SAT Solver

Zhaoguo Wang

Adapted From:

NYU G22.2390-001, Propositional Logic
教材《数理逻辑与集合论》1.4, 2.6
<<Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T)>>

Propositional Logic （命题逻辑）

1.1 Propositional Logic

Step 0. Proof.

Step 1. Convert program into mathematical formula.

Step 2. Ask the computer to solve the formula.

1.2 First Order Logic

Step 1. Convert it into first logic formula.

Step 2. Ask the computer to solve the formula.

1.3 Auto-active Proof

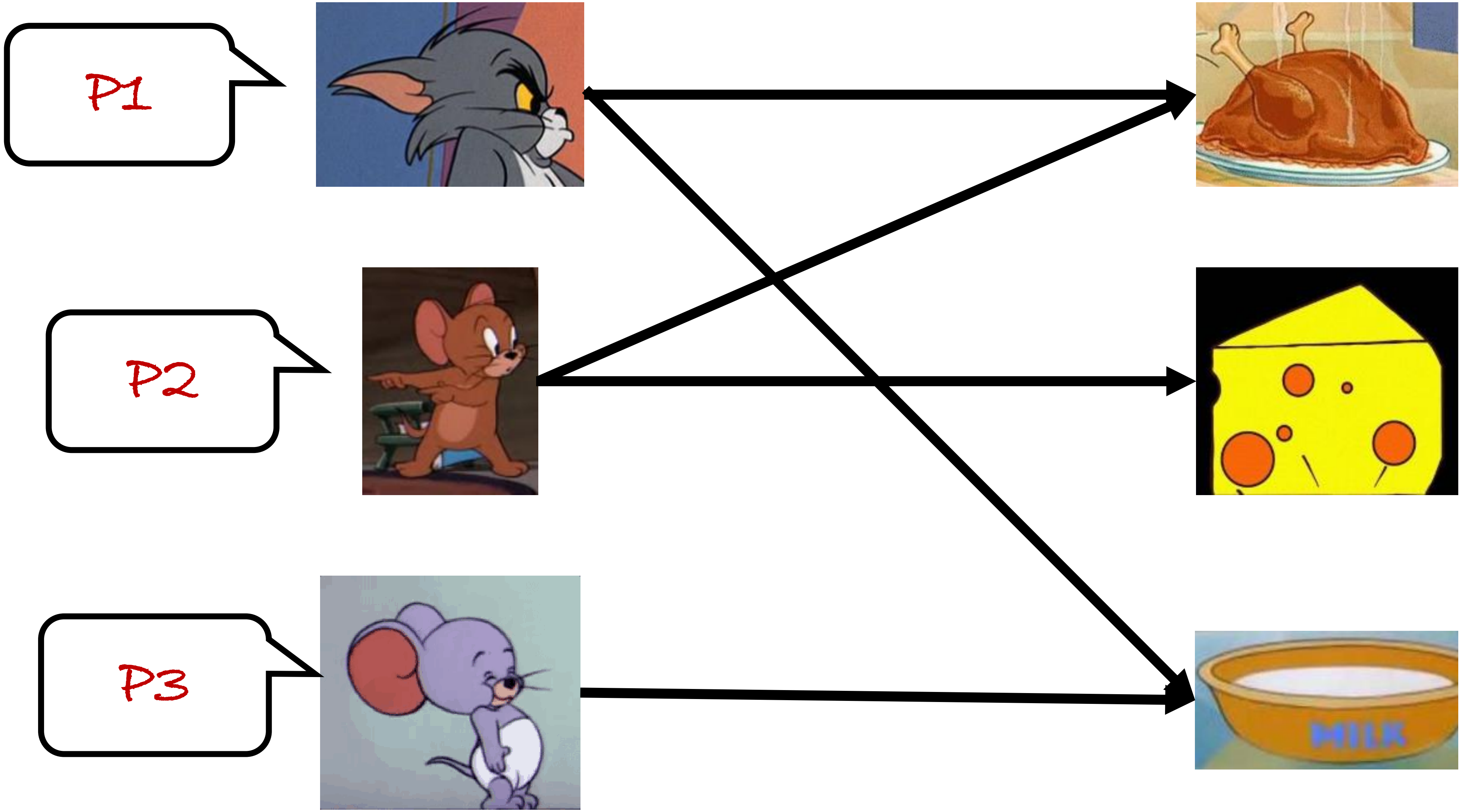
Step 1. Axiom system

Step 2. Ask the computer to check the invariants

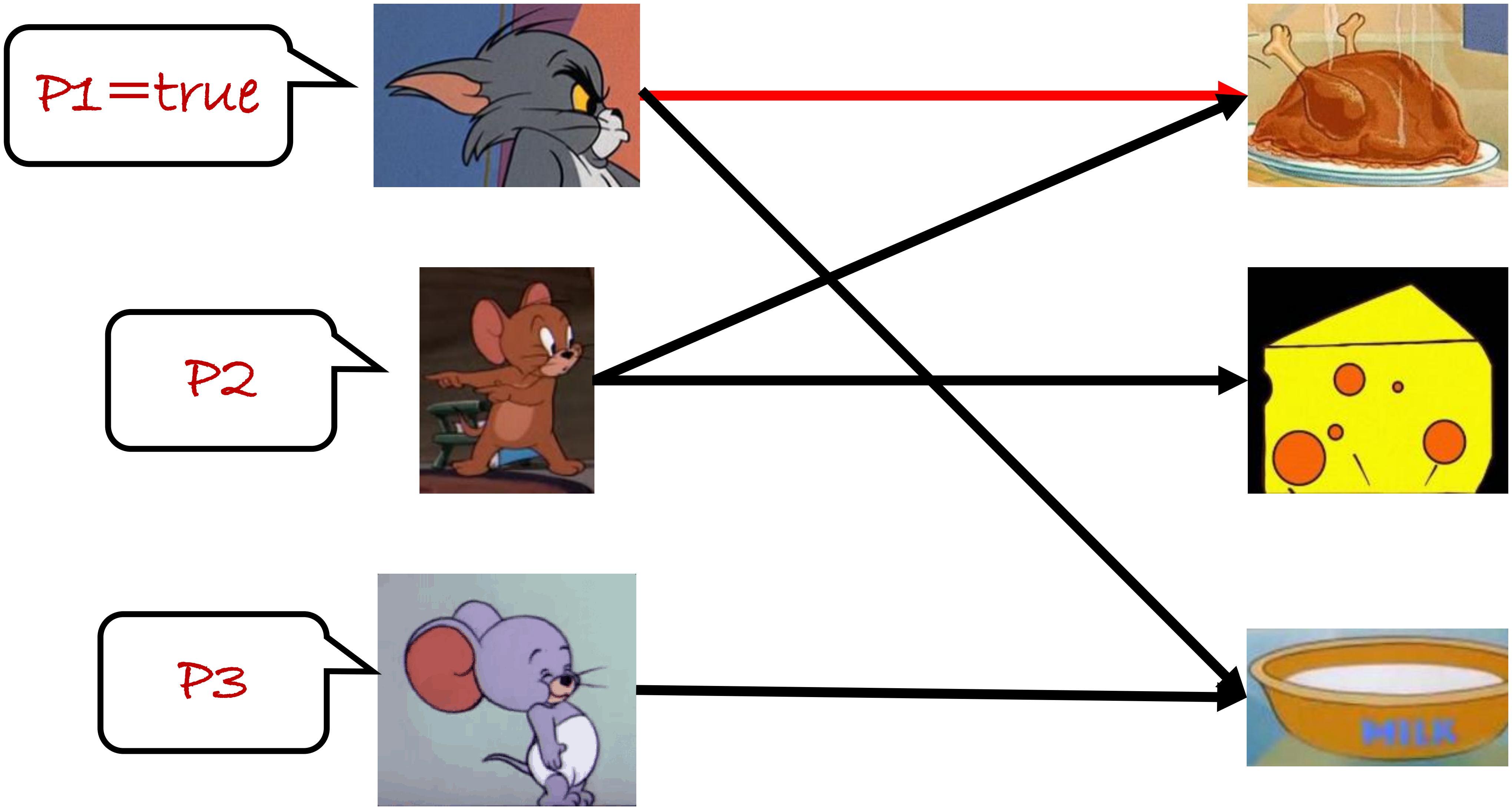
Outline – This Lecture

- SAT Problem
- Normal Form
- DPLL
- Program Analysis

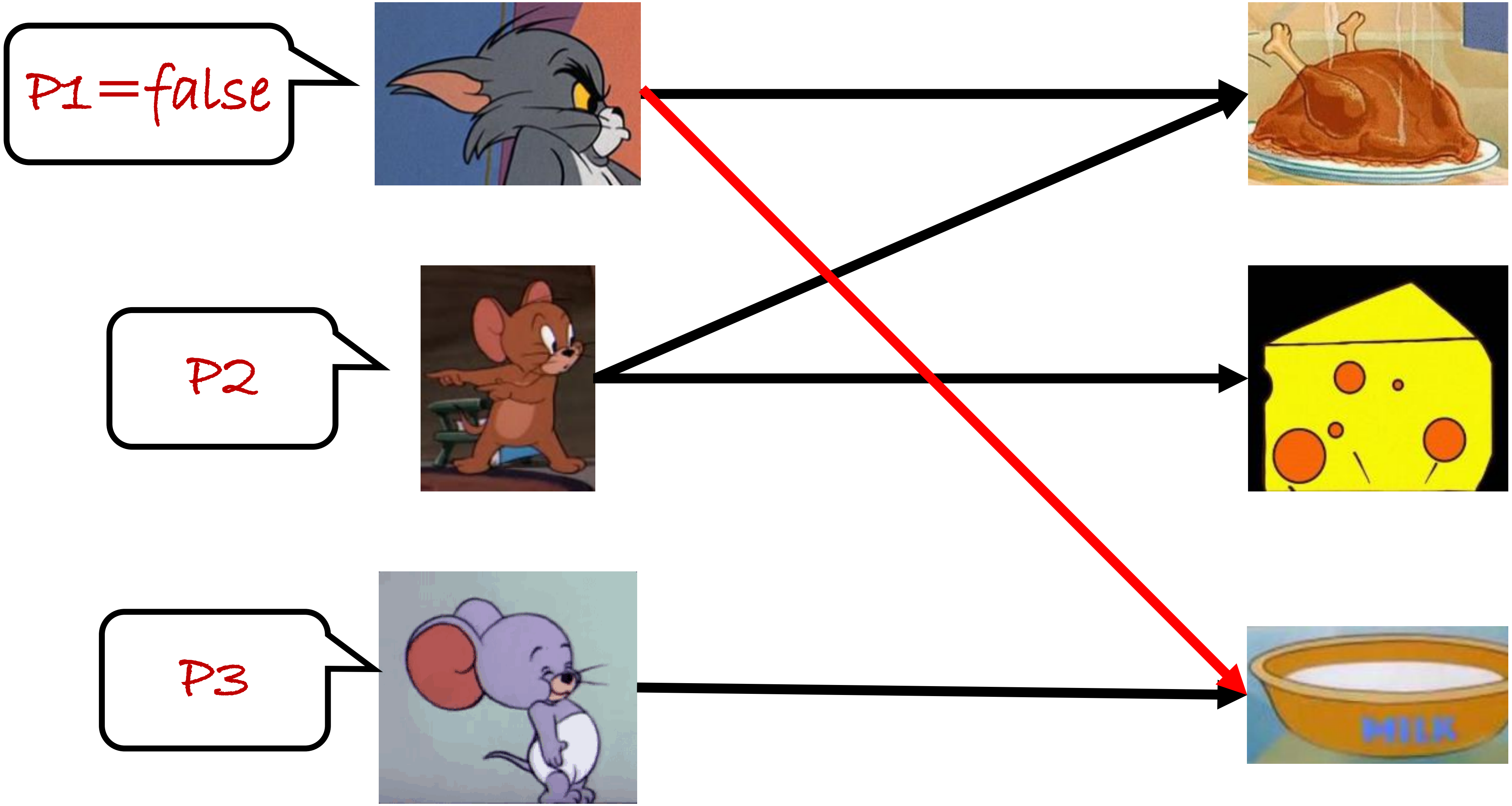
Matching Problem II



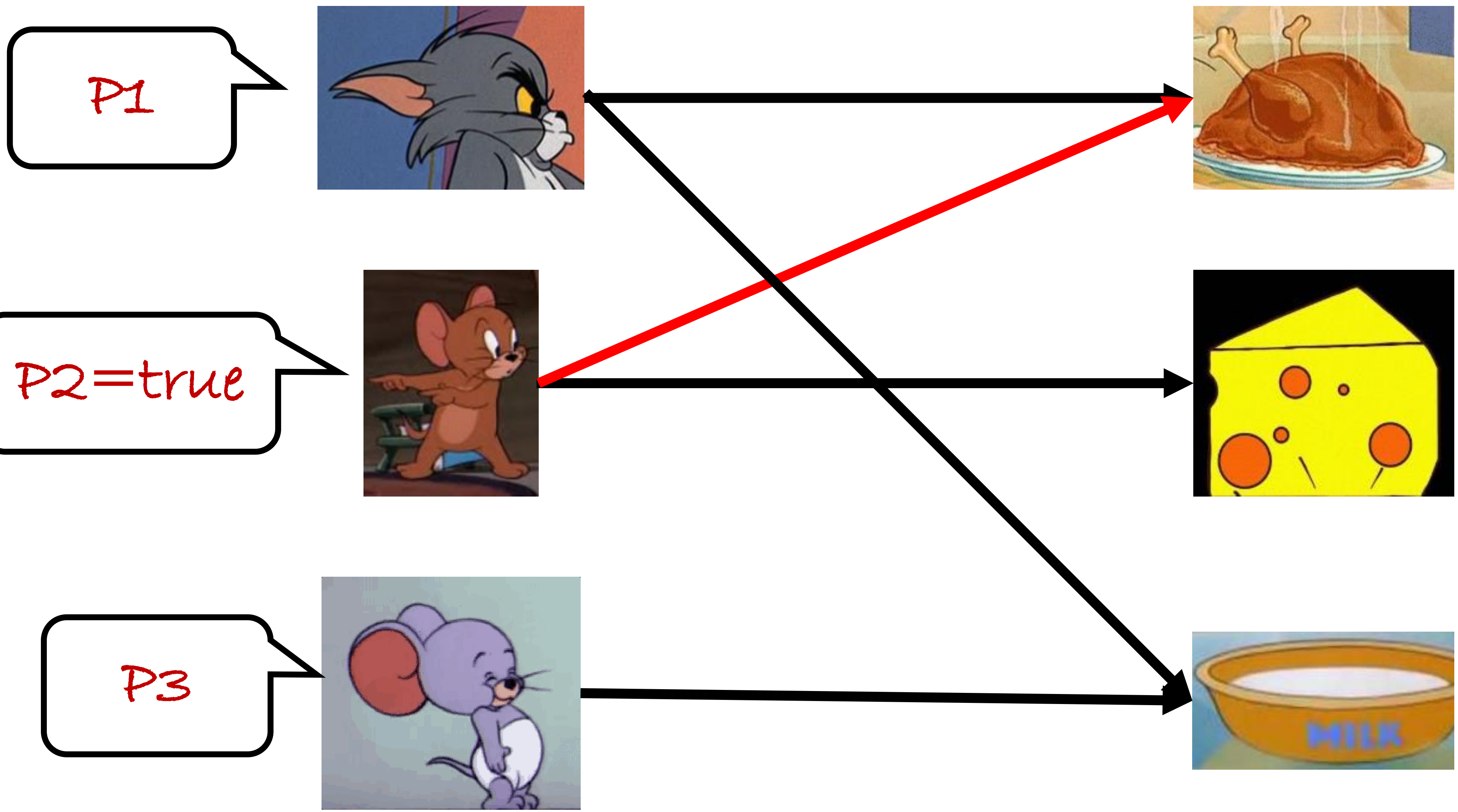
Matching Problem II



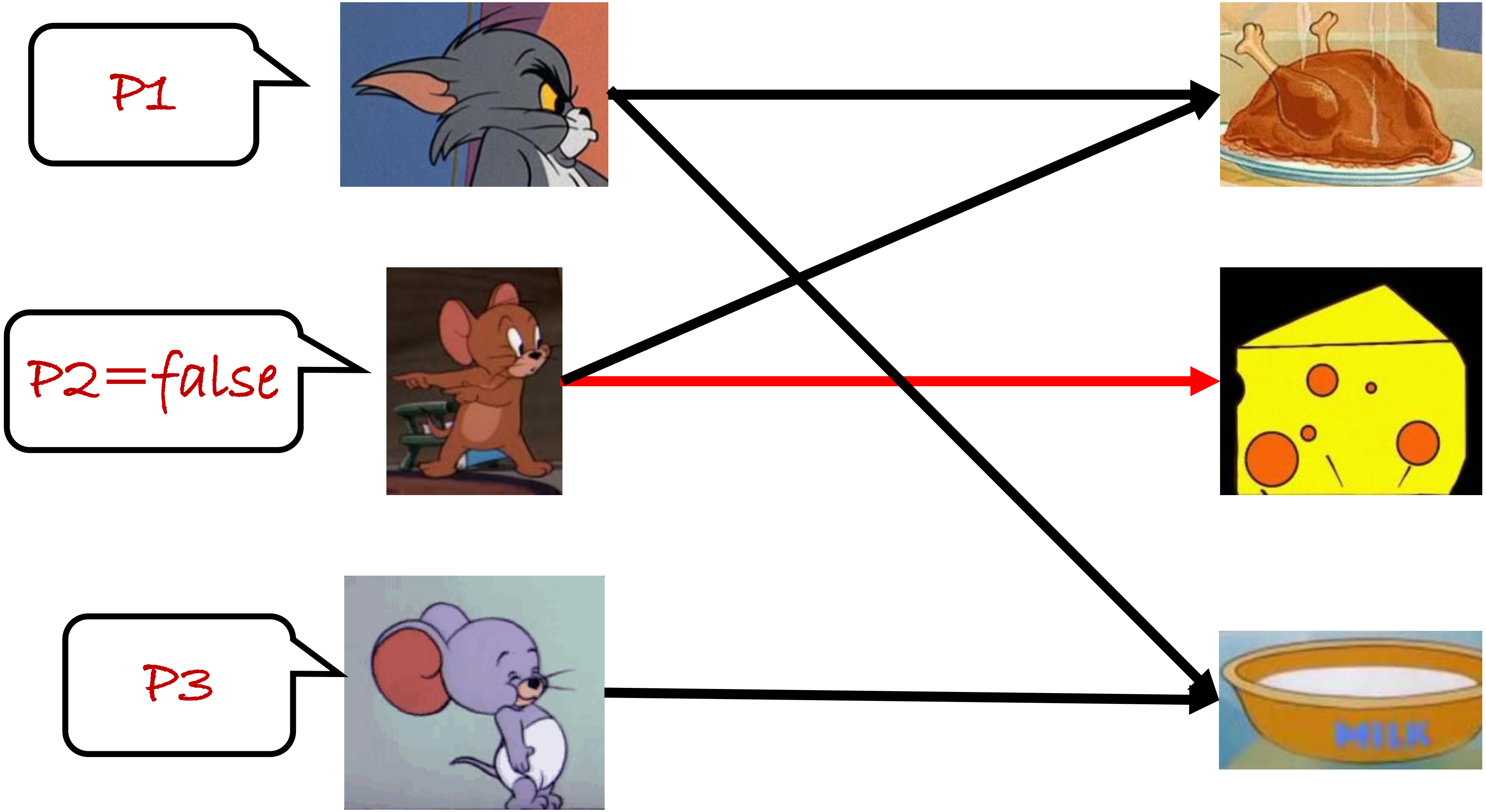
Matching Problem II



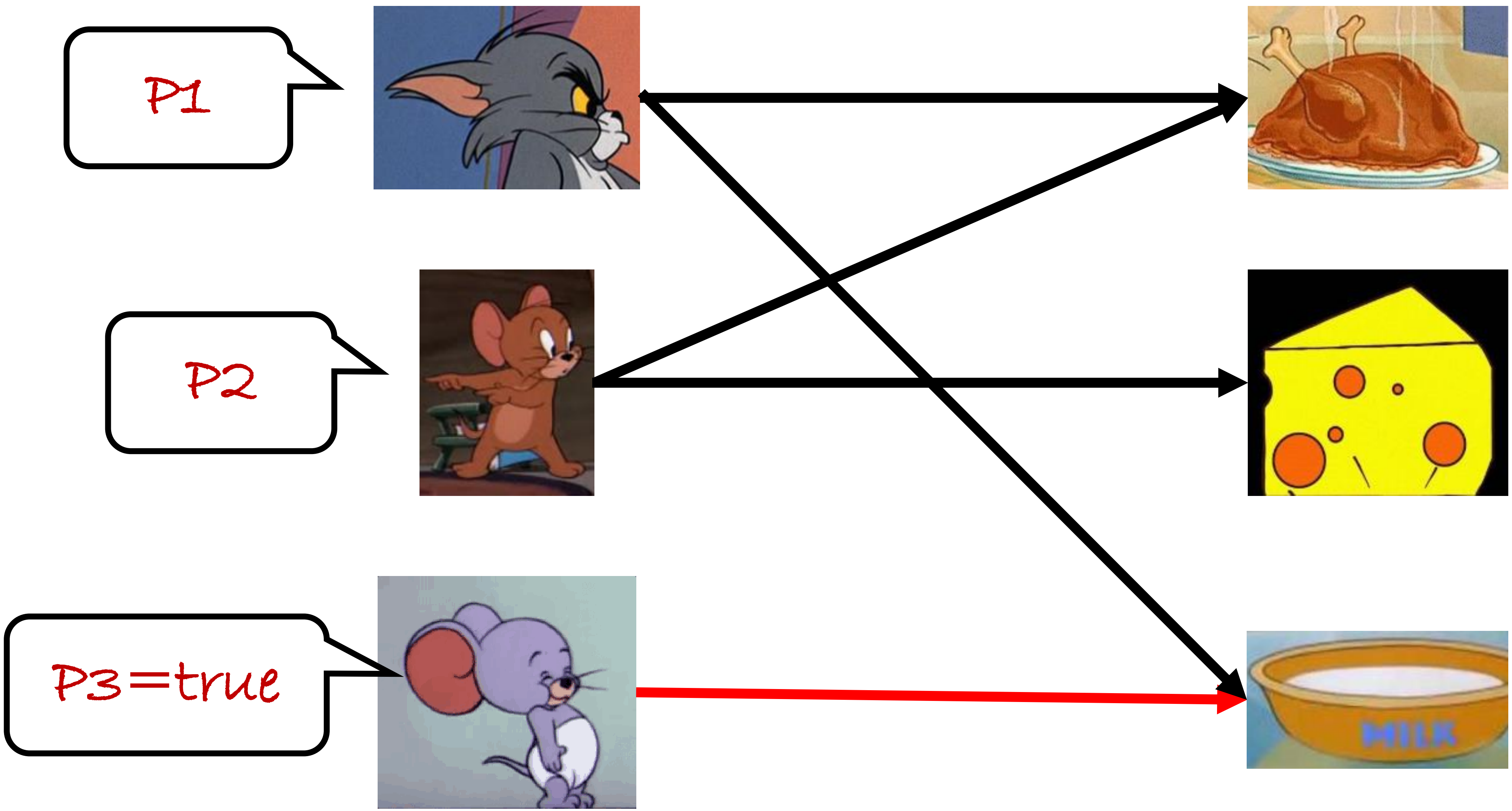
Matching Problem II



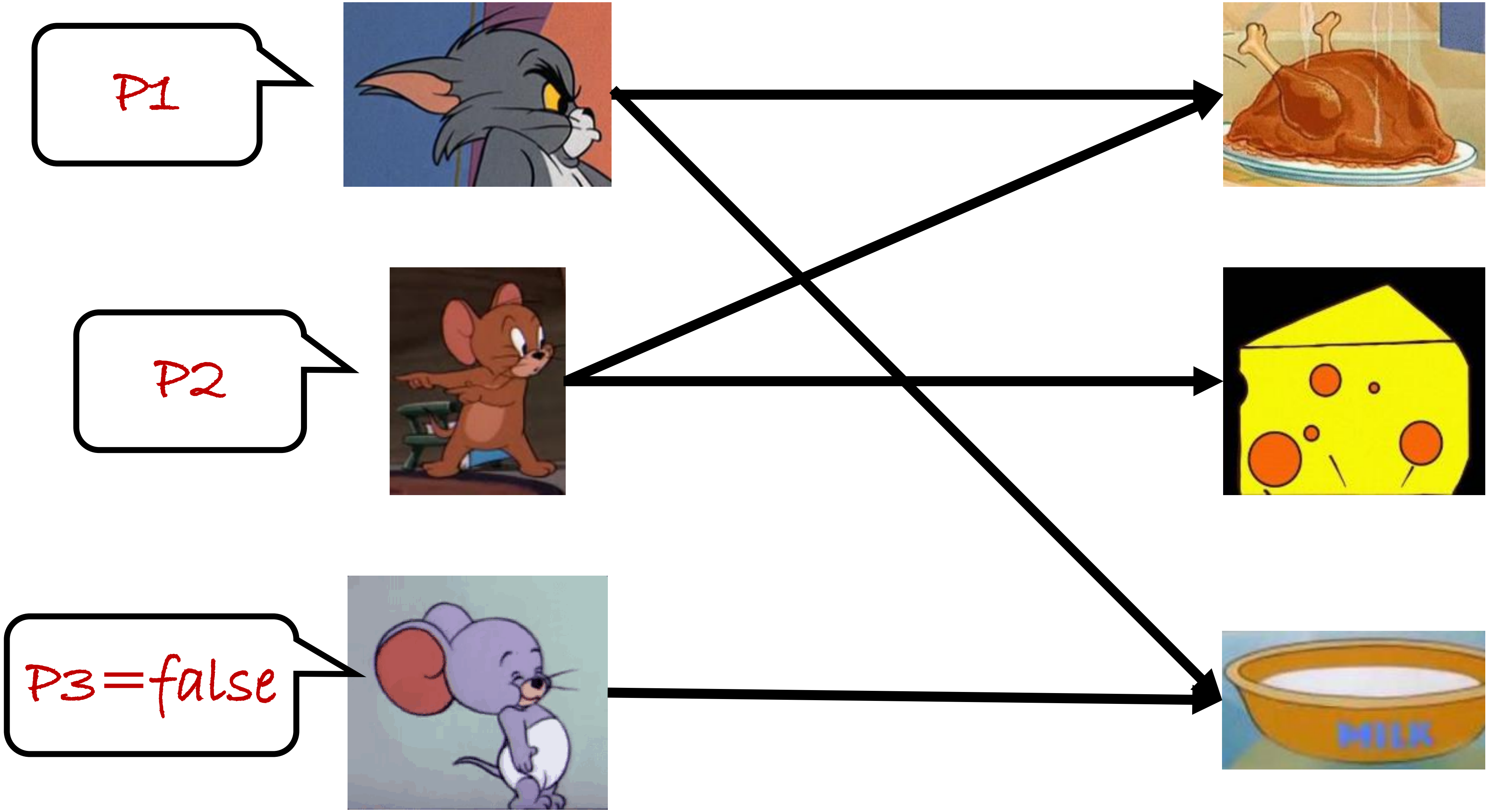
Matching Problem II



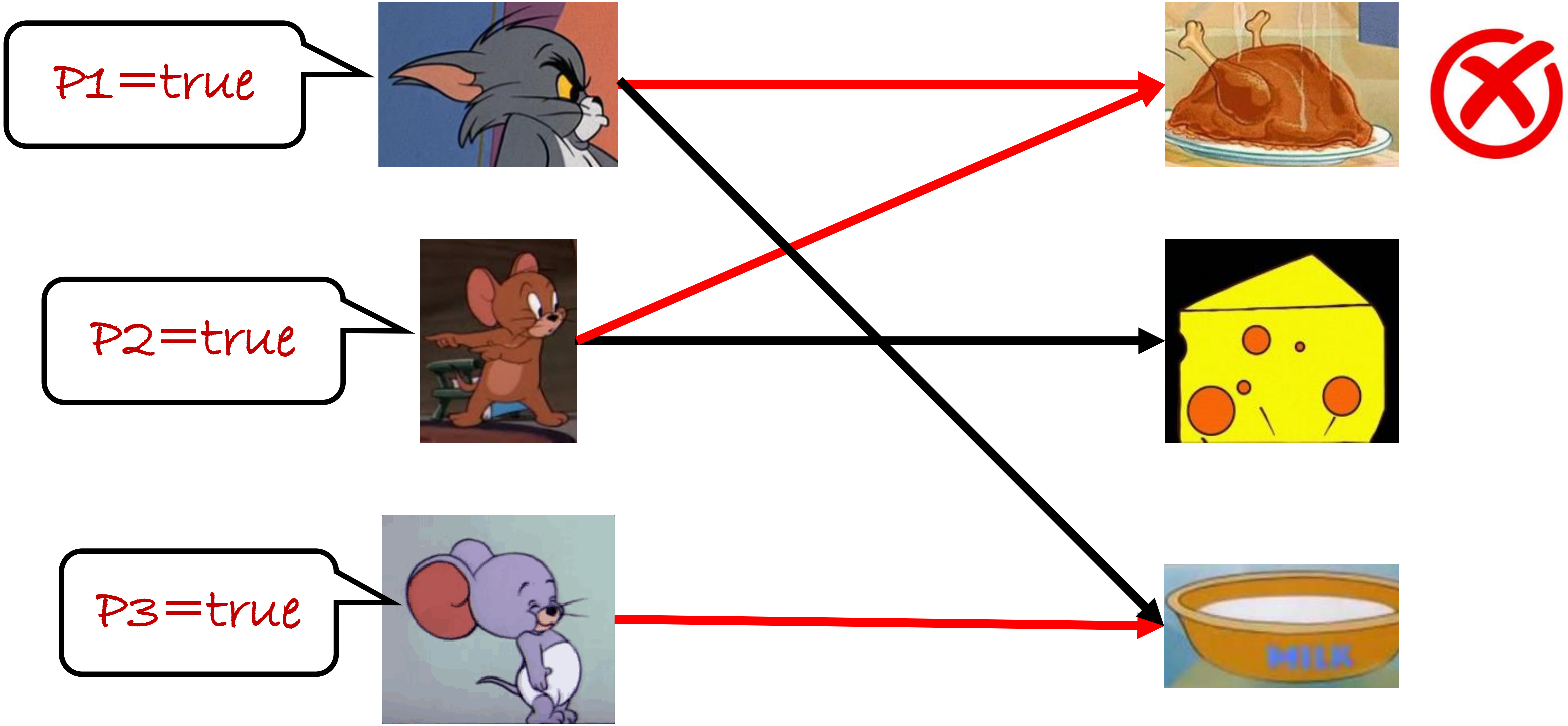
Matching Problem II



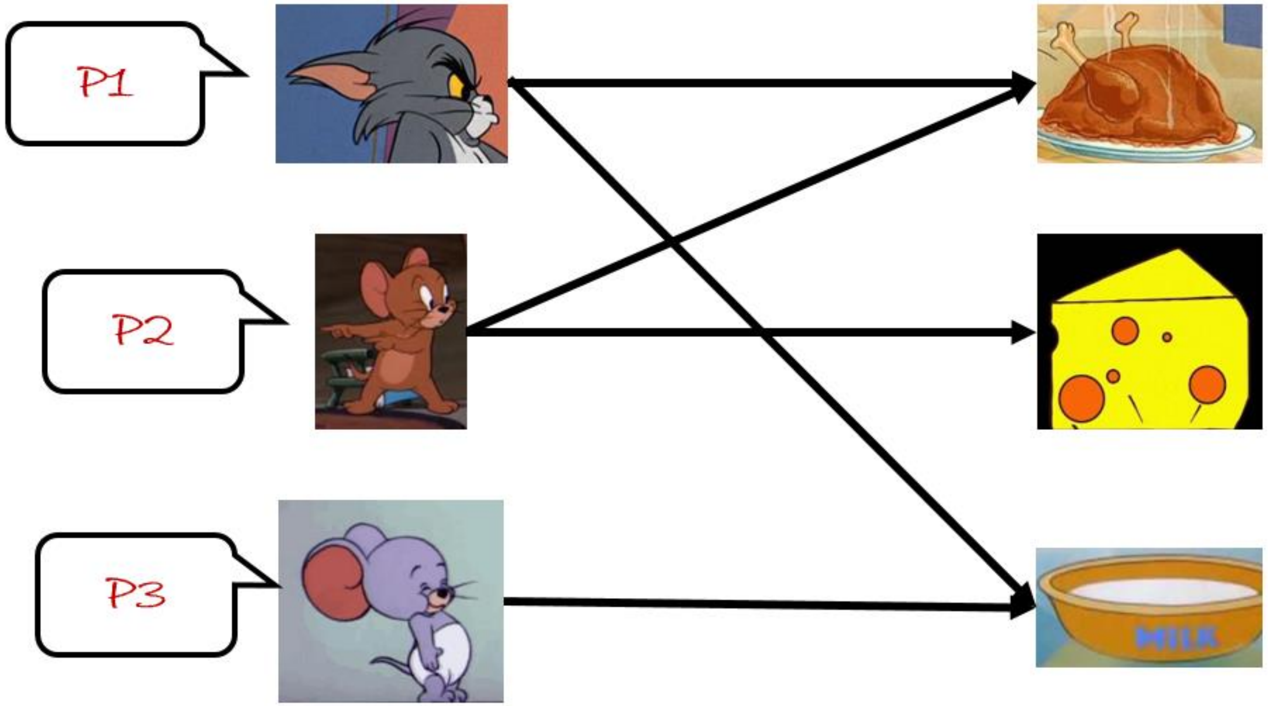
Matching Problem II



Matching Problem II

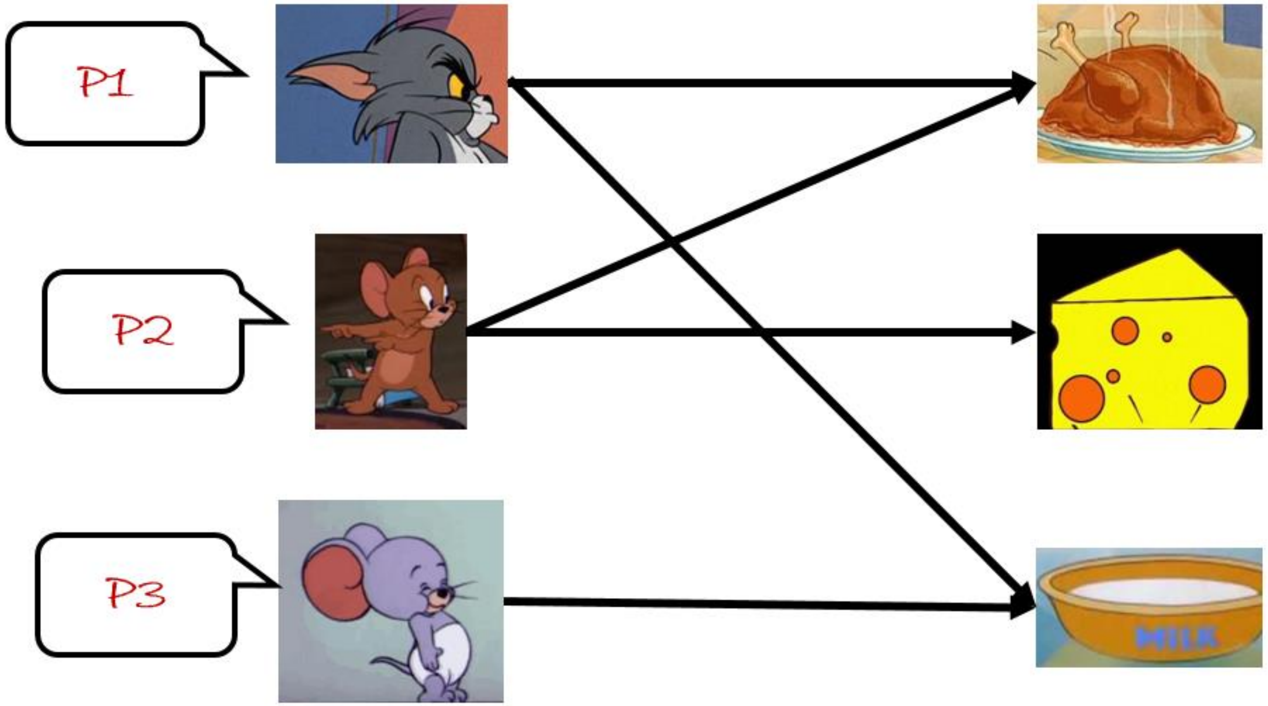


Matching Problem II



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

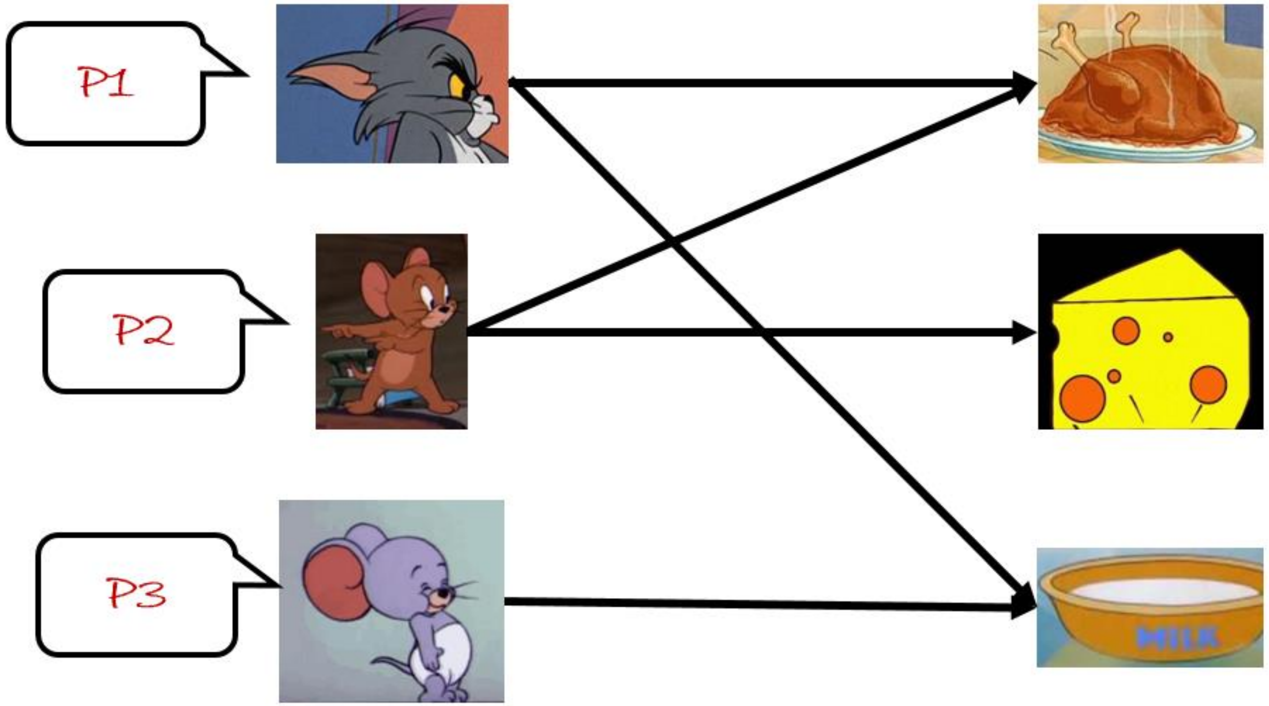
Matching Problem II



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

Nibbles must drink milk.

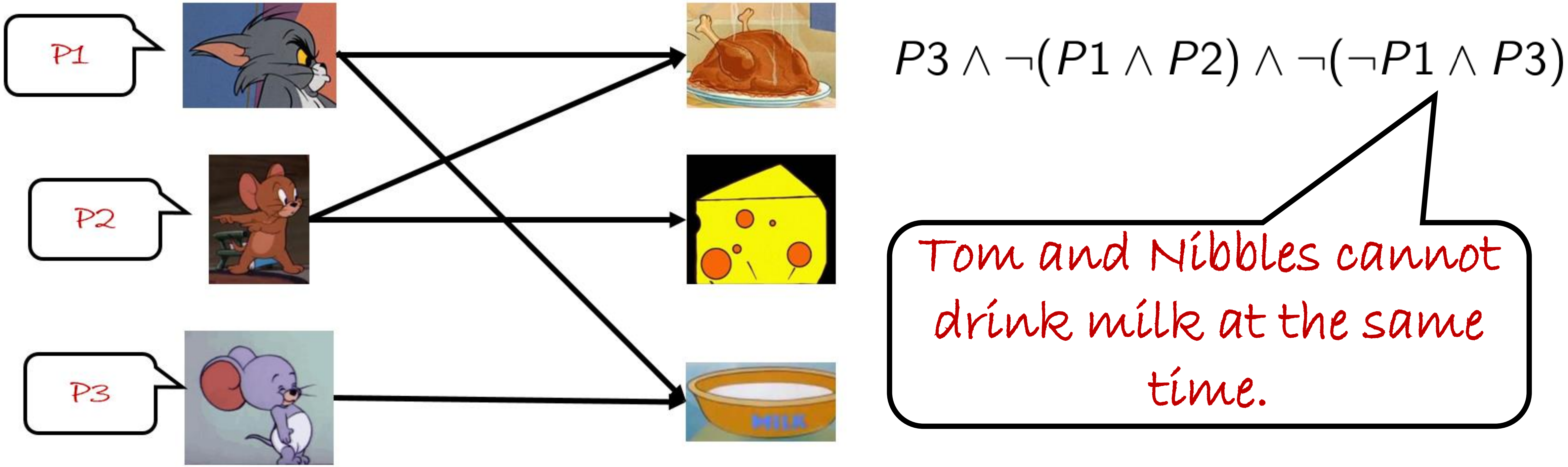
Matching Problem II



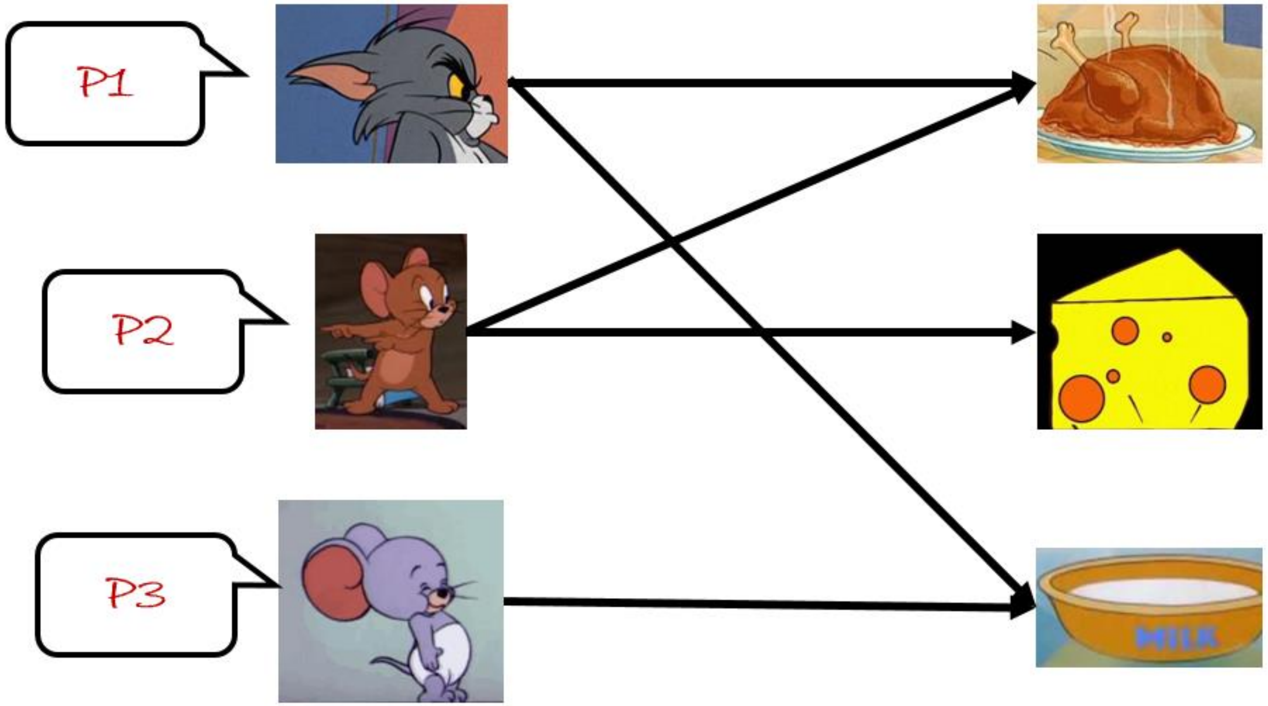
$$P3 \wedge \neg(P1 \wedge P2) \wedge \neg(\neg P1 \wedge P3)$$

Tom and Jerry cannot eat chicken at the same time.

Matching Problem II



Matching Problem II



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

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

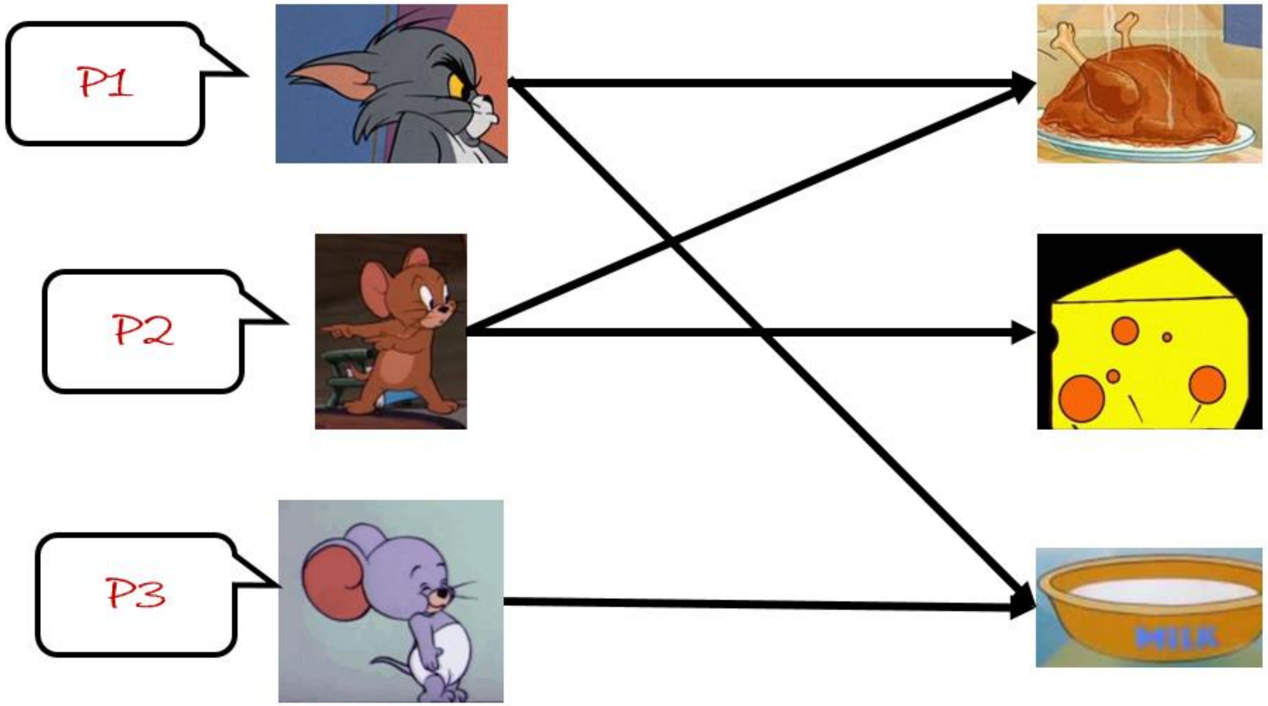
$$= P3 \wedge (P1 \vee \neg P3) \wedge (\neg P1 \vee \neg P2)$$

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

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

It is clear that P1=T,
P2=F and P3=T.

Matching Problem II



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

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

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

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

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

But it needs to be implemented by hand, which is not automatic.

Instead of proving tautology or contradiction, sometimes we just want to find an interpretation to make a formula true.

Satisfiability （可满足性）

DEFINITION

If a proposition can be true under some interpretation, it is satisfiable.

RELATION

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

2) P is satisfiable iff ¬P is not a tautology

3) P is unsatisfiable iff P is a contradiction

Satisfiability （可满足性）

RELATION

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

2) P is unsatisfiable iff P is a contradiction

So proving P is a tautology is to prove that ¬P is unsatisfiable.

Example

$(A \vee B) \wedge (\neg A \vee \neg B)$ **satisfiable, not tautology**

$(A \vee B) \wedge (\neg A \vee \neg B) \wedge (A \leftrightarrow B)$ **unsatisfiable/contradiction**

$(P \wedge \neg P) \to (P \wedge \neg P)$ **tautology**

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

Example

$(A \vee B) \wedge (\neg A \vee \neg B)$ **satisfiable, not tautology**

$(A \vee B) \wedge (\neg A \vee \neg B) \wedge (A \leftrightarrow B)$ **unsatisfiable/contradiction**

$(P \wedge \neg P) \rightarrow (P \wedge \neg P)$ **tautology**

$A \wedge (A \rightarrow B) \rightarrow B$ **tautology**

To prove it, we can prove
$A \wedge (A \rightarrow B) \wedge \neg B$ is unsatisfiable.

Step 2. Ask the computer to determine the satisfiability.

Step 2.1. **Validate** the formula.

Step 2.2. **Solve** the formula.

Quick Recap

INDUCTIVE DEFINITION of WFF

1). Every single proposition (symbol) is in WFF.

2). If A and B are WFF, so are $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, and $(A \leftrightarrow B)$.

3). No expression is WFF unless forced by 1) or 2).

$(\neg P)$

$(P \wedge Q)$

$(P \vee Q)$

$(P \rightarrow Q)$

$(P \leftrightarrow Q)$

WFF

Recognize well-formed formulas

Inductive Definition (WWF) $\Longleftrightarrow$ Recursive Function

Recursive Function

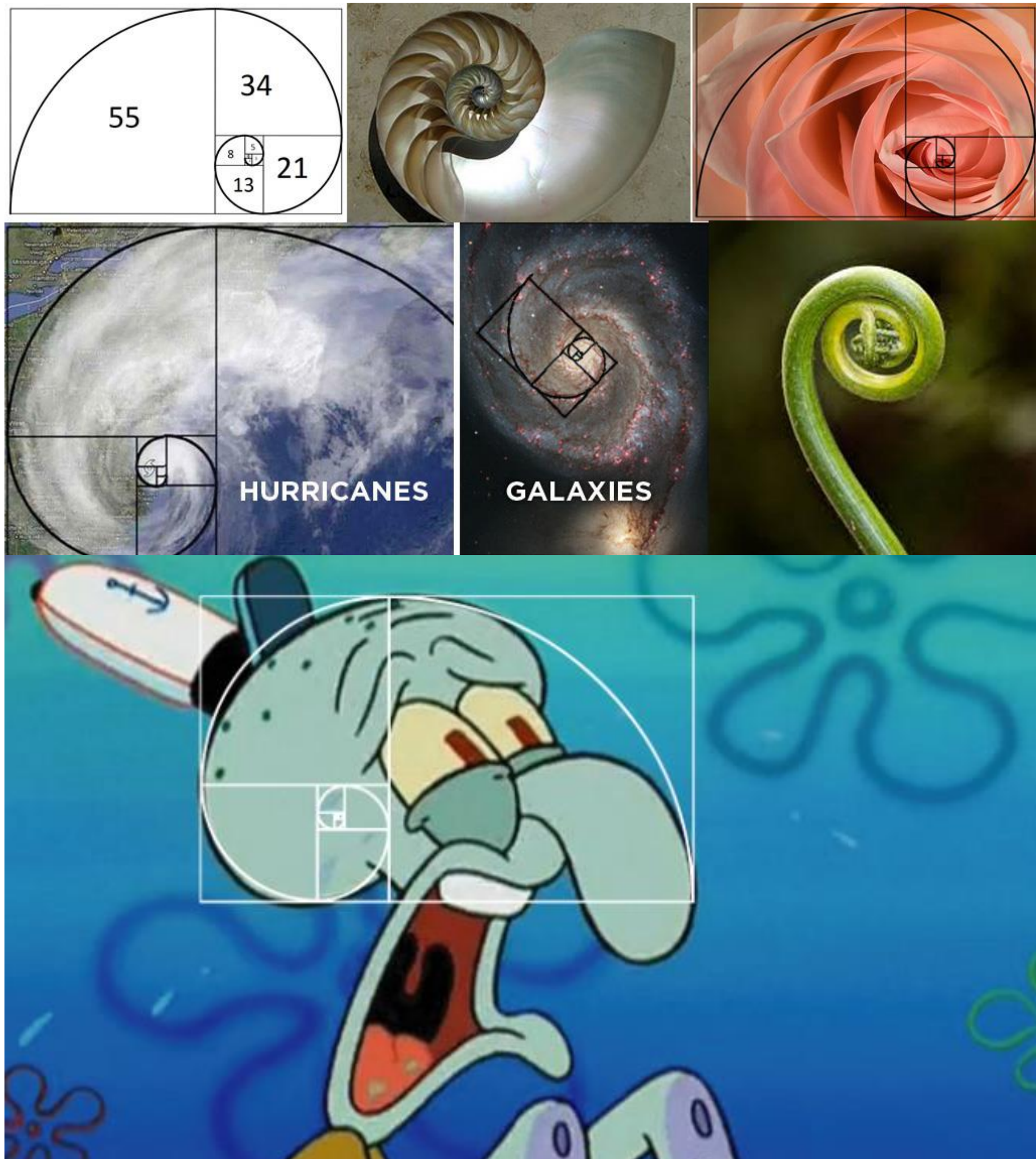
Operation for base elements.

Operation for elements built by elements-building operations.

Recursion（递归）

Fibonacci sequence function

- F(0) = 1
- F(1) = 1
- F(n) = F(n-1) + F(n-2)



Recognizing Well-formed Formulas

Input: a formula P; Output: true or false (indicating whether it is a wff)

0. Begin with an initial tree T containing a single node labeled with P

1. If all leaves of T are labeled with propositional variables, return true

2. Select a leaf labeled with an expression f which is not a propositional variable

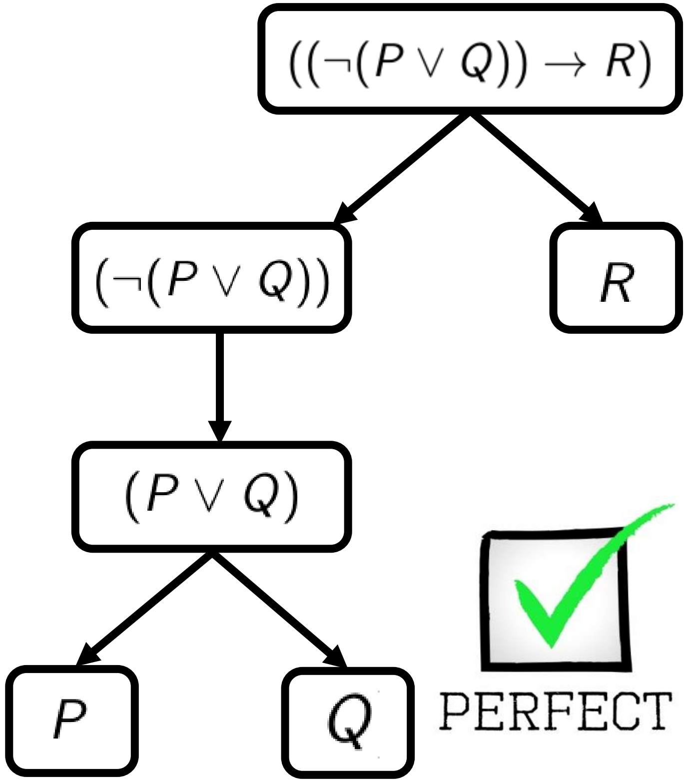
3. If f does not begin with (, return *false.* If f does not end with),return false.

4. If f = (¬Q), then add a child to the leaf labeled by f, label it with Q, and goto 1

5. Let f = (F), scan F until first reaching A, where A is a nonempty expression having the same number of left and right parentheses. If there is no such A, return false

6. If f = (A ⊙ B) where ⊙ is one of {∧, ∨,→,↔}, then add two children to the leaf labeled by f, label them with A and B, and goto 1

7. Return false



Recognizing Well-formed Formulas

Input: a formula P; Output: true or false (indicating whether it is a wff)

0. Begin with an initial tree T containing a single node labeled with P

1. If all leaves of T are labeled with propositional variables, return true

2. Select a leaf labeled with an expression f which is not a propositional variable

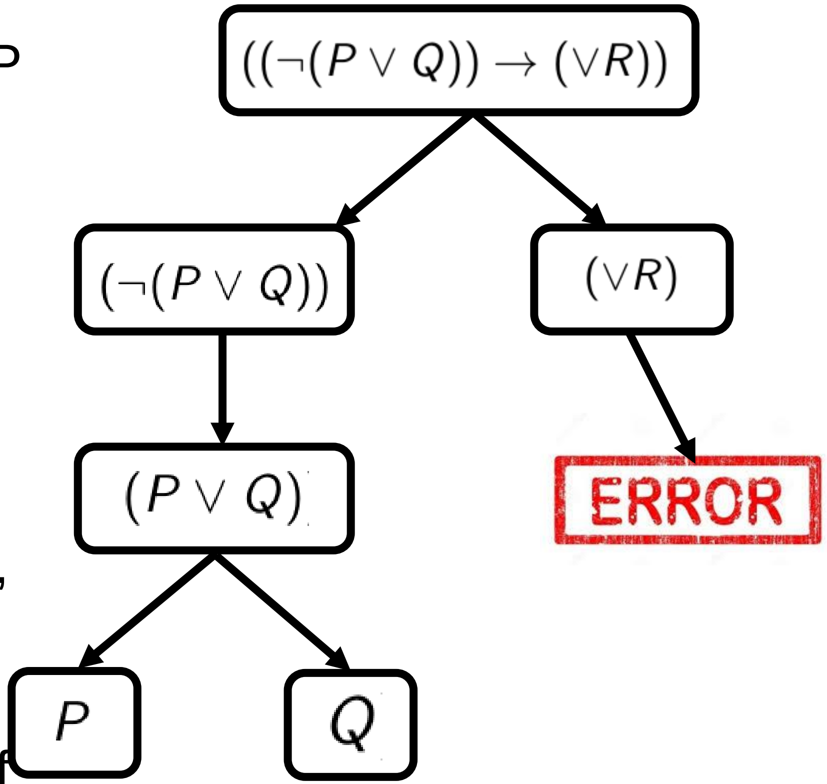
3. If f does not begin with (, return *false.* If f does not end with),return false.

4. If f = (¬Q), then add a child to the leaf labeled by f, label it with Q, and goto 1

5. Let f = (F), scan F until first reaching A, where A is a nonempty expression having the same number of left and right parentheses. If there is no such A, return false

6. If f = (A ⊙ B) where ⊙ is one of {∧, ∨,→,↔}, then add two children to the leaf labeled by f, label them with A and B, and goto 1

7. Return false



Recognizing Well-formed Formulas

Input: a formula P; Output: true or false (indicating whether it is a wff)

0. Begin with an initial tree T containing a single node labeled with P

1. If all leaves of T are labeled with propositional variables, return true

2. Select a leaf labeled with an expression f which is not a propositional variable

3. If f does not begin with (, return *false.* If f does not end with),return false.

4. If f = (¬Q), then add a child to the leaf labeled by f, label it with Q, and goto 1

5. Let f = (F), scan F until first reaching A, where A is a nonempty expression having the same number of left and right parentheses. If there is no such A, return false

6. If f = (A ⊙ B) where ⊙ is one of {∧, ∨,→,↔}, then add two children to the leaf labeled by f, label them with A and B, and goto 1

7. Return false

$$\boxed{((\neg(P \vee Q)))} \rightarrow R)$$

It is not a logical connective

Step 2. Ask the computer to determine the satisfiability.

Step 2.1. **Validate** the formula.

Step 2.2. **Solve** the formula.

Truth table? It is direct and simple.

Determining Satisfiability using Truth Tables

$$A \wedge (B \vee \neg A) \wedge (C \vee \neg B)$$

A	B	C	A	$\wedge$	$((B$	$\vee$	$\neg A)$	$\wedge$	$(C$	$\vee$	$\neg B))$

Determining Satisfiability using Truth Tables

$$A \wedge (B \vee \neg A) \wedge (C \vee \neg B)$$

A	B	C	A	$\wedge$	$((B$	$\vee$	$\neg A)$	$\wedge$	$(C$	$\vee$	$\neg B))$
F	**F**	**F**		**F**		**T**	**T**	**T**		**T**	**T**

Determining Satisfiability using Truth Tables

$$A \wedge (B \vee \neg A) \wedge (C \vee \neg B)$$

A	B	C	A	$\wedge$	$((B$	$\vee$	$\neg A)$	$\wedge$	$(C$	$\vee$	$\neg B))$
F	**F**	**F**		**F**		**T**	**T**	**T**		**T**	**T**
F	**F**	**T**		**F**		**T**	**T**	**T**		**T**	**T**

Determining Satisfiability using Truth Tables

$$A \wedge (B \vee \neg A) \wedge (C \vee \neg B)$$

A	B	C	A	$\wedge$	$((B$	$\vee$	$\neg A)$	$\wedge$	$(C$	$\vee$	$\neg B))$
F	**F**	**F**		**F**		**T**	**T**	**T**		**T**	**T**
F	**F**	**T**		**F**		**T**	**T**	**T**		**T**	**T**
F	**T**	**F**		**F**		**T**	**T**	**F**		**F**	**F**
F	**T**	**T**		**F**		**T**	**T**	**T**		**T**	**F**
T	**F**	**F**		**F**		**F**	**F**	**F**		**T**	**T**
T	**F**	**T**		**F**		**F**	**F**	**F**		**T**	**T**
T	**T**	**F**		**F**		**T**	**F**	**F**		**F**	**F**
T	**T**	**T**		**T**		**T**	**F**	**T**		**T**	**F**

Determining Satisfiability using Truth Tables

$$A \wedge (B \vee \neg A) \wedge (C \vee \neg B)$$

A	B	C	A	$\wedge$	$((B$	$\vee$	$\neg A)$	$\wedge$	$(C$	$\vee$	$\neg B))$
F	**F**	**F**		**F**		**T**	**T**	**T**		**T**	**T**
F	**F**	**T**		**F**		**T**	**T**	**T**		**T**	**T**
F	**T**	**F**		**F**		**T**	**T**	**F**		**F**	**F**
F	**T**	**T**		**F**		**T**	**T**	**T**		**T**	**F**
T	**F**	**F**		**F**		**F**	**F**	**F**		**T**	**T**
T	**F**	**T**		**F**		**F**	**F**	**F**		**T**	**T**
T	**T**	**F**		**F**		**T**	**F**	**F**		**F**	**F**
T	**T**	**T**		**T**		**T**	**F**	**T**		**T**	**F**

Determining Satisfiability using Truth Tables

Given n variables, What is the complexity of truth table?

Determining Satisfiability using Truth Tables

Given n variables, What is the complexity of truth table?

2^n !!!

It is really a very difficult problem.

SAT（可满足性问题）

DEFINITION

In computer science, the satisfiability problem (abbreviated SATISFIABILITY or SAT) is the problem of determining if there exists an interpretation that satisfies a given formula.

The first problem that was proven to be NP-complete (Cook-Levin theorem)

SAT（可满足性问题）

DEFINITION

In computer science, the satisfiability problem (abbreviated SATISFIABILITY or SAT) is the problem of determining if there exists an interpretation that satisfies a given formula.

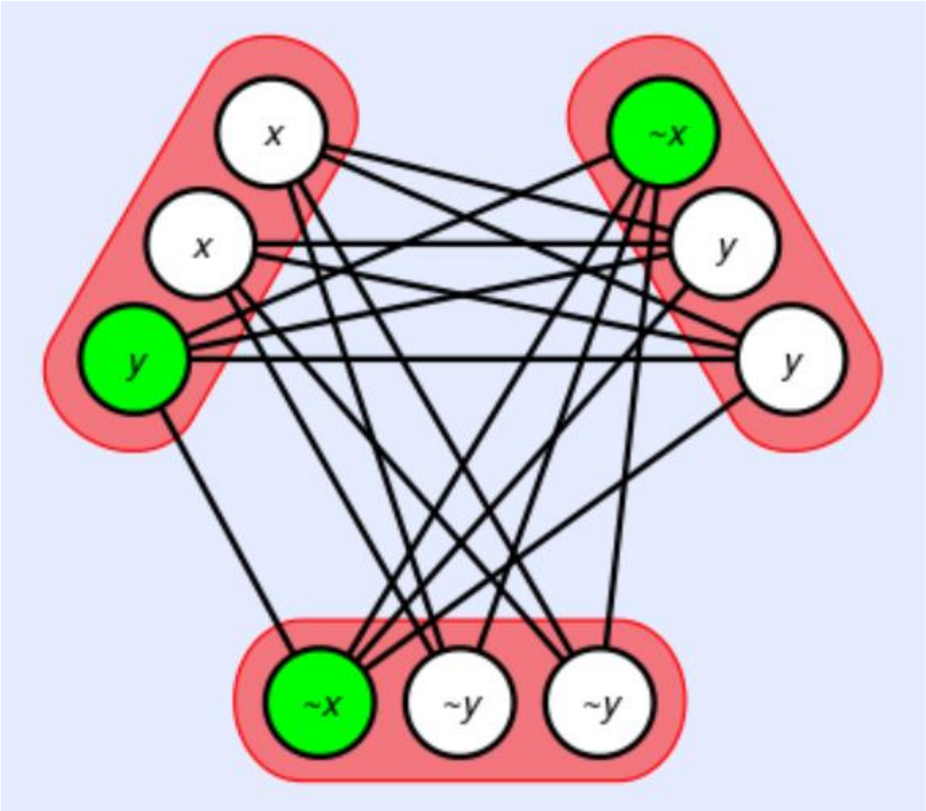
S. A. Cook.
The Complexity of Theorem Proving Procedures.
Proceedings of the Third Annual ACM Symposium on the Theory of Computing, 151-158, 1971.

Stephen Cook

Leonid Levin

Is there a way to speed up the algorithm?

SAT solver can do it!



http://www.satcompetition.org/

Normal Form（范式）

OBSERVATION

Given a wff, you can build infinite formulas that are equivalent to it.

$$A \to B, \neg A \vee B, \neg(A \wedge \neg B)$$

They are in different form
but they are equivalent.

Normal Form（范式）

OBSERVATION

Given a wff, you can build infinite formulas that are equivalent to it.

$$A \to B, \neg A \vee B, \neg(A \wedge \neg B)$$

If we can require a normal form, it is good for designing algorithm for SAT problem.

Normal Form（范式）

DEFINITION

A literal（文字）is a propositional variable or its negation.

A clause（析取式/子句）is a disjunction of one or more literals.

A conjunctive clause（合取式） is a conjunction of one or more literals.

A	literal
$\neg B$	literal
$\neg A \vee B$	disjunctive clause
$A \wedge \neg B$	conjunctive clause

Conjunction Normal Form（合取范式）

DEFINITION

A formula is in conjunctive normal form (CNF) or clausal normal form if it is a conjunction of one or more clauses. Otherwise put, it is an AND of ORs.

Modern SAT solvers use CNF.

$$(A \lor \neg B) \land (B \lor \neg A) \land A$$

$$A \land B$$

Converting to CNF

However, many problems are not in CNF. How to convert them to CNF?

Converting to CNF

P	Q	$g_0(P,Q)$	$g_1(P,Q)$
F	F	T	T
F	T	T	T
T	F	F	$\boxed{F}$
T	T	T	$\boxed{F}$

Step1: find all false rows

Step2: generate a formula for every row

Step3: use "and" to connect these formulas

$$g_1(P,Q) = ((\neg P) \vee Q) \wedge ((\neg P) \vee (\neg Q))$$

It is in CNF! Truth table can help convert formulas to CNF.

Converting to CNF

P	Q	$g_0(P,Q)$	$g_1(P,Q)$
F	F	T	T
F	T	T	T
T	F	F	F
T	T	T	F

Step1: find all false rows

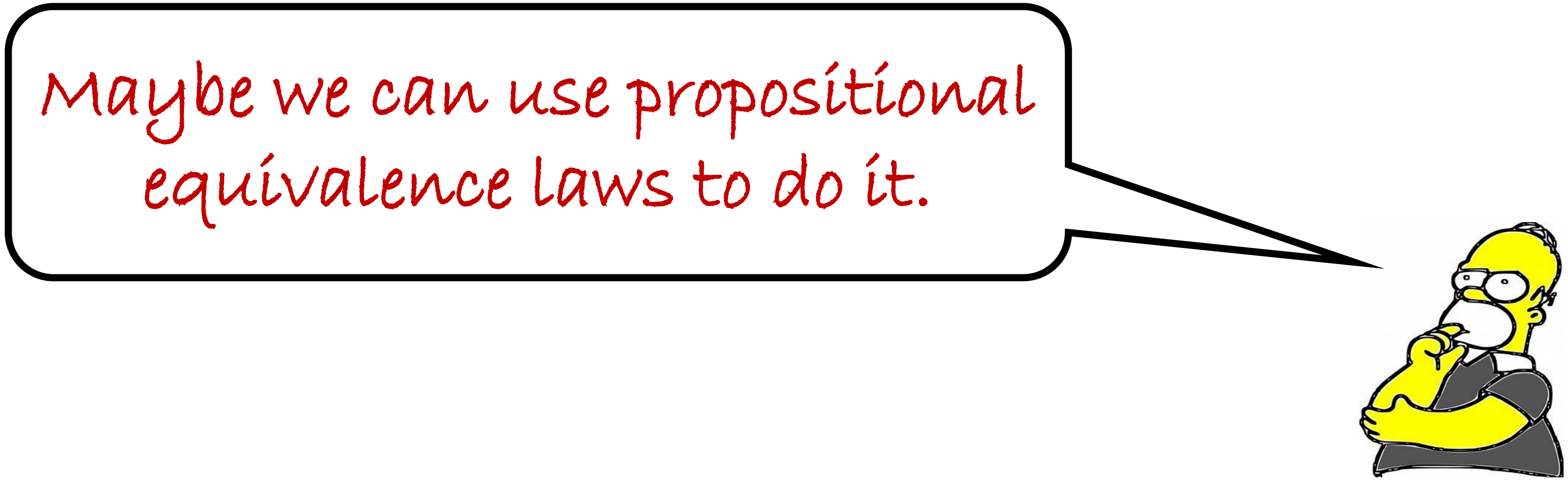
Step2: generate a formula for every row

Step3: use "and" to connect these formulas

$$g_1(P,Q) = ((\neg P) \vee Q) \wedge ((\neg P) \vee (\neg Q))$$

But if we can build the truth table, we don't need SAT solvers.

Converting to CNF



Converting to CNF

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

Example

Converting to CNF

Step1: eliminate implication and biconditional connectives.

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

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

Converting to CNF

$$
\begin{aligned}
&\neg(P \vee Q) \leftrightarrow (P \wedge Q)\\
=&(\neg\neg(P \vee Q) \vee (P \wedge Q)) \wedge (\neg(P \vee Q) \vee \neg(P \wedge Q))\\
=&((P \vee Q) \vee (P \wedge Q)) \wedge ((\neg P \wedge \neg Q) \vee (\neg P \vee \neg Q))
\end{aligned}
$$

Step1: eliminate implication and biconditional connectives.

Step2: repeatedly use De Morgan's laws and double negation law to move "not" inside

Converting to CNF

$$
\begin{aligned}
&\neg(P \vee Q) \leftrightarrow (P \wedge Q) \\
=&(\neg\neg(P \vee Q) \vee (P \wedge Q)) \wedge (\neg(P \vee Q) \vee \neg(P \wedge Q)) \\
=&((P \vee Q) \vee (P \wedge Q)) \wedge ((\neg P \wedge \neg Q) \vee (\neg P \vee \neg Q)) \\
=&(P \vee Q) \wedge (\neg P \vee \neg Q)
\end{aligned}
$$

Step1: eliminate implication and biconditional connectives.

Step2: repeatedly use De Morgan's laws and double negation law to move "not" inside

Step3: repeatedly use distributive law to get CNF

Exercise

$$(A \land B) \leftrightarrow E$$

$$(A \land B) \leftrightarrow E$$
$$((A \land B) \rightarrow E) \land (E \rightarrow (A \land B))$$
$$(\neg(A \land B) \lor E) \land (\neg E \lor (A \land B))$$
$$(\neg A \lor \neg B \lor E) \land (\neg E \lor A) \land (\neg E \lor B)$$

Are there any other normal forms?

Disjunction Normal Form（析取范式）

DEFINITION

A formula is in disjunctive normal form (DNF) if it is a disjunction of one or more conjunction clauses. Otherwise put, it is an OR of ANDs.

$$A \vee B$$

$$(A \wedge B) \vee (\neg A \wedge \neg B)$$

Converting to DNF

$$\neg(P \lor Q) \leftrightarrow (P \land Q)$$

Same example in CNF

Converting to DNF

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

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

Step1: eliminate implication and biconditional connectives.

Converting to DNF

Step1: eliminate implication and biconditional connectives.

$$
\begin{aligned}
&\neg(P \vee Q) \leftrightarrow (P \wedge Q) \\
=&(\neg\neg(P \vee Q) \wedge \neg(P \wedge Q)) \vee (\neg(P \vee Q) \wedge (P \wedge Q)) \\
=&((P \vee Q) \wedge (\neg P \vee \neg Q)) \vee (\neg P \wedge \neg Q \wedge P \wedge Q)
\end{aligned}
$$

Step2: repeatedly use De Morgan's laws and double negation law to move “not” inside

Converting to DNF

Step1: eliminate implication and biconditional connectives.

$$
\begin{aligned}
&\neg(P \vee Q) \leftrightarrow (P \wedge Q) \\
=&(\neg\neg(P \vee Q) \wedge \neg(P \wedge Q)) \vee (\neg(P \vee Q) \wedge (P \wedge Q)) \\
=&((P \vee Q) \wedge (\neg P \vee \neg Q)) \vee (\neg P \wedge \neg Q \wedge P \wedge Q) \\
=&(P \wedge \neg P) \vee (Q \wedge \neg P) \vee (P \wedge \neg Q) \vee (Q \wedge \neg Q) \vee (\neg P \wedge \neg Q \wedge P \wedge Q)
\end{aligned}
$$

Step3: repeatedly use distributive law to get DNF

Step2: repeatedly use De Morgan's laws and double negation law to move “not” inside

Exercise

$$(A \wedge B) \leftrightarrow E$$

$$\begin{aligned}
&(A \wedge B) \leftrightarrow E \\
=&(A \wedge B \wedge E) \vee (\neg(A \wedge B) \wedge \neg E) \\
=&(A \wedge B \wedge E) \vee ((\neg A \vee \neg B) \wedge \neg E) \\
=&(A \wedge B \wedge E) \vee (\neg A \wedge \neg E) \vee (\neg B \wedge \neg E)
\end{aligned}$$

Why does SAT Solvers use CNF instead of DNF?
If SAT solvers use DNF, what will happen?

CNF v.s. DNF

$$A_1 \wedge A_2 \wedge A_3 \wedge \ldots \wedge A_n$$

$$A_1 \vee A_2 \vee A_3 \vee \ldots \vee A_n$$

A formula in DNF is true when one of these conjunction clauses is true. One of them is true when P and ¬P don't exist at the same time.

CNF v.s. DNF

$$A_1 \wedge A_2 \wedge A_3 \wedge \ldots \wedge A_n$$

$$A_1 \vee A_2 \vee A_3 \vee \ldots \vee A_n$$

You just have to scan through the conjunction clauses and check whether one of them contains not both a variable and its negation.

CNF v.s. DNF

$$A_1 \wedge A_2 \wedge A_3 \wedge \ldots \wedge A_n$$

$$A_1 \vee A_2 \vee A_3 \vee \ldots \vee A_n$$

Things become much easier when we address DNF instead of CNF. Why SAT solvers use CNF?

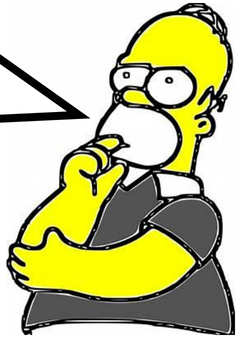
Disjunction Normal Form

$$n \left\{ \begin{array}{l} (A_1 \vee B_1) \wedge \\ (A_2 \vee B_2) \wedge \\ (A_3 \vee B_3) \wedge \\ \dots \\ (A_n \vee B_n) \end{array} \right. \quad \Longleftrightarrow \quad \left. \begin{array}{l} (A_1 \wedge A_2 \wedge \dots \wedge A_n) \vee \\ (B_1 \wedge A_2 \wedge \dots \wedge A_n) \vee \\ (A_1 \wedge B_2 \wedge \dots \wedge A_n) \vee \\ \dots \\ (B_1 \wedge B_2 \wedge \dots \wedge B_n) \end{array} \right\} 2^n$$

You can convert a formula into DNF, but the resulting formula might be very much larger than the original formula—in fact, exponentially so.

WHY CNF?

But what about CNF?



WHY CNF?

$$(A1 \land A2 \land A3) \lor (B1 \land B2 \land B3)$$

$$\Downarrow$$

3×3

$$(A1 \lor B1) \land (A1 \lor B2) \land (A1 \lor B3) \land$$
$$(A2 \lor B1) \land (A2 \lor B2) \land (A2 \lor B3) \land$$
$$(A3 \lor B1) \land (A3 \lor B2) \land (A3 \lor B3)$$

It seems to exponentially explode, too.

Switch Variables

$$(A1 \wedge A2 \wedge A3) \vee (B1 \wedge B2 \wedge B3)$$

$$\Downarrow$$

$$(Z \rightarrow A1 \wedge A2 \wedge A3) \wedge$$
$$(\neg Z \rightarrow B1 \wedge B2 \wedge B3)$$

The translation is equi-satisfiable.

The translation is not equivalent.

Exercise: convert it to CNF and check if there is explosion.

Switch Variables

$$(A1 \wedge A2 \wedge A3) \vee (B1 \wedge B2 \wedge B3)$$

$\Downarrow$

The translation is equi-satisfiable.

$$(Z \to A1 \wedge A2 \wedge A3) \wedge$$
$$(\neg Z \to B1 \wedge B2 \wedge B3)$$

The translation is not equivalent.

$\Downarrow$

$$\begin{aligned}
&(Z \to (A1 \wedge A2 \wedge A3)) \wedge (\neg Z \to (B1 \wedge B2 \wedge B3)) \\
=&(\neg Z \vee (A1 \wedge A2 \wedge A3)) \wedge (Z \vee (B1 \wedge B2 \wedge B3)) \\
=&(\neg Z \vee A1) \wedge (\neg Z \vee A2) \wedge (\neg Z \vee A3) \wedge \\
&(Z \vee B1) \wedge (Z \vee B2) \wedge (Z \vee B3)
\end{aligned}$$

3+3

There is not explosion.

Normal Form Theorem

THEOREM

Every formula can be converted to CNF and DNF.

We can use truth table to convert a formula to CNF and DNF.

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

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

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

Although we can convert formulas to CNF,
we can still get many different formulas.
Can we define a more precise form?

Principal Normal Form（主范式）

DEFINITION

Given n propositional variables, if every variable exists once in a conjunction clause, the clause is a minterm（极小项）.

$$P1 \quad P2 \Rightarrow P1 \wedge P2, \neg P1 \wedge P2, P1 \wedge \neg P2, \neg P1 \wedge \neg P2 \quad 2^n$$

Every minterm is true under only one interpretation.
Any two minterms are not equivalent and the conjunction of them is F.

Principal Disjunction Normal Form（主析取范式）

DEFINITION

Principal Disjunction Normal Form(PDNF) is the disjunction of minterms.

P	Q	$g_0(P,Q)$	$g_1(P,Q)$
F	F	$\boxed{T}$	T
F	T	$\boxed{T}$	T
T	F	F	F
T	T	$\boxed{T}$	F

Step1: find all true rows

Step2: generate a formula for every row

Step3: use “or” to connect these formulas

$$g_0(P,Q) = (((\neg P) \wedge (\neg Q)) \vee ((\neg P) \wedge Q)) \vee (P \wedge Q)$$

Truth table is all-purpose. Every formula con be converted to only one PDNF.

Principal Disjunction Normal Form（主析取范式）

DEFINITION

Principal Disjunction Normal Form(PDNF) is the disjunction of minterms.

P	Q	$g_0(P,Q)$	$g_1(P,Q)$
F	F	$\boxed{T}$	T
F	T	$\boxed{T}$	T
T	F	F	F
T	T	$\boxed{T}$	F

Step1: find all true rows

Step2: generate a formula for every row

Step3: use "or" to connect these formulas

$$g_0(P,Q) = (((\neg P) \wedge (\neg Q)) \vee ((\neg P) \wedge Q)) \vee (P \wedge Q)$$

If a PDNF includes all minterms, it must be true.

Converting to PDNF

$$
\begin{aligned}
& P \to Q \\
=& \neg P \vee Q \\
=& (\neg P \wedge (Q \vee \neg Q)) \vee Q \\
=& (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee Q \\
=& (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee (Q \wedge (P \vee \neg P)) \\
=& (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee (P \wedge Q) \vee (\neg P \wedge Q) \\
=& (\neg P \wedge \neg Q) \vee (\neg P \wedge Q) \vee (P \wedge Q) \\
=& m_0 \vee m_1 \vee m_3 = \vee_{0;1;3}
\end{aligned}
$$

Exercise

$$(A \wedge B) \leftrightarrow E$$

$$\begin{aligned}
&(A \wedge B) \leftrightarrow E \\
=&(A \wedge B \wedge E) \vee (\neg(A \wedge B) \wedge \neg E) \\
=&(A \wedge B \wedge E) \vee ((\neg A \vee \neg B) \wedge \neg E) \\
=&(A \wedge B \wedge E) \vee (\neg A \wedge \neg E) \vee (\neg B \wedge \neg E) \\
=&(A \wedge B \wedge E) \vee (\neg A \wedge \neg E \wedge (B \vee \neg B)) \vee (\neg B \wedge \neg E \wedge (A \vee \neg A)) \\
=&(A \wedge B \wedge E) \vee (\neg A \wedge B \wedge \neg E) \vee (\neg A \wedge \neg B \wedge \neg E) \vee (A \wedge \neg B \wedge \neg E) \vee (\neg A \wedge \neg B \wedge \neg E) \\
=&(\neg A \wedge \neg B \wedge \neg E) \vee (\neg A \wedge B \wedge \neg E) \vee (A \wedge \neg B \wedge \neg E) \vee (A \wedge B \wedge E) \\
=&m_0 \vee m_2 \vee m_4 \vee m_7 = \vee_{0;2;4;7}
\end{aligned}$$

Principal Normal Form（主范式）

DEFINITION

Given n propositional variables, if every variable exists once in a disjunction clause, the clause is a maxterm（极大项）.

$P1 \quad P2 \Rightarrow P1 \vee P2, \neg P1 \vee P2, P1 \vee \neg P2, \neg P1 \vee \neg P2 \qquad 2^n$

Every maxterm is false under only one interpretation.
Every two maxterms are not equivalent and the disjunction of them is T.

Principal Conjunction Normal Form（主合取范式）

DEFINITION

Principal Conjunction Normal Form(PCNF) is the conjunction of maxterms.

P	Q	$g_0(P, Q)$	$g_1(P, Q)$
F	F	T	T
F	T	T	T
T	F	F	$\boxed{F}$
T	T	T	$\boxed{F}$

Step1: find all false rows

Step2: generate a formula for every row

Step3: use "and" to connect these formulas

$$g_1(P, Q) = ((\neg P) \vee Q) \wedge ((\neg P) \vee (\neg Q))$$

Truth table is all-purpose. Every formula con be converted to only one PCNF.

Principal Conjunction Normal Form（主合取范式）

DEFINITION

Principal Conjunction Normal Form(PCNF) is the conjunction of maxterms.

P	Q	$g_0(P,Q)$	$g_1(P,Q)$
F	F	T	T
F	T	T	T
T	F	F	$\boxed{F}$
T	T	T	$\boxed{F}$

Step1: find all false rows

Step2: generate a formula for every row

Step3: use "and" to connect these formulas

$$g_1(P,Q) = ((\neg P) \vee Q) \wedge ((\neg P) \vee (\neg Q))$$

If a PCNF includes all maxterms, it must be false.

Converting to PCNF

$$
\begin{aligned}
& P \wedge Q \\
= & (P \vee (Q \wedge \neg Q)) \wedge (Q \vee (P \wedge \neg P)) \\
= & (P \vee Q) \wedge (P \vee \neg Q) \wedge (Q \vee P) \wedge (Q \vee \neg P) \\
= & (\neg P \vee Q) \wedge (P \vee \neg Q) \wedge (P \vee Q) \\
= & M_1 \wedge M_2 \wedge M_3 = \wedge_{1;2;3}
\end{aligned}
$$

Exercise

$$(A \wedge B) \leftrightarrow E$$

$$\begin{aligned}
&(A \wedge B) \leftrightarrow E \\
=&(A \wedge B \rightarrow E) \wedge (E \rightarrow A \wedge B) \\
=&(\neg(A \wedge B) \vee E) \wedge (\neg E \vee (A \wedge B)) \\
=&(\neg A \vee \neg B \vee E) \wedge (\neg E \vee A) \wedge (\neg E \vee B) \\
=&(\neg A \vee \neg B \vee E) \wedge (\neg E \vee A \vee (B \wedge \neg B)) \wedge (\neg E \vee B \vee (A \wedge \neg A)) \\
=&(\neg A \vee \neg B \vee E) \wedge (A \vee B \vee \neg E) \wedge (A \vee \neg B \vee \neg E) \wedge (A \vee B \vee \neg E) \wedge (\neg A \vee B \vee \neg E) \\
=&(\neg A \vee \neg B \vee E) \wedge (\neg A \vee B \vee \neg E) \wedge (A \vee \neg B \vee \neg E) \wedge (A \vee B \vee \neg E) \\
=&M_1 \wedge M_2 \wedge M_4 \wedge M_6 = \wedge_{1;2;4;6}
\end{aligned}$$

PCNF vs. PDNF

P	Q	$g_0(P,Q)$	$g_1(P,Q)$
F	F	T	T
F	T	T	T
T	F	F	F
T	T	T	F

P	Q	$g_0(P,Q)$	$g_1(P,Q)$
F	F	T	T
F	T	T	T
T	F	F	F
T	T	T	F

$$g_0(P,Q) = (((\neg P) \wedge (\neg Q)) \vee ((\neg P) \wedge Q)) \vee (P \wedge Q)$$

$$g_1(P,Q) = ((\neg P) \wedge (\neg Q)) \vee ((\neg P) \wedge Q)$$

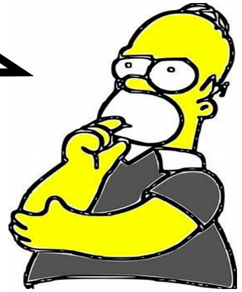
$$g_0(P,Q) = ((\neg P) \vee Q)$$

$$g_1(P,Q) = ((\neg P) \vee Q) \wedge ((\neg P) \vee (\neg Q))$$

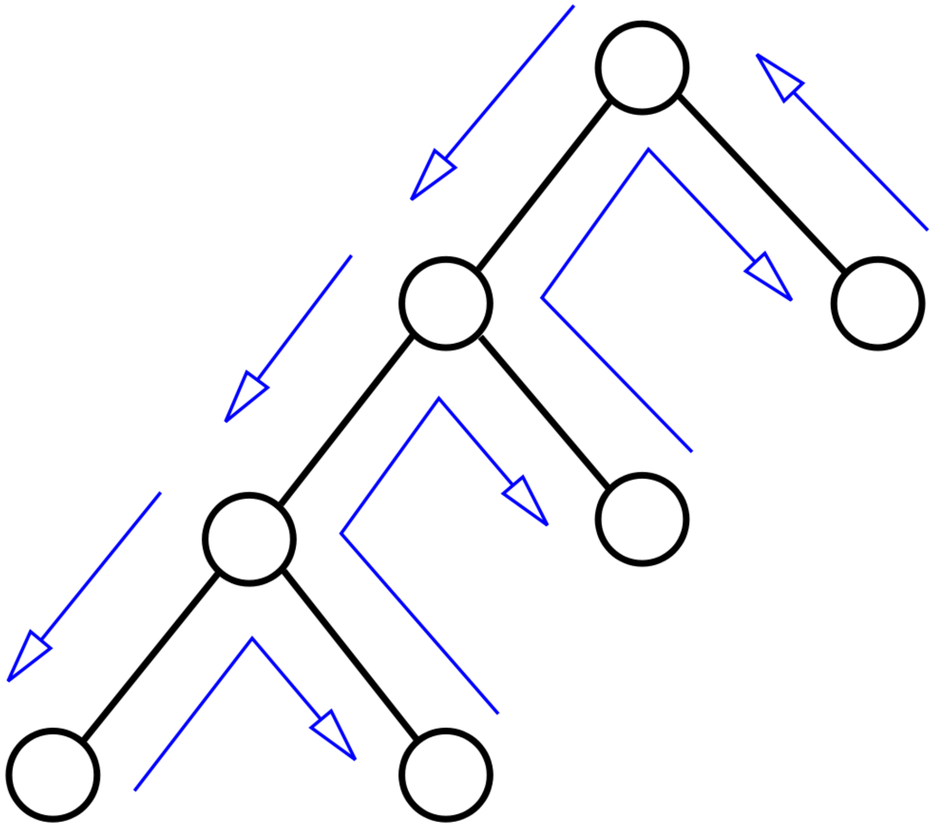
Can you find the relation between PCNF and PDNF according to truth table?

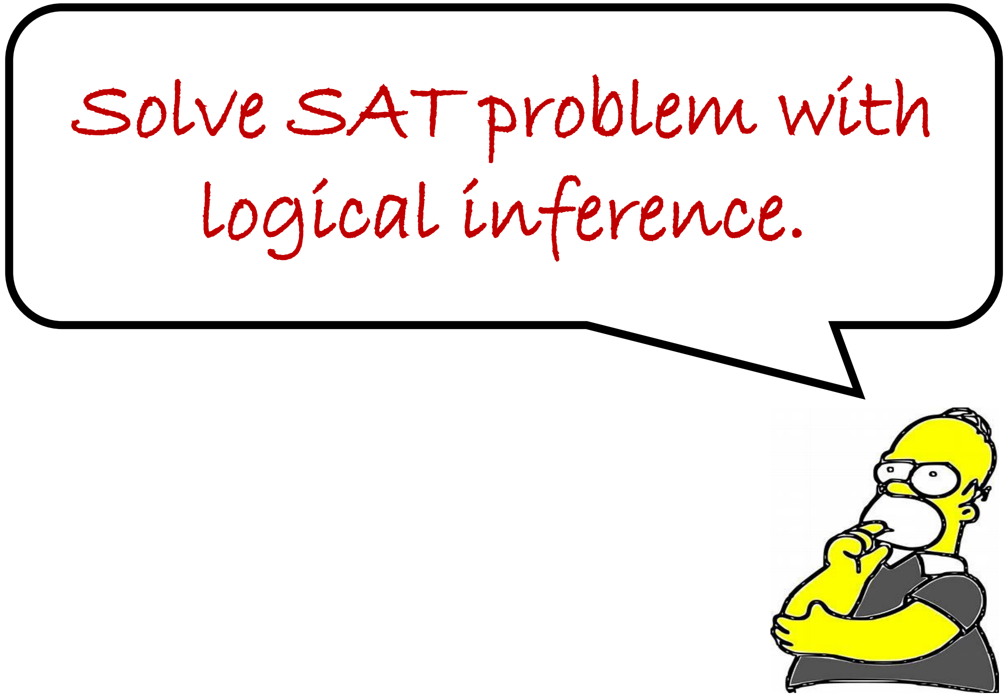
SAT Solver

In practice, CNF is enough and more convenient for SAT Solver.



DPLL (Davis-Putnam-Logemann-Loveland Algorithm)





The backtracking DPLL algorithm is
the basis for most modern SAT solvers.

DPLL

OVERVIEW

Iteratively process by following steps:

- Guess the truth value of an undefined literal
- Infer the truth value of other variables with inference rules
- If a clause is false and there is a decision literal, backtrack and re-guess a variable

DEFINITION

A literal is defined if its truth value has been guessed or inferred.

DPLL

OVERVIEW

Iteratively process by following steps:

- Guess the truth value of an undefined literal
- Infer the truth value of other variables with inference rules
- If a clause is false and there is a decision literal, backtrack and re-guess a variable

DEFINITION

A literal is defined if its truth value has been guessed or inferred.

DPLL

DECIDE RULE

A literal *l* is defined by ”guessing”, then it is annotated as a “decision literal”. If all defined literals can not satisfy formula F, then $\neg l$ must be concerned.

$$(\boxed{A} \vee \neg B) \wedge (B \vee \neg C) \wedge (C \vee A)$$

A is an undefined literal. We can guess A=F temporarily which makes A to be a decision literal.

DPLL

OVERVIEW

Iterative process by following steps:

- Guess the truth value of an undefined literal
- Infer the truth value of other variables with inference rules
- If a clause is false and there is a decision literal, backtrack and re-guess a variable

DEFINITION

A literal is defined if its truth value has been guessed or inferred.

DPLL

UNITPROPAGATE RULE

To satisfy a CNF formula, all its clauses have to be true. Hence, if a clause of *F* contains a literal *l* whose truth value is not defined by the current assignment while all the remaining literals of the clause are false, *l* must be defined to be true.

$$\boxed{(A \vee \neg B)} \wedge \boxed{(B \vee \neg C)} \wedge (C \vee A)$$

Because we guess A=F, then we can infer that B=F. Then we can infer C=F according to the second clause.

DPLL

OVERVIEW

Iteratively process by following steps:

- Guess the truth value of an undefined literal
- Infer the truth value of other variables with inference rules
- If a clause is false and there is a decision literal, backtrack and re-guess a variable

DEFINITION

A literal is defined if its truth value has been guessed or inferred.

DPLL

BACKTRACK RULE

If a conflicting clause C is detected and there is defined decision literal *l*, then the rule backtracks by replacing the most recent decision literal *l* by ¬*l* and removing any subsequent literals in the current assignment. Note that ¬*l* is annotated as a non-decision literal, since the other possibility *l* has already been explored.

$$(A \vee \neg B) \wedge (B \vee \neg C) \wedge \boxed{(C \vee A)}$$

A conflicting clause

Because A=B=C=F, then the last clause is F. Now we can backtrack and guess A=T. Now A is not a decision literal.

DPLL

FAIL RULE

This rule detects a conflicting clause *C* and produces the FailState state whenever there is no decision literals defined.

$$(A \vee \neg B) \wedge (B \vee \neg C) \wedge \boxed{(C \vee A)}$$

A conflicting clause

We can backtrack because A is a decision literal. If there is no decision literal, we can conclude the formula is unsatisfiable.

DPLL

OVERVIEW

Iteratively try to

- Guess the truth value of an undefined literal
- Infer the truth value of other variables with inference rules
- If a clause is false and there is a decision literal, backtrack and reguess a variable

$$(A \vee \neg B) \wedge (B \vee \neg C) \wedge (C \vee A)$$

Now A=T and we can repeat the three steps to get the answer.

DPLL

PURELITERAL RULE

If a literal *l* is pure in *F*, i.e., it occurs in *F* while its negation does not, then *F* is satisfiable only if it defines *l* to be true. Thus, we can define *l* to be true.

$$(\boxed{A} \vee \neg B) \wedge (B \vee \neg C) \wedge (C \vee \boxed{A})$$

Actually, we don't need to guess A here.
Because A exists and ¬A doesn't exist.
So we can directly determine A=T.

Example

$$\emptyset \parallel \overline{1} \vee \overline{2},\quad 2 \vee 3,\quad \overline{1} \vee \overline{3} \vee 4,\quad 2 \vee \overline{3} \vee \overline{4},\quad 1 \vee 4 \qquad \Longrightarrow \qquad \text{(Decide)}$$

Example

$$
\begin{array}{lllllllll}
\emptyset & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Decide)} \\
1^{\mathsf{d}} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)}
\end{array}
$$

Example

$$
\begin{array}{rcll}
\emptyset \;\|\; \overline{1}\vee\overline{2},\;\; 2\vee 3,\;\; \overline{1}\vee\overline{3}\vee 4,\;\; 2\vee\overline{3}\vee\overline{4},\;\; 1\vee 4 & \Longrightarrow & \text{(Decide)} \\
1^{\mathsf{d}} \;\|\; \overline{1}\vee\overline{2},\;\; 2\vee 3,\;\; \overline{1}\vee\overline{3}\vee 4,\;\; 2\vee\overline{3}\vee\overline{4},\;\; 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\,\overline{2} \;\|\; \overline{1}\vee\overline{2},\;\; 2\vee 3,\;\; \overline{1}\vee\overline{3}\vee 4,\;\; 2\vee\overline{3}\vee\overline{4},\;\; 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)}
\end{array}
$$

Example

$$
\begin{array}{rcllllllll}
\emptyset & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Decide)} \\
1^{\mathsf{d}} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)}
\end{array}
$$

Example

$$
\begin{array}{rcllllllr}
\emptyset & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Decide)} \\
1^{\mathsf{d}} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3\ 4 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & \boxed{2\vee\overline{3}\vee\overline{4},} & 1\vee 4 & \Longrightarrow & \text{(Backtrack)}
\end{array}
$$

Example

$$
\begin{array}{rcl}
\emptyset \ \| \ \overline{1}\vee\overline{2},\ \ 2\vee 3,\ \ \overline{1}\vee\overline{3}\vee 4,\ \ 2\vee\overline{3}\vee\overline{4},\ \ 1\vee 4 & \Longrightarrow & \text{(Decide)} \\
1^{\mathsf{d}} \ \| \ \overline{1}\vee\overline{2},\ \ 2\vee 3,\ \ \overline{1}\vee\overline{3}\vee 4,\ \ 2\vee\overline{3}\vee\overline{4},\ \ 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2} \ \| \ \overline{1}\vee\overline{2},\ \ 2\vee 3,\ \ \overline{1}\vee\overline{3}\vee 4,\ \ 2\vee\overline{3}\vee\overline{4},\ \ 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3 \ \| \ \overline{1}\vee\overline{2},\ \ 2\vee 3,\ \ \overline{1}\vee\overline{3}\vee 4,\ \ 2\vee\overline{3}\vee\overline{4},\ \ 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3\ 4 \ \| \ \overline{1}\vee\overline{2},\ \ 2\vee 3,\ \ \overline{1}\vee\overline{3}\vee 4,\ \ 2\vee\overline{3}\vee\overline{4},\ \ 1\vee 4 & \Longrightarrow & \text{(Backtrack)} \\
\overline{1} \ \| \ \overline{1}\vee\overline{2},\ \ 2\vee 3,\ \ \overline{1}\vee\overline{3}\vee 4,\ \ 2\vee\overline{3}\vee\overline{4},\ \ 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)}
\end{array}
$$

Example

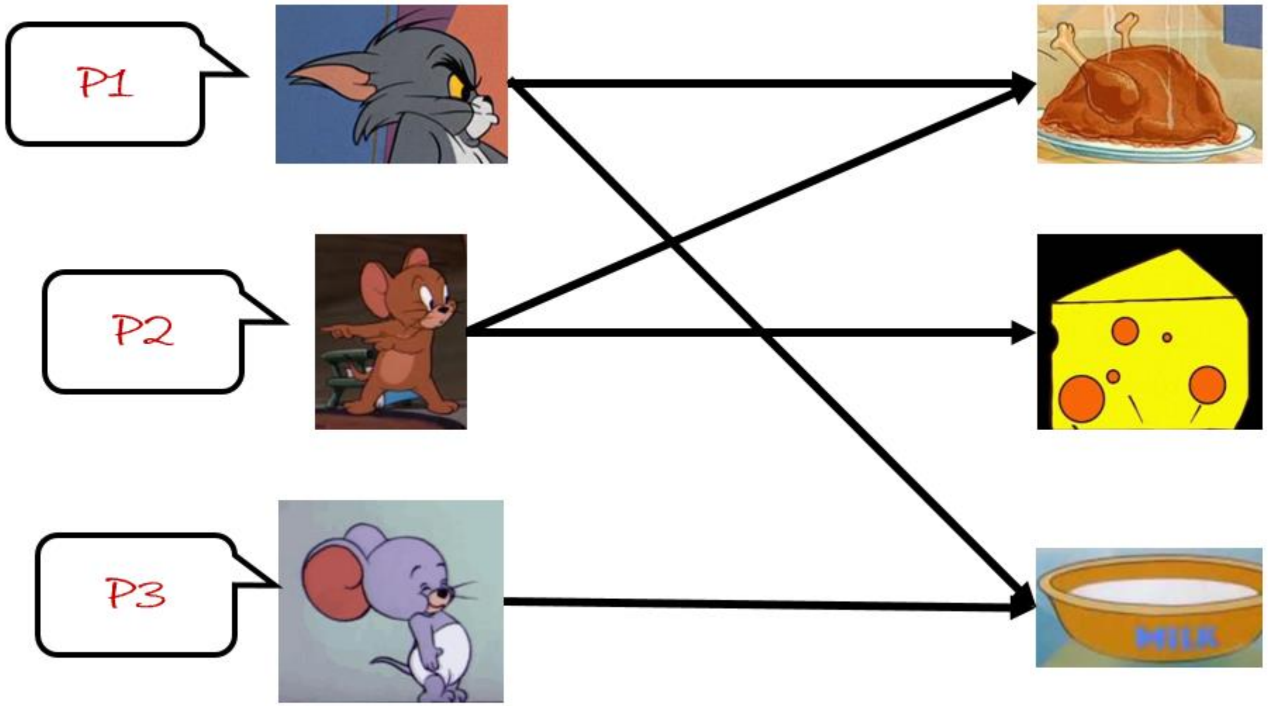
$$
\begin{array}{rcllllllll}
\emptyset & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Decide)} \\
1^{\mathsf{d}} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3\ 4 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Backtrack)} \\
\overline{1} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
\overline{1}\ 4 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Decide)}
\end{array}
$$

Example

$$
\begin{array}{rclllllcl}
\emptyset & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Decide)} \\
1^{\mathsf{d}} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3\ 4 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Backtrack)} \\
\overline{1} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
\overline{1}\ 4 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Decide)} \\
\overline{1}\ 4\ \overline{3}^{\mathsf{d}} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)}
\end{array}
$$

Example

$$
\begin{array}{rcllllllll}
\emptyset & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Decide)} \\
1^{\mathsf{d}} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
1^{\mathsf{d}}\ \overline{2}\ 3\ 4 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Backtrack)} \\
\overline{1} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
\overline{1}\ 4 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(Decide)} \\
\overline{1}\ 4\ \overline{3}^{\mathsf{d}} & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & \Longrightarrow & \text{(UnitPropagate)} \\
\overline{1}\ 4\ \overline{3}^{\mathsf{d}}\ 2 & \| & \overline{1}\vee\overline{2}, & 2\vee 3, & \overline{1}\vee\overline{3}\vee 4, & 2\vee\overline{3}\vee\overline{4}, & 1\vee 4 & &
\end{array}
$$



$$\emptyset \quad \| \quad 3,\ \ \overline{1} \vee \overline{2},\ \ 1 \vee \overline{3} \quad \Longrightarrow (PureLiteral)$$

$$\overline{2} \quad \| \quad 3,\ \ \overline{1} \vee \overline{2},\ \ 1 \vee \overline{3} \quad \Longrightarrow (UnitProp)$$

$$\overline{2}\ 3 \ \| \quad 3,\ \ \overline{1} \vee \overline{2},\ \ 1 \vee \overline{3} \quad \Longrightarrow (UnitProp)$$

$$\overline{2}\ 3\ 1 \ \| \quad 3,\ \ \overline{1} \vee \overline{2},\ \ 1 \vee \overline{3}$$

Exercise

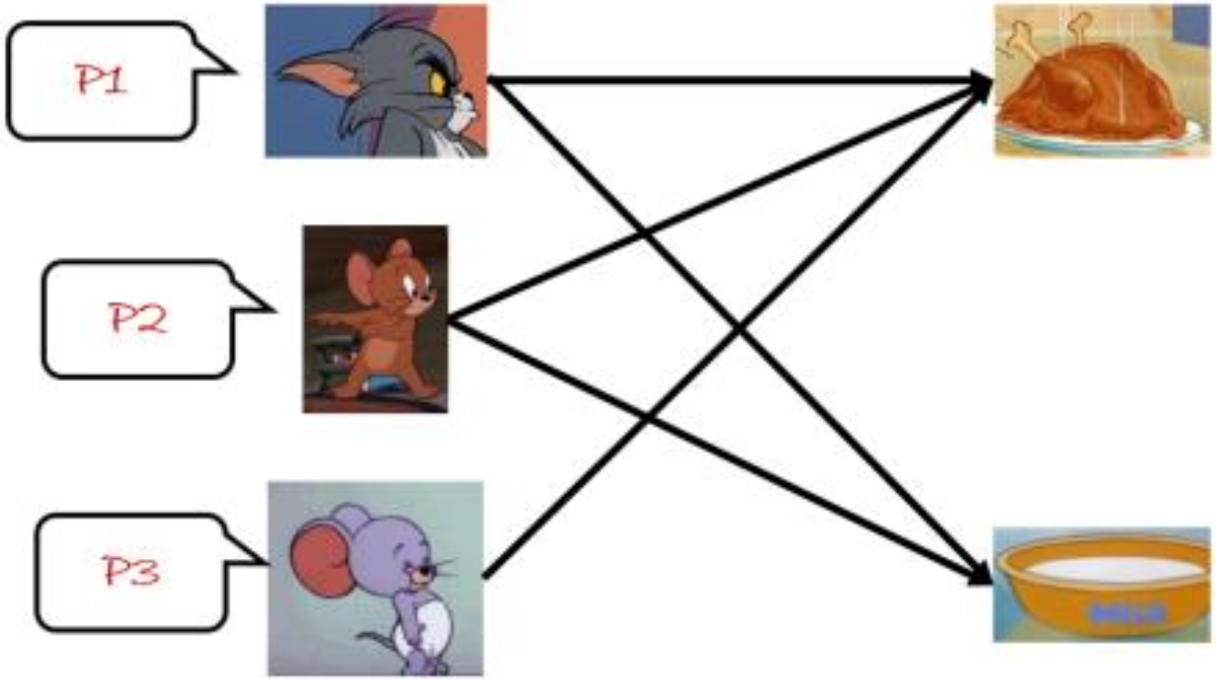
$$1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 2, \quad 1 \vee 4$$

Exercise

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

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

UNSAT



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

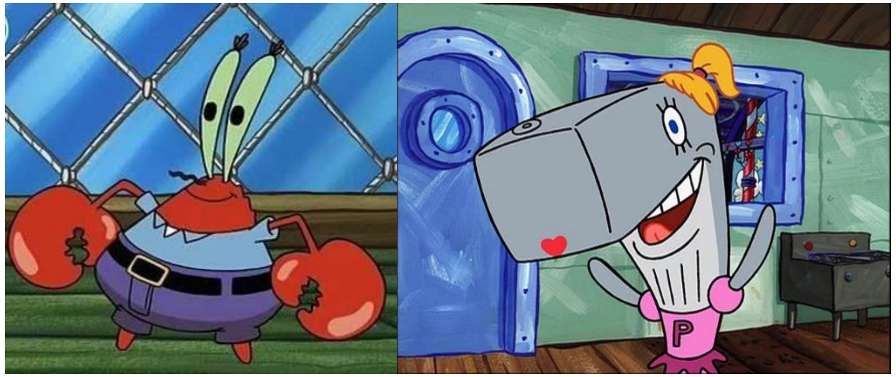
$$= P3 \wedge (\neg P1 \vee \neg P2) \wedge (\neg P2 \vee \neg P3) \wedge (\neg P1 \vee \neg P3) \wedge (P1 \vee P2)$$

UNSAT

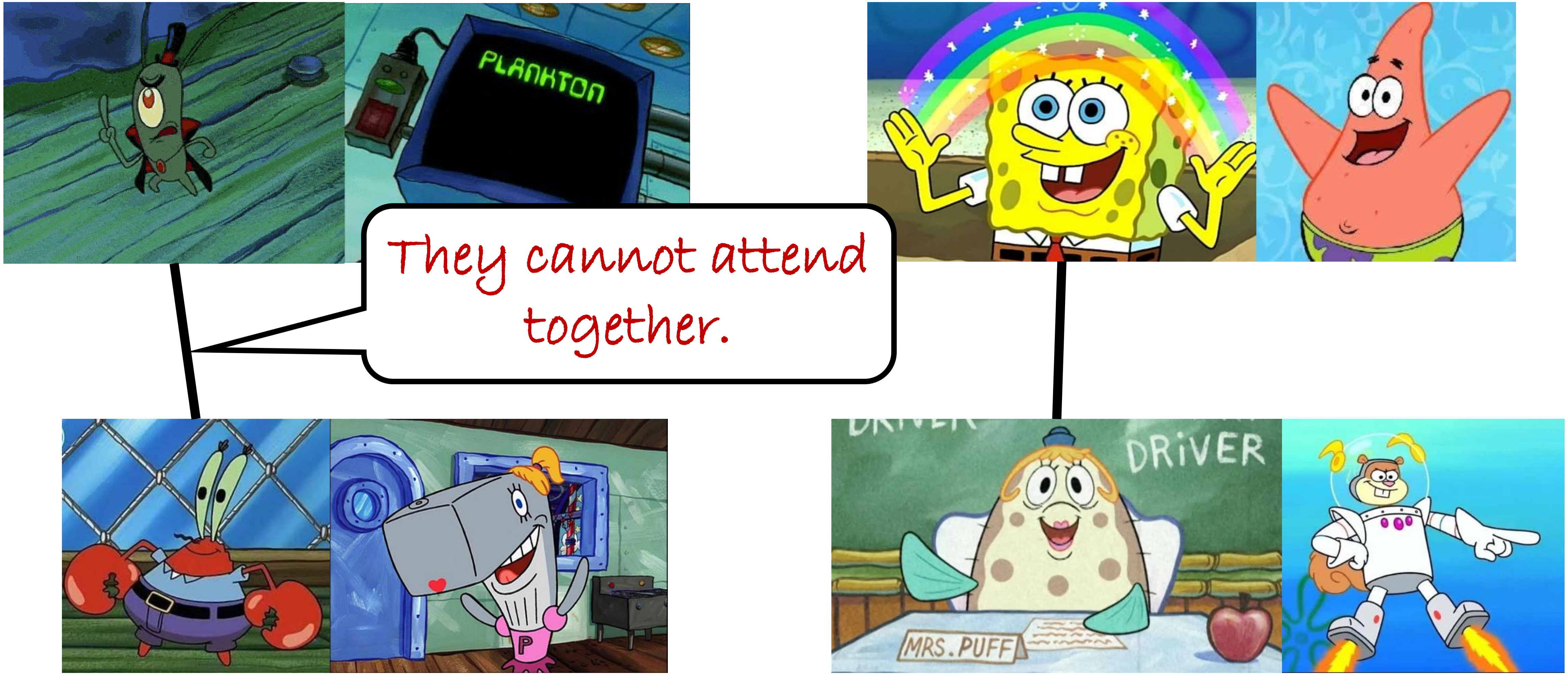
$$
\begin{array}{rl}
\emptyset \parallel & \boxed{3,}\ \overline{1} \vee \overline{2},\ \overline{2} \vee \overline{3},\ \overline{1} \vee \overline{3},\ 1 \vee 2 \quad \Longrightarrow (UnitProp) \\
3 \parallel & 3,\ \overline{1} \vee \overline{2},\ \boxed{\overline{2} \vee \overline{3}}\ \overline{1} \vee \overline{3},\ 1 \vee 2 \quad \Longrightarrow (UnitProp) \\
3\ \overline{2} \parallel & 3,\ \overline{1} \vee \overline{2},\ \overline{2} \vee \overline{3},\ \boxed{\overline{1} \vee \overline{3},}\ 1 \vee 2 \quad \Longrightarrow (UnitProp) \\
3\ \overline{2}\ \overline{1} \parallel & 3,\ \overline{1} \vee \overline{2},\ \overline{2} \vee \overline{3},\ \overline{1} \vee \overline{3},\ \boxed{1 \vee 2} \quad \Longrightarrow (Fail) \\
 & fail
\end{array}
$$

All in one

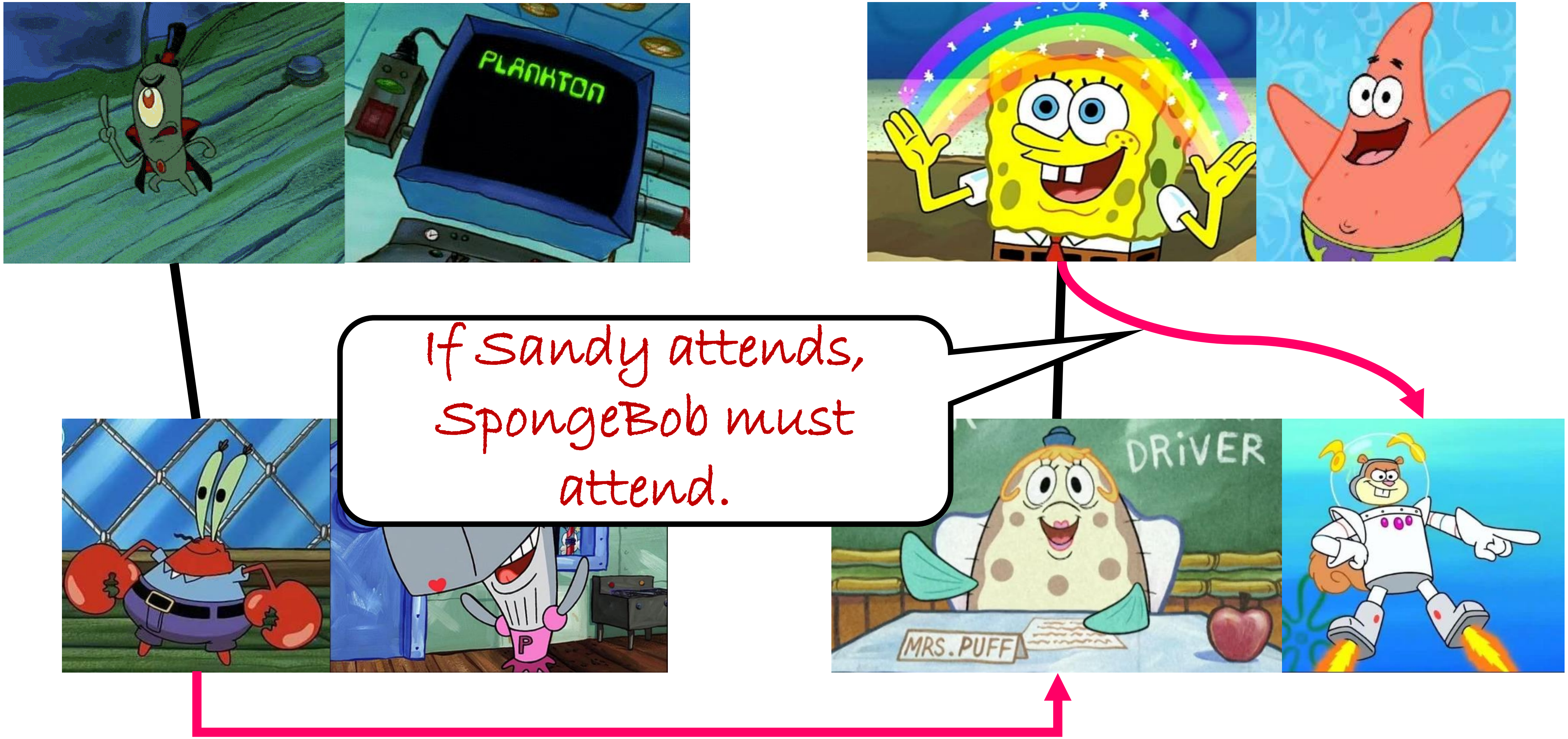
Inviting one of every group to have a meeting



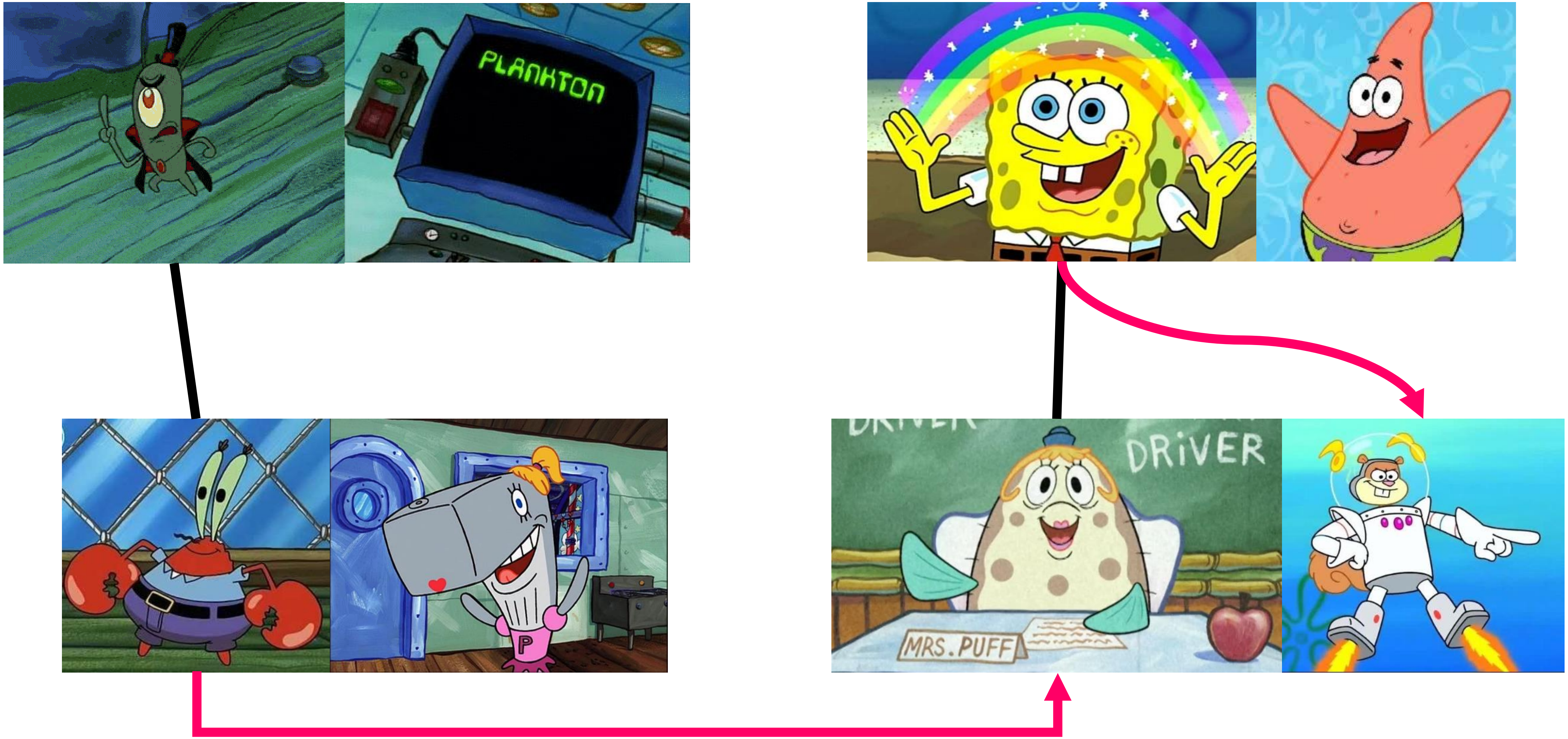
However, there are some requirements



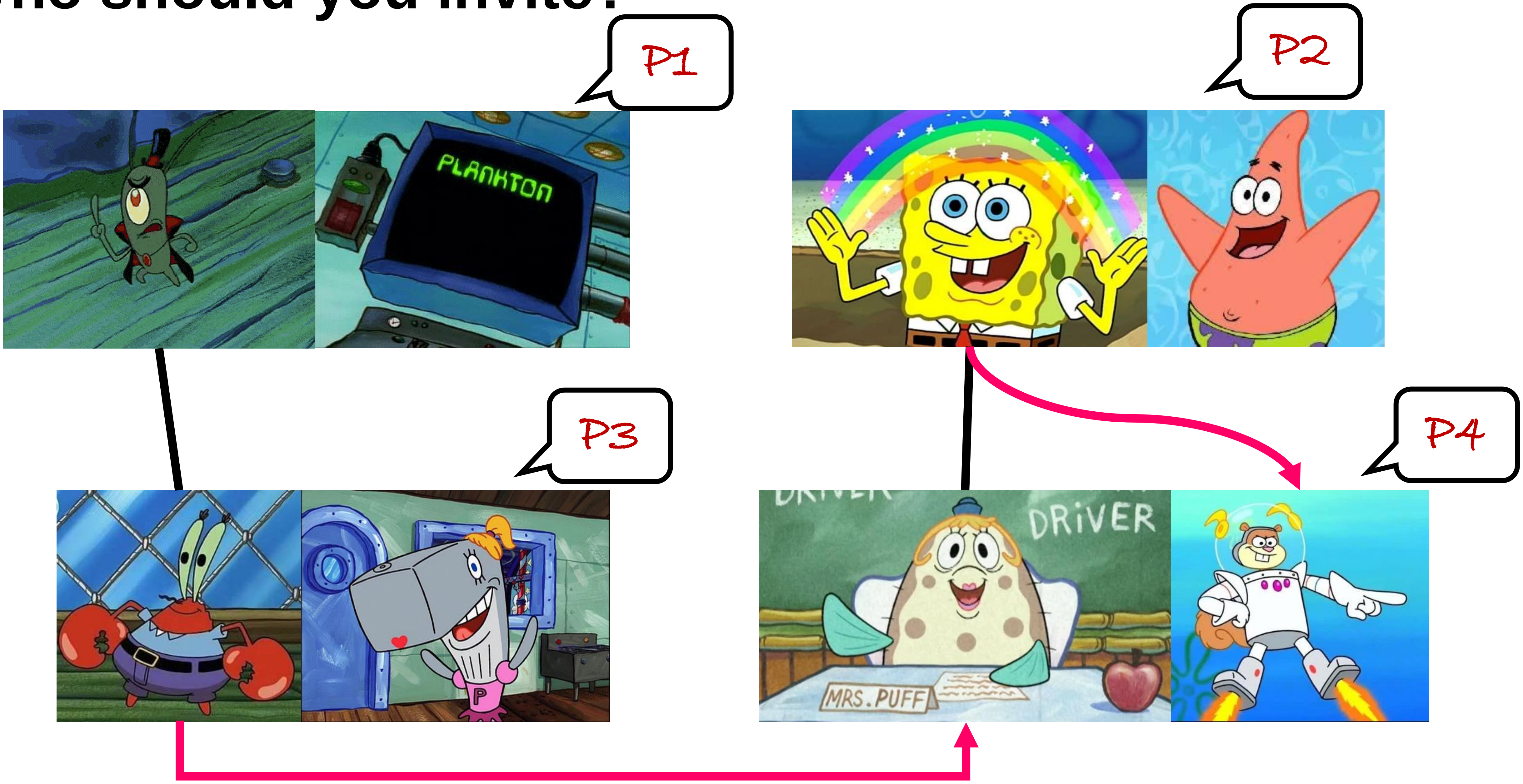
However, there are some requirements



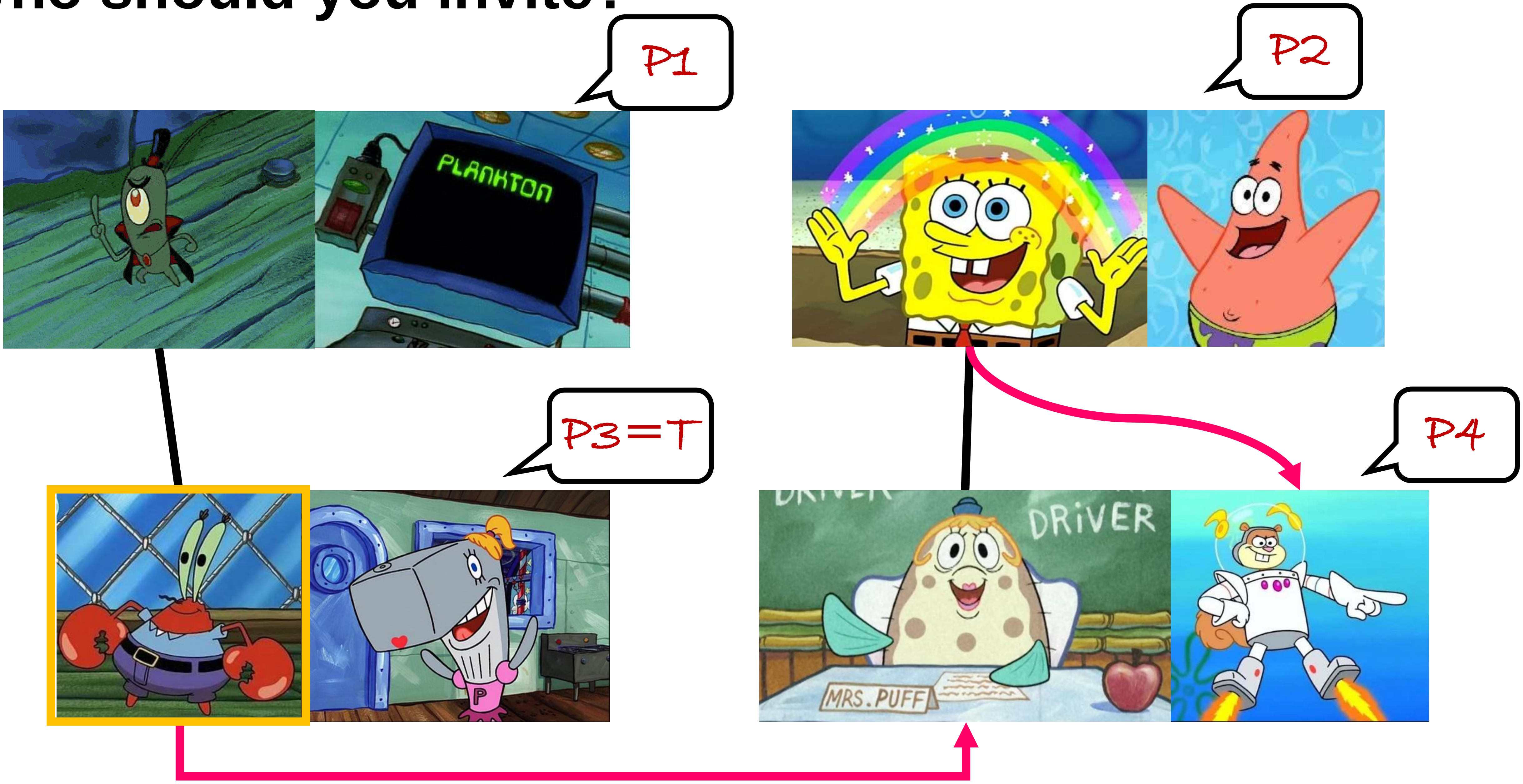
Who should you invite?



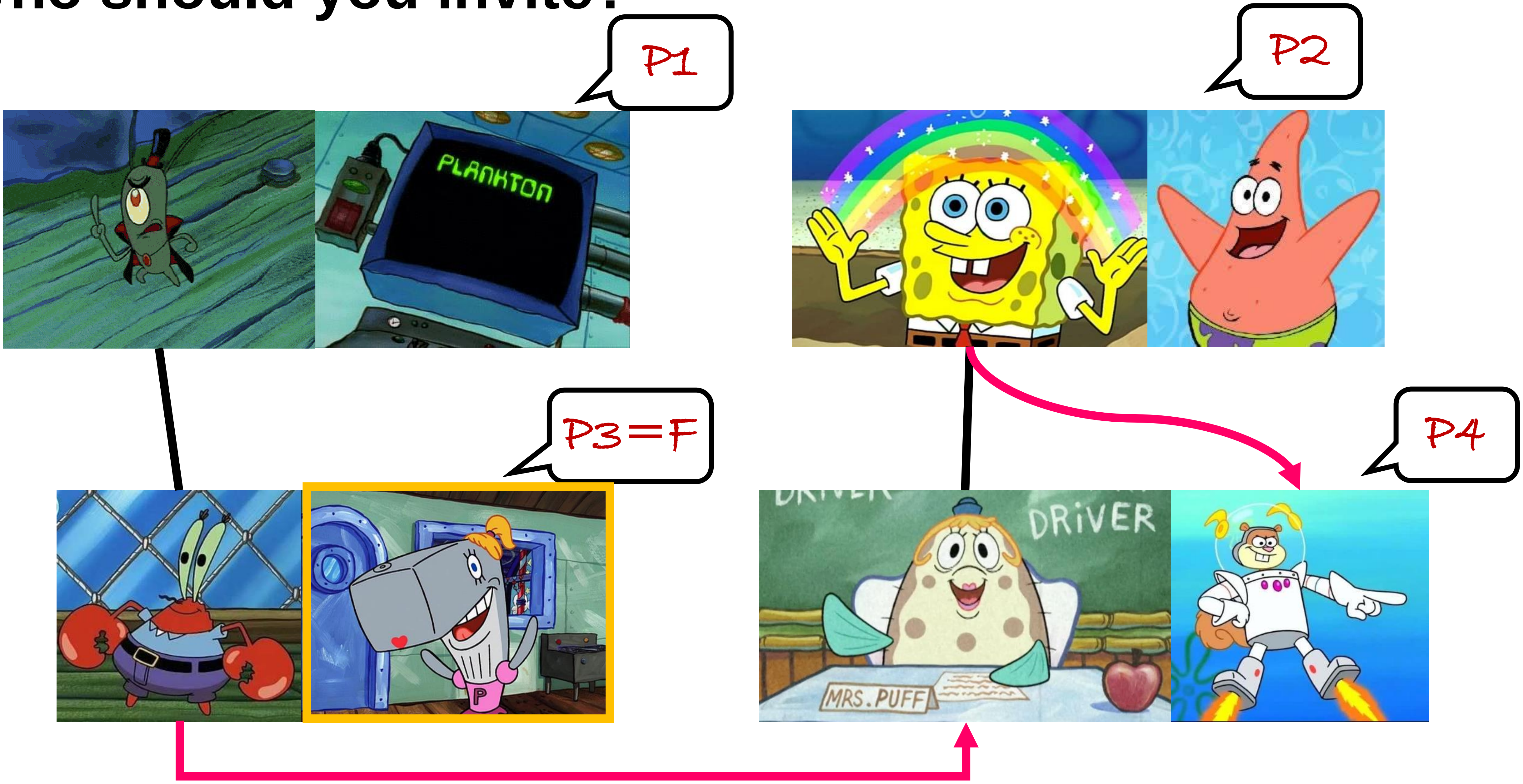
Who should you invite?



Who should you invite?



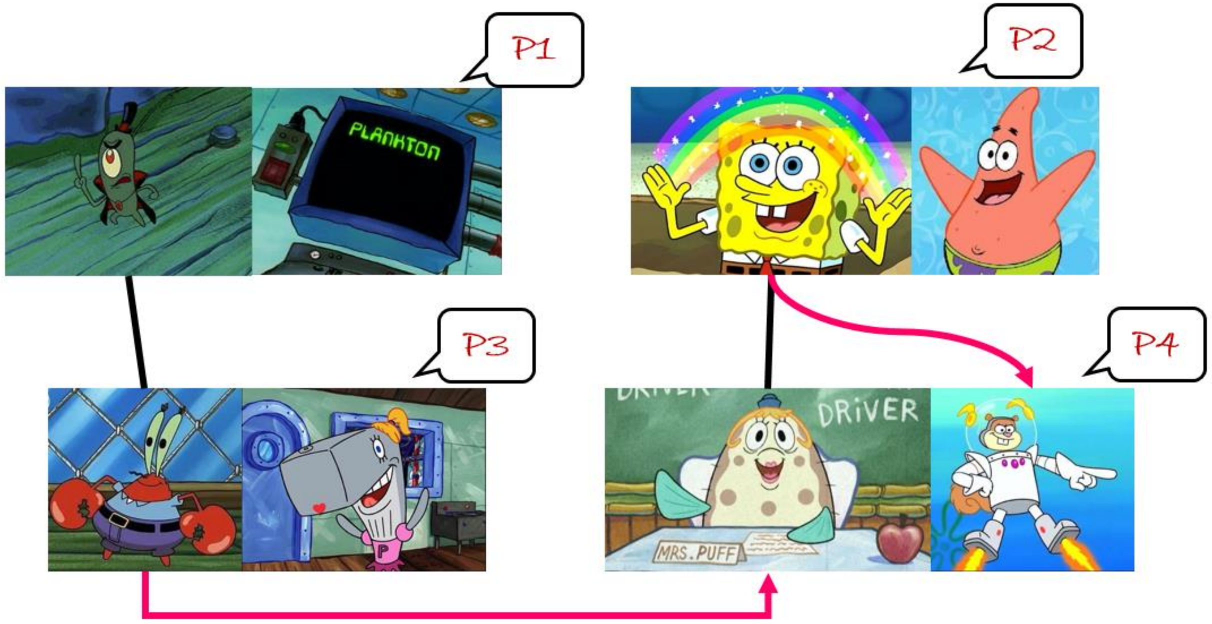
Who should you invite?



Formalize the problem

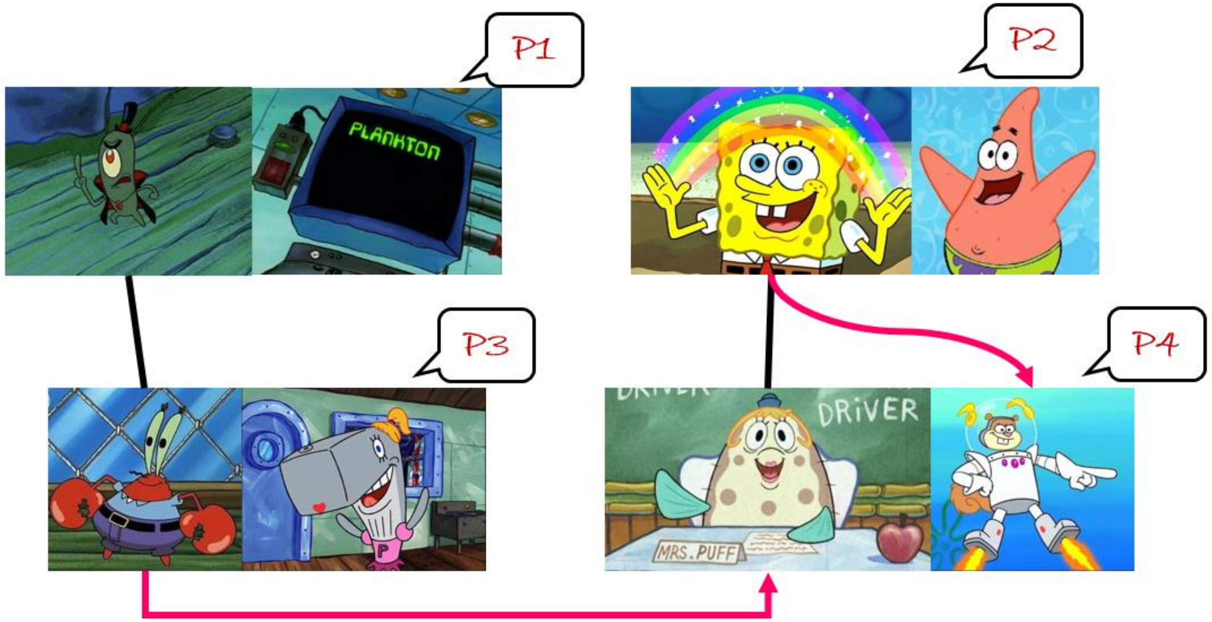
Krabs and Plankton cannot attend together.

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



Formalize the problem

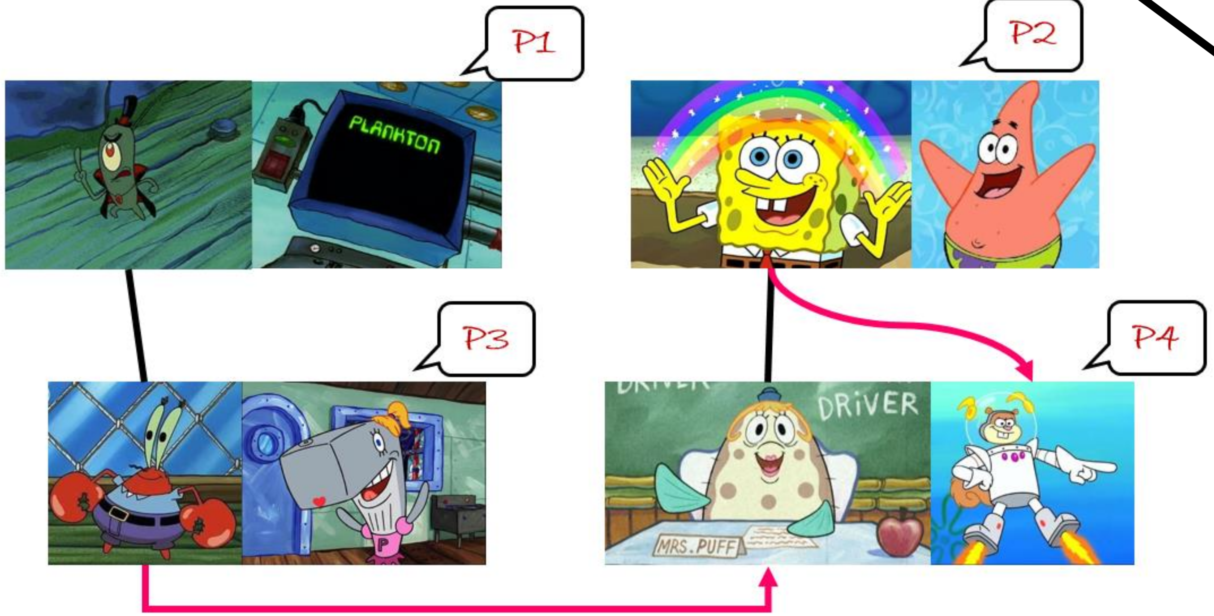
SpongeBob and Mrs. Puff
cannot attend together.



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

Formalize the problem

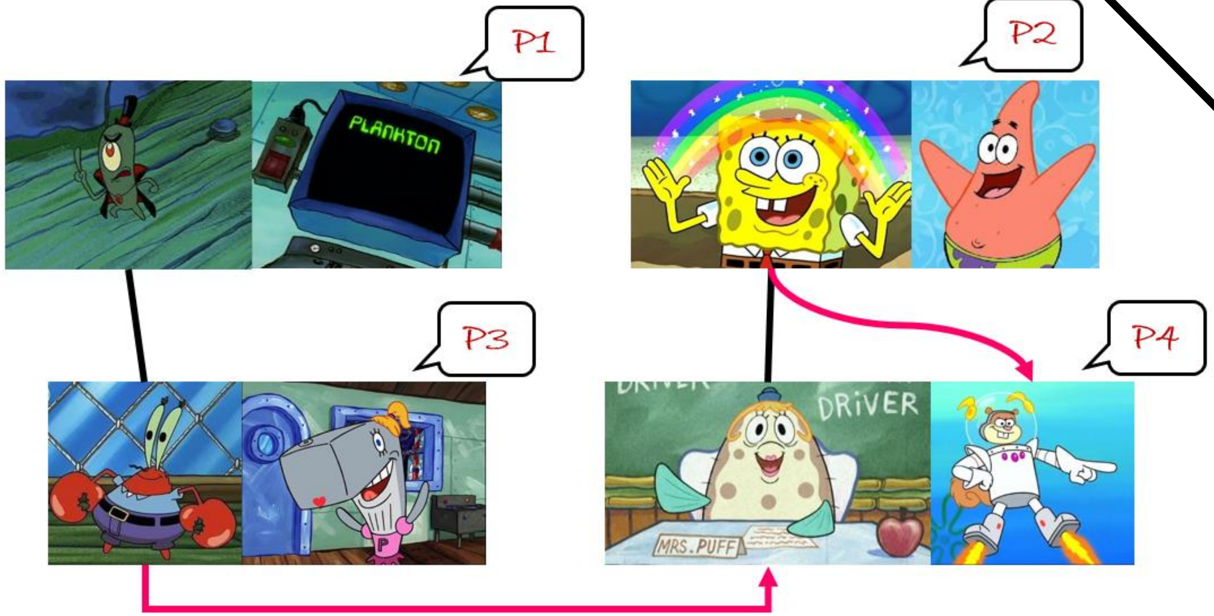
If Sandy attends, SpongeBob must attend.



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

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

$$\neg(P1 \land P3)\land$$
$$\neg(P2 \land P4)\land$$
$$(\neg P4 \to P2)\land$$
$$(P4 \to P3)$$

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

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

MiniSat

Define propositional variables

CNF

Solve the WFF

Print the interpretation

UNSAT

```
int main() {
    Solver solver;
    auto A = solver.newVar();
    auto B = solver.newVar();
    auto C = solver.newVar();
    auto D = solver.newVar();
    solver.addClause( ~mkLit(A),  ~mkLit(C) );
    solver.addClause( ~mkLit(B),  ~mkLit(D) );
    solver.addClause( mkLit(B), mkLit(D) );
    solver.addClause( mkLit(C), ~mkLit(D));

    auto sat = solver.solve();
    if(sat) {
        std::cout << "A := " << (solver.modelValue(A) == l_True) << '\n';
        std::cout << "B := " << (solver.modelValue(B) == l_True) << '\n';
        std::cout << "C := " << (solver.modelValue(C) == l_True) << '\n';
        std::cout << “D := " << (solver.modelValue(D) == l_True) << '\n';
    } else {
        std::cout << “UNSAT\n”;
    }
}
```

SymPy

```
from sympy.logic.inference import satisfiable
from sympy import Symbol

A = Symbol("A")
B = Symbol("B")
C = Symbol("C")
D = Symbol("D")

models = satisfiable(  (~A | ~C) & ( ~B | ~D) & ( B | D ) & ( C | ~D) )

print( models )
```

Define propositional variables

Solve the WFF

Print the interpretation

SymPy

```
from sympy.logic.inference import satisfiable
from sympy import Symbol

A = Symbol(“A”)
B = Symbol(“B”)
C = Symbol(“C”)
D = Symbol(“D”)

models = satisfiable(  (~A & ~C) | ( ~B & ~D) | ( B & D ) | ( C & ~D) )

print( models )
```

Define propositional variables

Print the interpretation

The expression does not need to be CNF

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

Let's focus on a simple case firstly, which has only bool variables.

Straight-Line Code

```
void swap(bool& a, bool& 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.

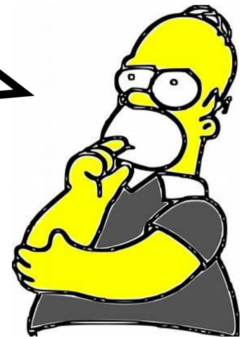
How to convert the program construct to a propositional logic formula?

Straight-Line Code

```
void swap(bool& a, bool& 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.

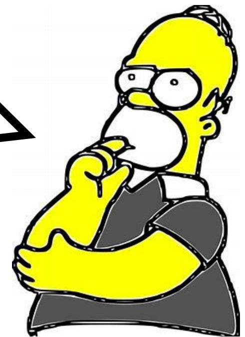


Straight-Line Code

```
void swap(bool& a, bool& 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, a bool variable has two values. A bool variable can be naturally represented by a propositional variable.



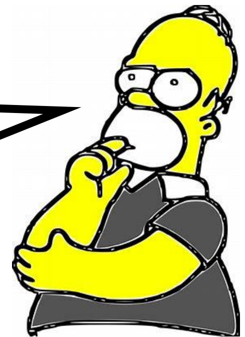
Straight-Line Code

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

a0 b0 } state0

The truth values of 2 propositional variables construct the program state.

How to represent operators in the program?



Straight-Line Code

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

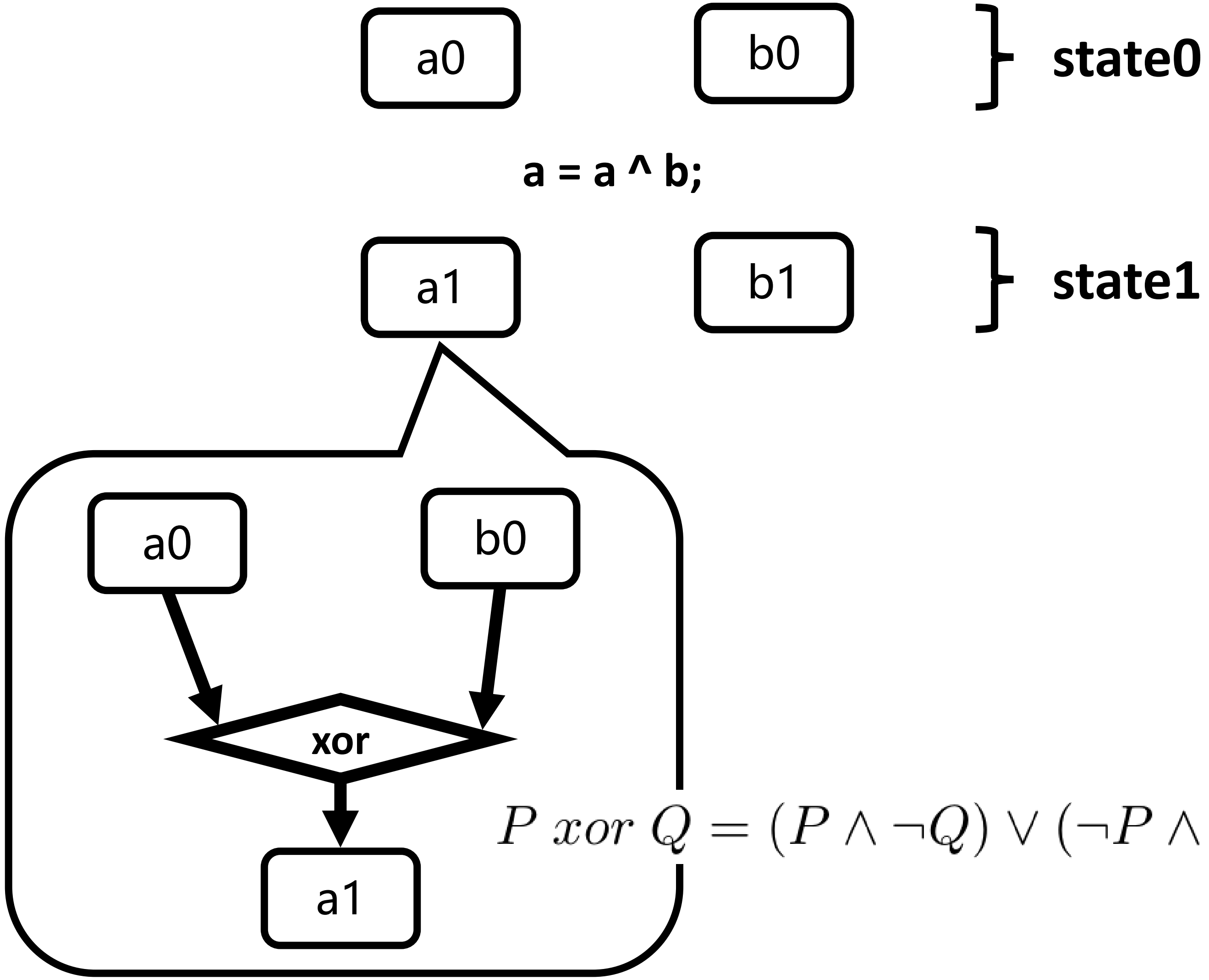
a0 b0 } **state0**

The truth values of 2 propositional variables construct the program state.

Operators can be naturally represented by connectives.

Straight-Line Code

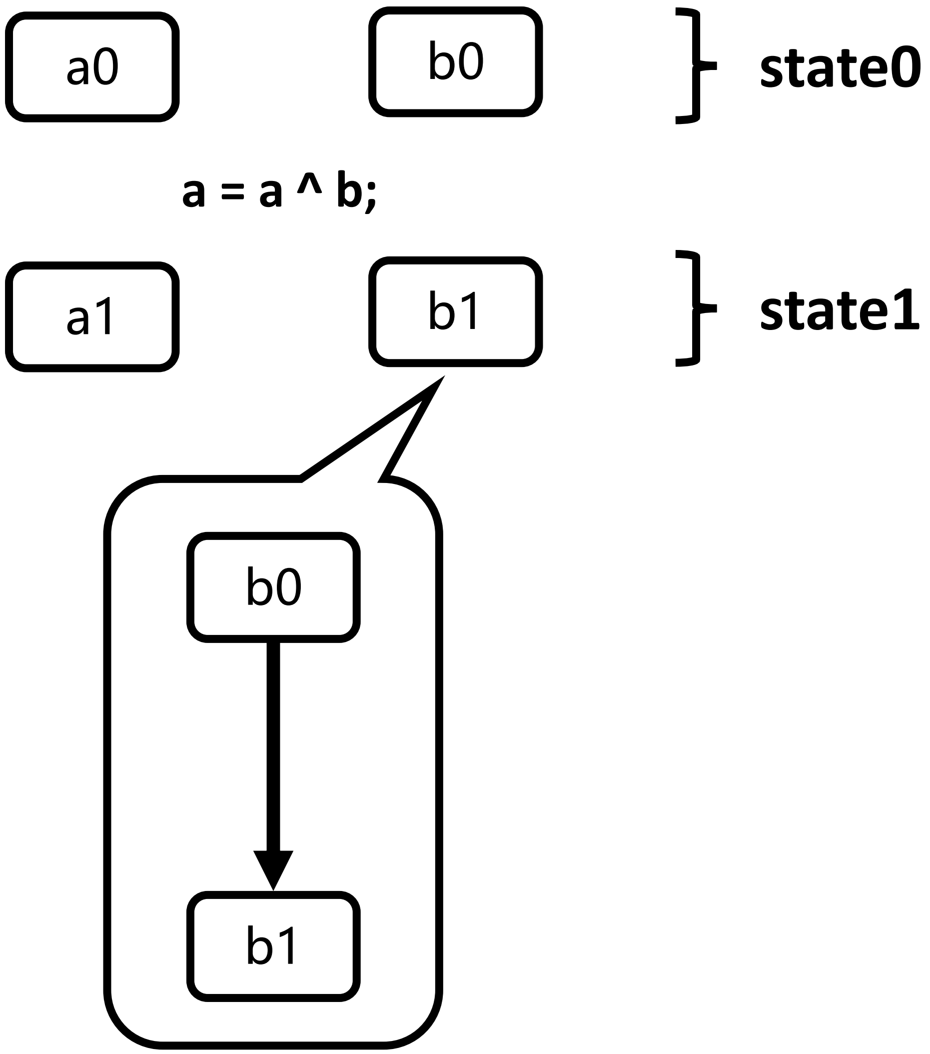
```
void swap(bool& a, bool& b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```



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

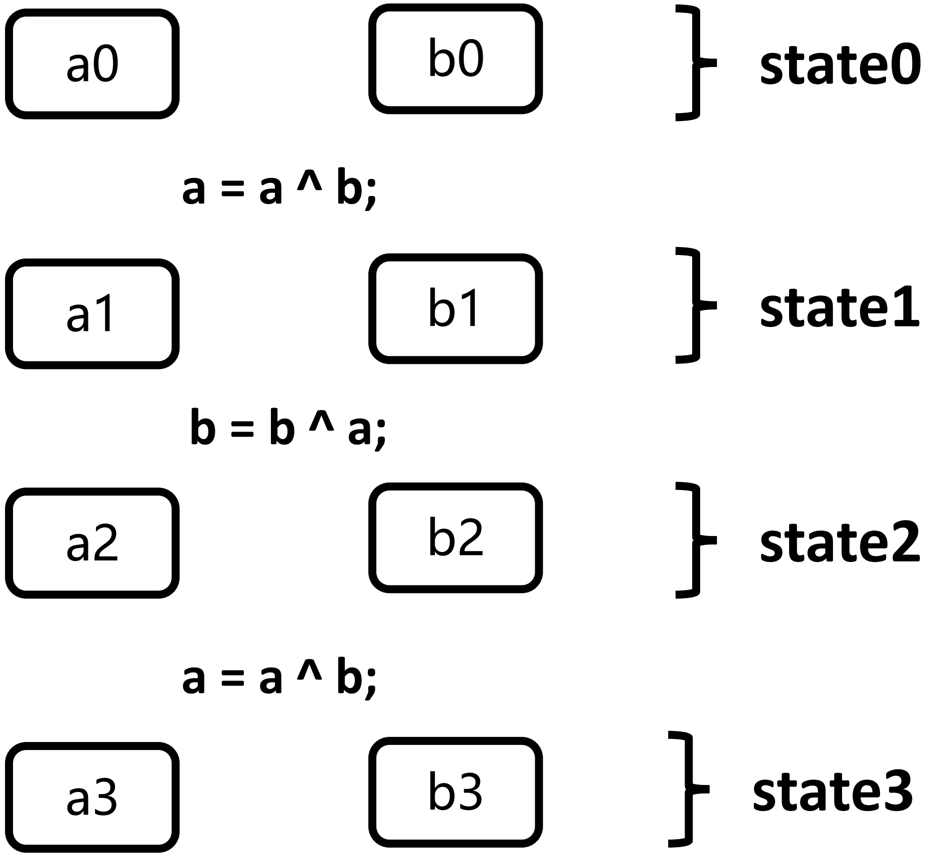
Straight-Line Code

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



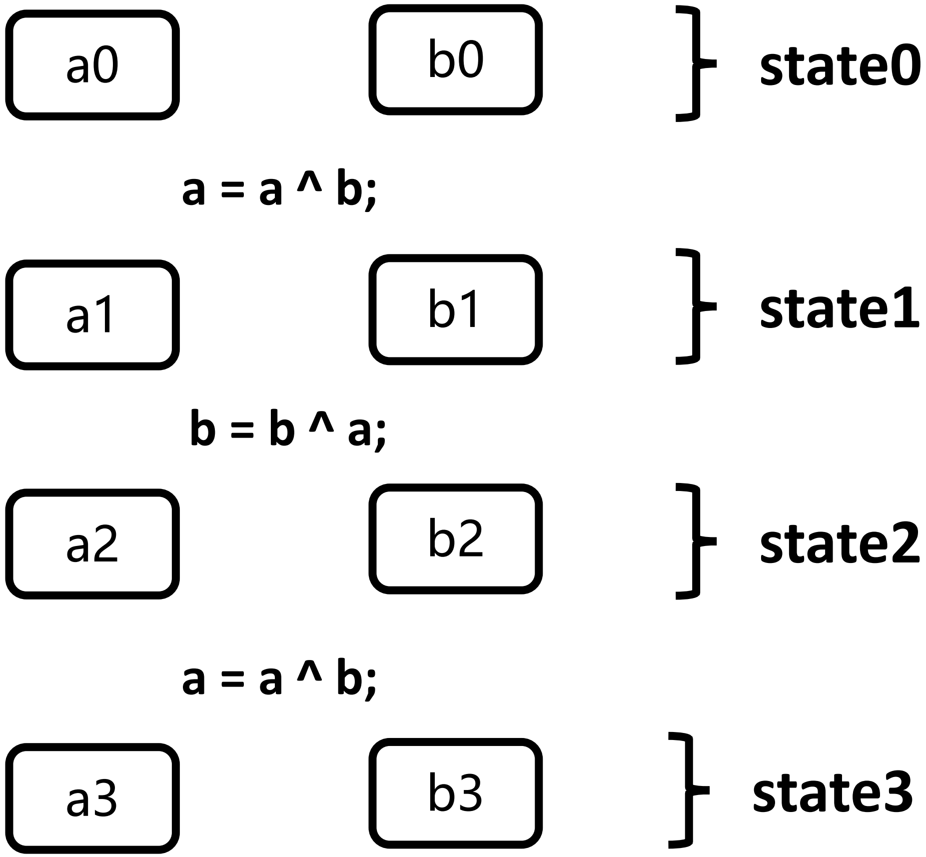
Straight-Line Code

```
void swap(bool& a, bool& 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



$$(A1 \leftrightarrow \boxed{(A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)}) \wedge$$
$$(B1 \leftrightarrow B0) \wedge$$

Straight-Line Code

a0 b0 state0

a = a ^ b;

a1 b1 state1

b = b ^ a;

a2 b2 state2

a = a ^ b;

a3 b3 state3

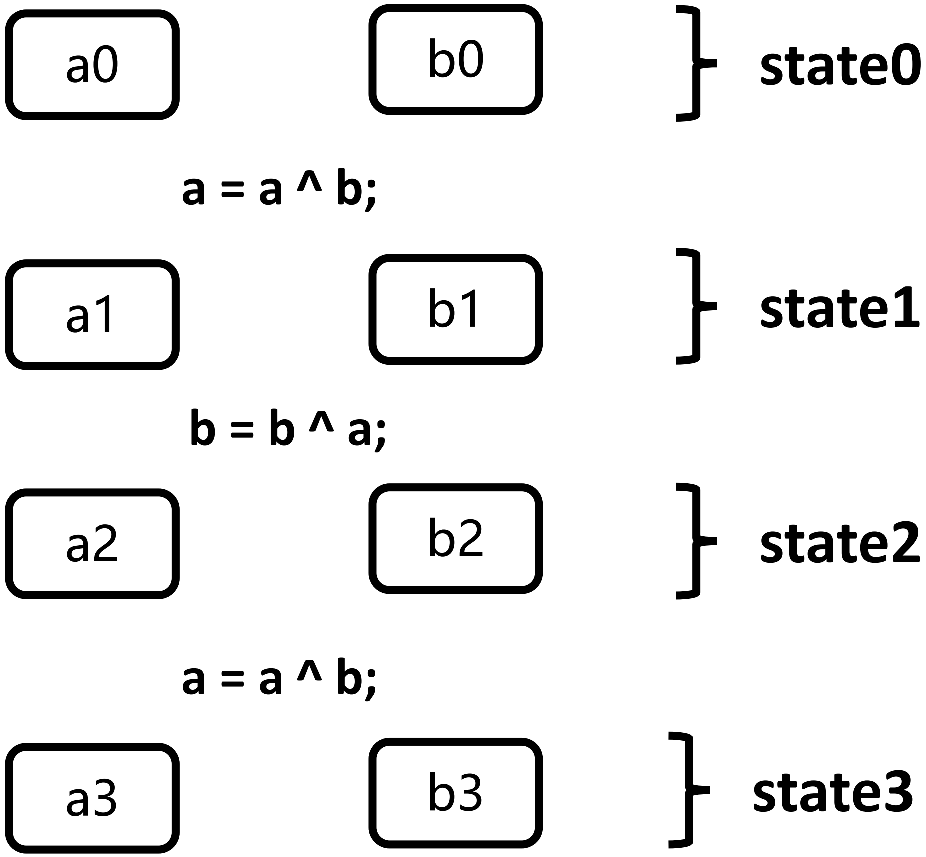
$$(A1 \leftrightarrow \boxed{(A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)}) \wedge$$

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

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

$$(B2 \leftrightarrow \boxed{(A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)}) \wedge$$

Straight-Line Code



$$(A1 \leftrightarrow \boxed{(A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)}) \wedge$$

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

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

$$(B2 \leftrightarrow \boxed{(A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)}) \wedge$$

$$(A3 \leftrightarrow \boxed{(A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)}) \wedge$$

$$(B3 \leftrightarrow B2)$$

Straight-Line Code

a0 b0 } **state0**

a = a ^ b;

a1 b1 } **state1**

b = b ^ a;

a2 b2 } **state2**

a = a ^ b;

a3 b3 } **state3**

How to prove a and b are swapped?

$$(A1 \leftrightarrow \boxed{(A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)}) \wedge$$

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

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

$$(B2 \leftrightarrow \boxed{(A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)}) \wedge$$

$$(A3 \leftrightarrow \boxed{(A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)}) \wedge$$

$$(B3 \leftrightarrow B2)$$

Straight-Line Code

$$\left.\begin{array}{l}
(A1 \leftrightarrow \boxed{(A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)}) \wedge \\
(B1 \leftrightarrow B0) \wedge \\
(A2 \leftrightarrow A1) \wedge \\
(B2 \leftrightarrow \boxed{(A1 \wedge \neg B1) \vee (\neg A1 \wedge B1)}) \wedge \\
(A3 \leftrightarrow \boxed{(A2 \wedge \neg B2) \vee (\neg A2 \wedge B2)}) \wedge \\
(B3 \leftrightarrow B2) \rightarrow
\end{array}\right\} \text{Program}$$

$$\left.(A3 \leftrightarrow B0) \wedge (A0 \leftrightarrow B3)\right\} \text{Assertion}$$

Straight-Line Code

$$
\begin{aligned}
&(A1 \leftrightarrow \boxed{(A0 \land \neg B0) \lor (\neg A0 \land B0)}) \land \\
&(B1 \leftrightarrow B0) \land \\
&(A2 \leftrightarrow A1) \land \\
&(B2 \leftrightarrow \boxed{(A1 \land \neg B1) \lor (\neg A1 \land B1)}) \land \\
&(A3 \leftrightarrow \boxed{(A2 \land \neg B2) \lor (\neg A2 \land B2)}) \land \\
&(B3 \leftrightarrow B2) \rightarrow \\
&(A3 \leftrightarrow B0) \land (A0 \leftrightarrow B3)
\end{aligned}
$$

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

$$\begin{aligned}
&(A1 \leftrightarrow (A0 \land \neg B0) \lor (\neg A0 \land B0))\land \\
&(B1 \leftrightarrow B0)\land \\
&(A2 \leftrightarrow A1)\land \\
&(B2 \leftrightarrow (A1 \land \neg B1) \lor (\neg A1 \land B1))\land \\
&(A3 \leftrightarrow (A2 \land \neg B2) \lor (\neg A2 \land B2))\land \\
&(B3 \leftrightarrow B2)\boxed{\rightarrow} \\
&\boxed{(A3 \leftrightarrow B0) \land (A0 \leftrightarrow B3)}
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
&(A1 \leftrightarrow (A0 \land \neg B0) \lor (\neg A0 \land B0))\land \\
&(B1 \leftrightarrow B0)\land \\
&(A2 \leftrightarrow A1)\land \\
&(B2 \leftrightarrow (A1 \land \neg B1) \lor (\neg A1 \land B1))\land \\
&(A3 \leftrightarrow (A2 \land \neg B2) \lor (\neg A2 \land B2))\land \\
&(B3 \leftrightarrow B2)\boxed{\land} \\
&\boxed{\neg((A3 \leftrightarrow B0) \land (A0 \leftrightarrow B3))}
\end{aligned}$$

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

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

But now the formula is unsatisfiable. So a and b are swapped.

Straight-Line Code

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

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

$$(A1 \leftrightarrow (A0 \land \neg B0) \lor (\neg A0 \land B0)) \land$$
$$(B1 \leftrightarrow B0) \land$$
$$(A2 \leftrightarrow A1) \land$$
$$(B2 \leftrightarrow (A1 \land \neg B1) \lor (\neg A1 \land B1)) \land$$
$$(A3 \leftrightarrow (A2 \land \neg B2) \lor (\neg A2 \land B2)) \land$$
$$(B3 \leftrightarrow B2) \land$$
$$\neg((A3 \leftrightarrow B0) \land (A0 \leftrightarrow B3))$$

$$(A1 \leftrightarrow (A0 \land \neg B0) \lor (\neg A0 \land B0)) \land$$
$$(B2 \leftrightarrow (A1 \land \neg B0) \lor (\neg A1 \land B0)) \land$$
$$(A3 \leftrightarrow (A1 \land \neg B2) \lor (\neg A1 \land B2)) \land$$
$$\neg((A3 \leftrightarrow B0) \land (A0 \leftrightarrow B2))$$

Exercise

```
void f(bool x, bool y)
{
    bool 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

What if there are some "if"?

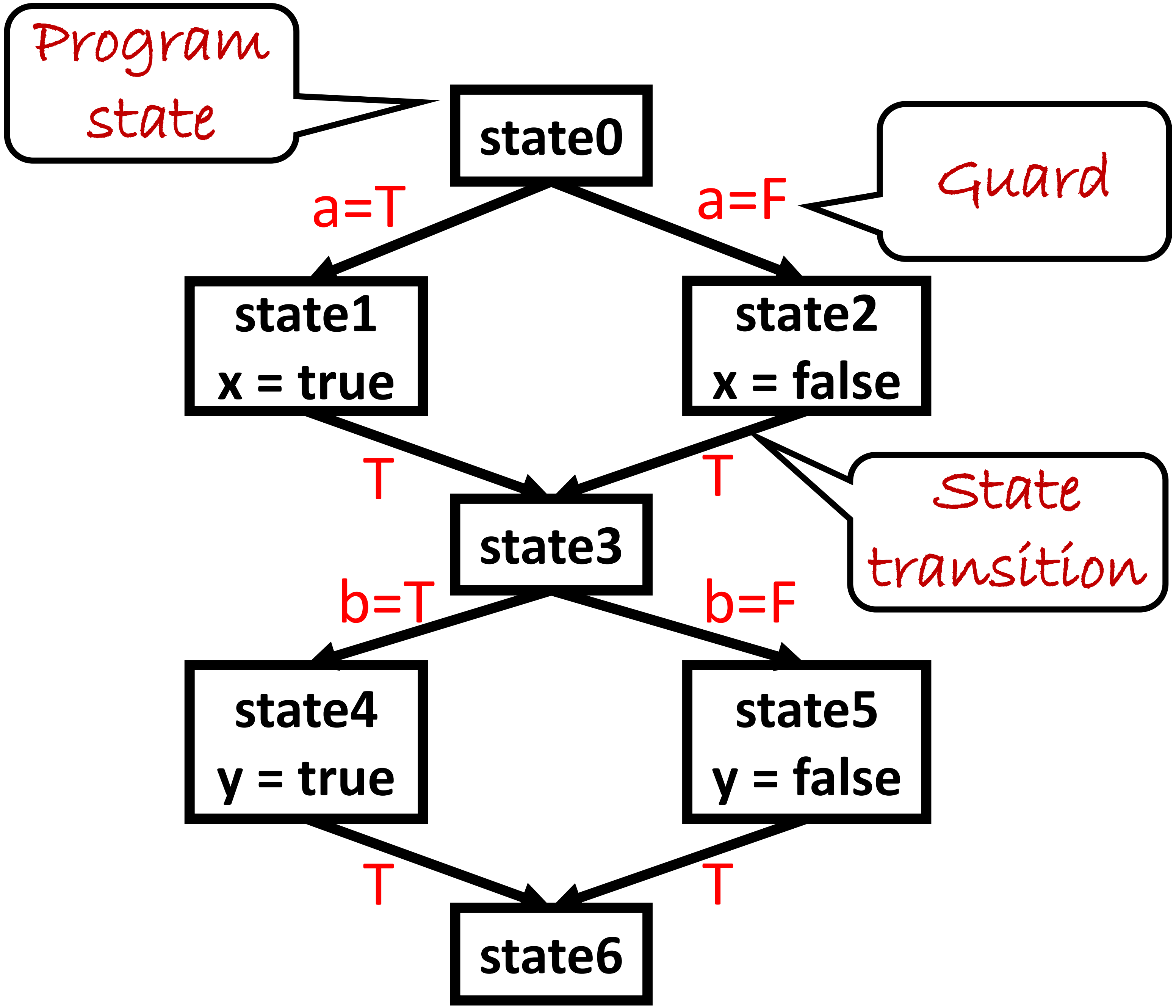
Control Flow

```
void f(bool a, bool b)
{
    bool x, y;
    if (a)
        x = true;
    else
        x = false;
    if (b)
        y = true;
    else
        y = false;
    assert(x || y);
}
```

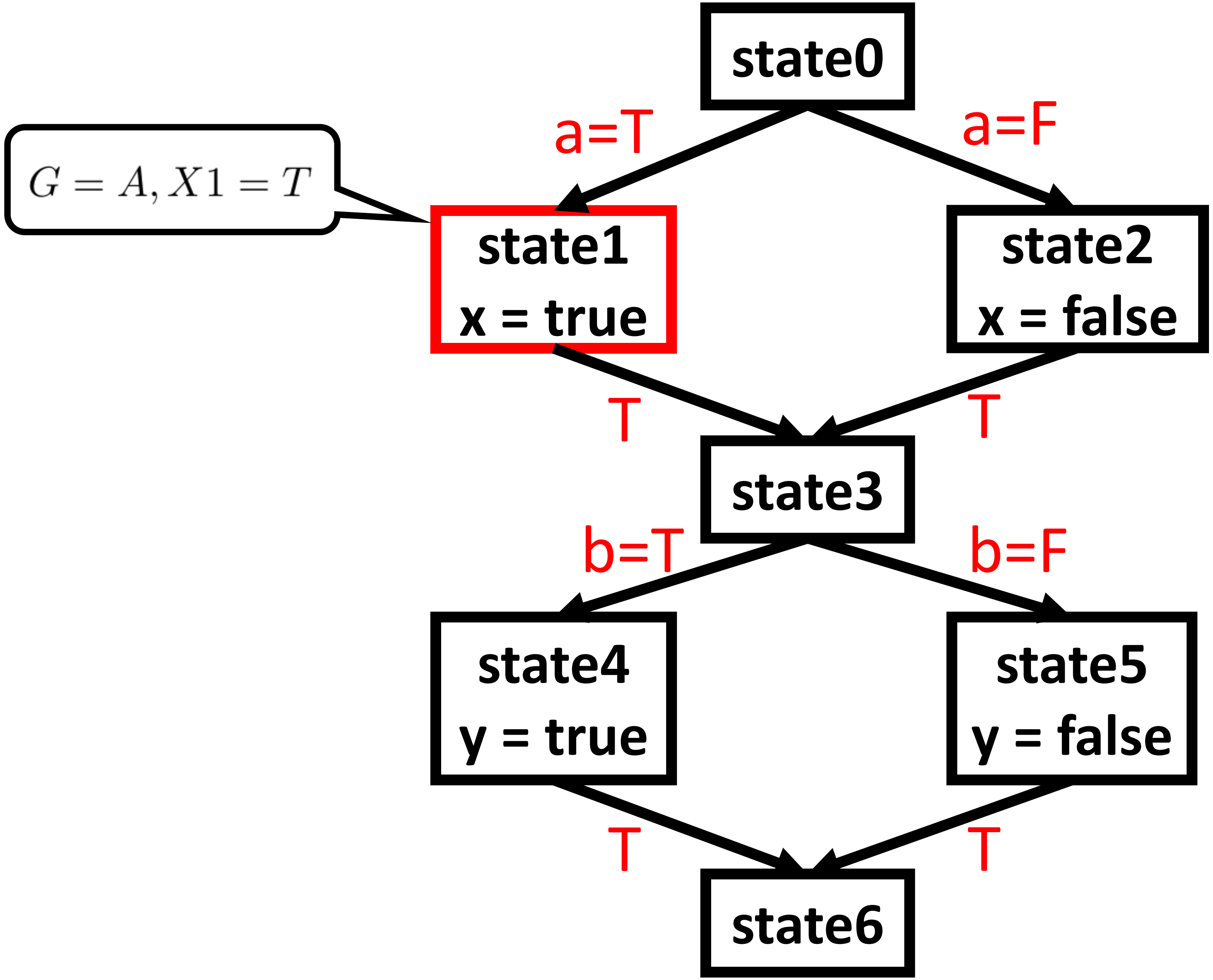
How to determine if the assertion will fail?

Control Flow

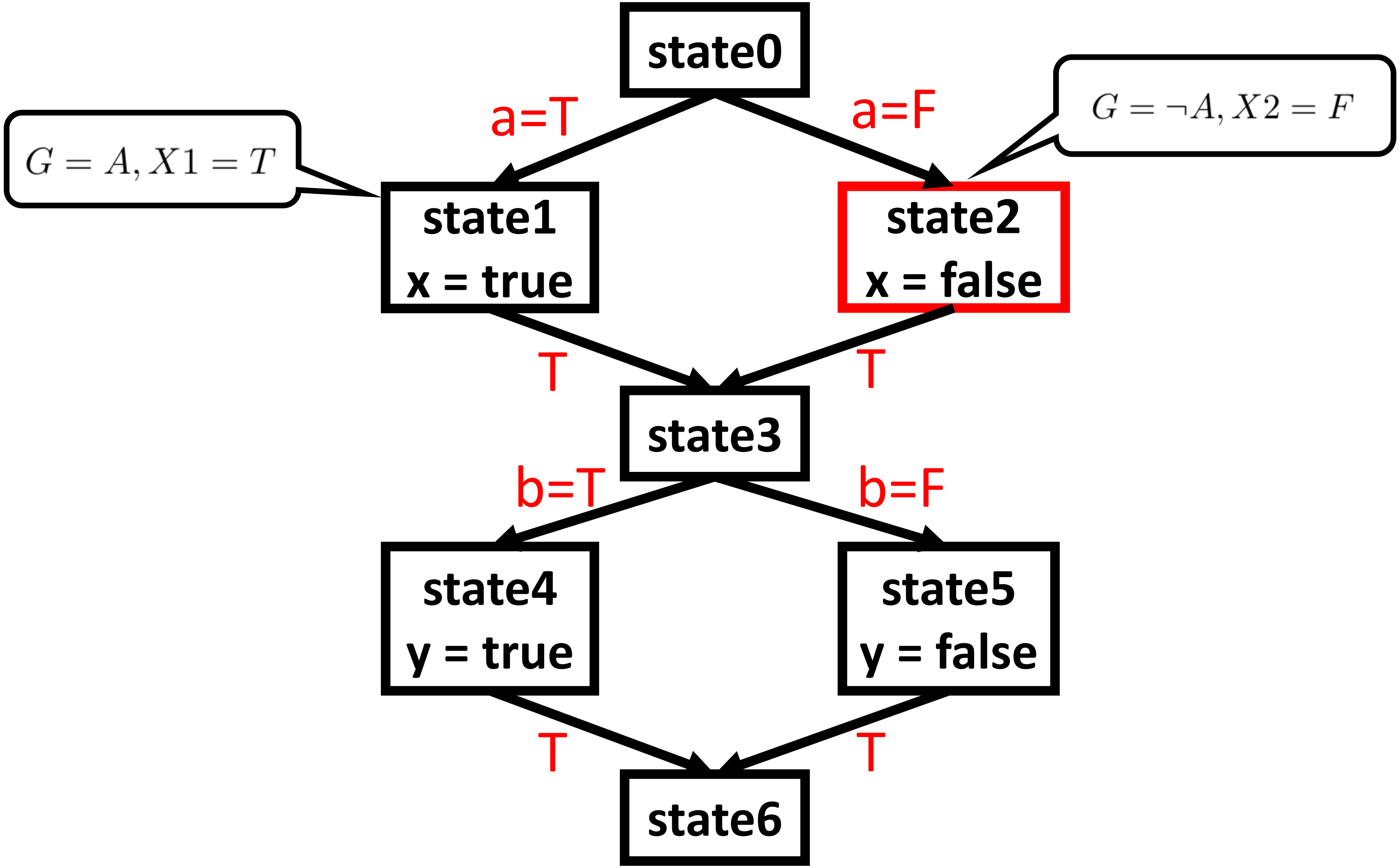
```
void f(bool a, bool b)
{
    bool x, y;
    if (a)
        x = true;
    else
        x = false;
    if (b)
        y = true;
    else
        y = false;
    assert(x || y);
}
```



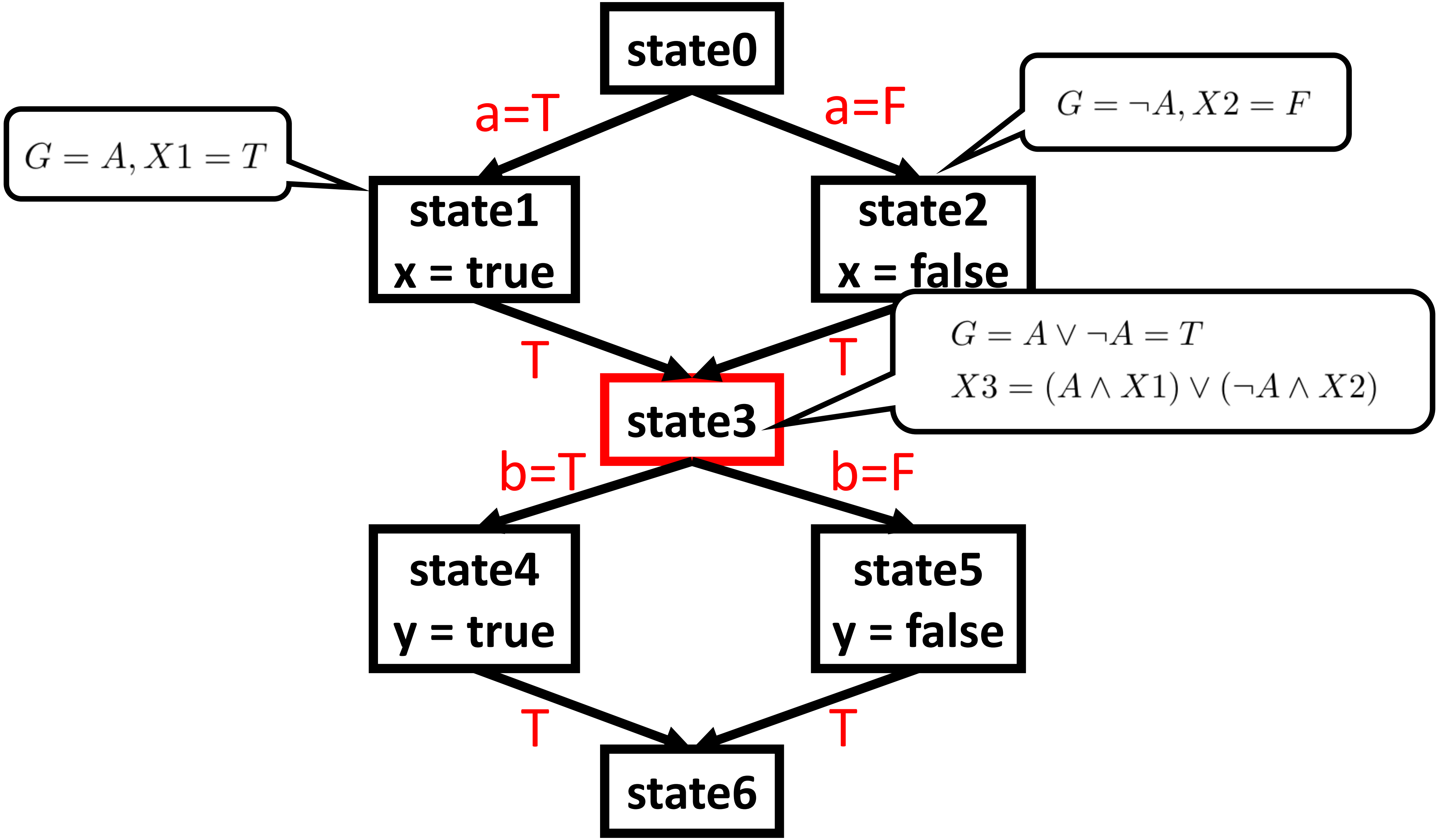
Control Flow



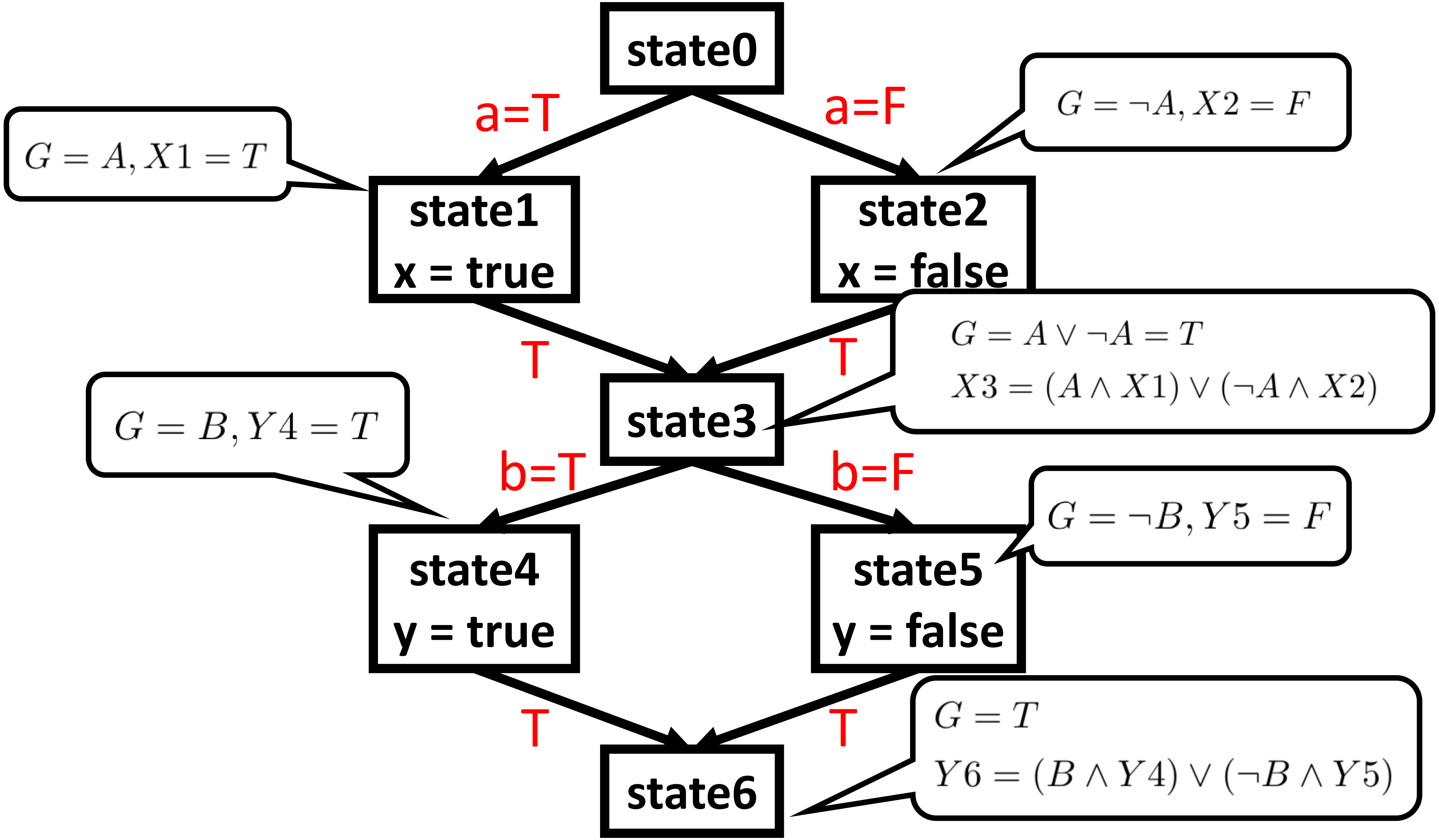
Control Flow



Control Flow



Control Flow



Control Flow

```
void f(bool a, bool b)
{
    bool x, y;
    if (a)
        x = true;
    else
        x = false;
    if (b)
        y = true;
    else
        y = false;
    assert(x || y);
}
```

This can be represented in propositional logic

$$X1 \wedge \neg X2 \wedge Y4 \wedge \neg Y5 \wedge$$
$$(X3 \leftrightarrow ((A \wedge X1) \vee (\neg A \wedge X2))) \wedge$$
$$(Y6 \leftrightarrow ((B \wedge Y4) \vee (\neg B \wedge Y5)))$$
$$\rightarrow$$
$$(X3 \vee Y6)$$

Prove it is a tautology.

Control Flow

```
void f(bool a, bool b)
{
    bool x, y;
    if (a)
        x = true;
    else
        x = false;
    if (b)
        y = true;
    else
        y = false;
    assert(x || y);
}
```

$$
\begin{aligned}
&X1 \wedge \neg X2 \wedge Y4 \wedge \neg Y5 \wedge \\
&(X3 \leftrightarrow ((A \wedge X1) \vee (\neg A \wedge X2)))\wedge \\
&(Y6 \leftrightarrow ((B \wedge Y4) \vee (\neg B \wedge Y5))) \\
&\wedge \\
&\neg(X3 \vee Y6)
\end{aligned}
$$

Prove it is unsatisfiable

Control Flow

```
void f(bool a, bool b)
{
    bool x, y;
    if (a)
        x = true;
    else
        x = false;
    if (b)
        y = true;
    else
        y = false;
    assert(x || y);
}
```

$$X1 \land \neg X2 \land Y4 \land \neg Y5 \land$$
$$(X3 \leftrightarrow ((A \land X1) \lor (\neg A \land X2))) \land$$
$$(Y6 \leftrightarrow ((B \land Y4) \lor (\neg B \land Y5)))$$
$$\land$$
$$\neg(X3 \lor Y6)$$

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

Integer

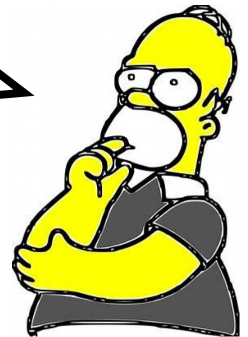
What about integers?

Integer

```
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.

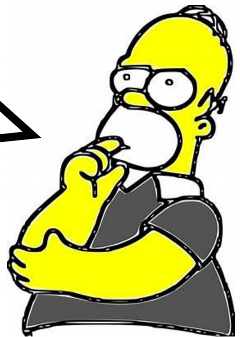


Integer

```
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.



Integer

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

a31	…	a0

b31	…	b0

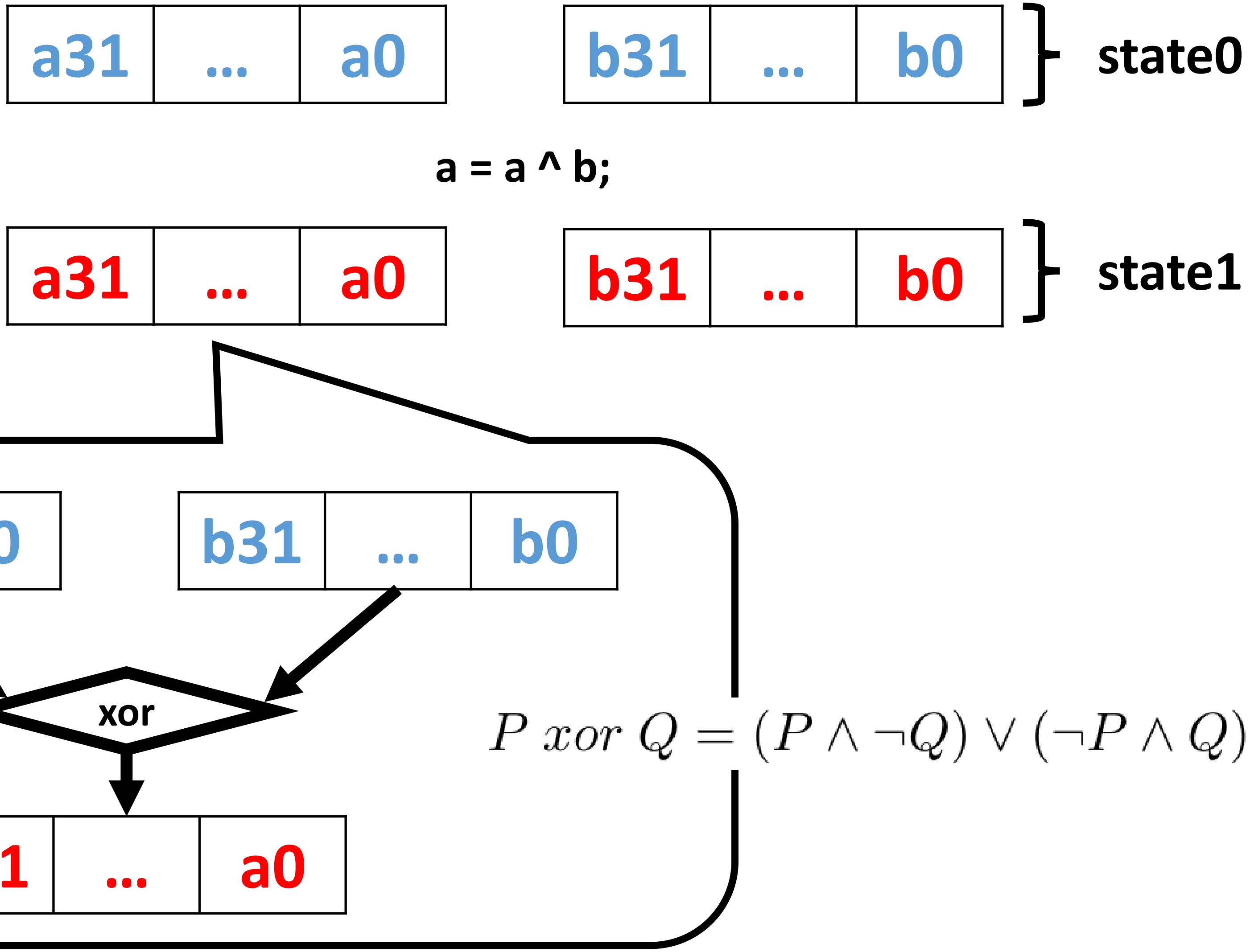
} state0

The truth values of 64 variables construct the program state.

Operators can be naturally represented by connectives.

Integer

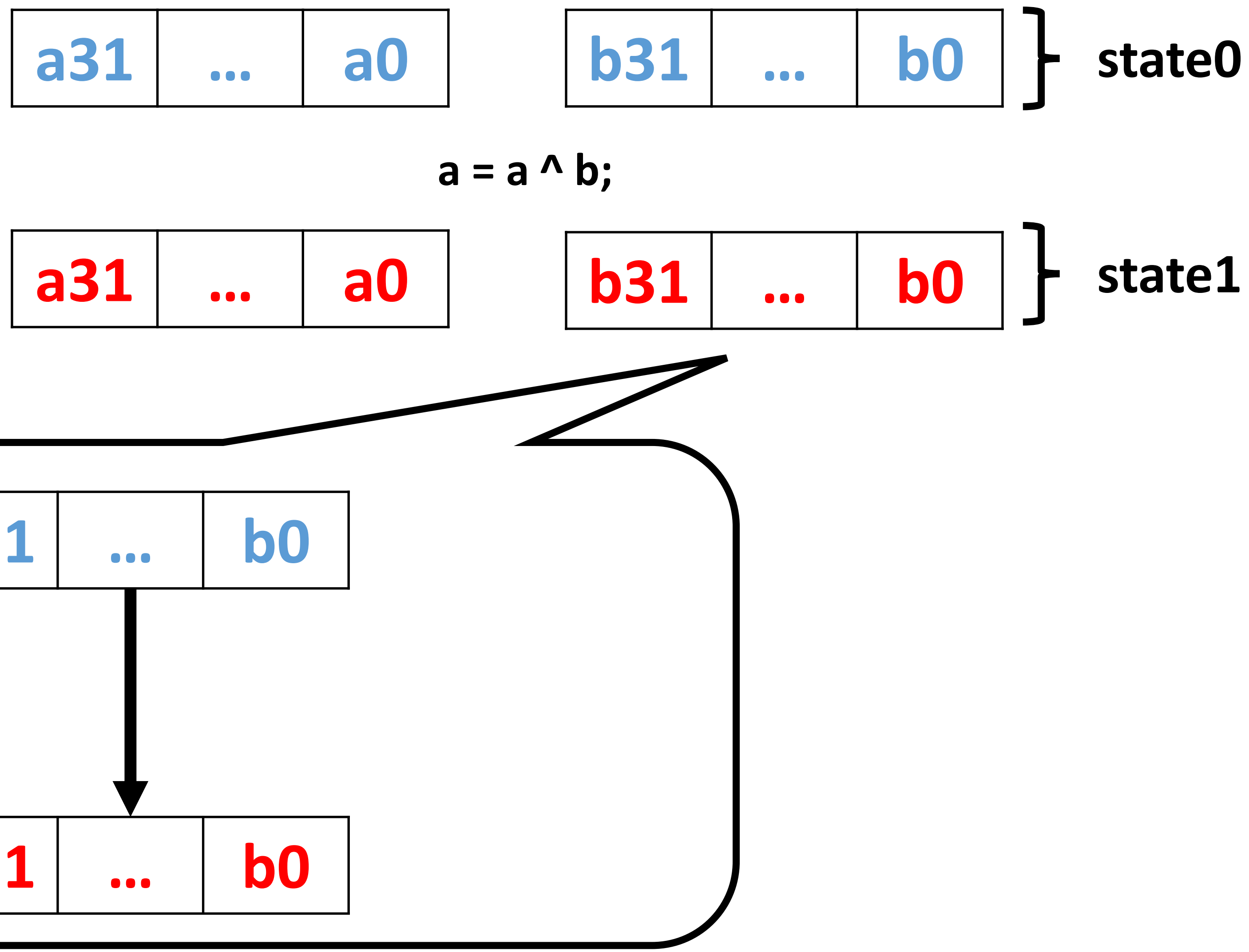
```
void swap(int& a, int& b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```



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

Integer

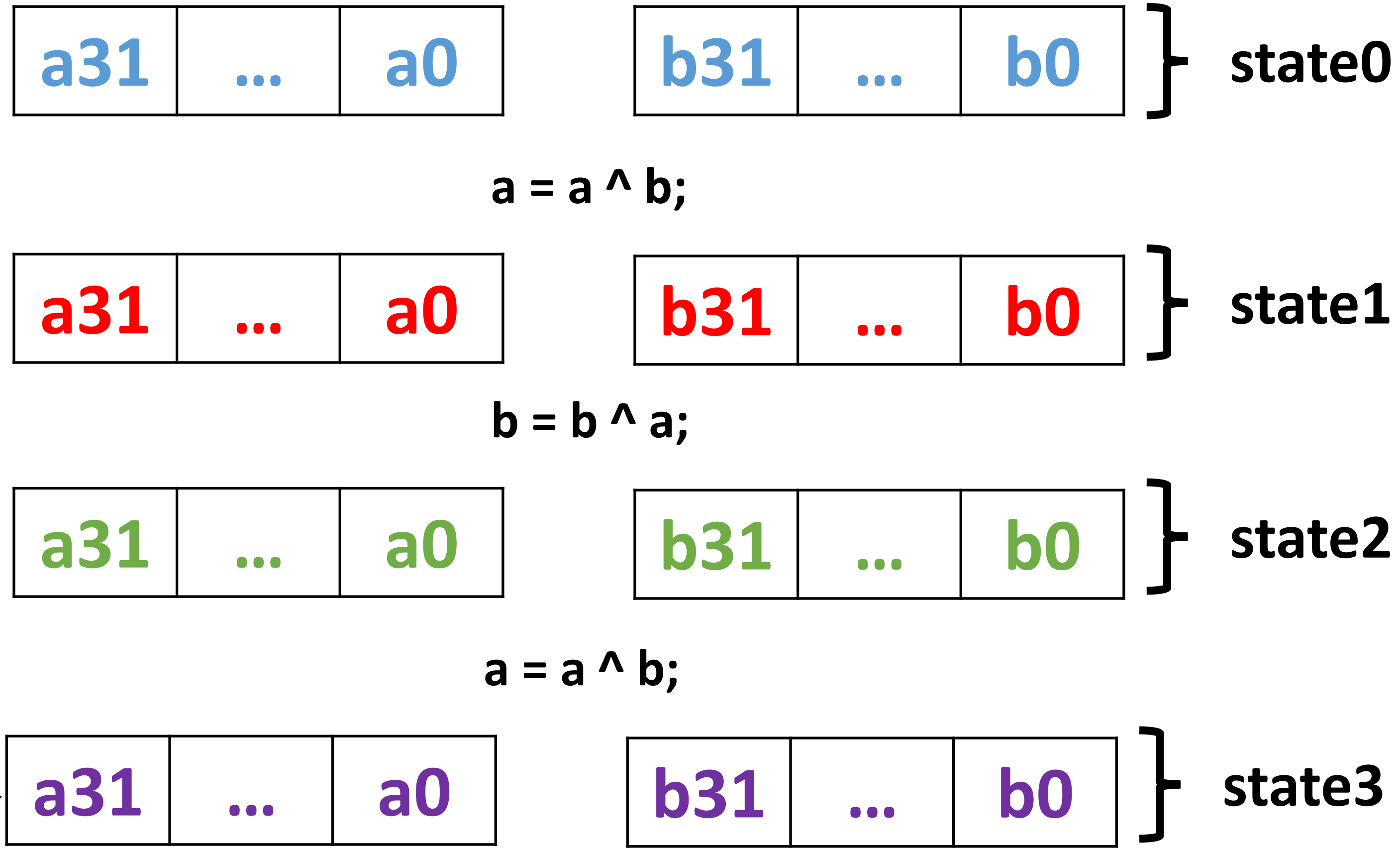
```
void swap(int& a, int& b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```



Integer

```
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.

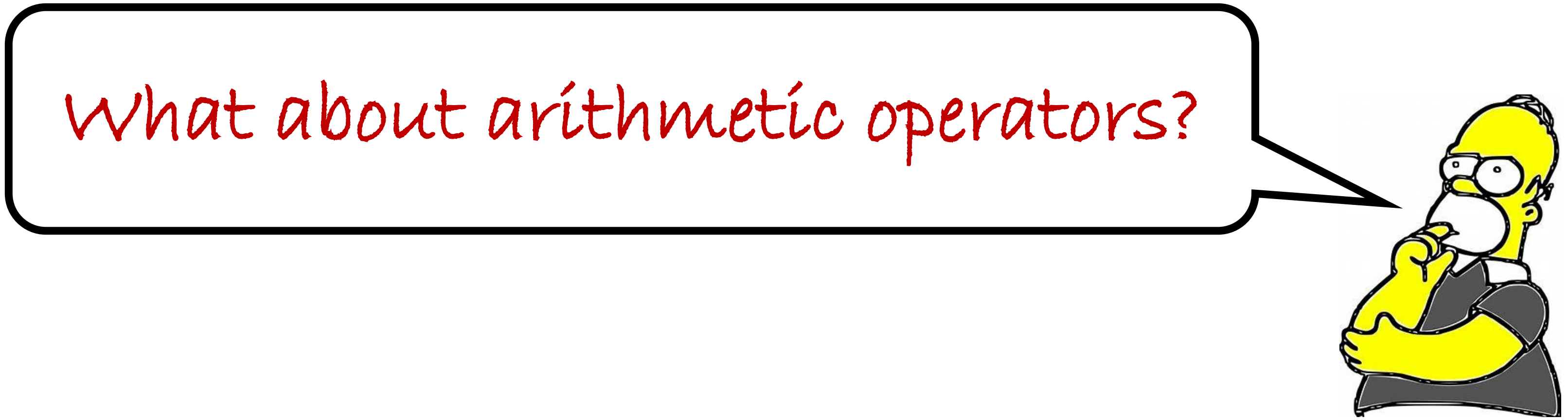


Operators

What language operators can be represented by connectives?

Operators

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



Operators

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

All of operators can be represented by connectives.

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

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);
}
```

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?

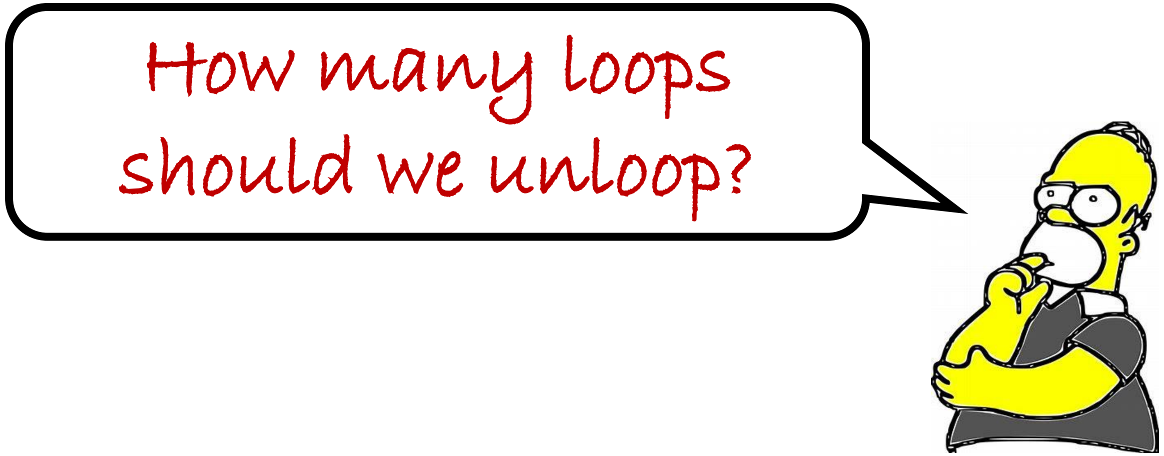
Loops

Stra

ine Code

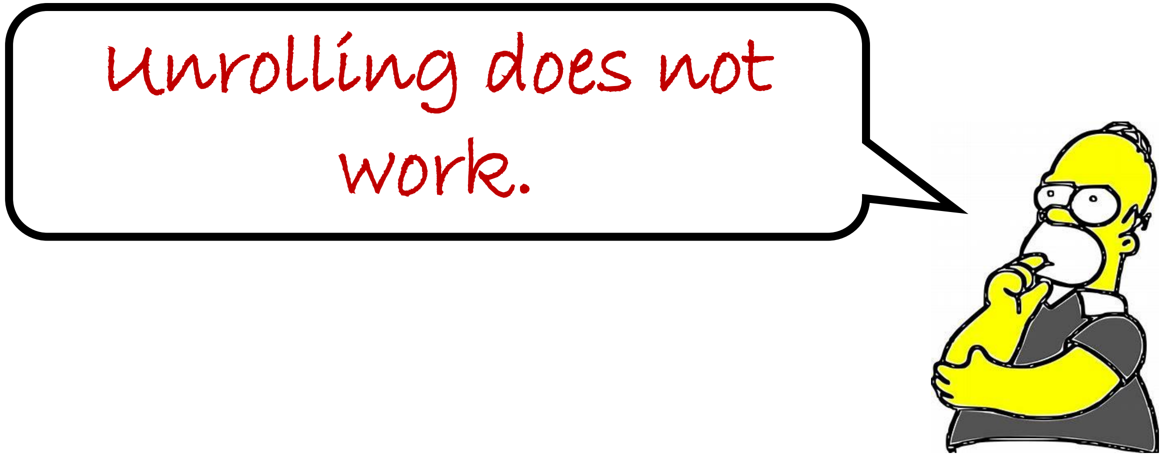
Loops

```
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;
    }
}
```



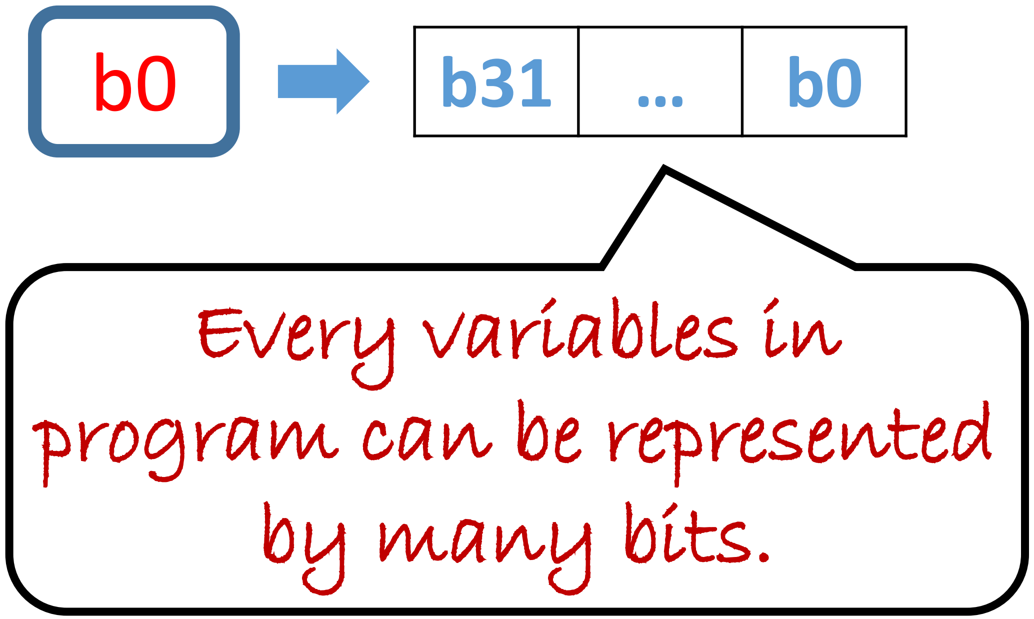
- Unroll loops k times, drop backedges
- May miss errors that are deeply buried
- Simplicity

Summary

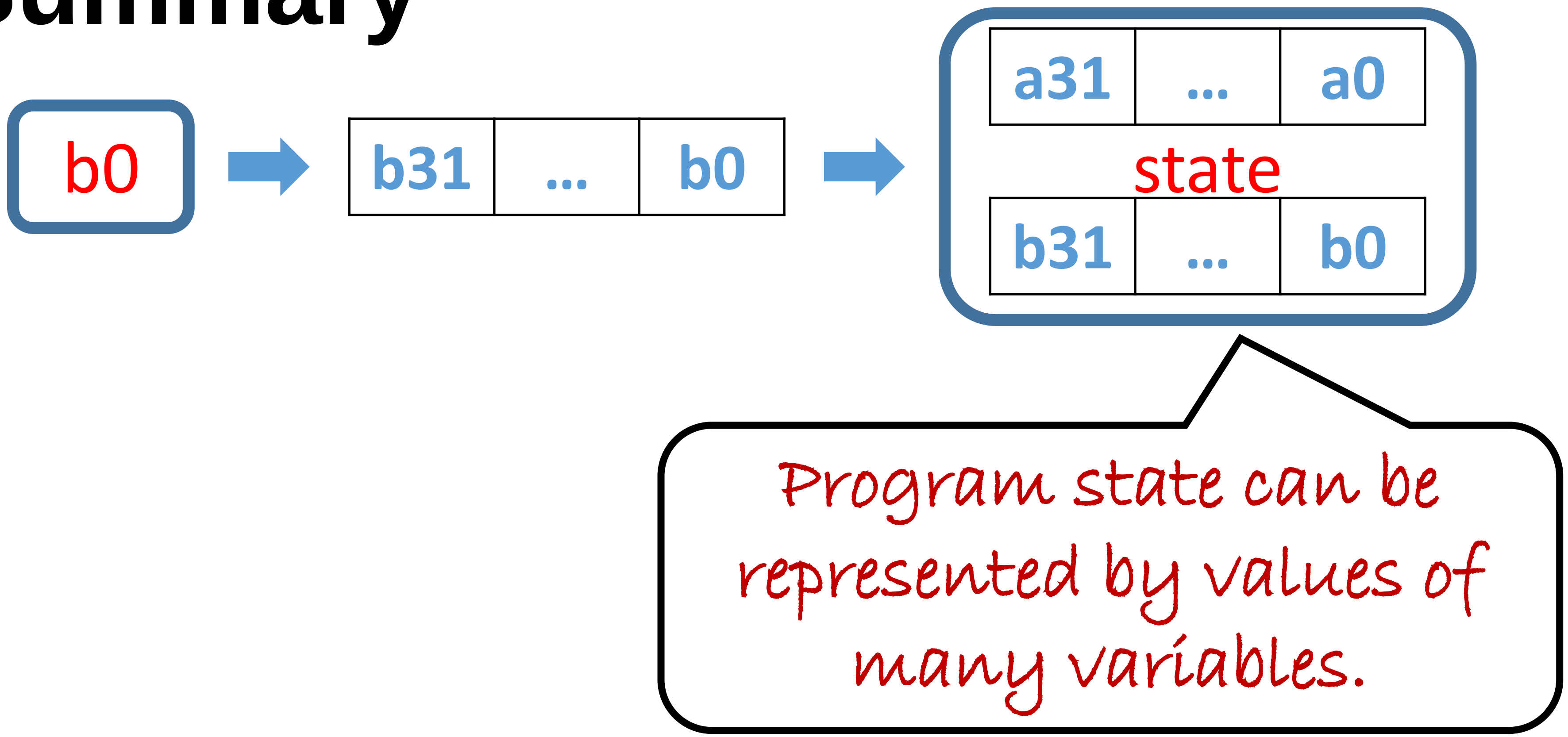
b0

Every bit can be represented by a propositional variable.

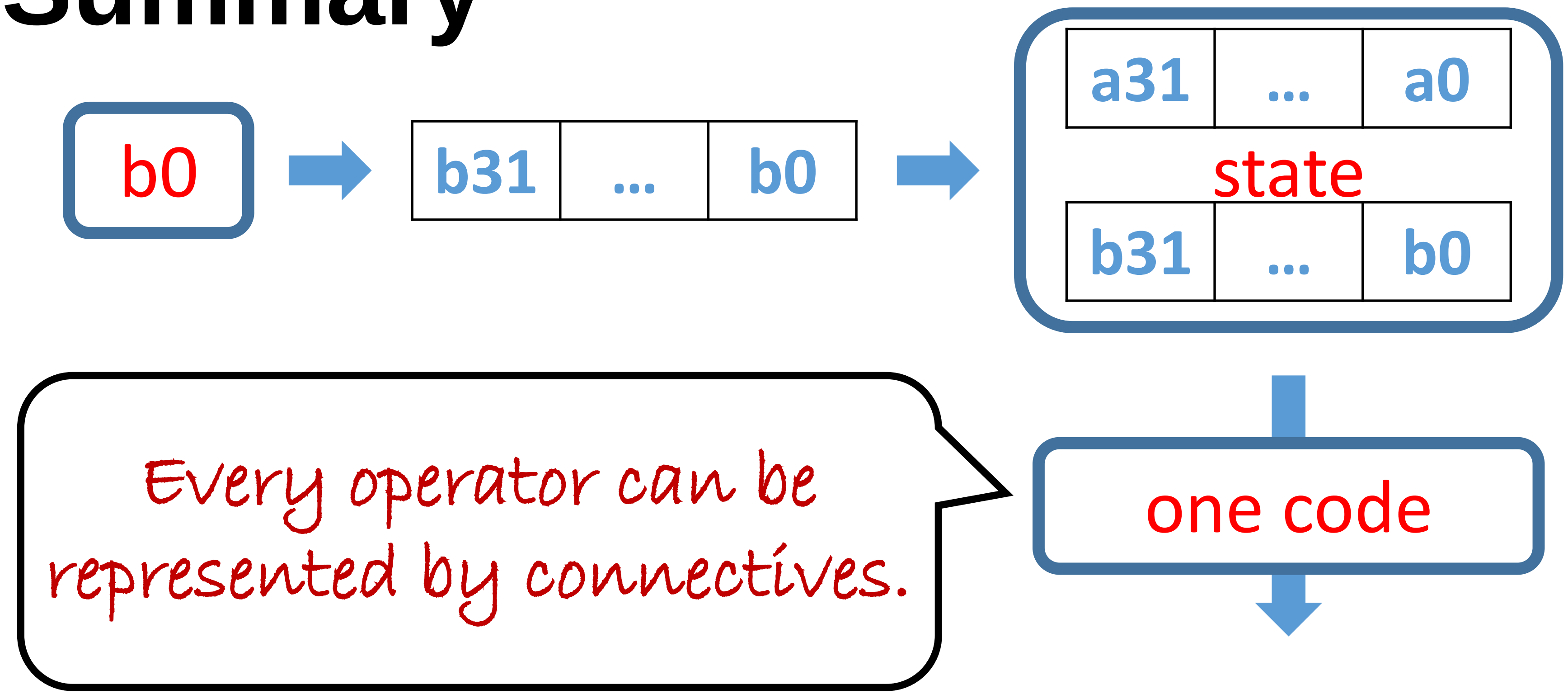
Summary


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

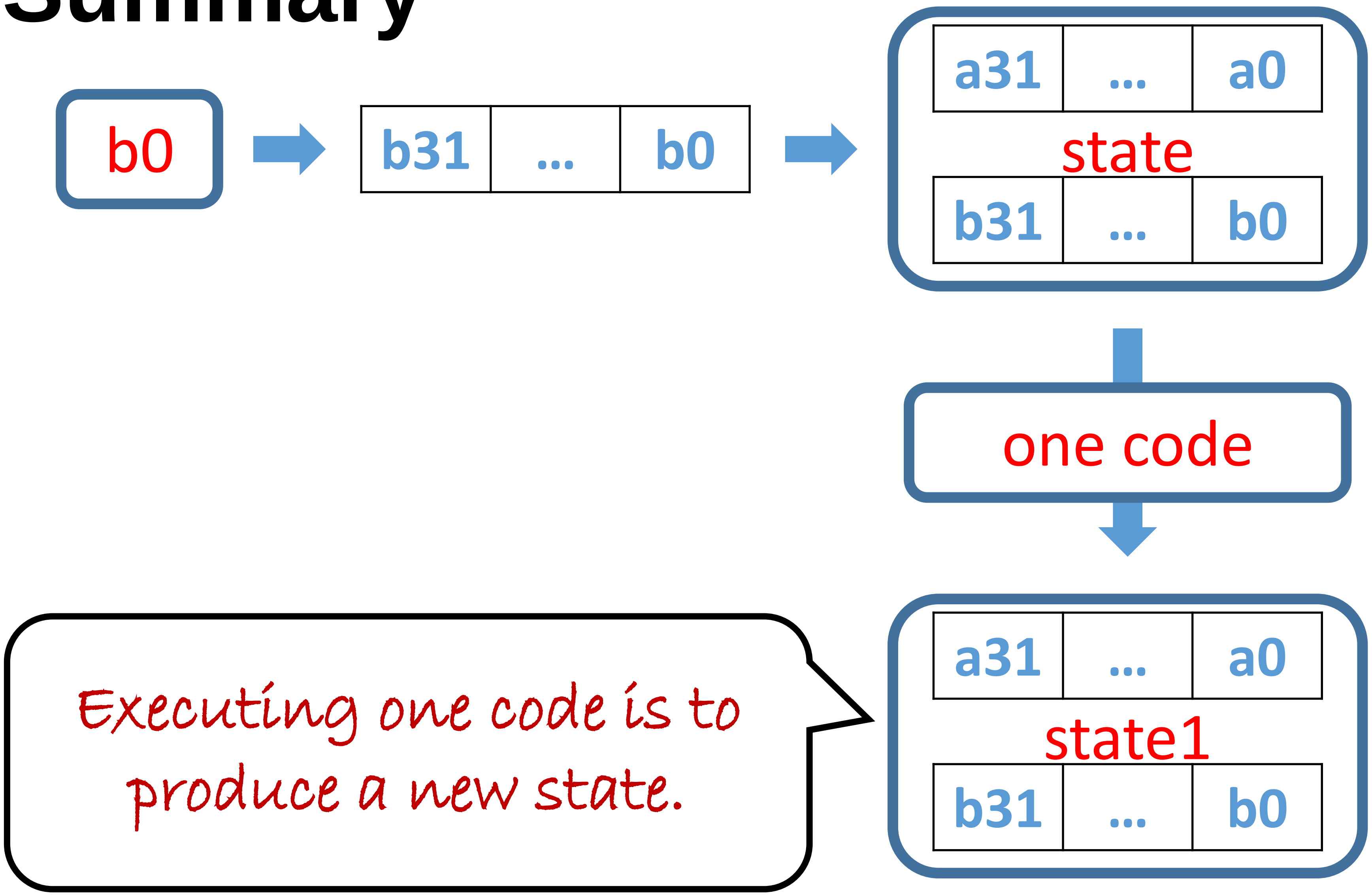
Summary


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

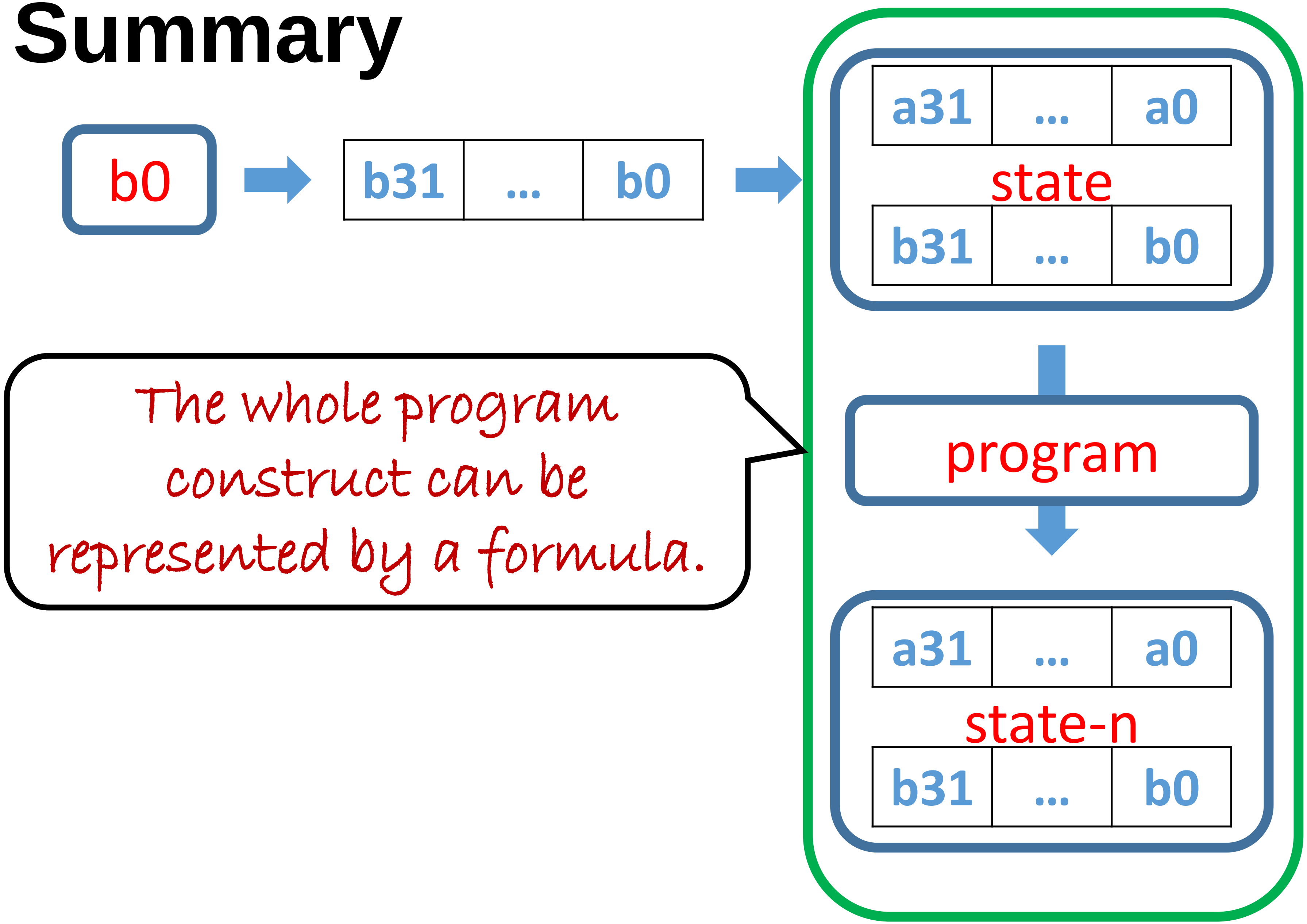
Summary


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

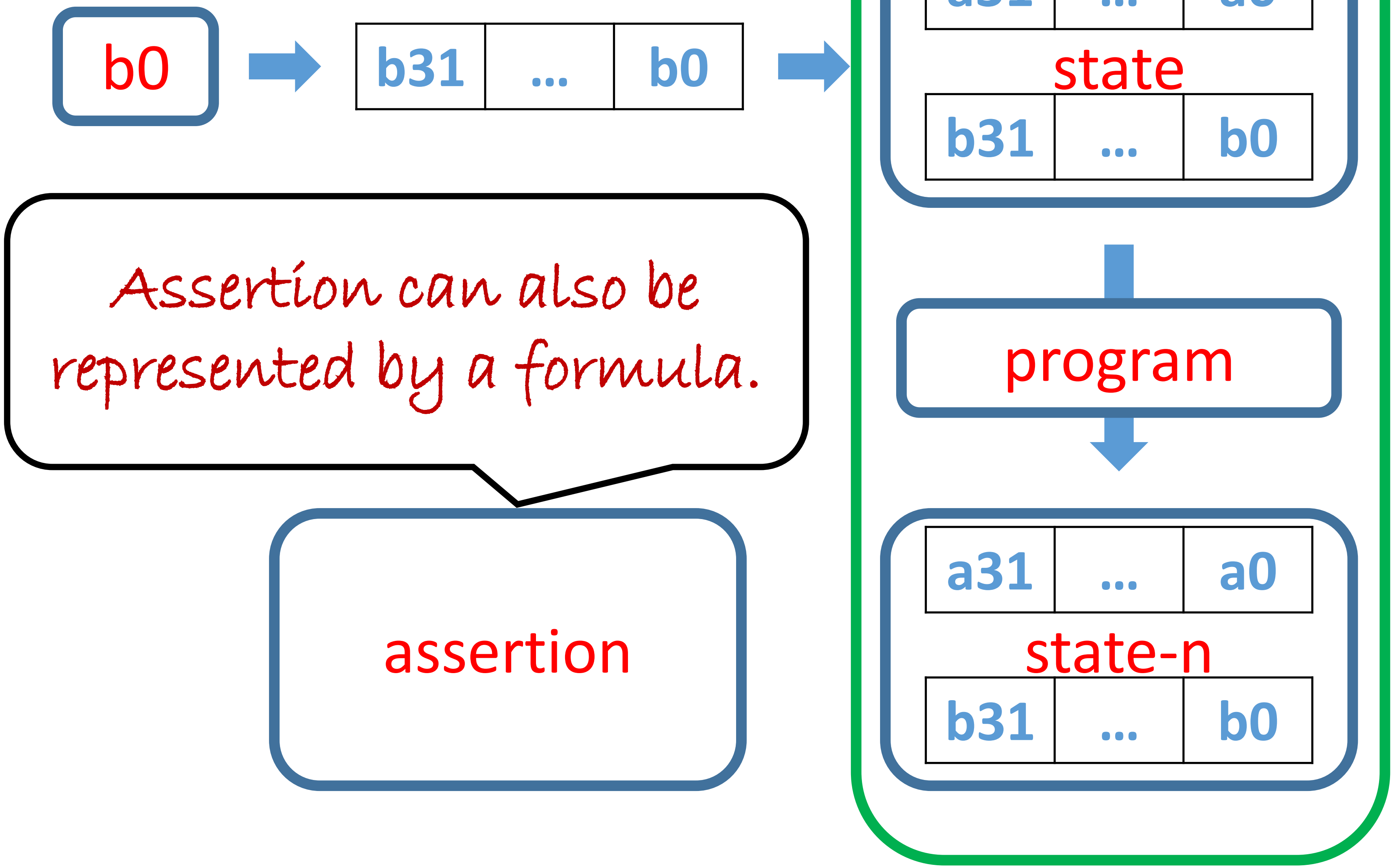
Summary


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

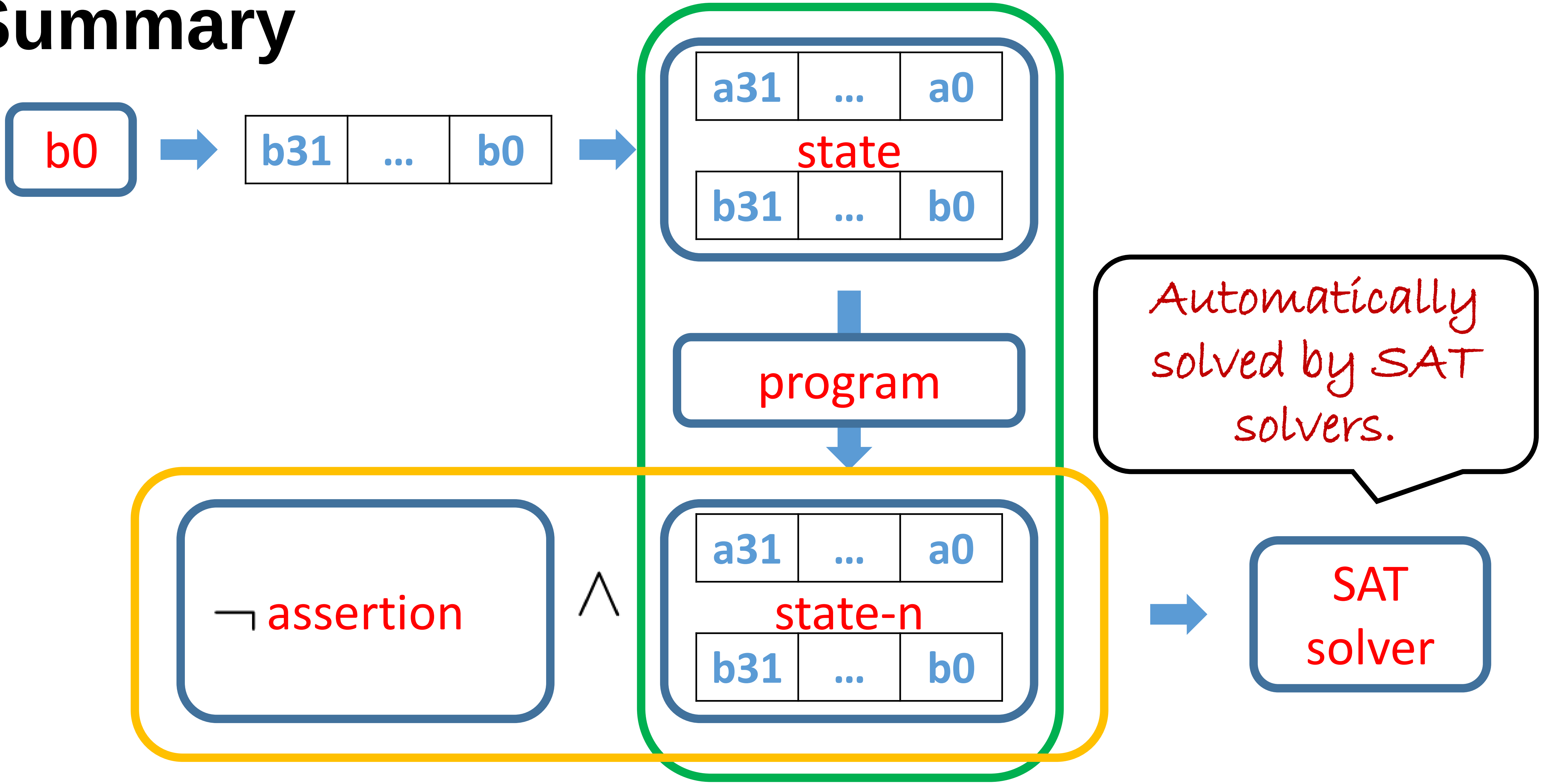
Summary


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

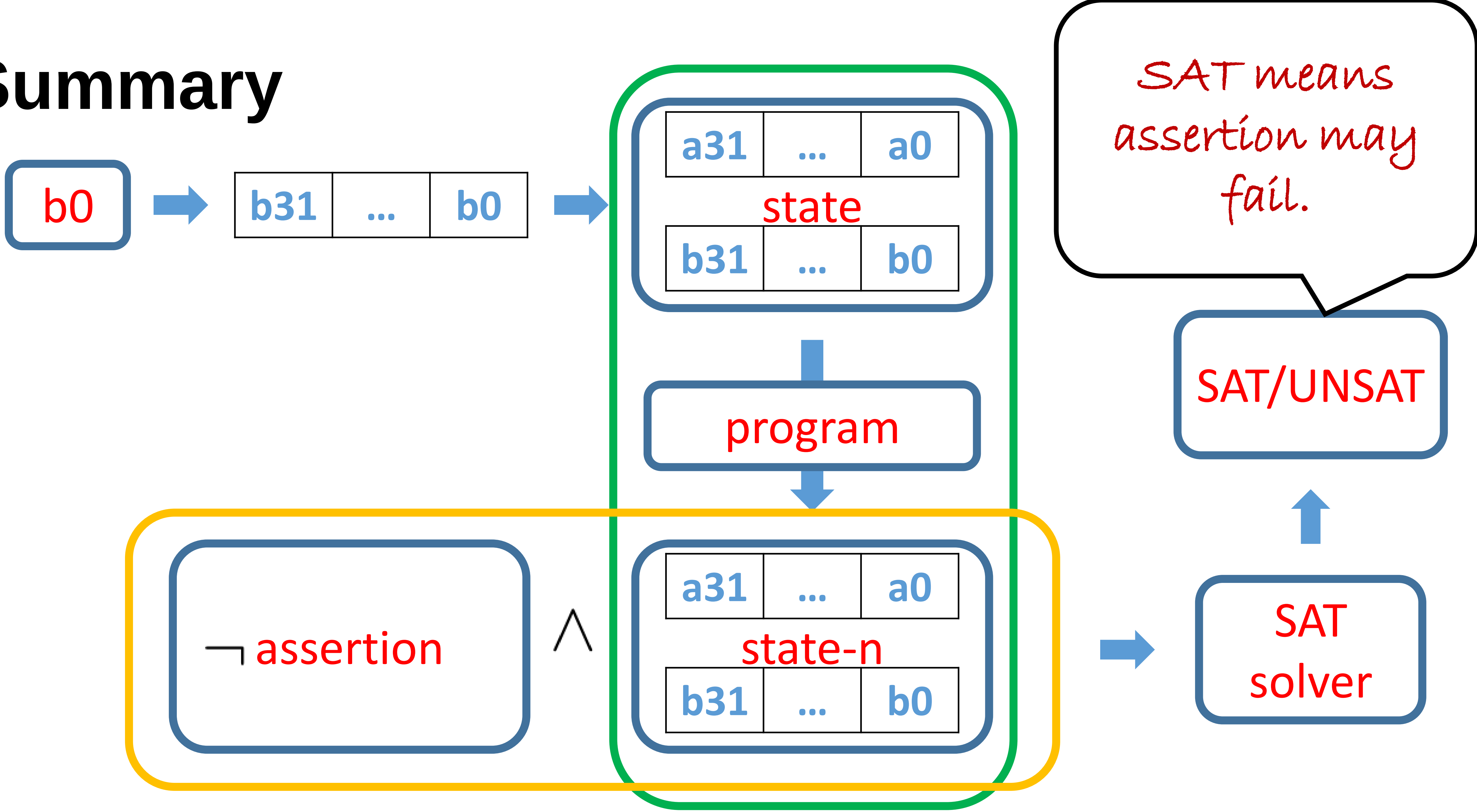

b0
b31 … b0
a31 … a0
state
b31 … b0
Assertion can also be represented by a formula.
assertion
program
a31 … a0
state-n
b31 … b0

Summary



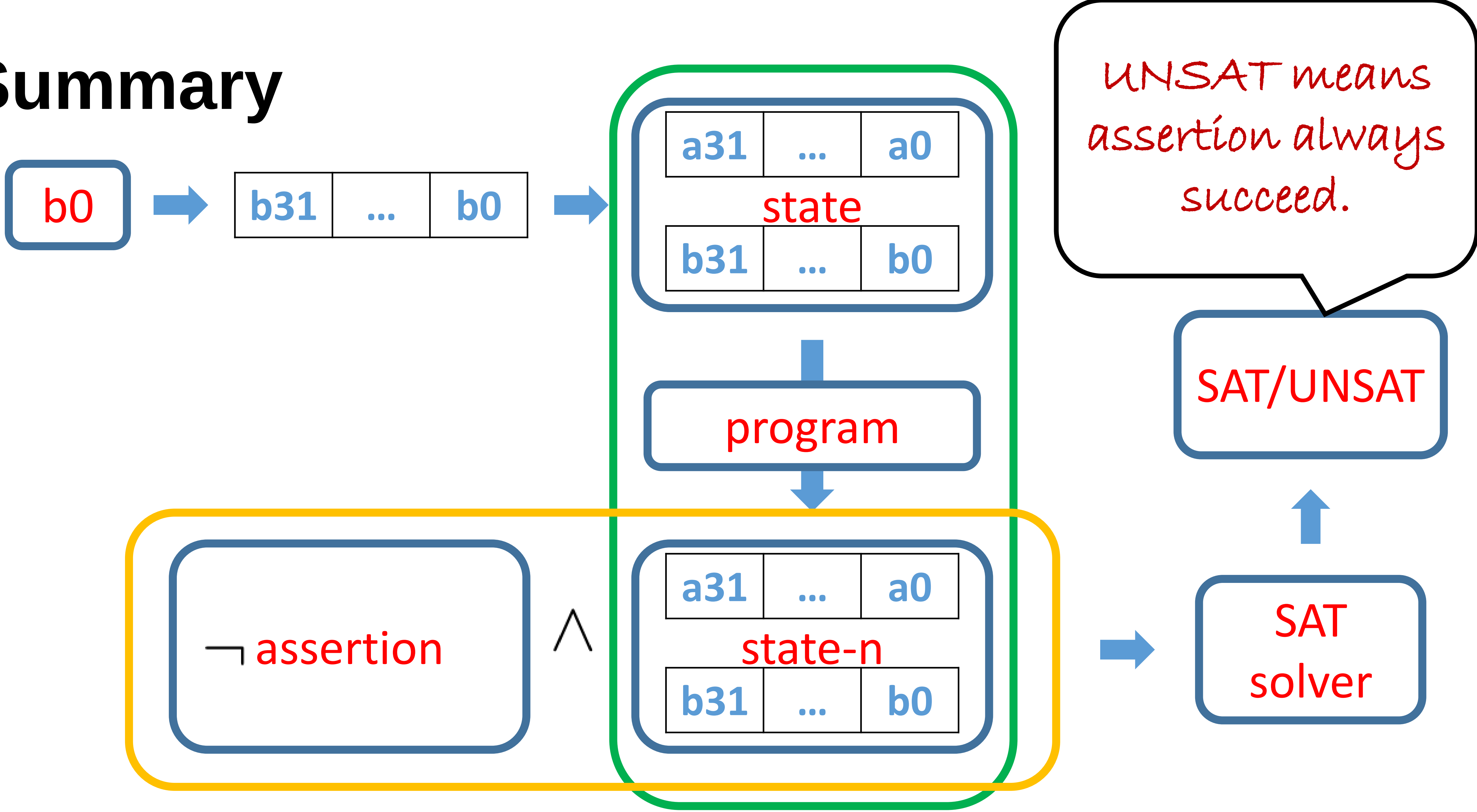
b0
b31
…
b0
a31
…
a0
state
b31
…
b0
program
¬ assertion
∧
a31
…
a0
state-n
b31
…
b0
Automatically solved by SAT solvers.
SAT solver

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.

Thanks & Questions