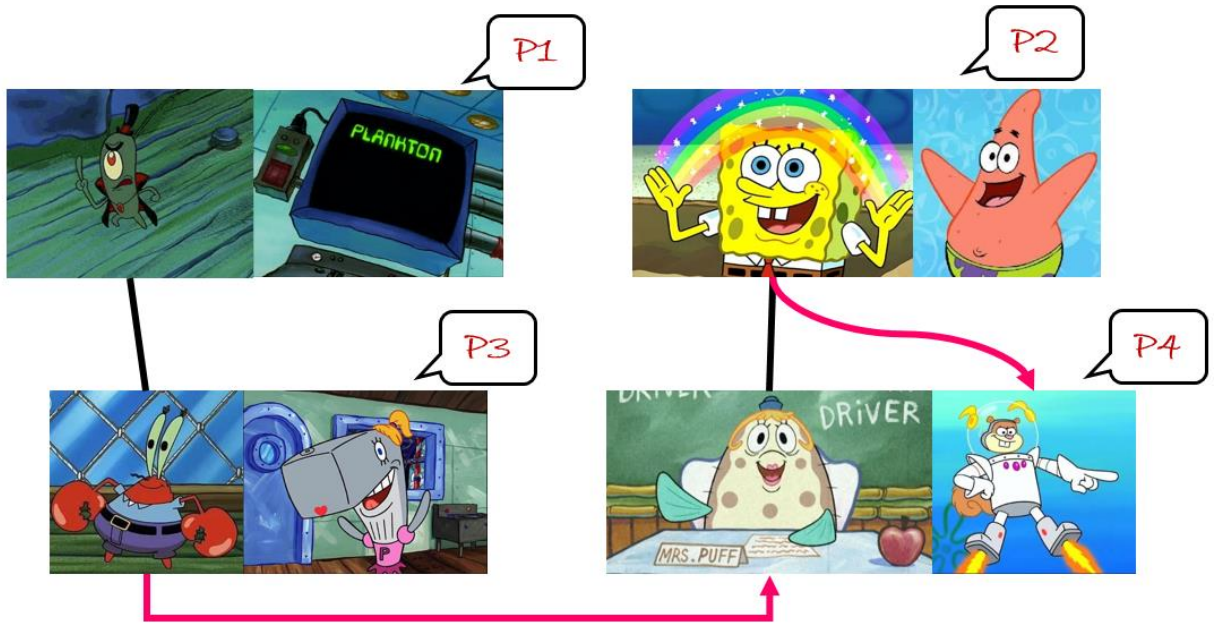
Krabs and Plankton cannot attend together.



$$\neg(P1 \wedge P3) \wedge$$
$$\neg(P2 \wedge P4) \wedge$$
$$(\neg P4 \rightarrow P2) \wedge$$
$$(P4 \rightarrow P3)$$

Formalize the problem

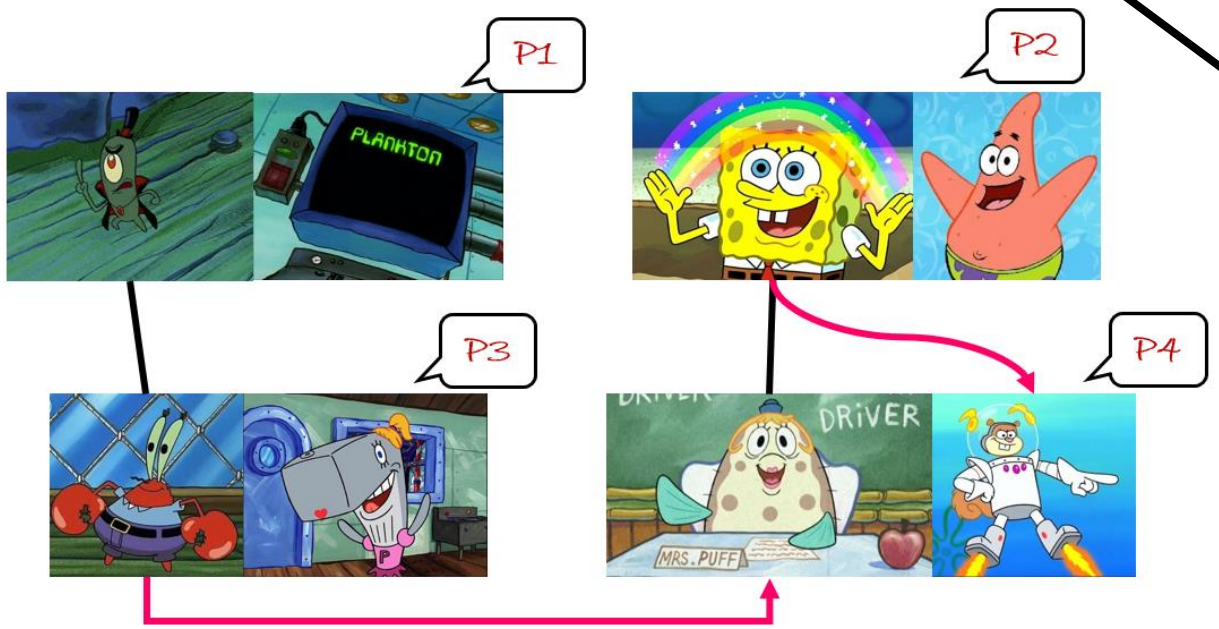
SpongeBob and Mrs. Puff cannot attend together.



$$\neg(P1 \wedge P3) \wedge$$
$$\neg(P2 \wedge P4) \wedge$$
$$(\neg P4 \rightarrow P2) \wedge$$
$$(P4 \rightarrow P3)$$

Formalize the problem

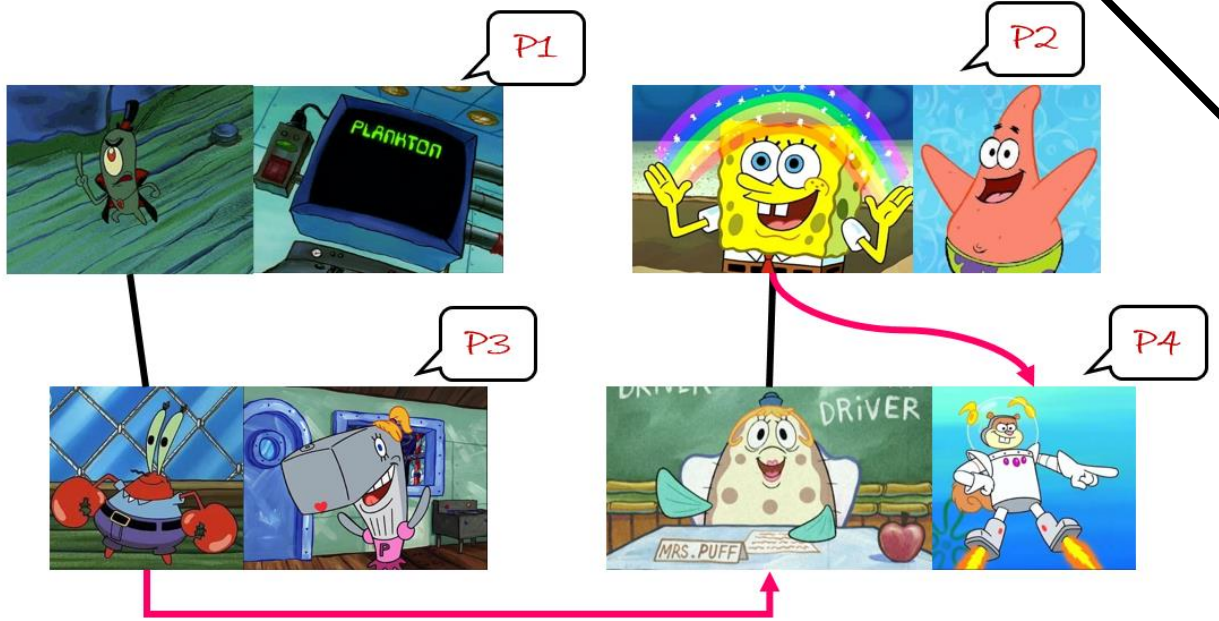
If Sandy attends,
SpongeBob must attend.



$$\neg(P1 \wedge P3) \wedge$$
$$\neg(P2 \wedge P4) \wedge$$
$$(\neg P4 \rightarrow P2) \wedge$$
$$(P4 \rightarrow P3)$$

Formalize the problem

If Mrs. Puff attends, Krabs must attend.



$$\neg(P1 \wedge P3) \wedge$$
$$\neg(P2 \wedge P4) \wedge$$
$$(\neg P4 \rightarrow P2) \wedge$$
$$(P4 \rightarrow P3)$$

Determine if it is a wff

$$\neg(P1 \wedge P3) \wedge$$
$$\neg(P2 \wedge P4) \wedge$$
$$(\neg P4 \rightarrow P2) \wedge$$
$$(P4 \rightarrow P3)$$

Convert it to CNF

$$\neg(P1 \wedge P3) \wedge \neg(P2 \wedge P4) \wedge (\neg P4 \rightarrow P2) \wedge (P4 \rightarrow P3)$$

$$=(\neg P1 \vee \neg P3) \wedge (\neg P2 \vee \neg P4) \wedge (\neg\neg P4 \vee P2) \wedge (\neg P4 \vee P3)$$

$$=(\neg P1 \vee \neg P3) \wedge (\neg P2 \vee \neg P4) \wedge (P4 \vee P2) \wedge (\neg P4 \vee P3)$$

Solve it with DPLL

$$\begin{aligned}
\emptyset \parallel\ & \overline{1} \vee \overline{3},\ \overline{2} \vee \overline{4},\ 2 \vee 4,\ 3 \vee \overline{4} && \Longrightarrow (PureLiteral)\\
\overline{1} \parallel\ & \overline{1} \vee \overline{3},\ \overline{2} \vee \overline{4},\ 2 \vee 4,\ 3 \vee \overline{4} && \Longrightarrow (Decide)\\
\overline{1}\ 3^d \parallel\ & \overline{1} \vee \overline{3},\ \overline{2} \vee \overline{4},\ 2 \vee 4,\ 3 \vee \overline{4} && \Longrightarrow (Decide)\\
\overline{1}\ 3^d\ 2^d \parallel\ & \overline{1} \vee \overline{3},\ \overline{2} \vee \overline{4},\ 2 \vee 4,\ 3 \vee \overline{4} && \Longrightarrow (UnitProp)\\
\overline{1}\ 3^d\ 2^d\ \overline{4} \parallel\ & \overline{1} \vee \overline{3},\ \overline{2} \vee \overline{4},\ 2 \vee 4,\ 3 \vee \overline{4}
\end{aligned}$$

Program Analysis with SAT solvers

truth table | propositional equivalence | SAT | program analysis

Program Analysis with SAT solvers

What?

- SAT-based approach to program analysis

How?

- Program constructs → propositional logic constraints
- Inference → SAT solving

Why SAT?

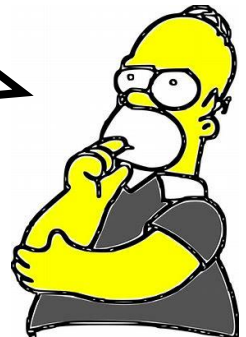
- Program states naturally expressed as bits
- The theory for bits is SAT
- Efficient SAT solvers available

Straight-Line Code

```
void swap(int a, int b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```

We want to prove that the program really swaps the value of a and b.

How to convert the program construct to a propositional logic formula?

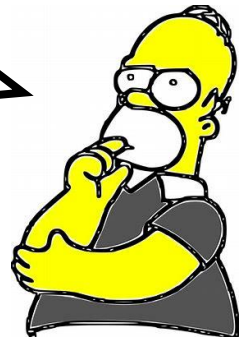


Straight-Line Code

```
void swap(int a, int b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```

We want to prove that the program really swaps the value of a and b.

Firstly we should represent the value of a and b in propositional logic.

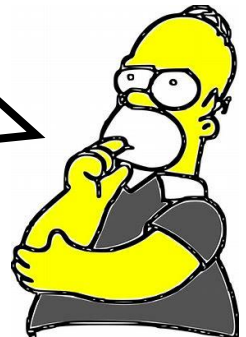


Straight-Line Code

```
void swap(int a, int b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```

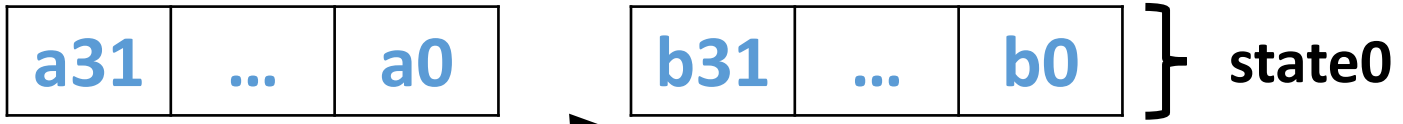
We want to prove that the program really swaps the value of a and b.

In computer, an integer is composed of 32 bits. Every bit is 1 or 0. A bit can be naturally represented by a propositional variable.



Straight-Line Code

```
void swap(int a, int b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```

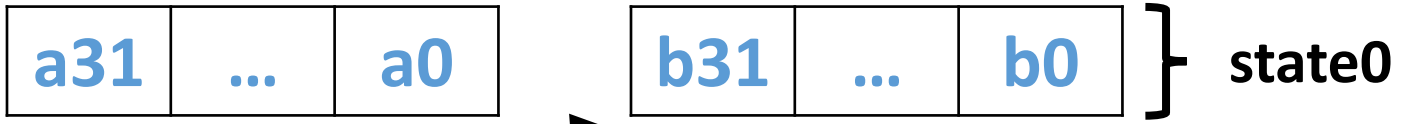


The truth values of 64 variables construct the program state.

How to represent operators in the program?

Straight-Line Code

```
void swap(int a, int b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```

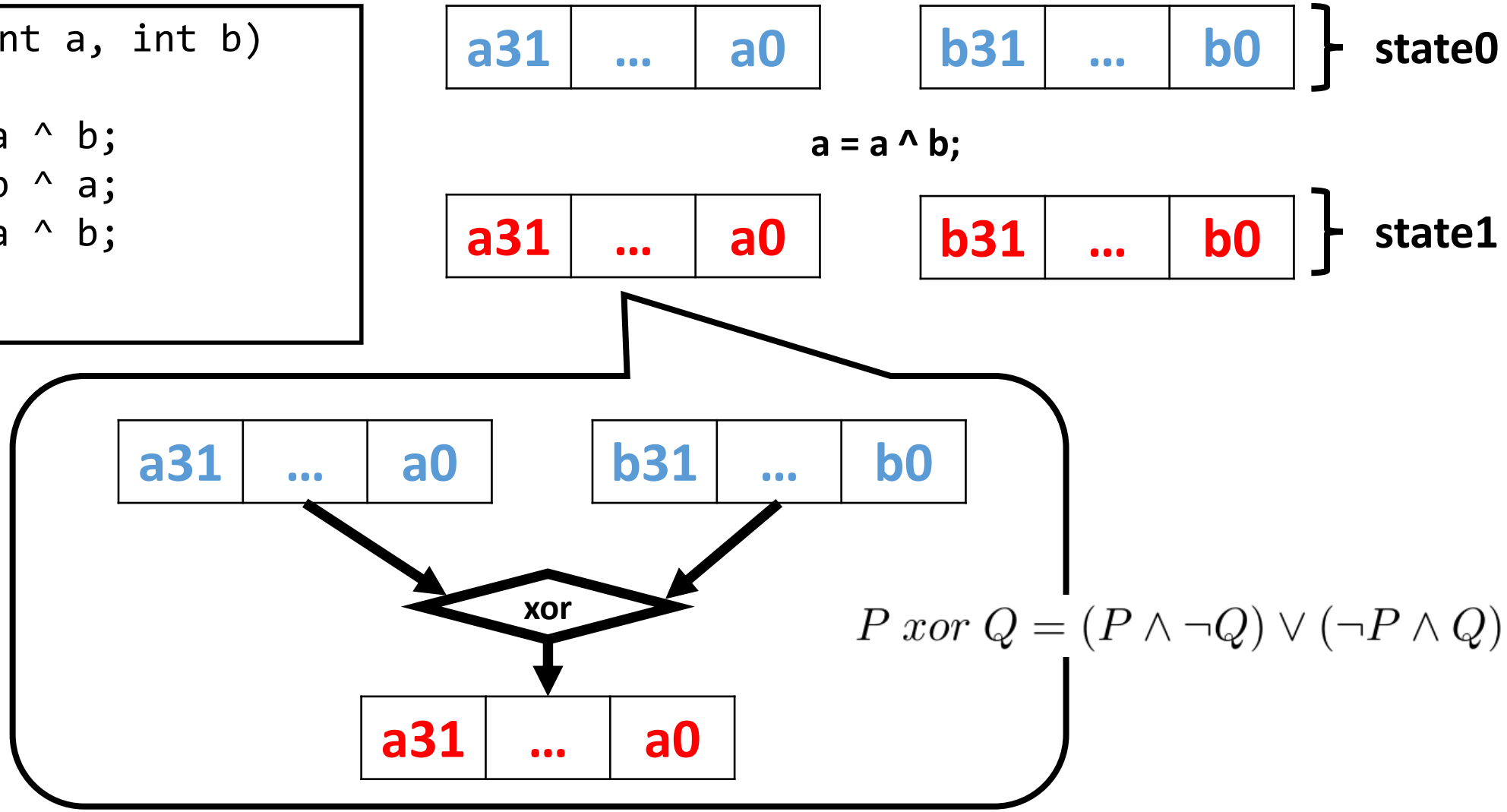


The truth values of 64 variables construct the program state.

Operators can be naturally represented by connectives.

Straight-Line Code

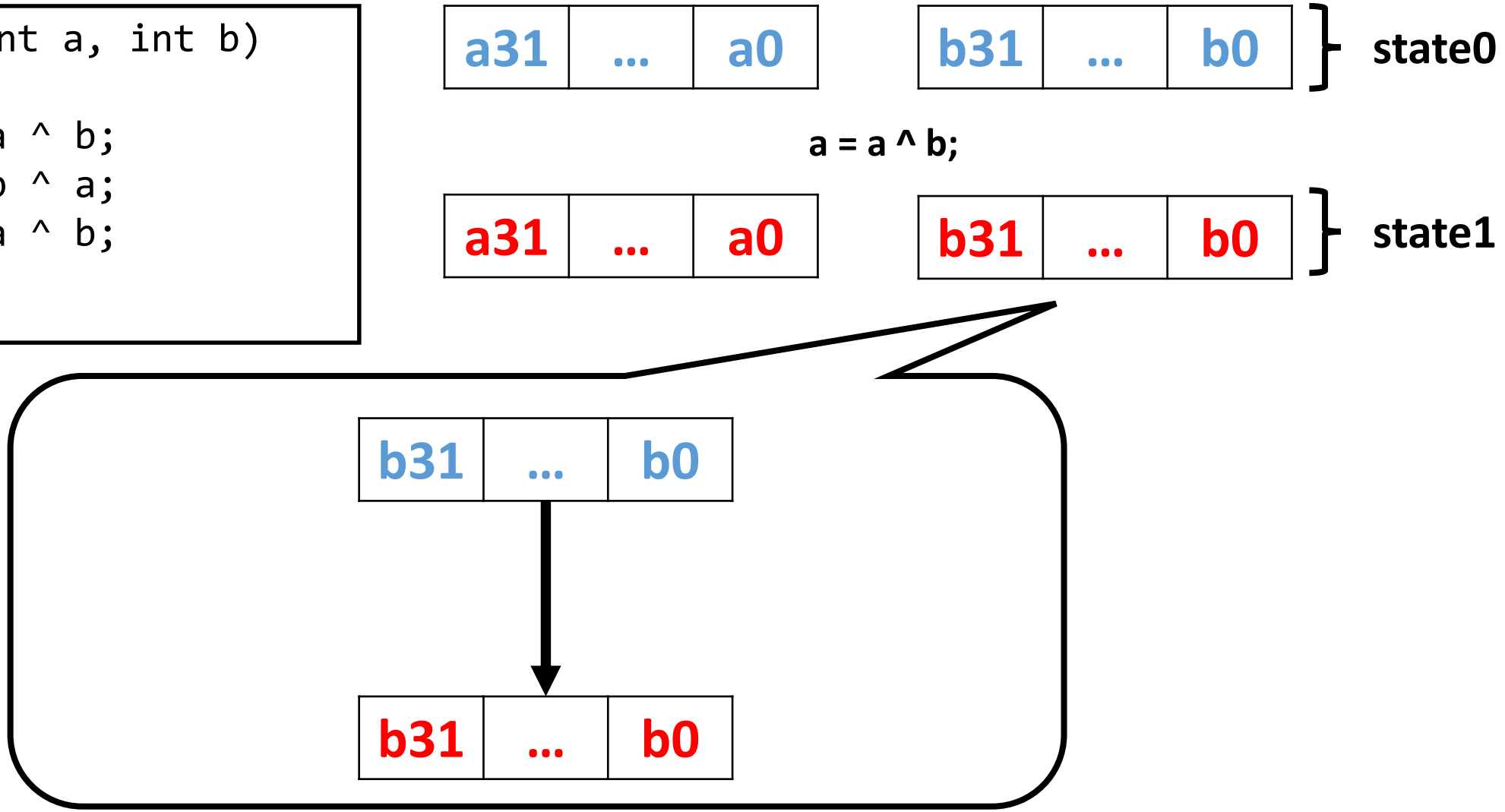
```
void swap(int a, int b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```



$$P\ xor\ Q = (P \wedge \neg Q) \vee (\neg P \wedge Q)$$

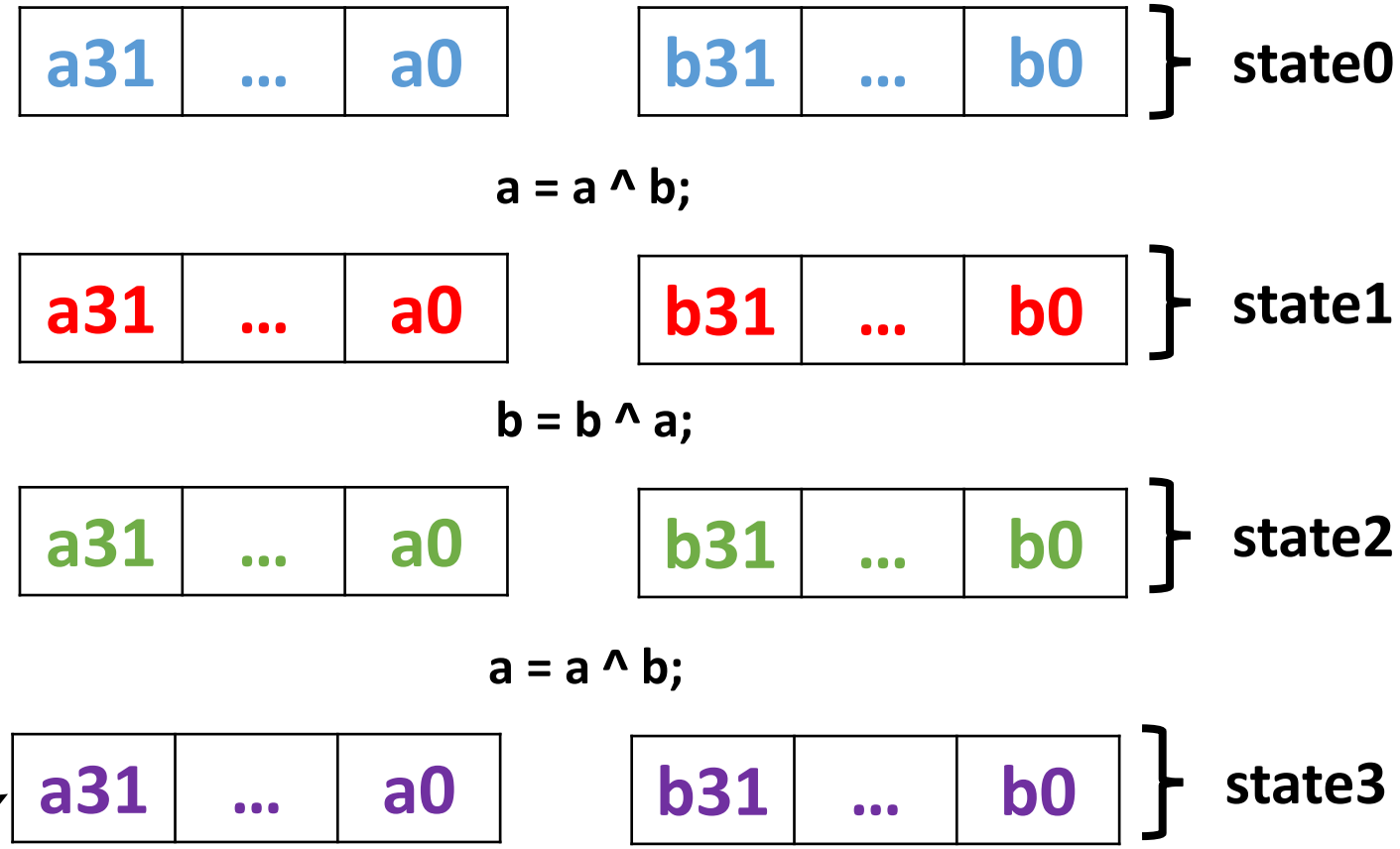
Straight-Line Code

```
void swap(int a, int b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```



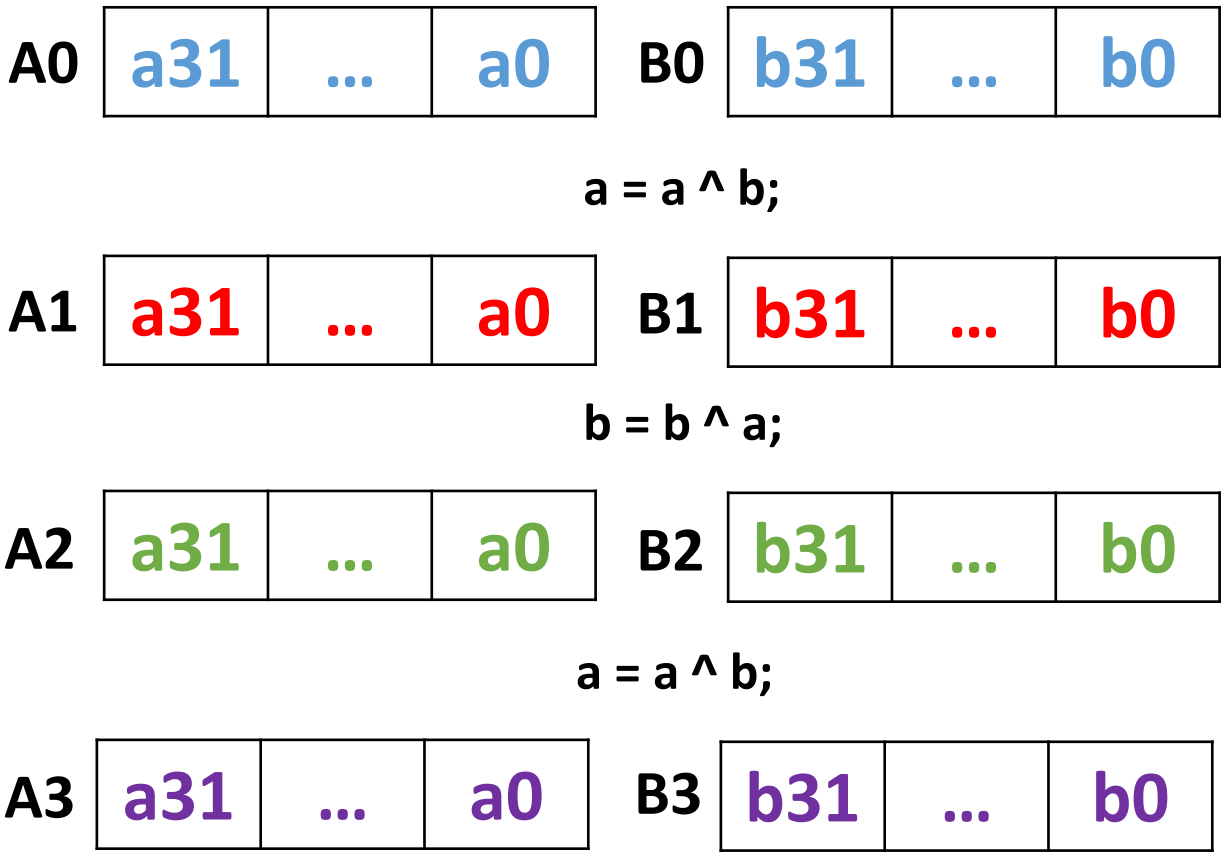
Straight-Line Code

```
void swap(int a, int b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```



Final a and b can be represented by formulas composed of a and b in previous states.

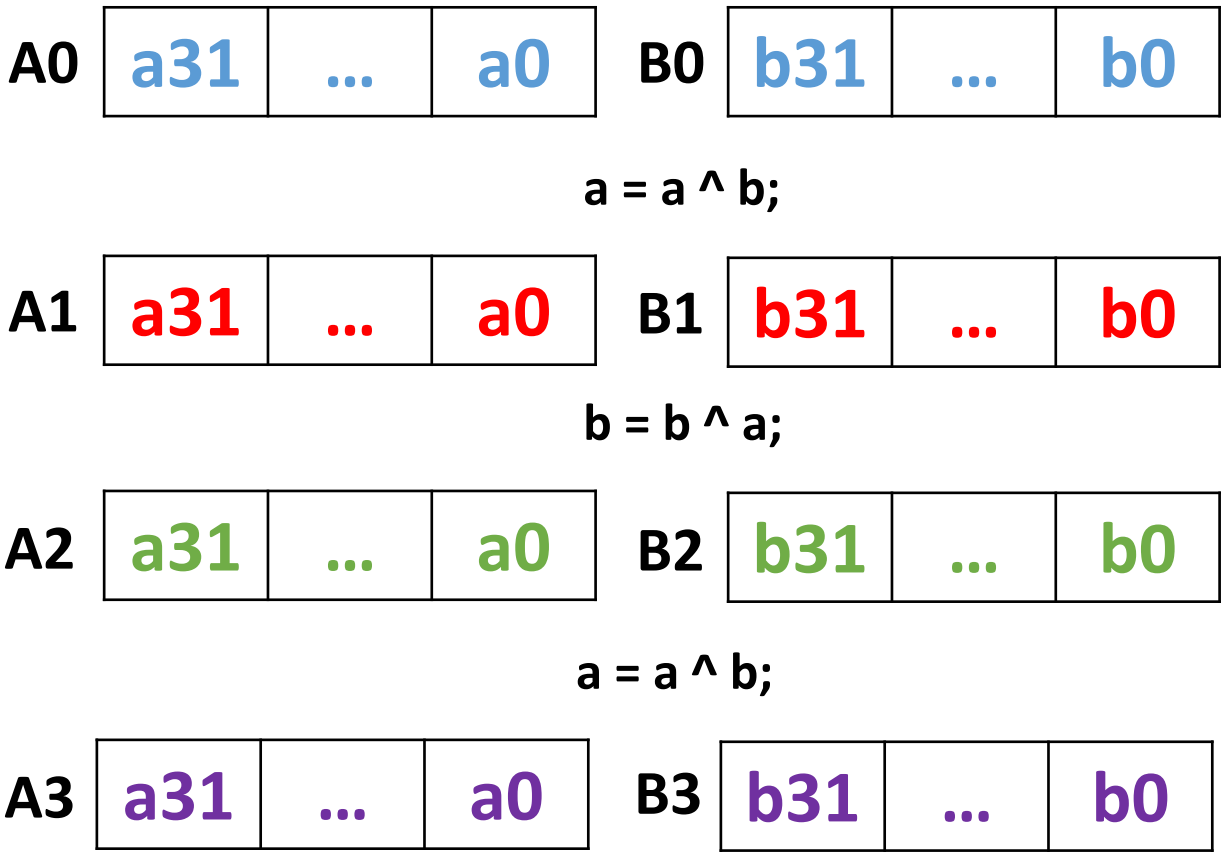
Straight-Line Code



$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$

$$(B1 \leftrightarrow B0) \wedge$$

Straight-Line Code



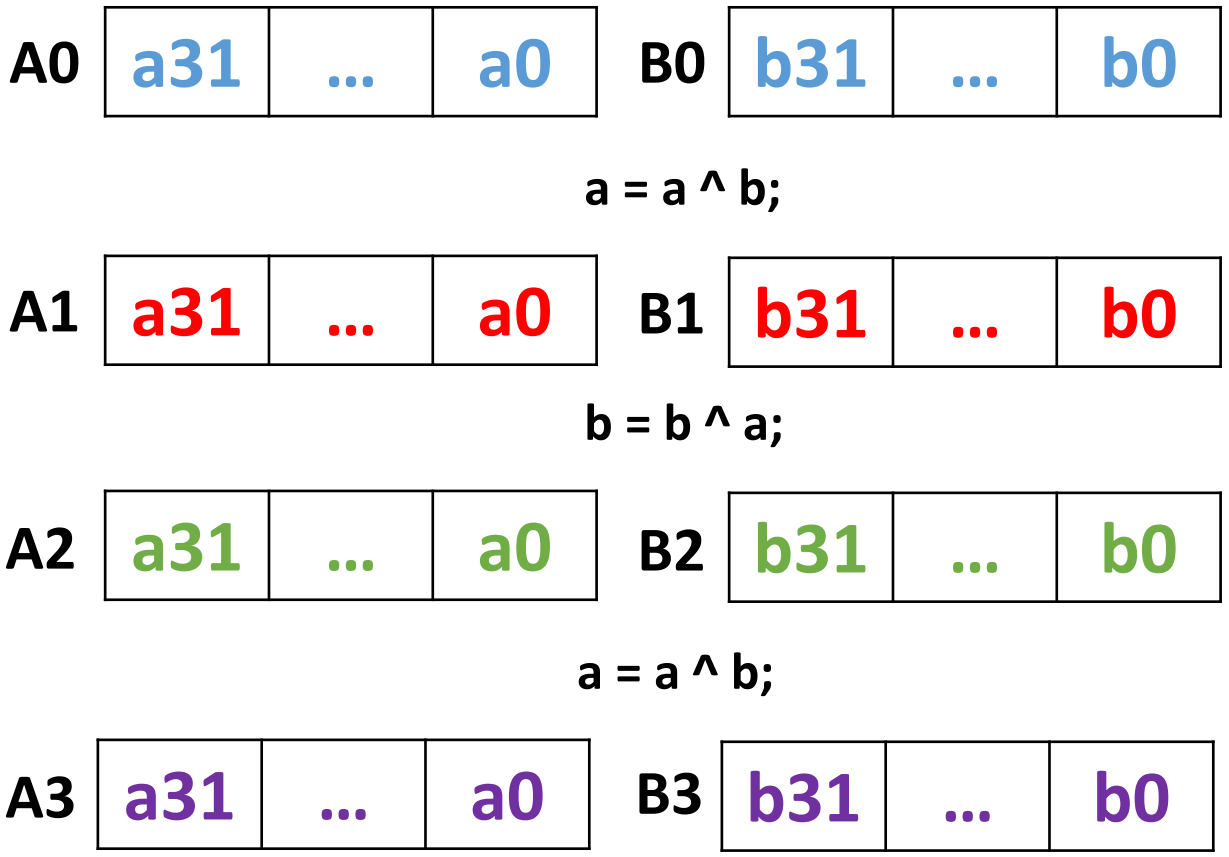
$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$

$$(B1 \leftrightarrow B0) \wedge$$

$$(A2 \leftrightarrow A1) \wedge$$

$$(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)) \wedge$$

Straight-Line Code



$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$

$$(B1 \leftrightarrow B0) \wedge$$

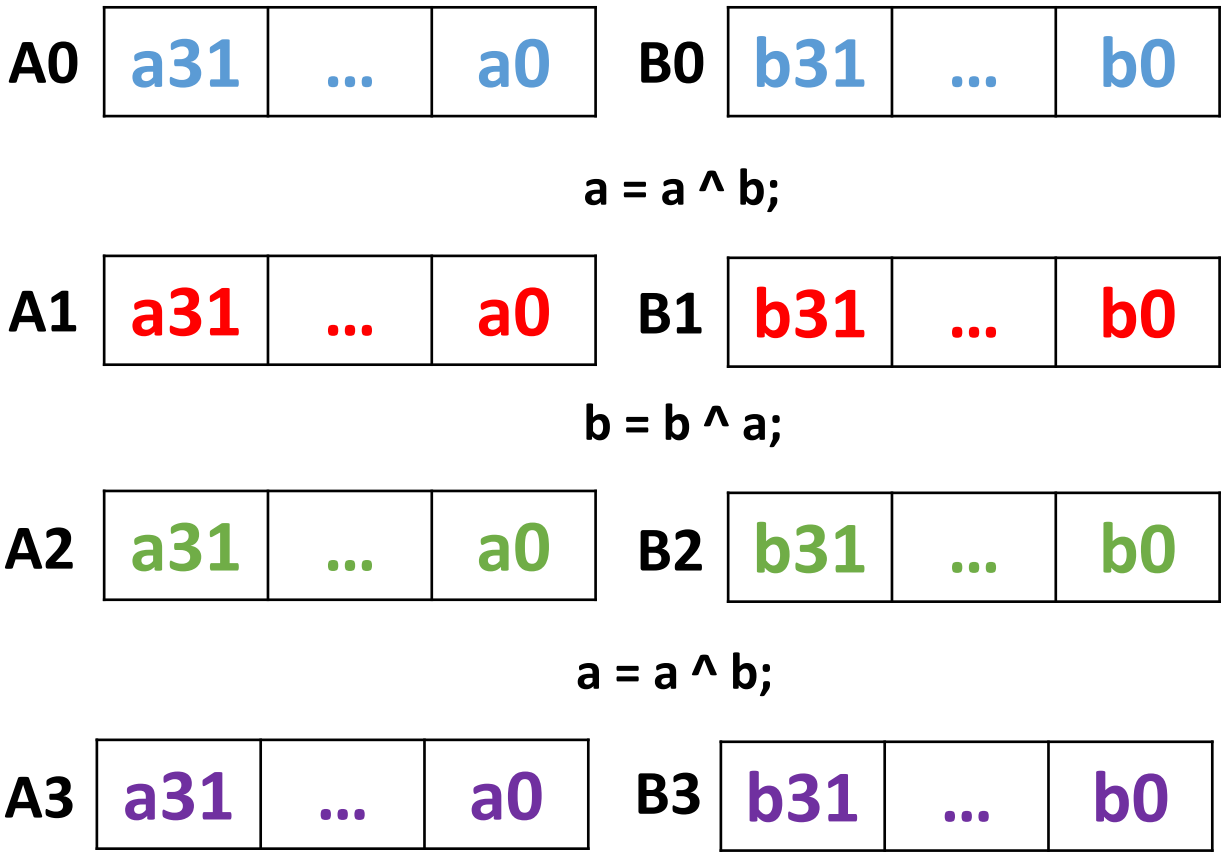
$$(A2 \leftrightarrow A1) \wedge$$

$$(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)) \wedge$$

$$(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)) \wedge$$

$$(B3 \leftrightarrow B2)$$

Straight-Line Code



$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$

$$(B1 \leftrightarrow B0) \wedge$$

$$(A2 \leftrightarrow A1) \wedge$$

$$(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)) \wedge$$

$$(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)) \wedge$$

$$(B3 \leftrightarrow B2)$$

How to prove a and b are swapped?

Straight-Line Code

$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$

$$(B1 \leftrightarrow B0) \wedge$$

$$(A2 \leftrightarrow A1) \wedge$$

$$(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)) \wedge$$

$$(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)) \wedge$$

$$(B3 \leftrightarrow B2) \rightarrow$$

$$(A3 \leftrightarrow B0) \wedge (A0 \leftrightarrow B3)$$

Program

Assertion

Straight-Line Code

$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$

$$(B1 \leftrightarrow B0) \wedge$$

$$(A2 \leftrightarrow A1) \wedge$$

$$(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)) \wedge$$

$$(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)) \wedge$$

$$(B3 \leftrightarrow B2) \rightarrow$$

$$(A3 \leftrightarrow B0) \wedge (A0 \leftrightarrow B3)$$

How to prove it is a tautology?

Program

Assertion

Straight-Line Code

RECAP

Tautology is a proposition which is always true under any interpretation.

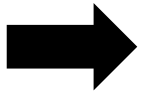
1) P is a tautology iff ¬P is a contradiction

2) P is unsatisfiable iff P is a contradiction

tautology ➡ UNSAT

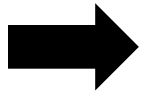
Straight-Line Code

$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$
$$(B1 \leftrightarrow B0) \wedge$$
$$(A2 \leftrightarrow A1) \wedge$$
$$(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)) \wedge$$
$$(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)) \wedge$$
$$(B3 \leftrightarrow B2) \rightarrow$$
$$(A3 \leftrightarrow B0) \wedge (A0 \leftrightarrow B3)$$



$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$
$$(B1 \leftrightarrow B0) \wedge$$
$$(A2 \leftrightarrow A1) \wedge$$
$$(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)) \wedge$$
$$(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)) \wedge$$
$$(B3 \leftrightarrow B2) \wedge$$
$$\neg((A3 \leftrightarrow B0) \wedge (A0 \leftrightarrow B3))$$

tautology



UNSAT

Straight-Line Code

$$
\begin{aligned}
&(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge \\
&(B1 \leftrightarrow B0) \wedge \\
&(A2 \leftrightarrow A1) \wedge \\
&(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)) \wedge \\
&(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)) \wedge \\
&(B3 \leftrightarrow B2) \wedge \\
&\neg((A3 \leftrightarrow B0) \wedge (A0 \leftrightarrow B3))
\end{aligned}
$$

If the formula is satisfiable, SAT solver will give an assignment,which is a counterexample.

Straight-Line Code

$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$

$$(B1 \leftrightarrow B0) \wedge$$

$$(A2 \leftrightarrow A1) \wedge$$

$$(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)) \wedge$$

$$(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)) \wedge$$

$$(B3 \leftrightarrow B2) \wedge$$

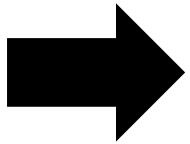
$$\neg((A3 \leftrightarrow B0) \wedge (A0 \leftrightarrow B3))$$

Some variables in different states are the same because the code does not change them.

We can simplify the formula by merging redundant variables.

Straight-Line Code

$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$
$$(B1 \leftrightarrow B0) \wedge$$
$$(A2 \leftrightarrow A1) \wedge$$
$$(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)) \wedge$$
$$(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)) \wedge$$
$$(B3 \leftrightarrow B2) \wedge$$
$$\neg((A3 \leftrightarrow B0) \wedge (A0 \leftrightarrow B3))$$



$$(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)) \wedge$$
$$(B2 \leftrightarrow (A1 \wedge \neg B0) \vee (\neg A1 \wedge B0)) \wedge$$
$$(A3 \leftrightarrow (A1 \wedge \neg B2) \vee (\neg A1 \wedge B2)) \wedge$$
$$\neg((A3 \leftrightarrow B0) \wedge (A0 \leftrightarrow B2))$$

Exercise

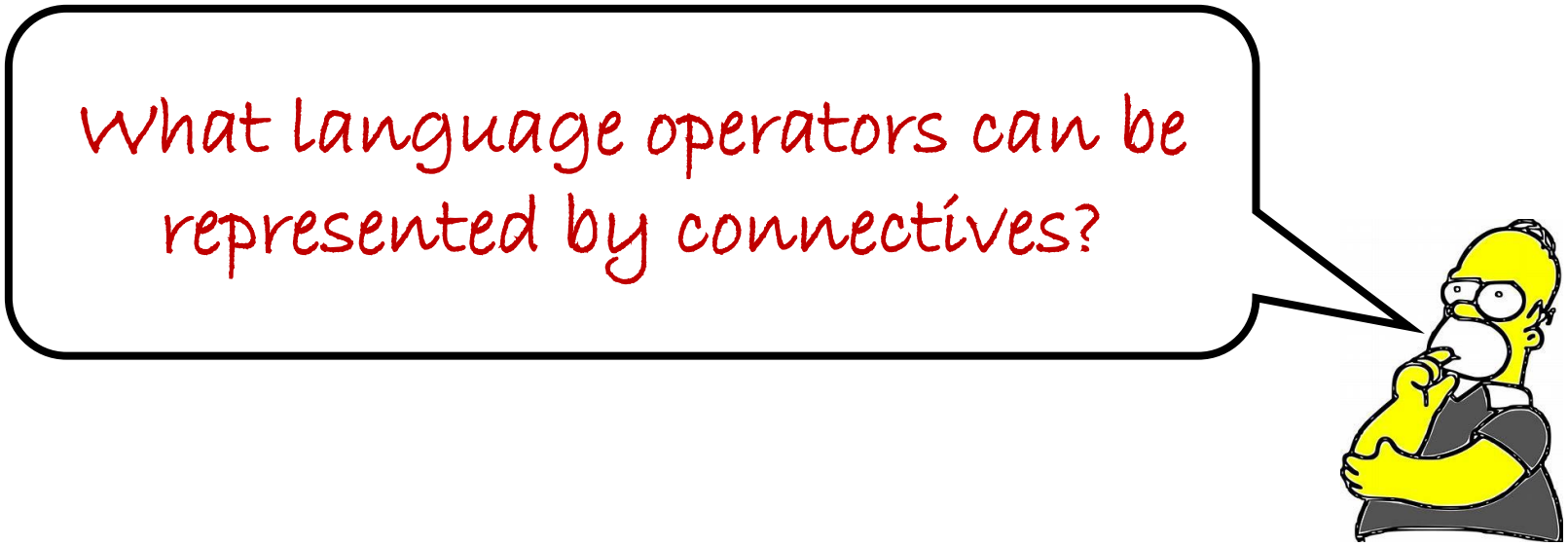
```
void f(int x, int y)
{
    int z = x & y ;
    z = z | x;
    assert(z == x);
}
```

How to prove this?

$$(Z1 \leftrightarrow (X \wedge Y)) \wedge (Z2 \leftrightarrow (Z1 \vee X)) \wedge \neg(Z2 \leftrightarrow X)$$

Prove this is UNSAT

Straight-Line Code



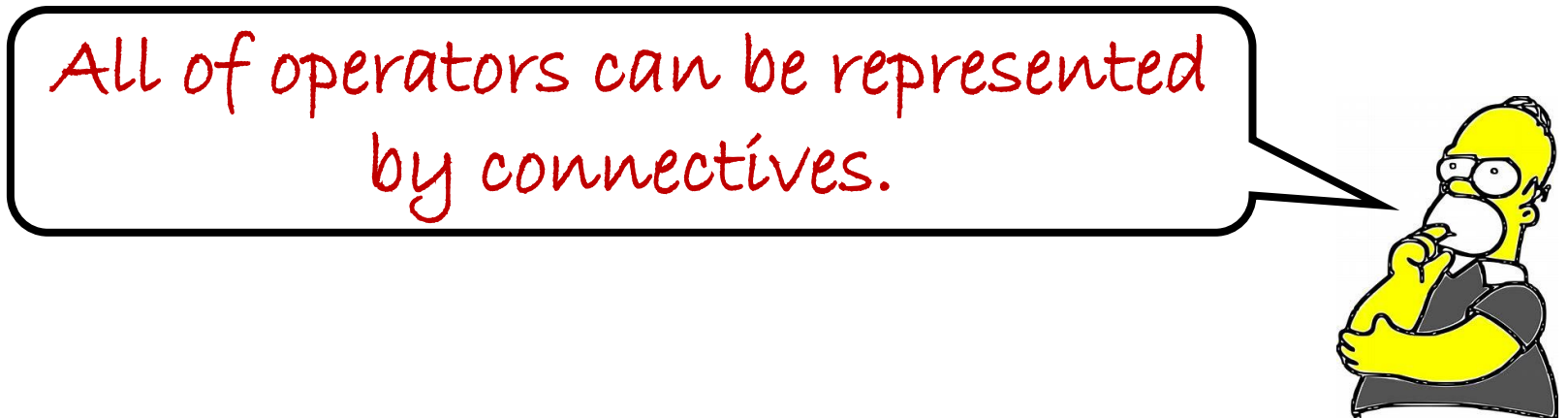
Straight-Line Code

- Logical operators, bit operators
  - &, |, ^, >>, !, ……
- Relational operators
  - >, <, ==, ……

What about arithmetic operators?

Straight-Line Code

Arithmetic operator, such as + and -, can be implemented with combinational circuit, which can be represented by propositional formulas.

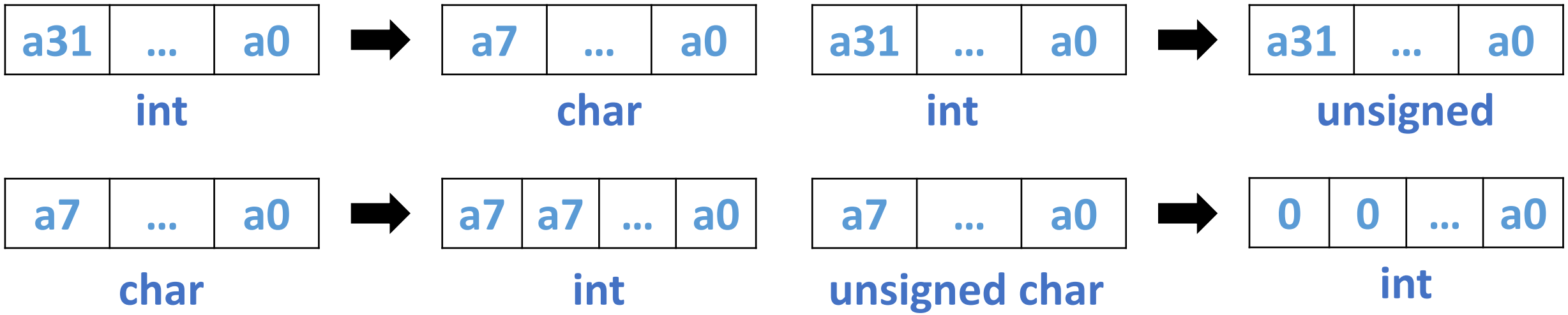


Straight-Line Code

Casts don’t change the value of any bits, But

- May increase or decrease the number of bits
- Widening casts pad
- Drop appropriate bits for narrowing casts

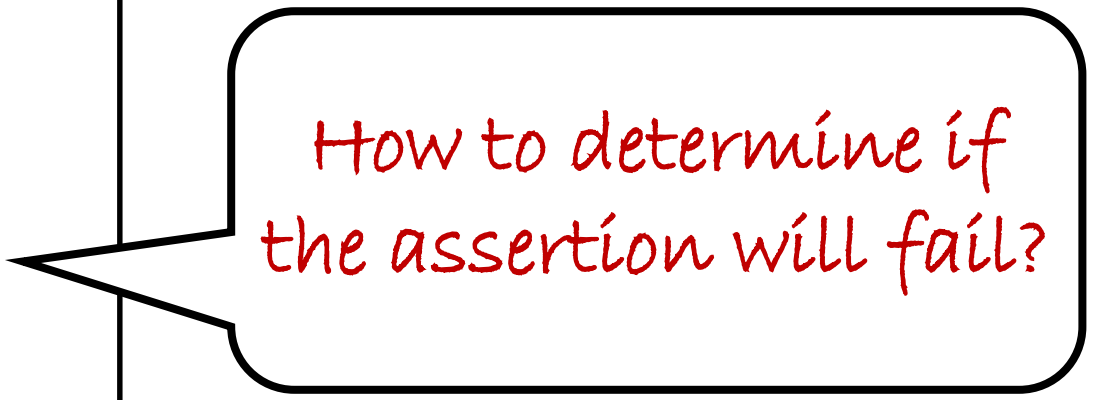
Casts are easy in this representation



What if there are some "if"?

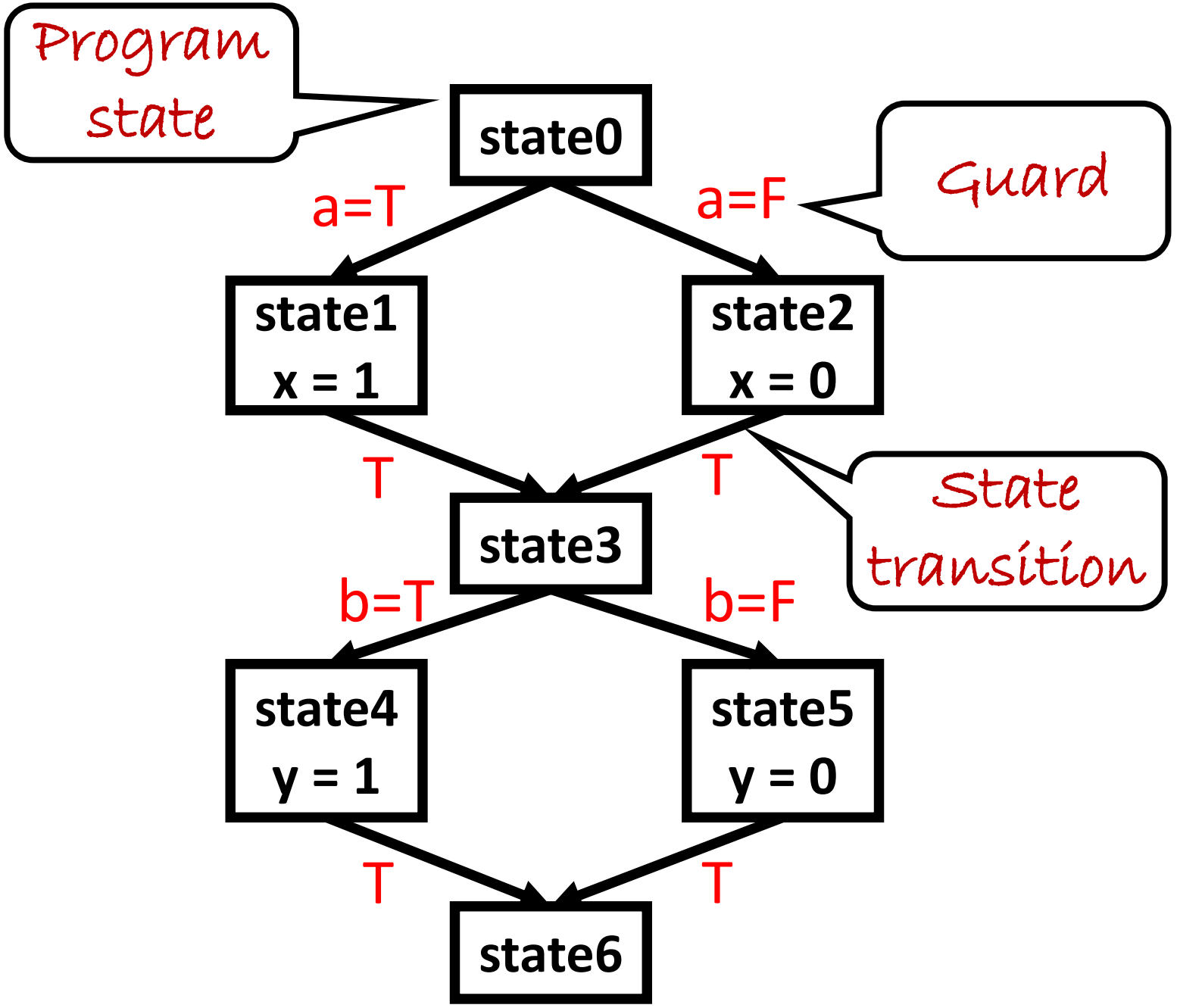
Control Flow

```
void f(bool a, bool b)
{
    unsigned x, y;
    if (a)
        x = 1;
    else
        x = 0;
    if (b)
        y = 1;
    else
        y = 0;
    assert(x + y > 0);
}
```

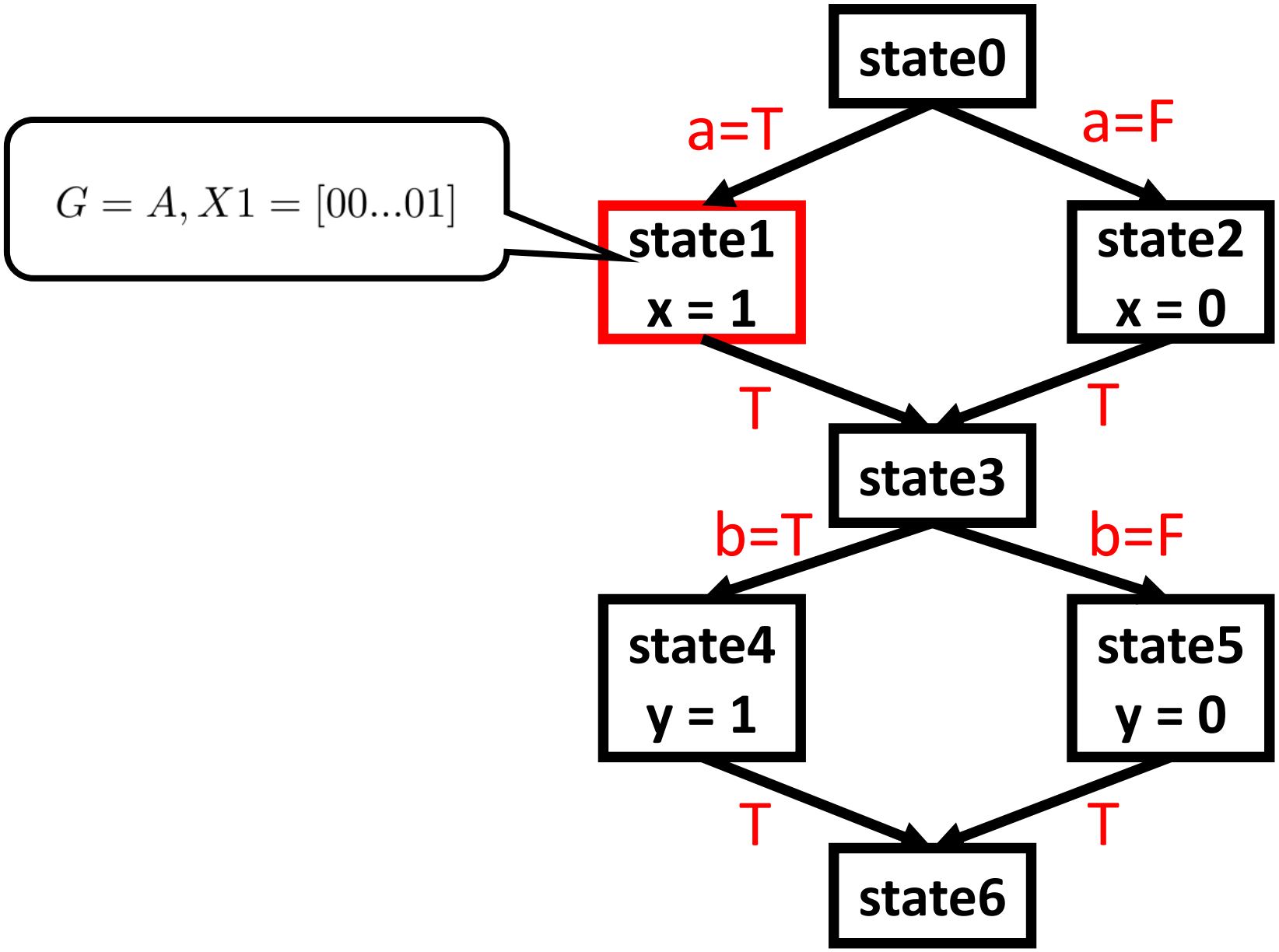


Control Flow

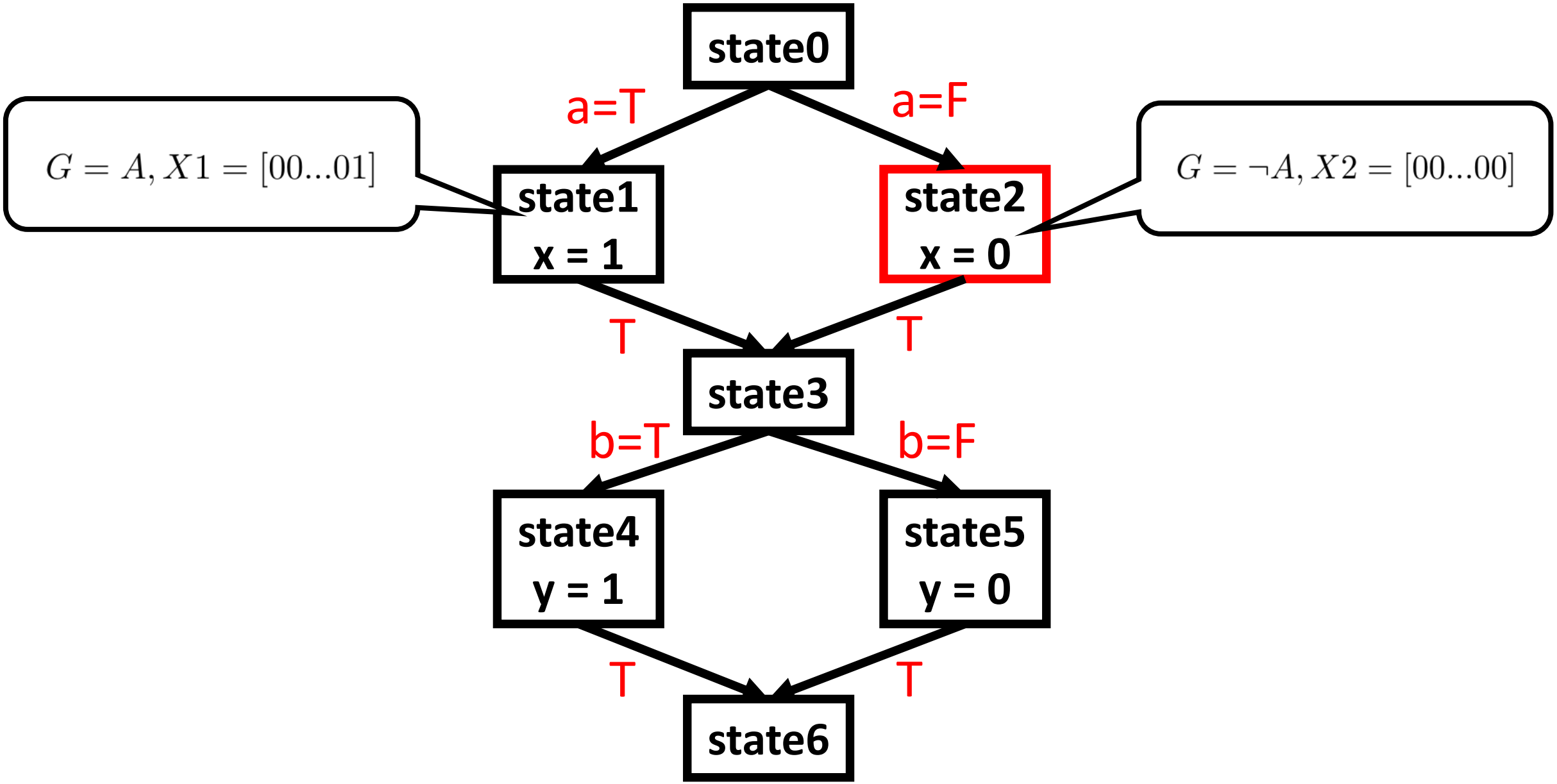
```
void f(bool a, bool b)
{
    unsigned x, y;
    if (a)
        x = 1;
    else
        x = 0;
    if (b)
        y = 1;
    else
        y = 0;
    assert(x + y > 0);
}
```



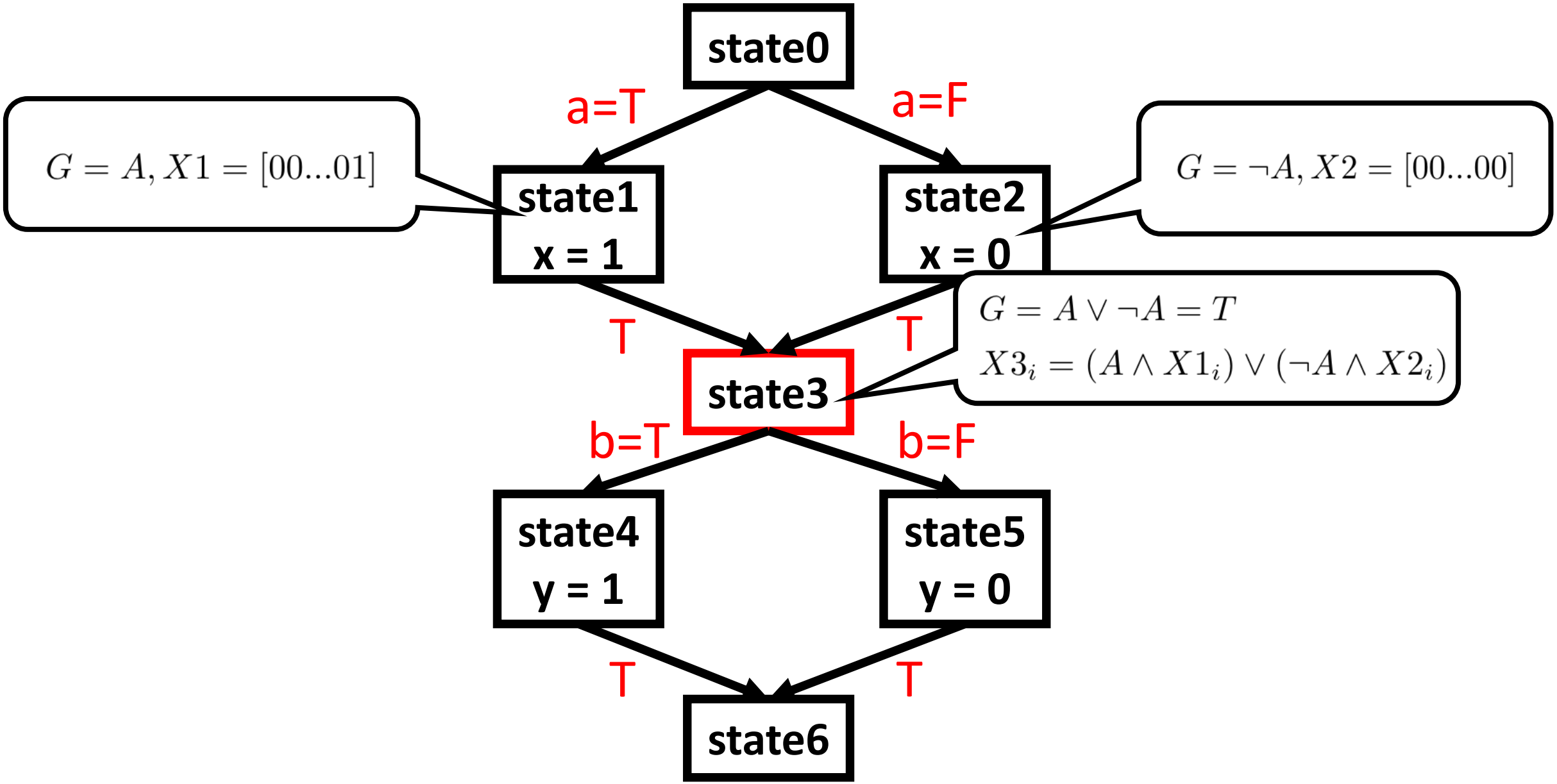
Control Flow



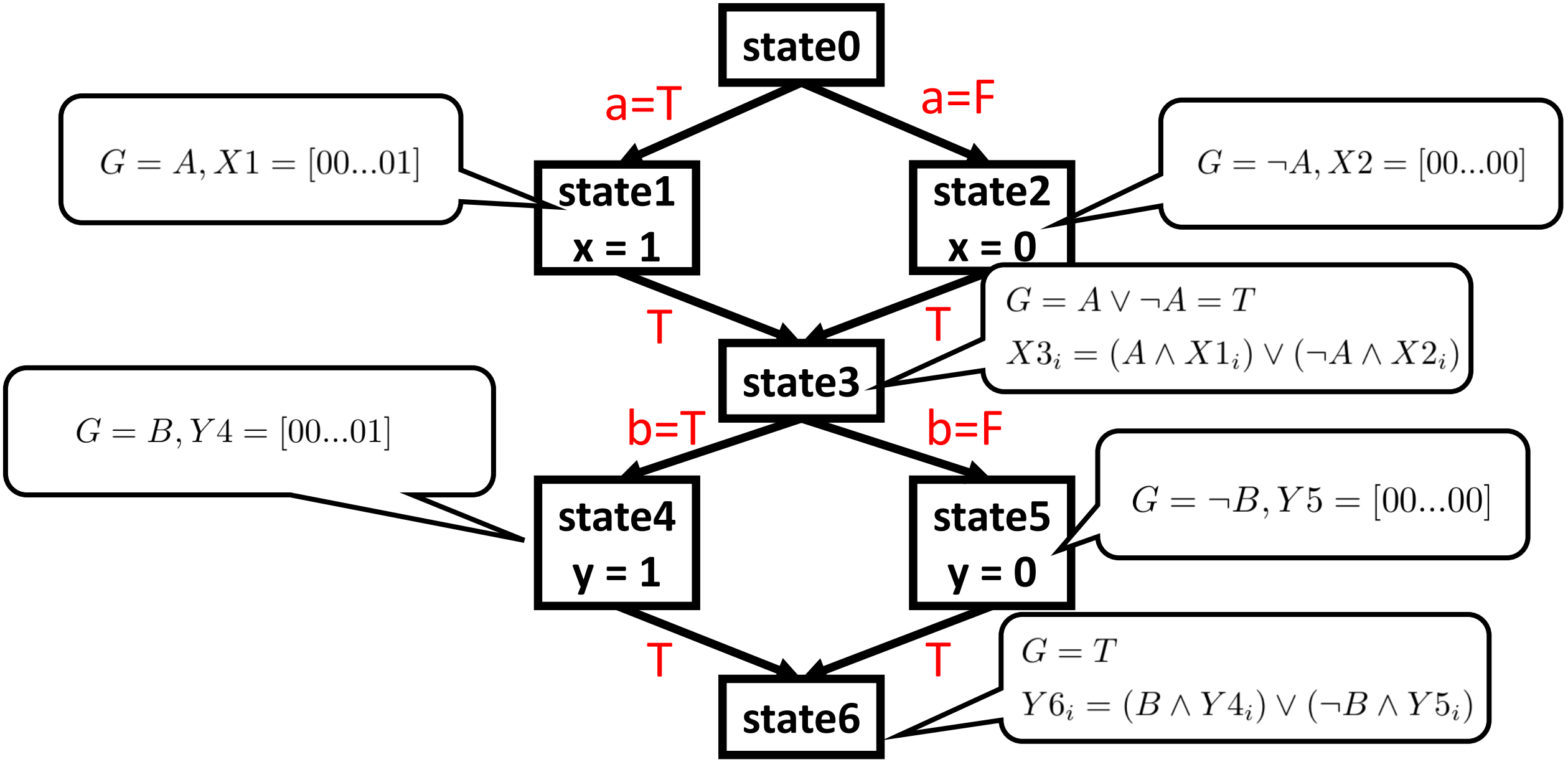
Control Flow



Control Flow

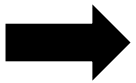


Control Flow



Control Flow

```
void f(bool a, bool b)
{
    unsigned x, y;
    if (a)
        x = 1;
    else
        x = 0;
    if (b)
        y = 1;
    else
        y = 0;
    assert(x + y > 0);
}
```



This can be represented in propositional logic

$$formula\{X1 = [00...01]\} \wedge$$
$$formula\{X2 = [00...00]\} \wedge$$
$$formula\{Y4 = [00...01]\} \wedge$$
$$formula\{Y5 = [00...00]\} \wedge$$
$$\wedge_{i=0,...,31} (X3_i \leftrightarrow (A \wedge X1_i) \vee (\neg A \wedge X2_i)) \wedge$$
$$\wedge_{i=0,...,31} (Y6_i \leftrightarrow (B \wedge Y4_i) \vee (\neg B \wedge Y5_i)) \rightarrow$$
$$formula\{X3 + Y6 > 0\}$$

Prove it is a tautology.

Control Flow

```
void f(bool a, bool b)
{
    unsigned x, y;
    if (a)
        x = 1;
    else
        x = 0;
    if (b)
        y = 1;
    else
        y = 0;
    assert(x + y > 0);
}
```

$$formula\{X1 = [00...01]\} \wedge$$
$$formula\{X2 = [00...00]\} \wedge$$
$$formula\{Y4 = [00...01]\} \wedge$$
$$formula\{Y5 = [00...00]\} \wedge$$
$$\wedge_{i=0,...,31} (X3_i \leftrightarrow (A \wedge X1_i) \vee (\neg A \wedge X2_i)) \wedge$$
$$\wedge_{i=0,...,31} (Y6_i \leftrightarrow (B \wedge Y4_i) \vee (\neg B \wedge Y5_i)) \rightarrow$$
$$formula\{X3 + Y6 > 0\}$$

Prove it is unsatisfiable

Control Flow

```
void f(bool a, bool b)
{
    unsigned x, y;
    if (a)
        x = 1;
    else
        x = 0;
    if (b)
        y = 1;
    else
        y = 0;
    assert(x + y > 0);
}
```

A=F, B=F can satisfy it.
So the assertion will fail.

$$formula\{X1 = [00...01]\} \wedge$$
$$formula\{X2 = [00...00]\} \wedge$$
$$formula\{Y4 = [00...01]\} \wedge$$
$$formula\{Y5 = [00...00]\} \wedge$$
$$\wedge_{i=0,...,31} (X3_i \leftrightarrow (A \wedge X1_i) \vee (\neg A \wedge X2_i)) \wedge$$
$$\wedge_{i=0,...,31} (Y6_i \leftrightarrow (B \wedge Y4_i) \vee (\neg B \wedge Y5_i)) \rightarrow$$
$$formula\{X3 + Y6 > 0\}$$

Loops

```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    while(i > 0)
    {
        a = a + 1;
        i = i - 1;
    }
    assert(a == 3);
}
```

What if there are some "while"?

Loops

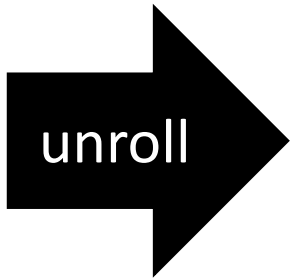
```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    while(i > 0)
    {
        a = a + 1;
        i = i - 1;
    }
    assert(a == 3);
}
```

We can unroll it for 3 times.

Loops

```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    while(i > 0)
    {
        a = a + 1;
        i = i - 1;
    }
    assert(a == 3);
}
```

Loops




```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    assert(a == 3);
}
```

Straight-Line Code

Loops

```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    while(i > 0)
    {
        a = a + 1;
        i = i - 1;
    }
    assert(a == 3);
}
```

unroll

```
void f(unsigned a)
{
    unsigned i = 3;
    a = 0;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    a = a + 1;
    i = i - 1;
    assert(a == 3);
```

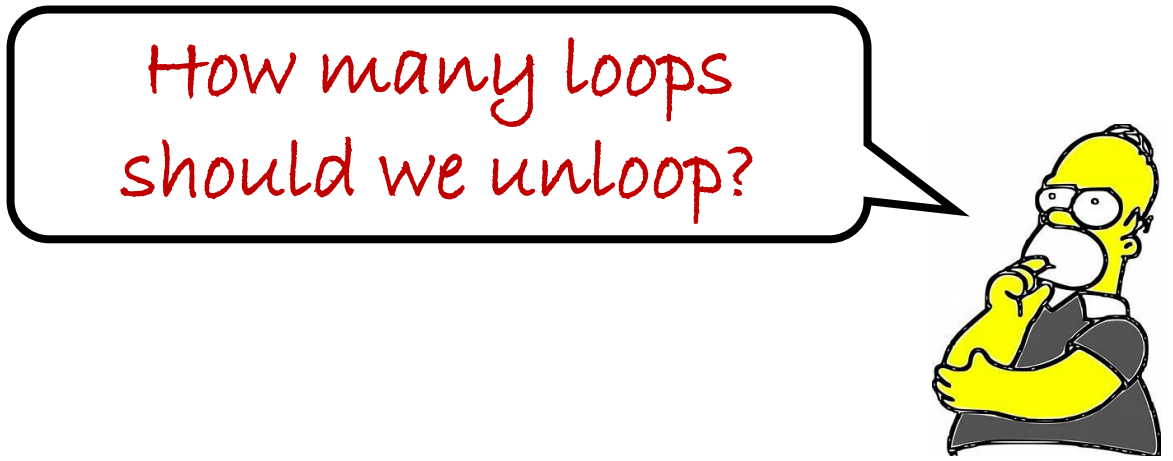
Is unrolling a master key?

Loops

Str ine Code

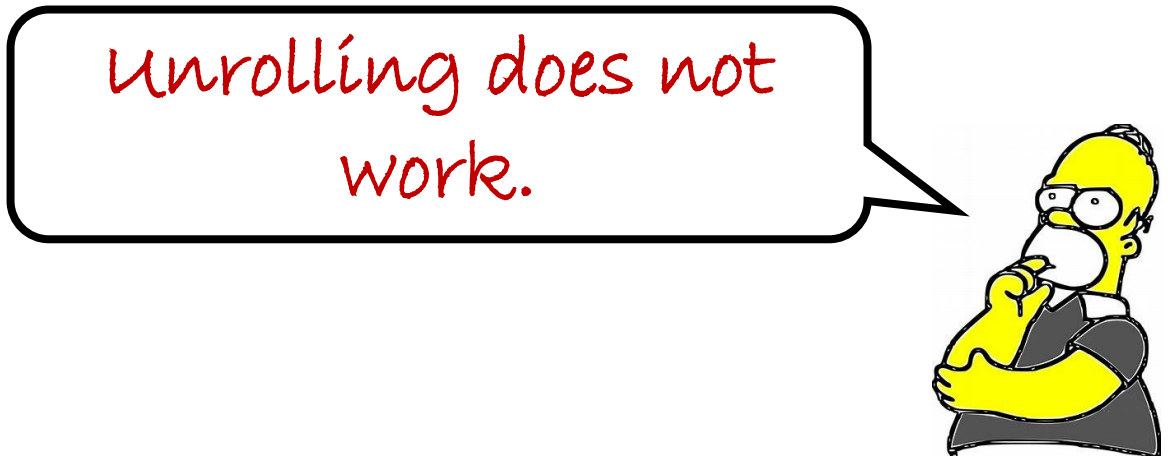
Loops

```
void f(unsigned a)
{
    unsigned i = 0;
    while(i < a)
    {
        i = i + 1;
    }
}
```



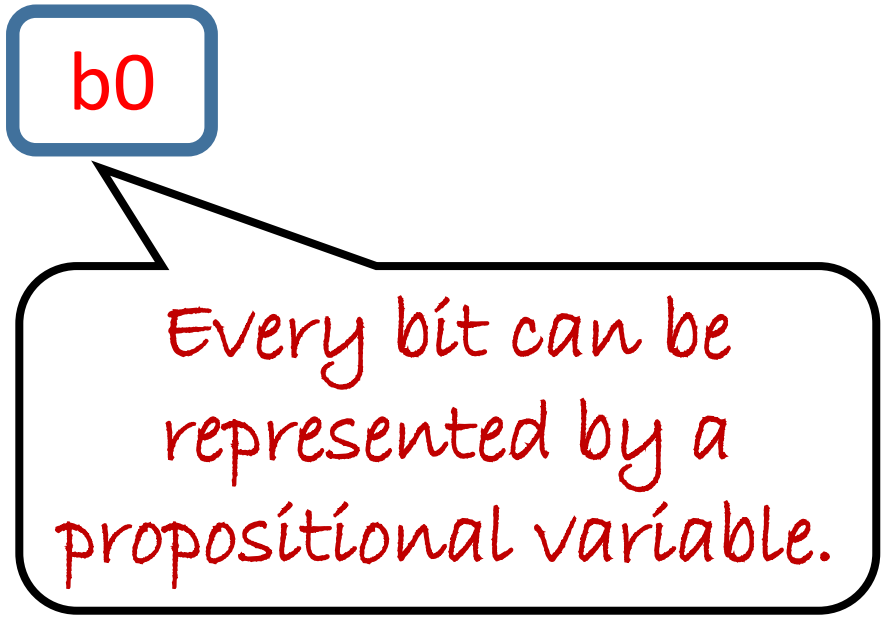
Loops

```
void f(unsigned a)
{
    unsigned i = 0;
    while(i < a)
    {
        i = i + 1;
    }
}
```

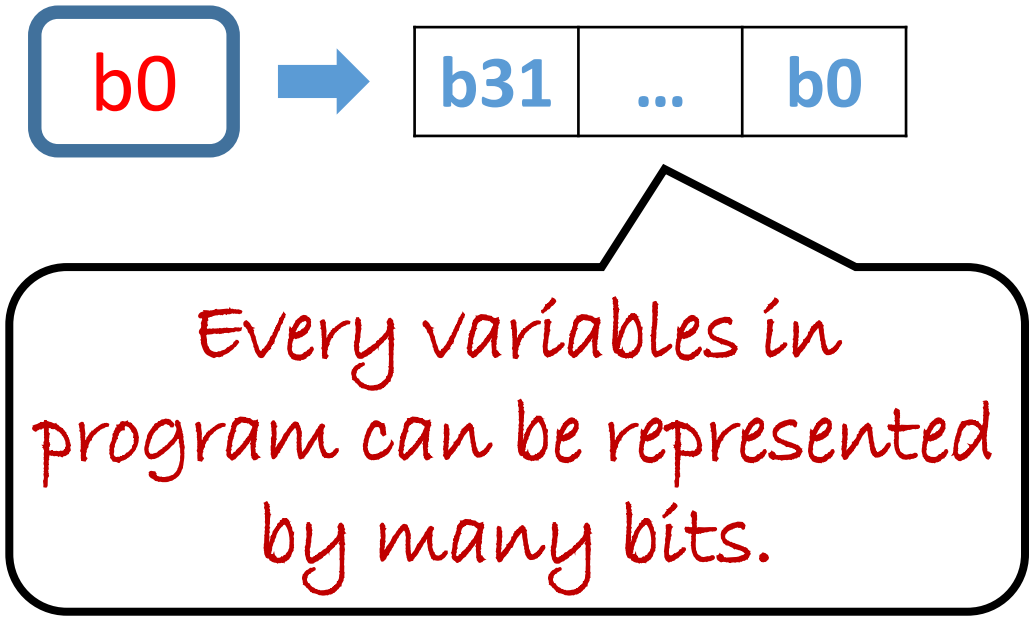


- Unroll loops k times, drop backedges
- May miss errors that are deeply buried
- Simplicity

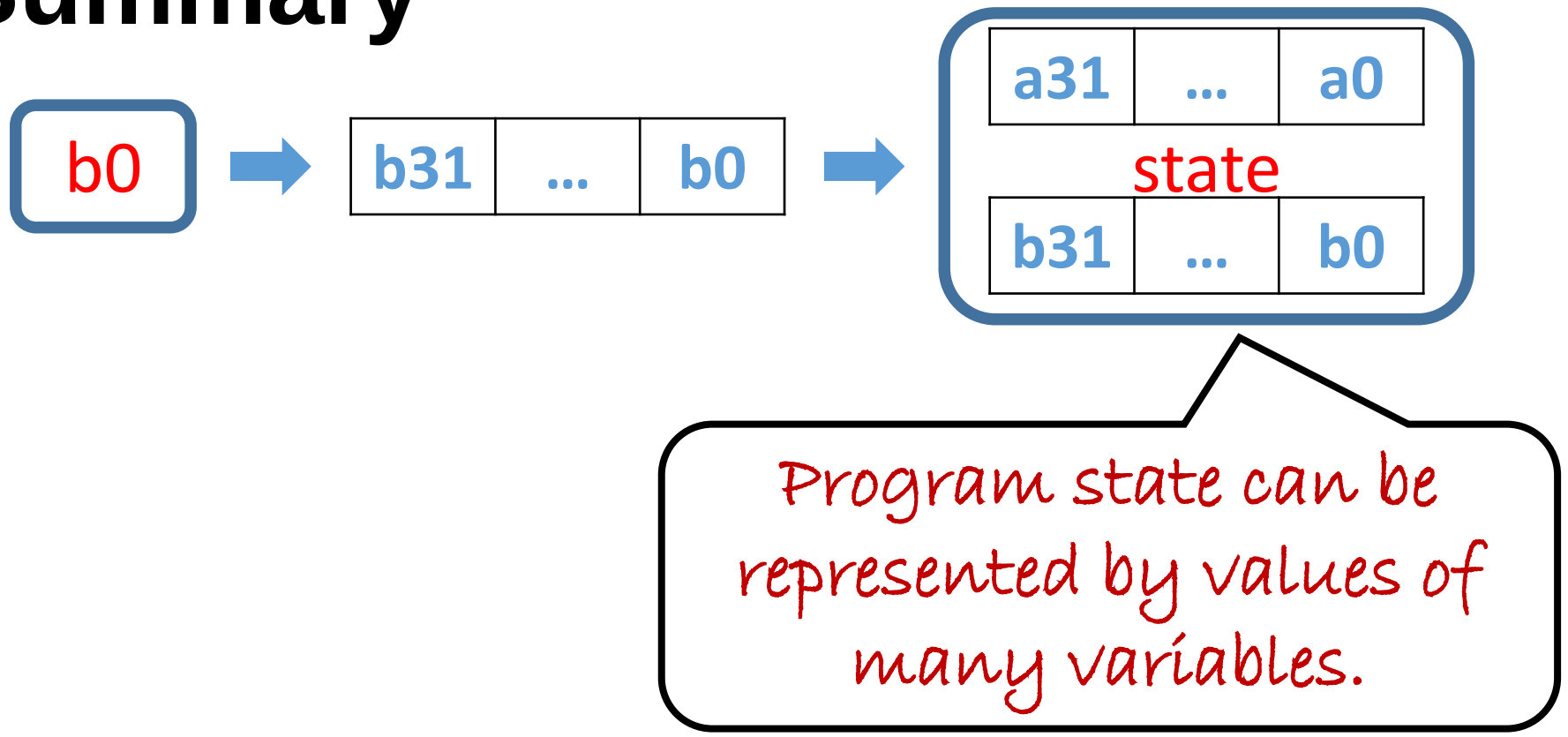
Summary


b0
Every bit can be represented by a propositional variable.

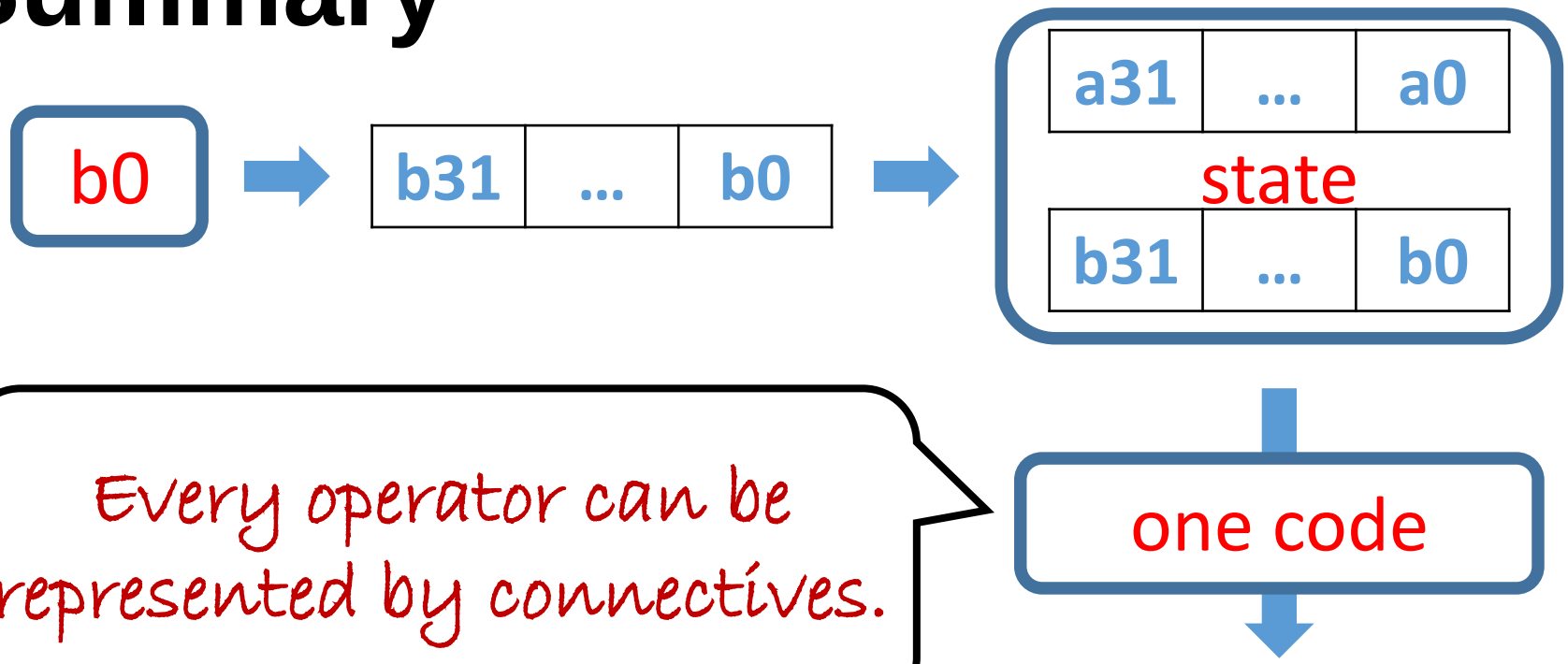
Summary


b0
b31
…
b0
Every variables in program can be represented by many bits.

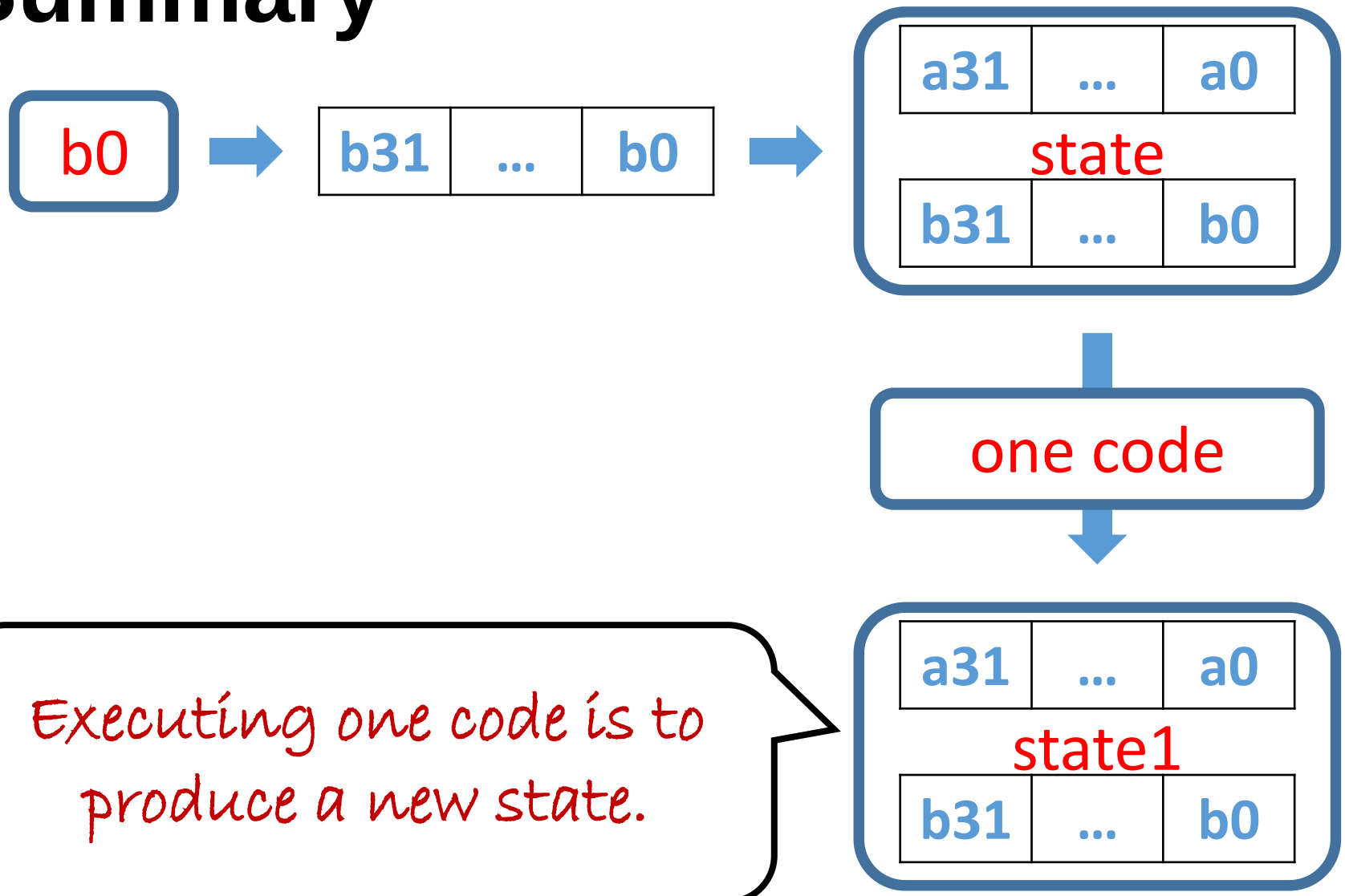
Summary


b0
b31
…
b0
a31
…
a0
state
b31
…
b0
Program state can be represented by values of many variables.

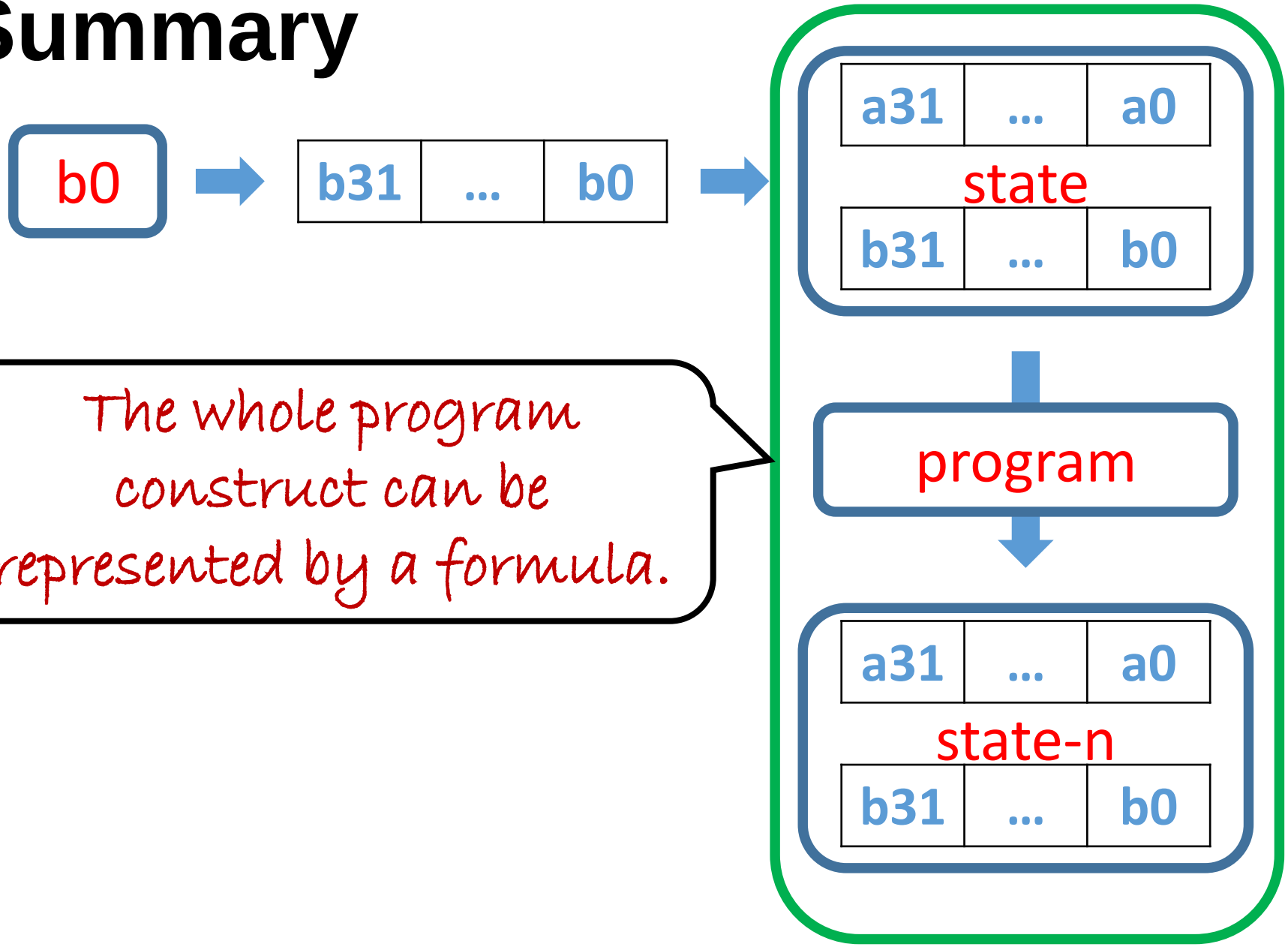
Summary


b0
b31
…
b0
a31
…
a0
state
b31
…
b0
Every operator can be represented by connectives.
one code

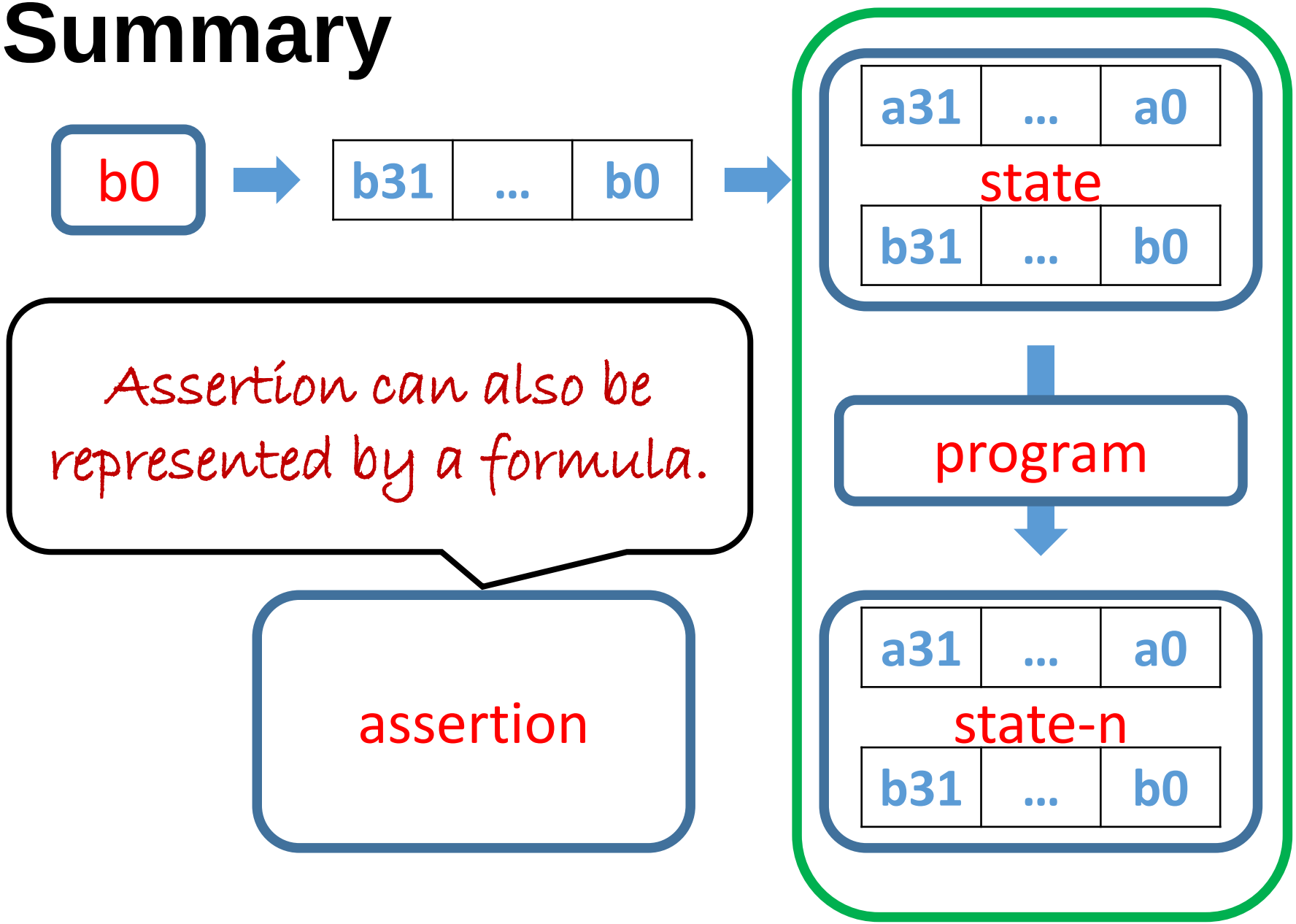
Summary


b0
b31
…
b0
a31
…
a0
state
b31
…
b0
one code
a31
…
a0
state1
b31
…
b0
Executing one code is to produce a new state.

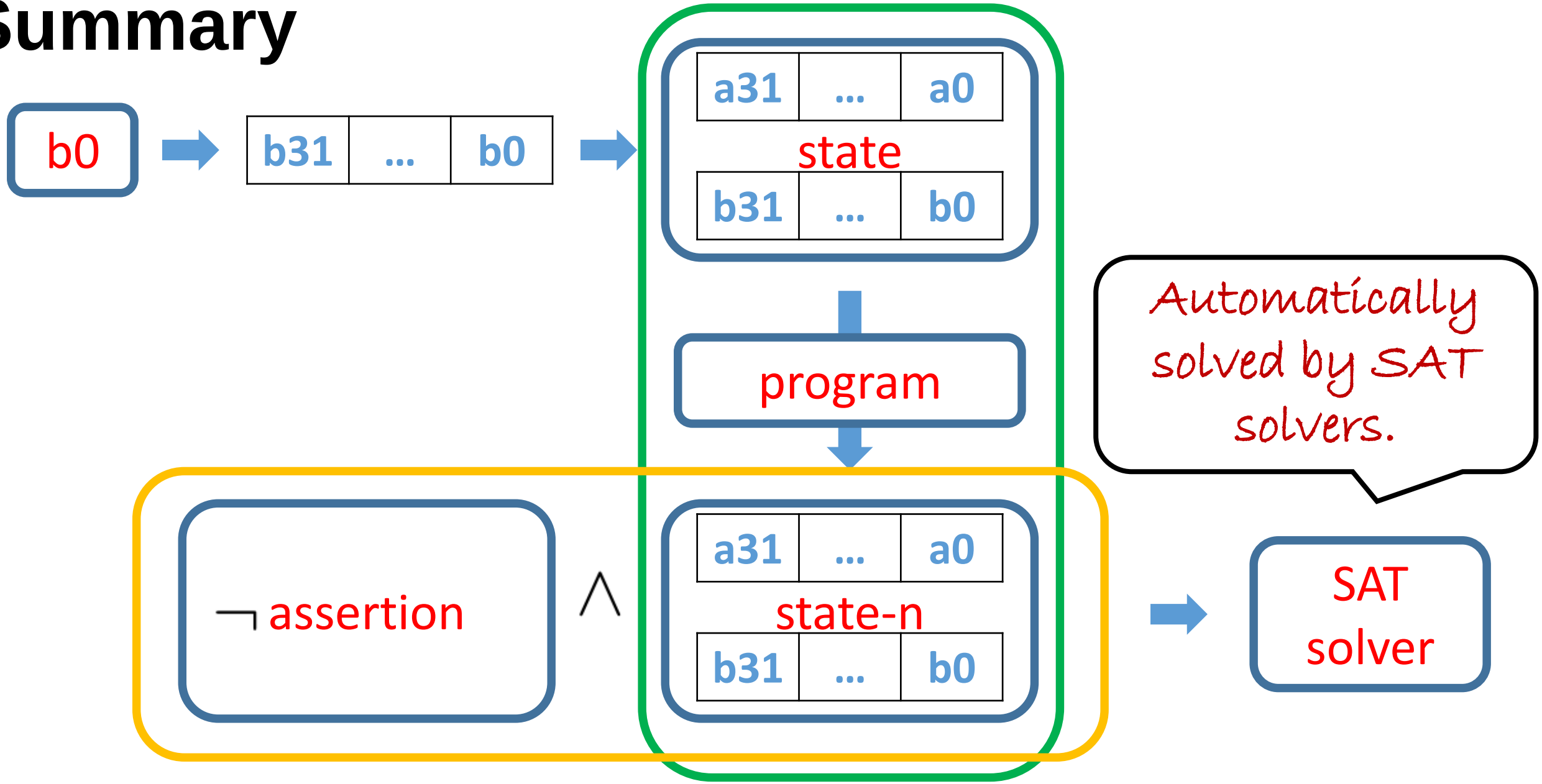
Summary


b0
b31 … b0
a31 … a0
state
b31 … b0
program
a31 … a0
state-n
b31 … b0
The whole program construct can be represented by a formula.

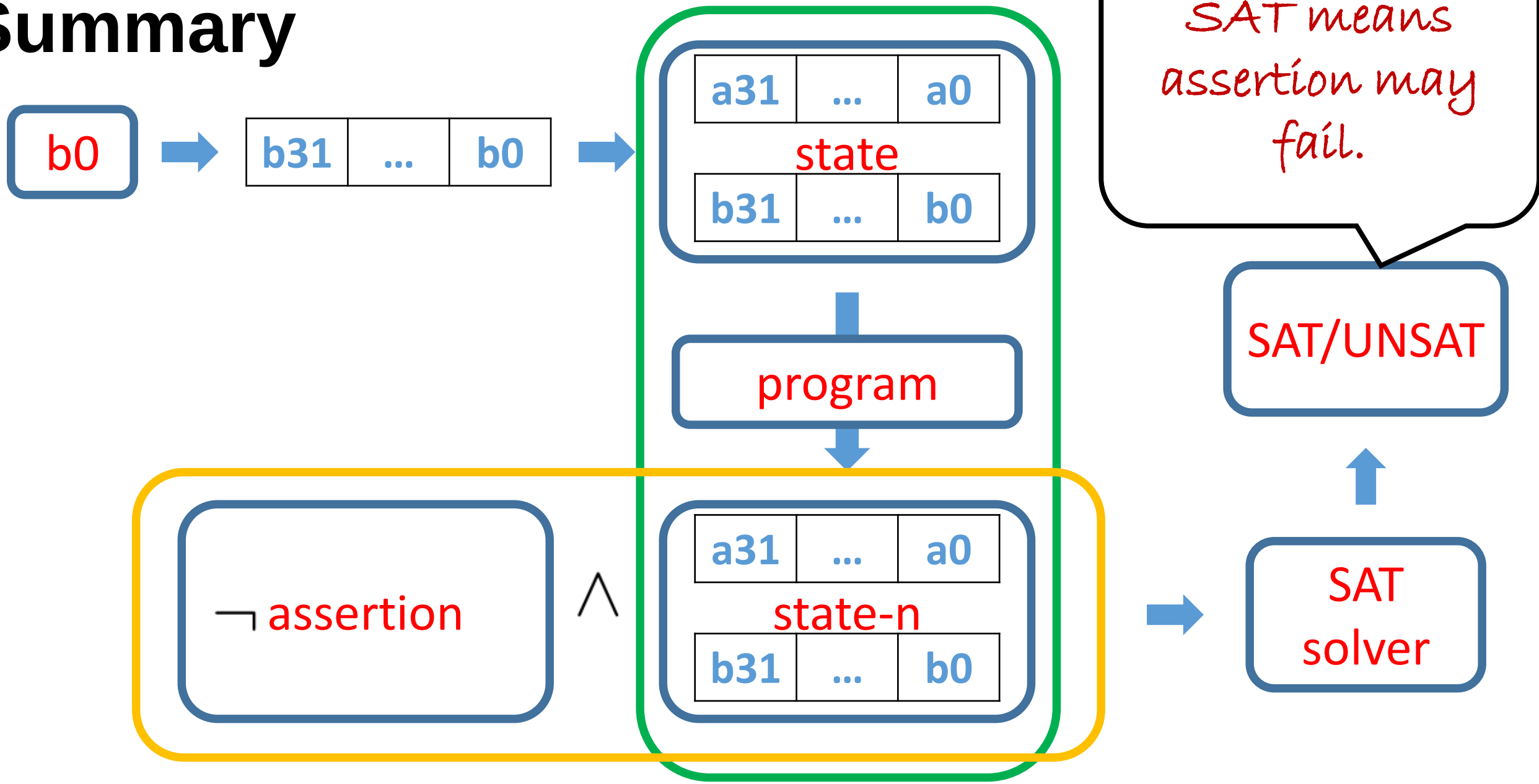
Summary


b0
b31
…
b0
a31
…
a0
state
b31
…
b0
program
a31
…
a0
state-n
b31
…
b0
Assertion can also be represented by a formula.
assertion

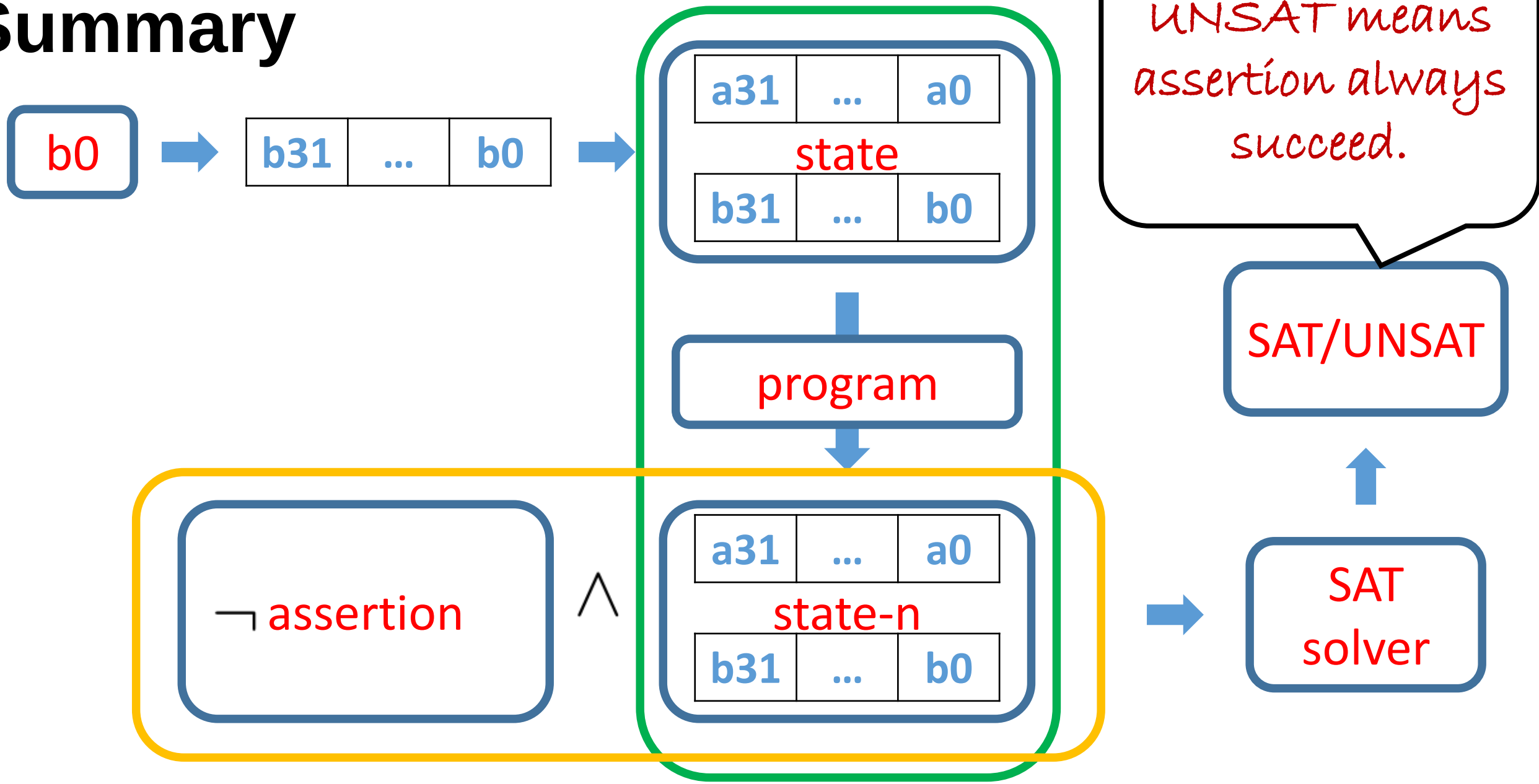
Summary


b0
b31 … b0
a31 … a0
state
b31 … b0
program
¬ assertion
∧
a31 … a0
state-n
b31 … b0
Automatically solved by SAT solvers.
SAT solver

Summary


b0
b31 … b0
a31 … a0
state
b31 … b0
program
¬ assertion
∧
a31 … a0
state-n
b31 … b0
SAT solver
SAT/UNSAT
SAT means assertion may fail.

Summary


b0
b31 … b0
a31 … a0
state
b31 … b0
program
¬ assertion
∧
a31 … a0
state-n
b31 … b0
SAT
solver
SAT/UNSAT
UNSAT means assertion always succeed.

Deduction

DEFINITION

A proof is an argument that demonstrates why a conclusion is true, subject to certain standards of truth, which is made up of a finite sequence of fixed indisputable steps.

DEFINITION

Proofs in natural language are informal proof. They are arguments about mathematical objects.

Proofs in logic are formal proofs or deductions(推理). They are mathematical objects themselves.

Deduction

DEFINITION

A proof is an argument that demonstrates why a conclusion is true, subject to certain standards of truth, which is made up of a finite sequence of fixed indisputable steps.

DEFINITION

Proofs in natural language are informal proof. They are arguments about mathematical objects.

Proofs in logic are formal proofs or de

mathematical objects themselves.

Remember that an "informal" proof must still be convincing!

Deduction

facts assumed to be true for the purpose of the deduction

Axioms or Assumptions

Deduction

Inference rules

a group of rules for creating new facts from old

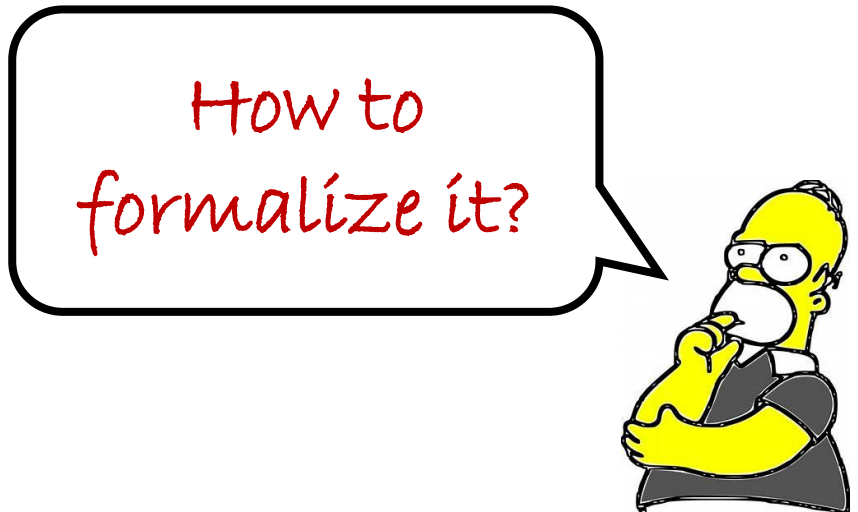
Deduction is a sequence of theorems obtained using a specific set of assumptions and rules of inference.

Deduction

DEFINITION

Formalize the deduction relation and we can get a deduction form (推理形式).

- If I am ill, I don’t go to school.
- I am ill.
- So I don’t go to school.



Deduction

DEFINITION

Formalize the deduction relation and we can get a deduction form(推理形式).

A — I am ill.

B — I don't go to school.

$A \to B$ — If I am ill, I don't go to school.

$$A \wedge (A \to B) \to B$$

$$\frac{\begin{array}{ll} A \to B & assumption \\ A & assumption \end{array}}{B \quad conclusion}$$

Deduction

$A \wedge (A \to B) \to B$

P Q

This is a correct deduction form iff it is a tautology.

DEFINITION

Given two formulas A and B, if B must be true when A is true, then A tautologically implies(重言蘊含) B.

$A \Rightarrow B$

Deduction form is correct iff P tautologically implies Q.

It's not a connective.

Deduction

$$A \wedge (A \to B) \Rightarrow B$$

It can be proved by truth table, equivalence calculus and interpretation in natural language.

THEOREM

1) If $A \Longrightarrow B$, A is a tautology, then B is a tautology.

2) If $A \Longrightarrow B, B \Longrightarrow A$, then $A = B$.

3) If $A \Longrightarrow B, B \Longrightarrow C$, then $A \Longrightarrow C$.

4) If $A \Longrightarrow B, A \Longrightarrow C$, then $A \Longrightarrow B \wedge C$.

5) If $A \Longrightarrow C, B \Longrightarrow C$, then $A \vee B \Longrightarrow C$.

Deduction

There are many important tautology implication expressions.

They can be used as inference rules to produce new theorems.

$$\neg(P \to Q) \Rightarrow P$$

$$\neg(P \to Q) \Rightarrow \neg Q$$

$$P \wedge (P \to Q) \Rightarrow Q$$

hypothetical reasoning(假言推理/分离规则)

$$(P \to Q) \wedge (Q \to R) \Rightarrow P \to R$$

Syllogism(三段论)

Deduction

We now introduce an example to explain deduction calculus.

Prove R when $P \to Q$, $Q \to R$ and P.

Deduction

We now introduce an example to explain deduction calculus.

Prove R when $P \to Q$, $Q \to R$ and P.

(1) P *introduce premise*

We can introduce assumptions/premise(前提引入规则) anytime in deduction process.

Deduction

We now introduce an example to explain deduction calculus.

Prove R when $P \to Q$, $Q \to R$ and P.

$(1) P$	*introduce premise*
$(2) P \to Q$	*introduce premise*

Deduction

We now introduce an example to explain deduction calculus.

Prove R when $P \to Q$, $Q \to R$ and P.

(1) P	*introduce premise*
(2) $P \to Q$	*introduce premise*
(3) Q	*Hypothetical reasoning on* (1)(2)

We can use hypothetical reasoning to produce new theorems(分离规则), which can be used as premise(结论引用规则).

Deduction

We now introduce an example to explain deduction calculus.

Prove R when $P \to Q$, $Q \to R$ and P.

(1) P	*introduce premise*
(2) $P \to Q$	*introduce premise*
(3) Q	*Hypothetical reasoning on* (1)(2)
(4) $Q \to R$	*introduce premise*

Deduction

We now introduce an example to explain deduction calculus.

Prove R when $P \to Q$, $Q \to R$ and P.

(1) P	*introduce premise*
(2) $P \to Q$	*introduce premise*
(3) Q	*Hypothetical reasoning on* (1)(2)
(4) $Q \to R$	*introduce premise*
(5) R	*Hypothetical reasoning on* (3)(4)

Substitution rule

THEOREM

Substitution rule(代入规则):

For any propositional variable P that occurs in tautology A, replace each and every occurrence of P in A with another formula and get a new formula B. Then B is also a tautology.

$$P \vee \neg P \quad \xrightarrow{\frac{P}{(R \vee S)}} \quad (R \vee S) \vee \neg (R \vee S)$$

Replacement rule

THEOREM

Replacement rules(置换规则) are rules of replacing sub-formula in A and still have a wff B with the same truth-value; in other words, they are a list of logical equivalencies.

We have implicitly applied replacement rules in propositional equivalence calculus.

Deduction

Prove $(P \to (Q \to S)) \land (\neg R \lor P) \land Q \Rightarrow R \to S$.

Deduction

Prove $(P \to (Q \to S)) \land (\neg R \lor P) \land Q \Rightarrow R \to S$.

$(1) \neg R \lor P$ *introduce premise*

Deduction

Prove $(P \to (Q \to S)) \land (\neg R \lor P) \land Q \Rightarrow R \to S$.

$(1) \neg R \lor P$	*introduce premise*
$(2) R \to P$	*replacement rule*

Deduction

Prove $(P \to (Q \to S)) \wedge (\neg R \vee P) \wedge Q \Rightarrow R \to S$.

$(1) \neg R \vee P$	*introduce premise*
$(2) R \to P$	*replacement rule*
$(3) R$	*introduce premise*

We can introduce R because
$A \wedge B \Longrightarrow C$ iff $A \Longrightarrow B \to C$.(
条件证明规则)

Deduction

Prove $(P \to (Q \to S)) \land (\neg R \lor P) \land Q \Rightarrow R \to S$.

(1)$\neg R \lor P$	*introduce premise*
(2)$R \to P$	*replacement rule*
(3)R	*introduce premise*
(4)P	*Hypothetical reasoning on* (2)(3)

Deduction

Prove $(P \to (Q \to S)) \wedge (\neg R \vee P) \wedge Q \Rightarrow R \to S$.

$(1) \neg R \vee P$	*introduce premise*
$(2) R \to P$	*replacement rule*
$(3) R$	*introduce premise*
$(4) P$	*Hypothetical reasoning on* (2)(3)
$(5) P \to (Q \to S)$	*introduce premise*

Deduction

Prove $(P \to (Q \to S)) \wedge (\neg R \vee P) \wedge Q \Rightarrow R \to S$.

$(1) \neg R \vee P$	*introduce premise*
$(2) R \to P$	*replacement rule*
$(3) R$	*introduce premise*
$(4) P$	*Hypothetical reasoning on* (2)(3)
$(5) P \to (Q \to S)$	*introduce premise*
$(6) Q \to S$	*Hypothetical reasoning on* (4)(5)

Deduction

Prove $(P \to (Q \to S)) \land (\neg R \lor P) \land Q \Rightarrow R \to S$.

(1) $\neg R \lor P$	*introduce premise*
(2) $R \to P$	*replacement rule*
(3) R	*introduce premise*
(4) P	*Hypothetical reasoning on* (2)(3)
(5) $P \to (Q \to S)$	*introduce premise*
(6) $Q \to S$	*Hypothetical reasoning on* (4)(5)
(7) Q	*introduce premise*

Deduction

Prove $(P \to (Q \to S)) \wedge (\neg R \vee P) \wedge Q \Rightarrow R \to S$.

$(1) \neg R \vee P$	*introduce premise*
$(2) R \to P$	*replacement rule*
$(3) R$	*introduce premise*
$(4) P$	*Hypothetical reasoning on* (2)(3)
$(5) P \to (Q \to S)$	*introduce premise*
$(6) Q \to S$	*Hypothetical reasoning on* (4)(5)
$(7) Q$	*introduce premise*
$(8) S$	*Hypothetical reasoning on* (6)(7)

Deduction

Prove $(P \to (Q \to S)) \wedge (\neg R \vee P) \wedge Q \Rightarrow R \to S$.

(1) $\neg R \vee P$	*introduce premise*
(2) $R \to P$	*replacement rule*
(3) R	*introduce premise*
(4) P	*Hypothetical reasoning on* (2)(3)
(5) $P \to (Q \to S)$	*introduce premise*
(6) $Q \to S$	*Hypothetical reasoning on* (4)(5)
(7) Q	*introduce premise*
(8) S	*Hypothetical reasoning on* (6)(7)
(9) $R \to S$	*deduction theorem*

条件证明规则

Resolution Reasoning(归结推理法)

Because $A \Longrightarrow B$ iff $A \wedge \neg B$ is a contradiction. We can prove $A \wedge \neg B$ is a contradiction.

Resolution Reasoning

DEFINITION

For formula $A = P \vee Q$ and $B = \neg P \vee R$, $R(A, B) = Q \vee R$ is a resolvent(归结式) of A and B.

- To prove $\boldsymbol{A \Longrightarrow B}$, we just need to prove $\boldsymbol{A \wedge \neg B}$ is a contradiction
- Convert $\boldsymbol{A \wedge \neg B}$ to CNF P1∧P2∧… ∧Pn
- Repeatly produce resolvents from Pi
- Find contradiction and proof ends

$$(\boldsymbol{P} \vee \boldsymbol{Q}) \wedge (\neg \boldsymbol{P} \vee \boldsymbol{R}) \Longrightarrow \boldsymbol{Q} \vee \mathbf{R}$$

Resolution Reasoning

Prove that $(P \rightarrow Q) \wedge P \Rightarrow Q$.

Resolution Reasoning

Prove that $(P \to Q) \wedge P \Rightarrow Q$.

$$(P \to Q) \wedge P \wedge \neg Q = (\neg P \vee Q) \wedge P \wedge \neg Q$$

Resolution Reasoning

Prove that $(P \to Q) \wedge P \Rightarrow Q$.

$$(P \to Q) \wedge P \wedge \neg Q = (\neg P \vee Q) \wedge P \wedge \neg Q$$

$$S = \{\neg P \vee Q, P, \neg Q\}$$

Resolution Reasoning

Prove that $(P \to Q) \wedge P \Rightarrow Q$.

$(P \to Q) \wedge P \wedge \neg Q = (\neg P \vee Q) \wedge P \wedge \neg Q$

$S = \{\neg P \vee Q, P, \neg Q\}$

(1) $\neg P \vee Q$ *introduce premise*

Resolution Reasoning

Prove that $(P \to Q) \wedge P \Rightarrow Q$.

$(P \to Q) \wedge P \wedge \neg Q = (\neg P \vee Q) \wedge P \wedge \neg Q$

$S = \{\neg P \vee Q, P, \neg Q\}$

$(1) \neg P \vee Q$	*introduce premise*
$(2) P$	*introduce premise*

Resolution Reasoning

Prove that $(P \to Q) \wedge P \Rightarrow Q$.

$(P \to Q) \wedge P \wedge \neg Q = (\neg P \vee Q) \wedge P \wedge \neg Q$

$S = \{\neg P \vee Q, P, \neg Q\}$

$(1) \neg P \vee Q$	*introduce premise*
$(2) P$	*introduce premise*
$(3) Q$	*resolution of* (1) *and* (2)

Resolution Reasoning

Prove that $(P \to Q) \wedge P \Rightarrow Q$.

$$(P \to Q) \wedge P \wedge \neg Q = (\neg P \vee Q) \wedge P \wedge \neg Q$$

$$S = \{\neg P \vee Q, P, \neg Q\}$$

$(1) \neg P \vee Q$	*introduce premise*
$(2) P$	*introduce premise*
$(3) Q$	*resolution of* (1) *and* (2)
$(4) \neg Q$	*introduce premise*

Resolution Reasoning

Prove that $(P \to Q) \wedge P \Rightarrow Q$.

$$(P \to Q) \wedge P \wedge \neg Q = (\neg P \vee Q) \wedge P \wedge \neg Q$$

$$S = \{\neg P \vee Q, P, \neg Q\}$$

$(1) \neg P \vee Q$	*introduce premise*
$(2) P$	*introduce premise*
$(3) Q$	*resolution of* (1) *and* (2)
$(4) \neg Q$	*introduce premise*
$(5) \square$	*resolution of* (3) *and* (4)

Exercise

Prove $((P \to Q) \wedge (Q \to R)) \Rightarrow (P \to R)$

证明 先将$(P \to Q) \wedge (Q \to R) \wedge \neg (P \to R)$

化成合取范式

$$(\neg P \vee Q) \wedge (\neg Q \vee R) \wedge P \wedge \neg R$$

建立子句集

$$S = \{\neg P \vee Q, \neg Q \vee R, P, \neg R\}$$

归结过程：

(1) $\neg P \vee Q$	
(2) $\neg Q \vee R$	
(3) P	
(4) $\neg R$	
(5) $\neg P \vee R$	(1) (2) 归结
(6) R	(3) (5) 归结
(7) $\square$	(4) (6) 归结

Dual formula

Duality underlines the world

Dual formula

DEFINITION

Given a formula A, replace $\vee$, $\wedge$, T, F in A with $\wedge$, $\vee$, F, T and get A^*.

Then A and A^* are dual formulas(对偶式) with each other.

Dual formula

THEOREM

$$A = A(P_1, ..., P_n), \quad A^- = A(\neg P_1, ..., \neg P_n)$$

$$\neg(A^*) = (\neg A)^*, \quad \neg(A^-) = (\neg A)^-$$

$$(A^*)^* = A, \quad (A^-)^- = A$$

$$(\neg A) = A^{*-}$$

How to prove these theorems?

Dual formula

THEOREM

$$A = A(P_1, ..., P_n), \quad A^- = A(\neg P_1, ..., \neg P_n)$$

$$\neg(A^*) = (\neg A)^*, \quad \neg(A^-) = (\neg A)^-$$

$$(A^*)^* = A, \quad (A^-)^- = A$$

$$(\neg A) = A^{*-}$$

Recap: induction is a natural way to prove theorems about inductive-defined objects.

Dual formula

THEOREM

$$A = A(P_1, ..., P_n), \quad A^- = A(\neg P_1, ..., \neg P_n)$$

$$\neg(A^*) = (\neg A)^*, \quad \neg(A^-) = (\neg A)^-$$

$$(A^*)^* = A, \quad (A^-)^- = A$$

$$(\neg A) = A^{*-}$$

Proof is on p23 in the textbook.

Dual formula

THEOREM

1) If $A = B$, then $A^* = B^*$

2) If $A \longrightarrow B$ is a tautology, then $B^* \longrightarrow A^*$ is a tautology

3) A is a tautology iff A^- is a tautology

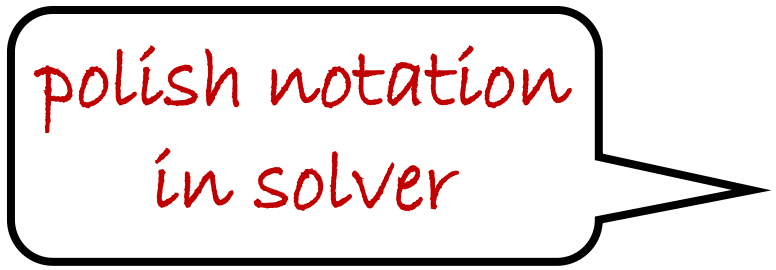
4) A is satisfiable iff A^- is satisfiable

5) $\neg A$ is a tautology iff A^* is a tautology

6) $\neg A$ is satisfiable iff A^* is satisfiable

Polish notation（波兰表达式）

- The previous wff is called infix notation（中缀表达式）
- There is a different notation called polish notation（波兰表达式）or prefix notation（前缀表达式）
- Polish notation can achieve higher performance than infix notation

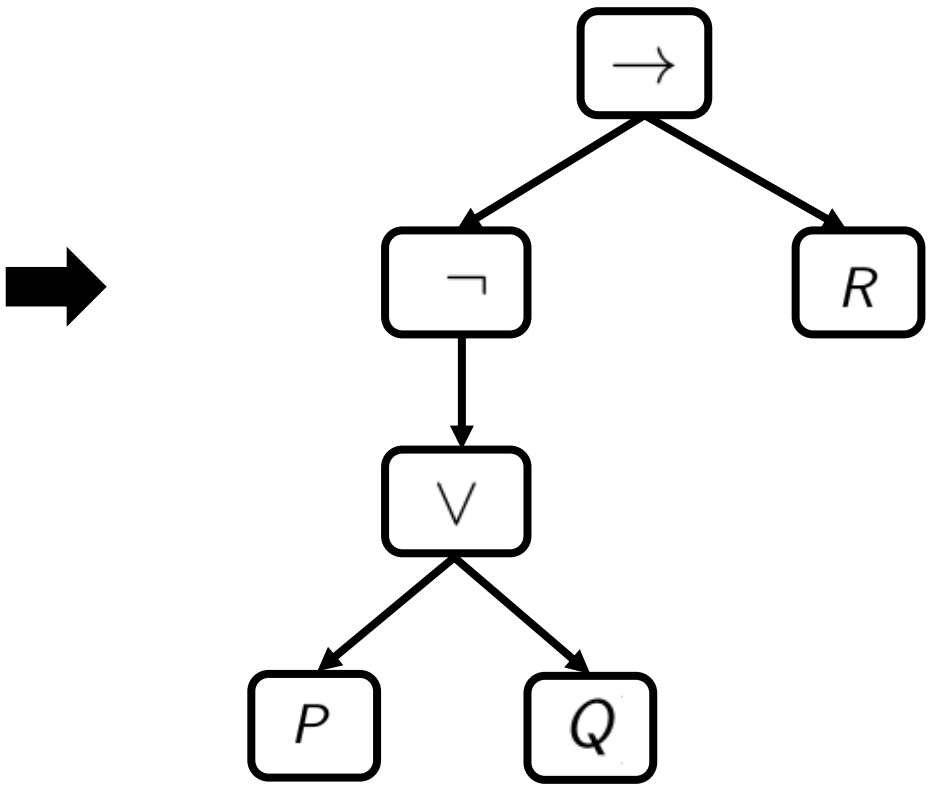



```
(declare-const a Int)
(declare-fun f (Int Bool) Int)
(assert (> a 10))
(assert (< (f a true) 100))
(check-sat)
```

Polish notation

- We show here how to convert the infix notation to polish notation

$((\neg(P \vee Q)) \rightarrow R)$

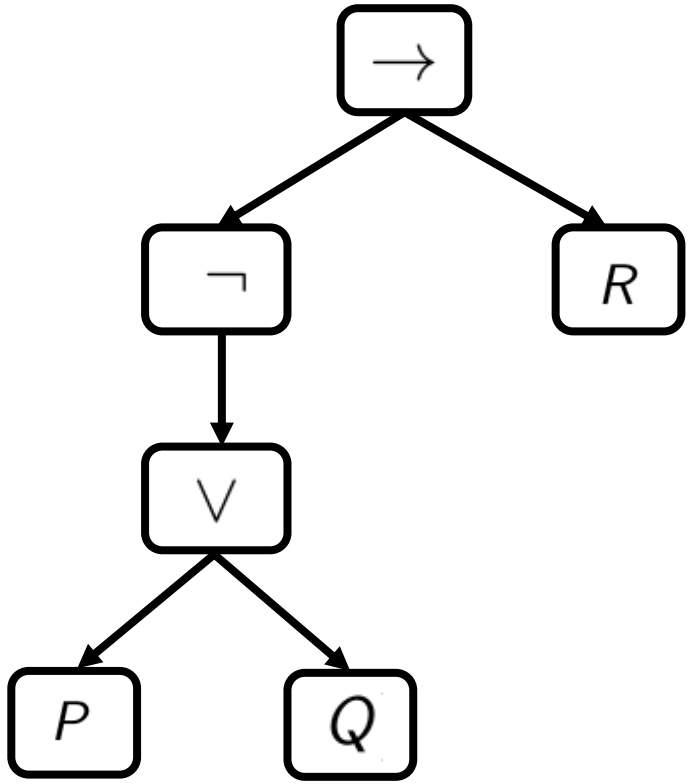


$\rightarrow \neg \vee PQR$

Reverse polish notation（逆波兰表达式）

- There is also a different notation called reverse polish notation（逆波兰表达式）or postfix notation（后缀表达式）

$((\neg(P \vee Q)) \rightarrow R)$ ➡



➡ $PQ \vee \neg R \rightarrow$

Thanks & Questions