Propositional Logic III

Zhaoguo Wang

Adapted From:

NYU G22.2390-001, Propositional Logic
MIT 6.825 Techniques in Artificial Intelligence (SMA 5504) Satisfiability and Validity
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 Interactive Proof

Step 1. Axiom system

Step 2. Ask the computer to check the invariants

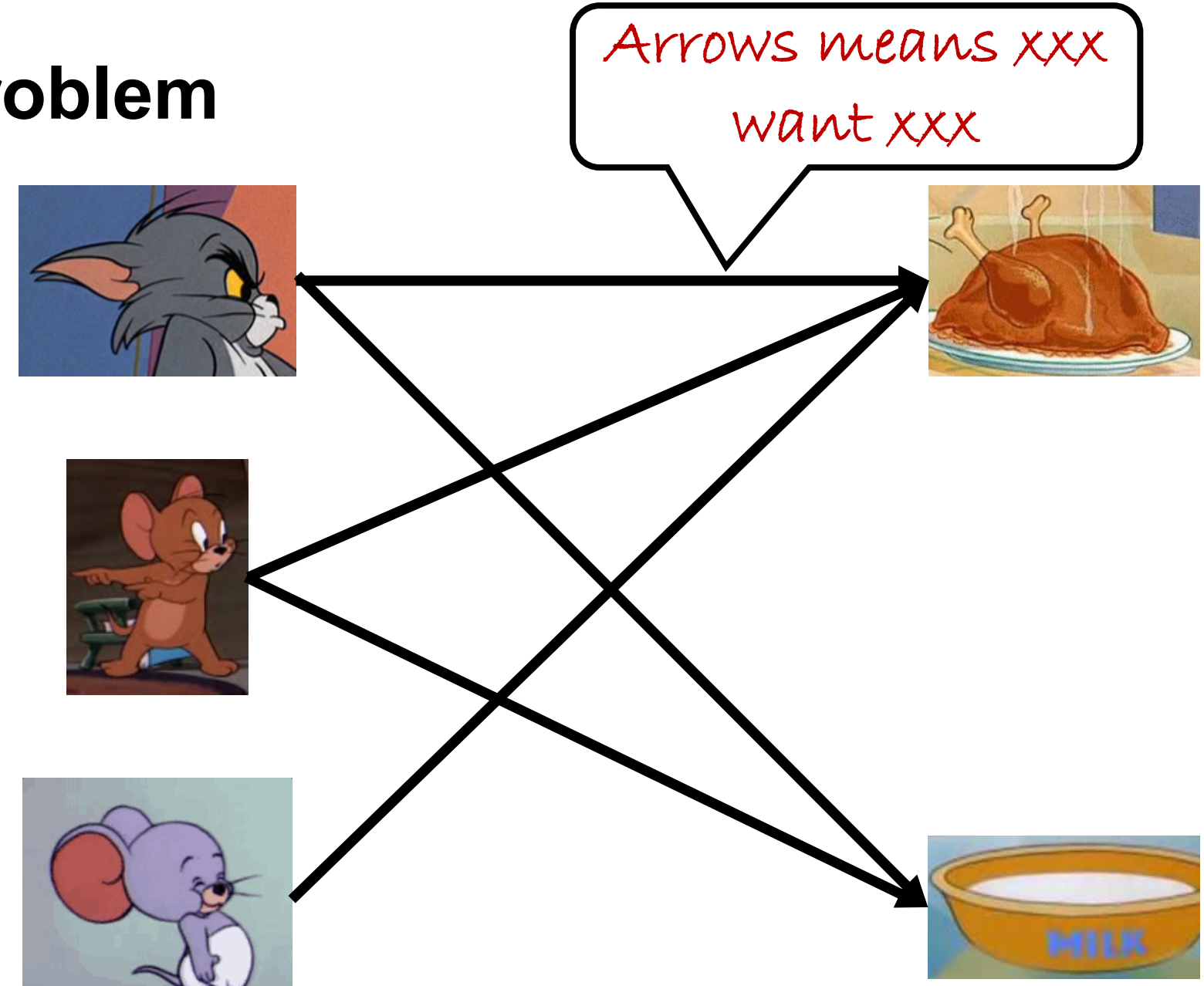
A Quick Recap – Last Lecture

- Completeness of Connectives
- Operator Precedence
- Propositional Equivalence

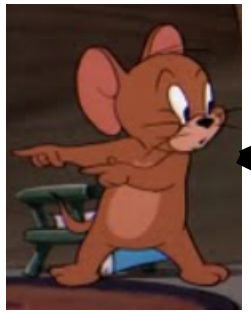
Outline – This Lecture

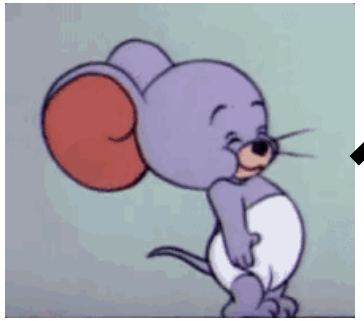
- Tautology
- SAT Problem
- Normal Form
- DPLL

Matching Problem



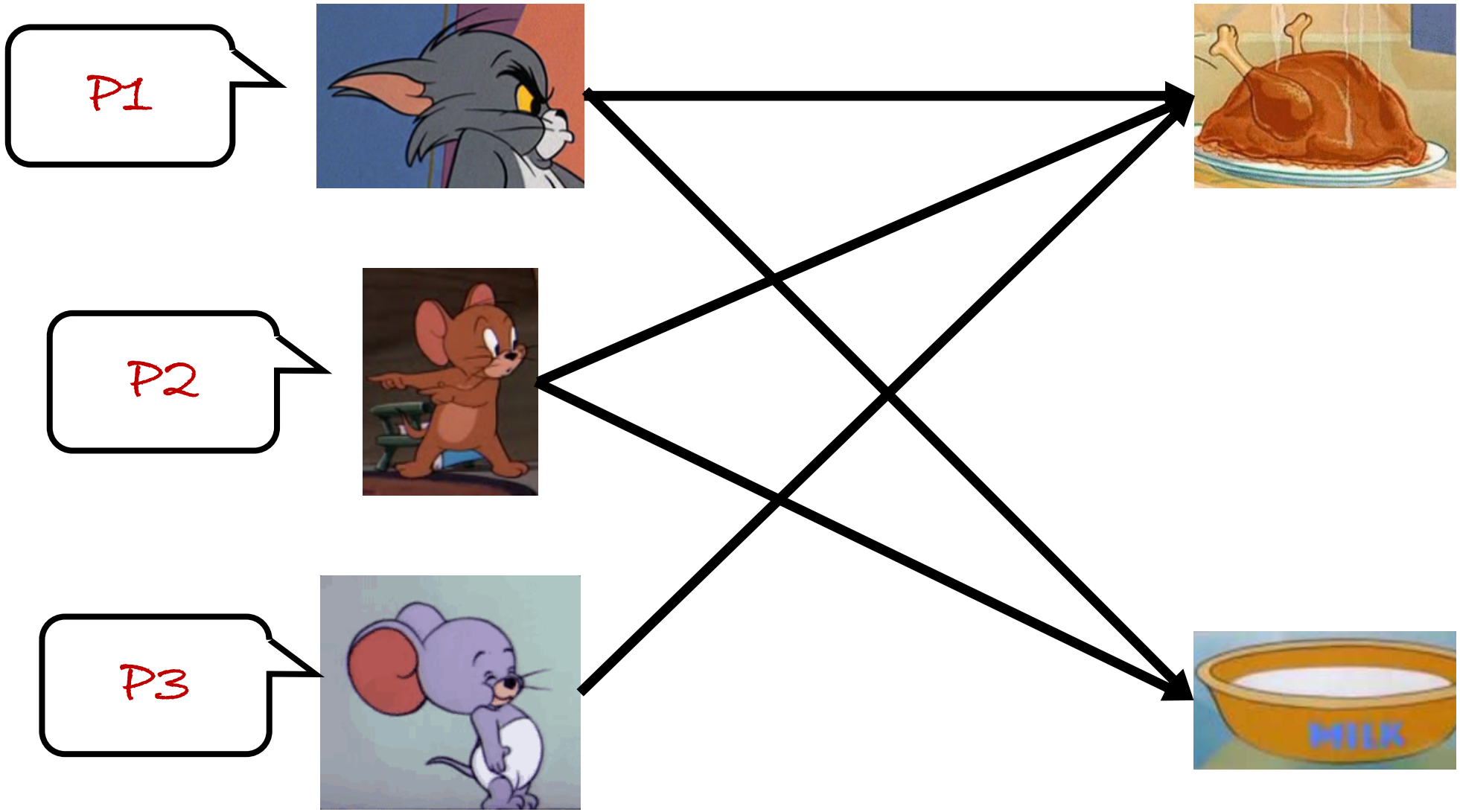
Matching Them With Food And Ensuring No Conflict



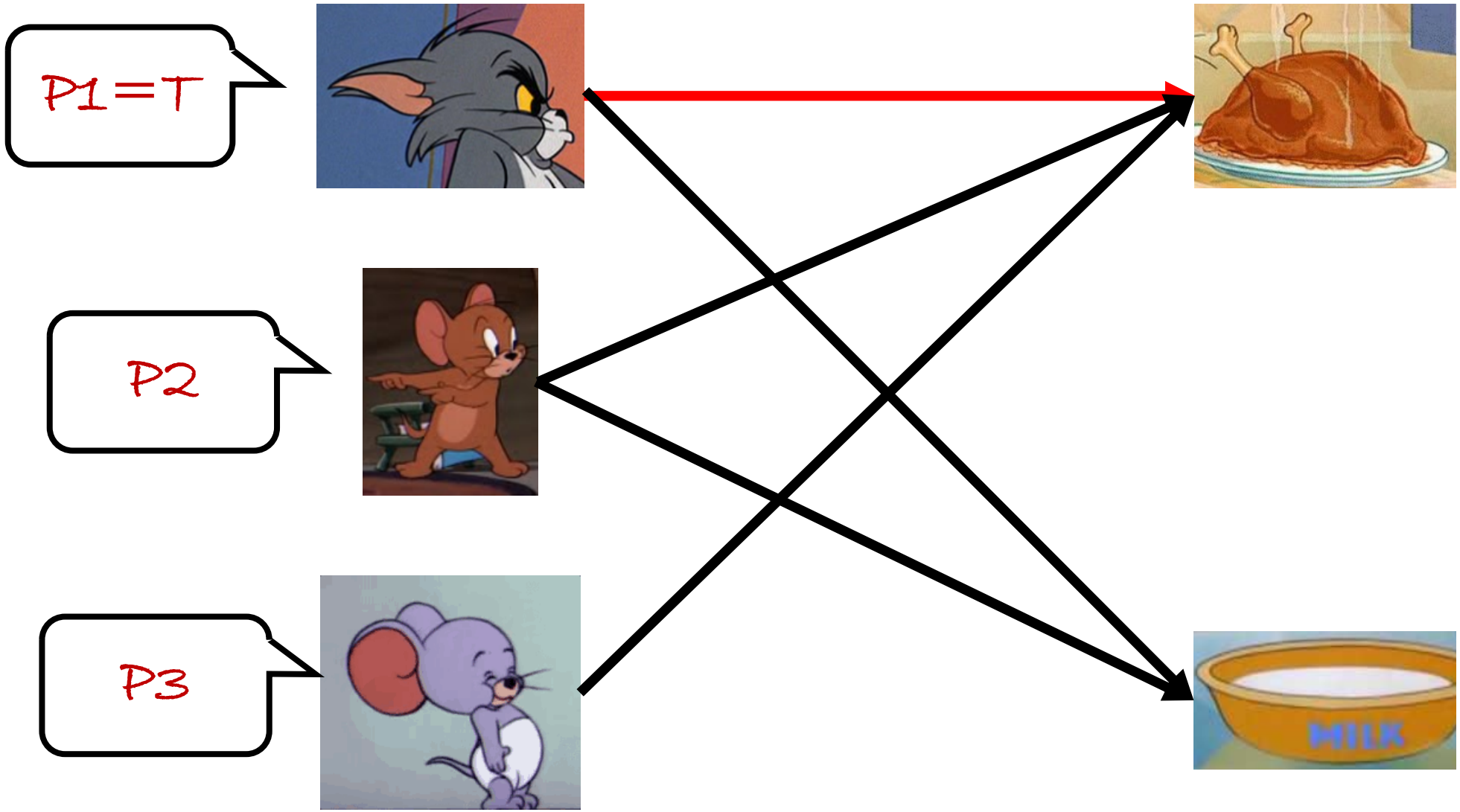


How to prove there is no legal matching with propositional logic?

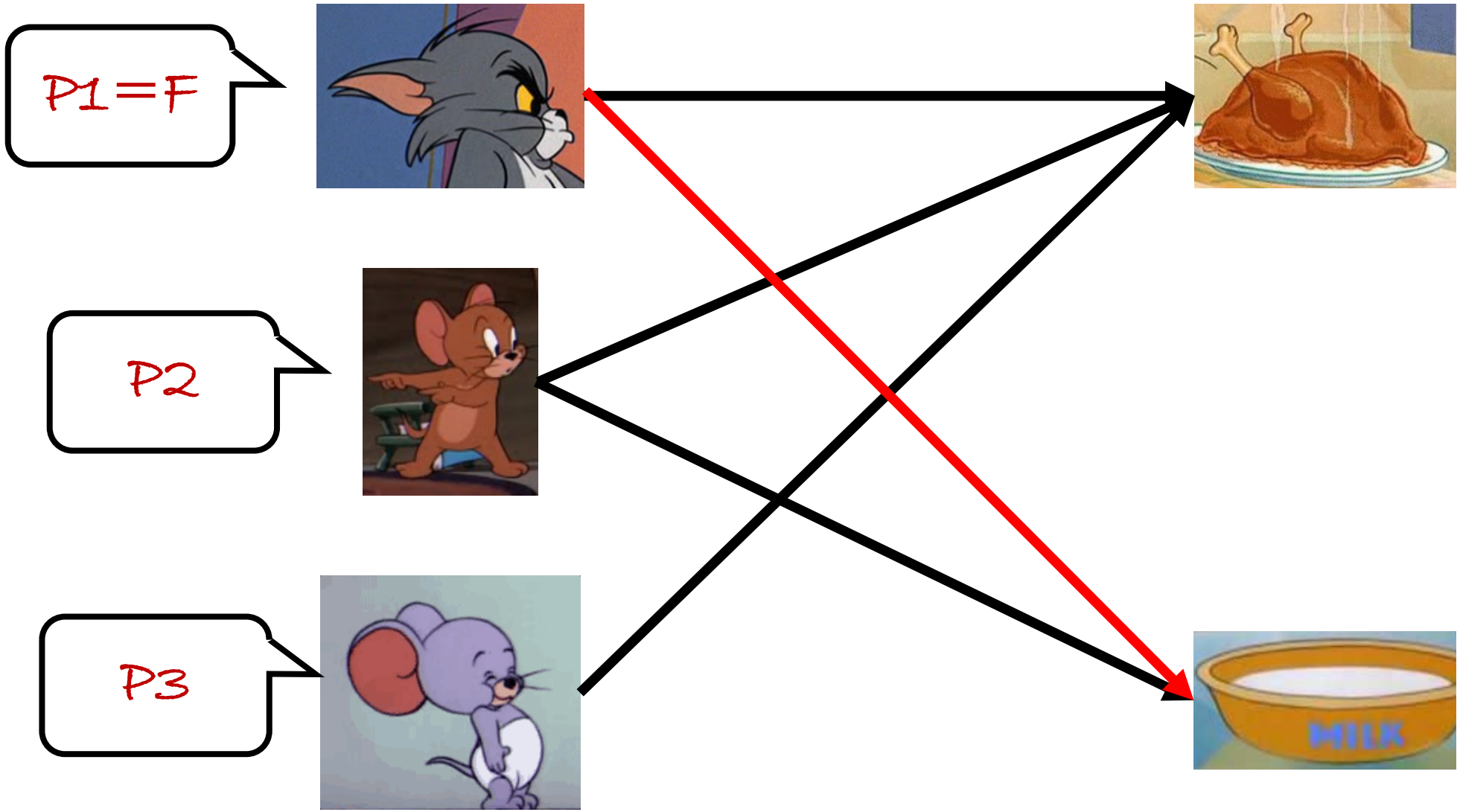
Matching Problem



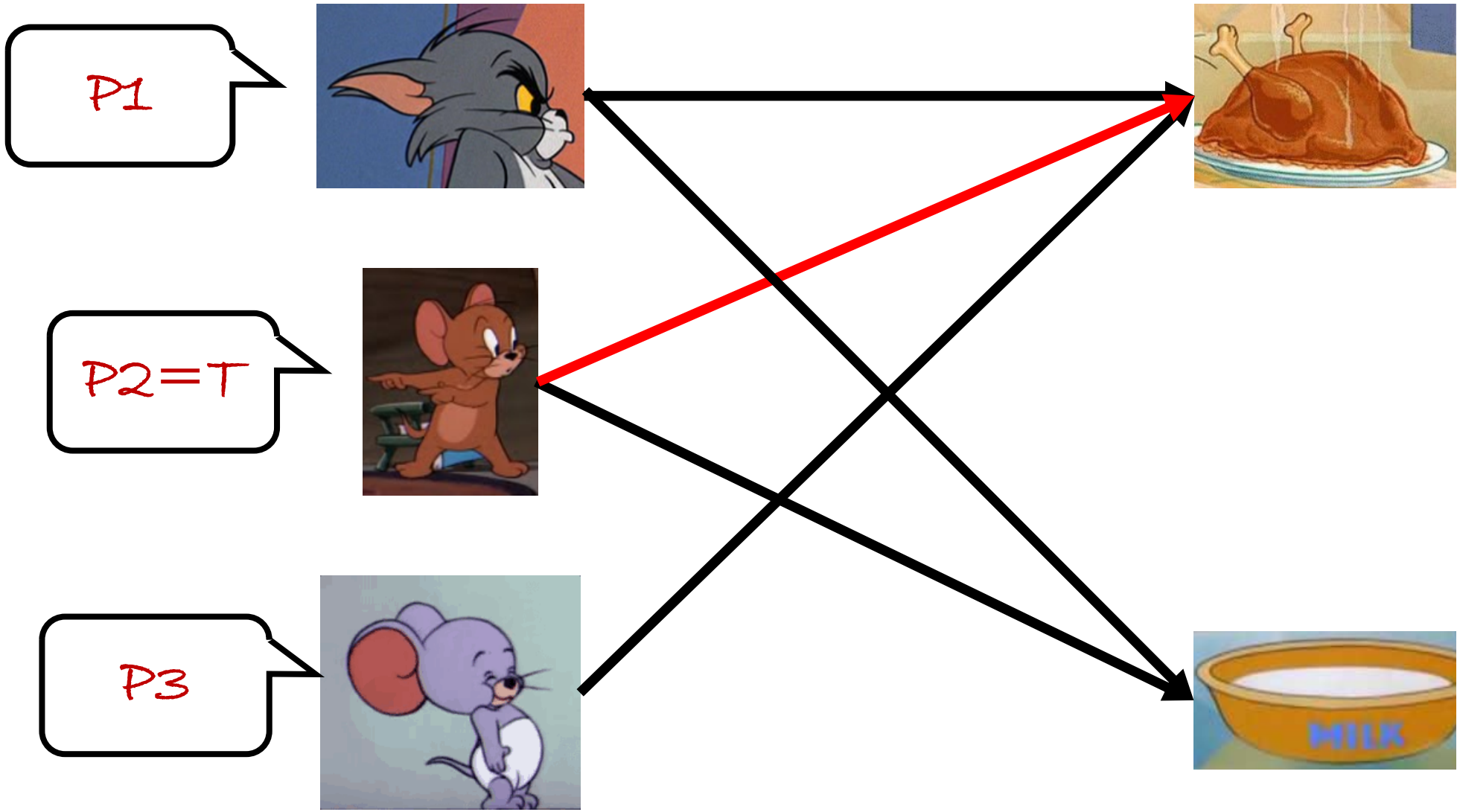
Matching Problem



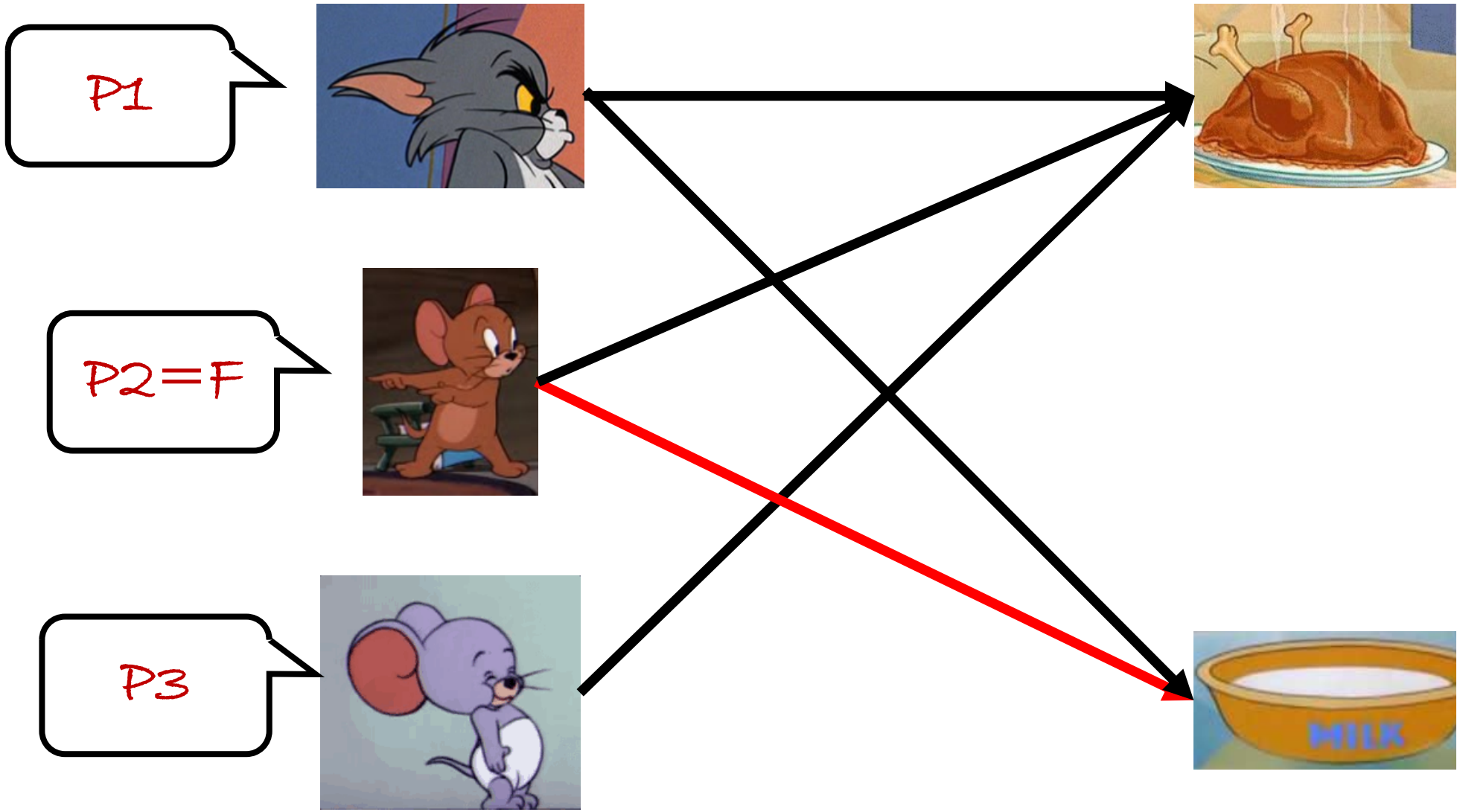
Matching Problem



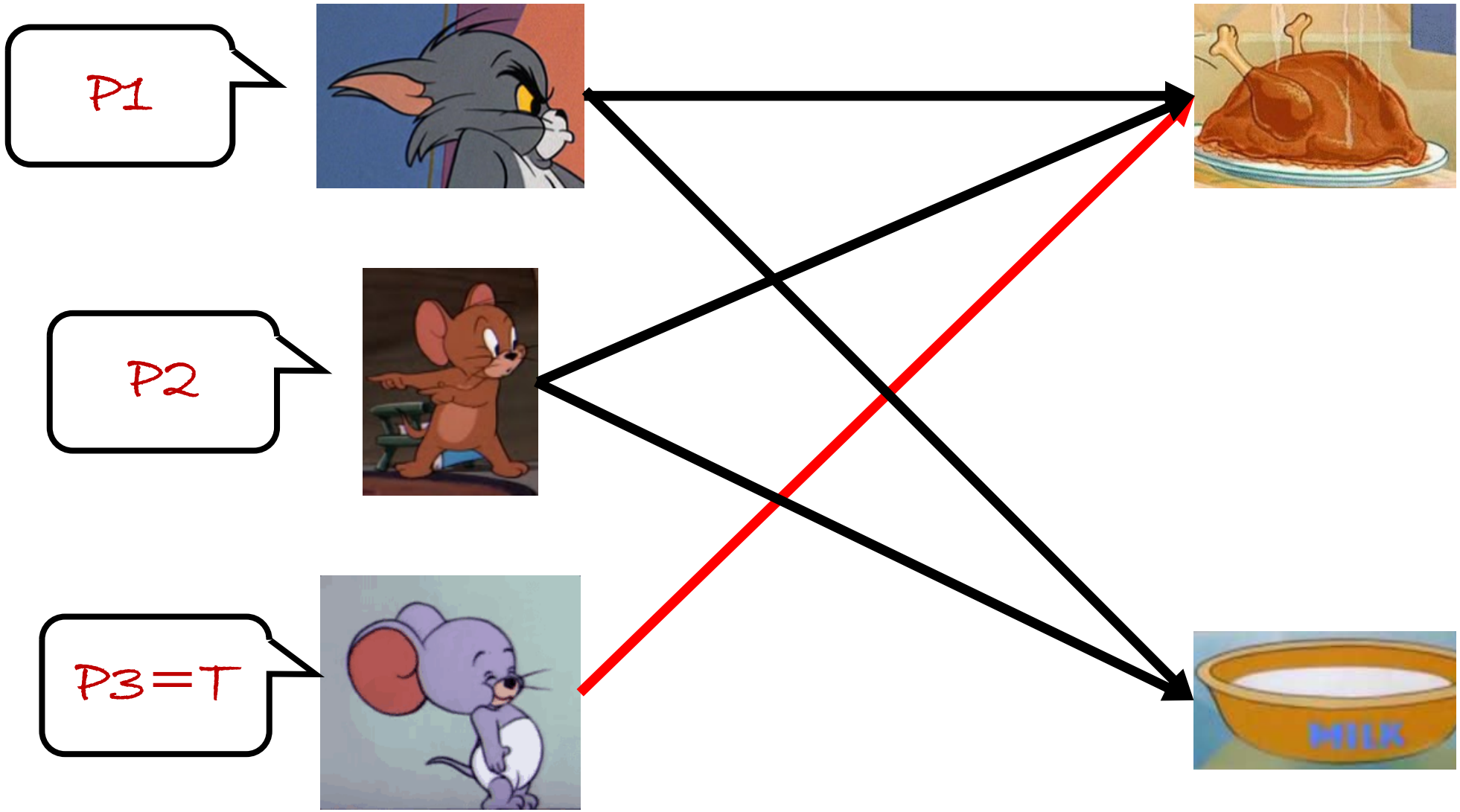
Matching Problem



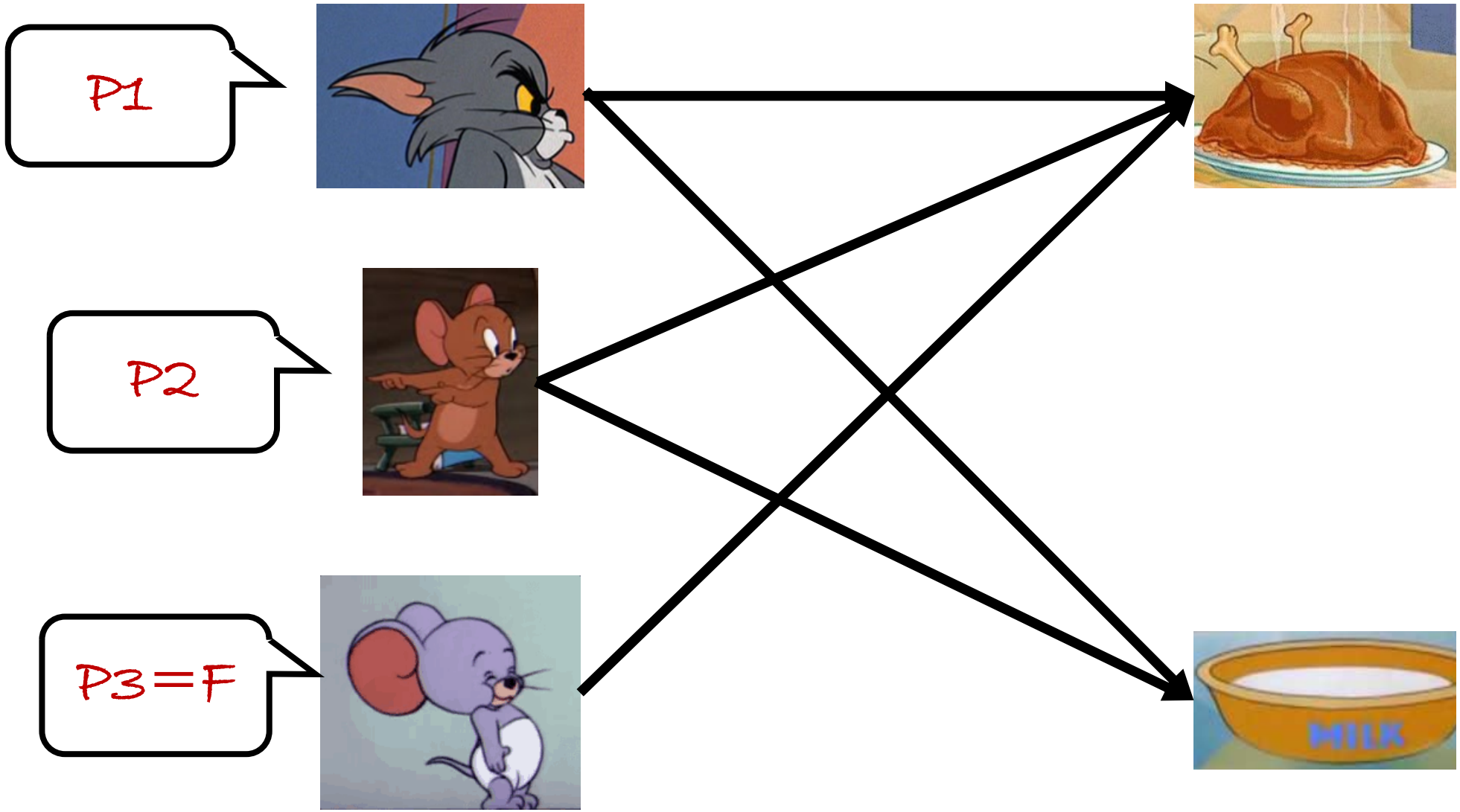
Matching Problem



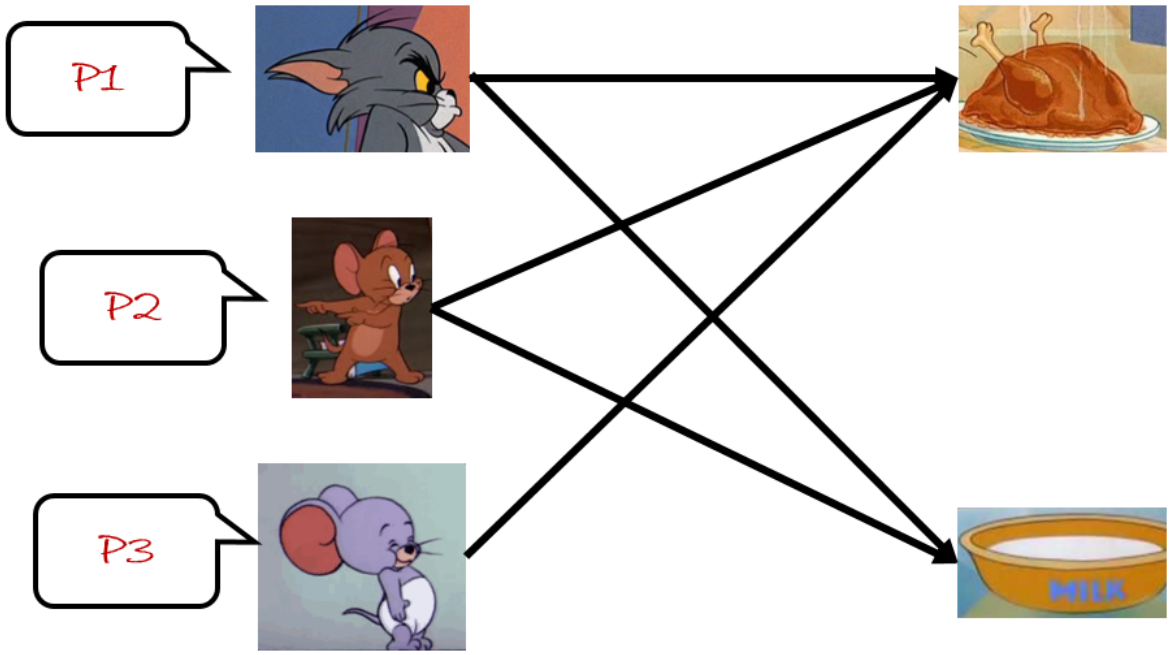
Matching Problem



Matching Problem



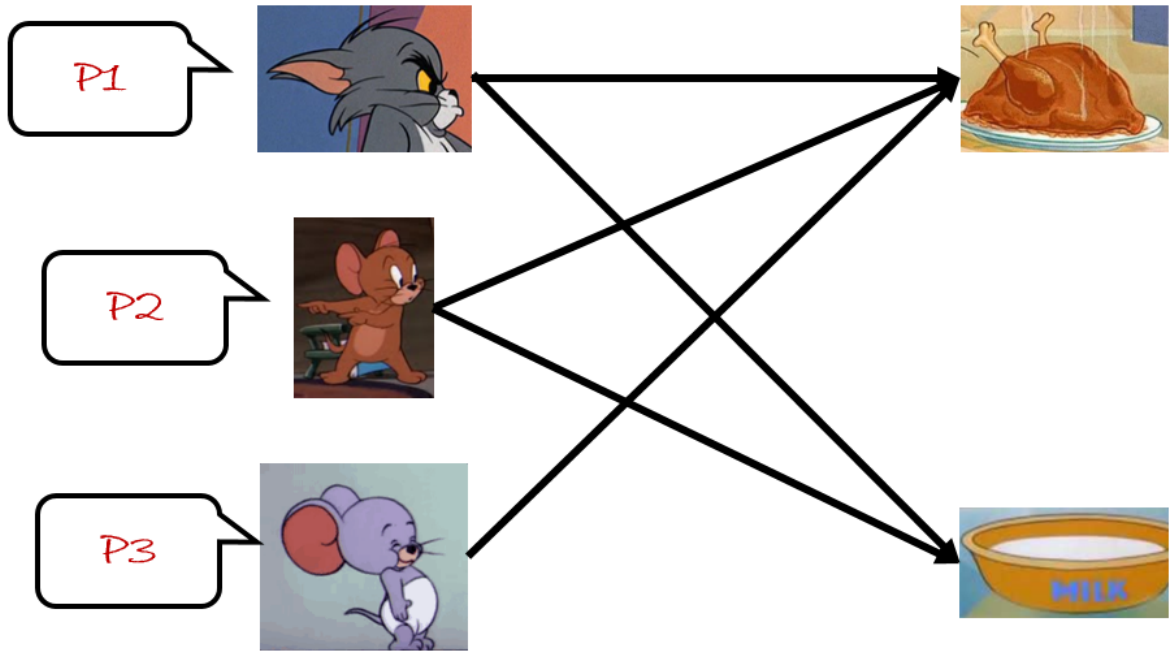
Matching Problem



Nibbles must eat chicken.

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

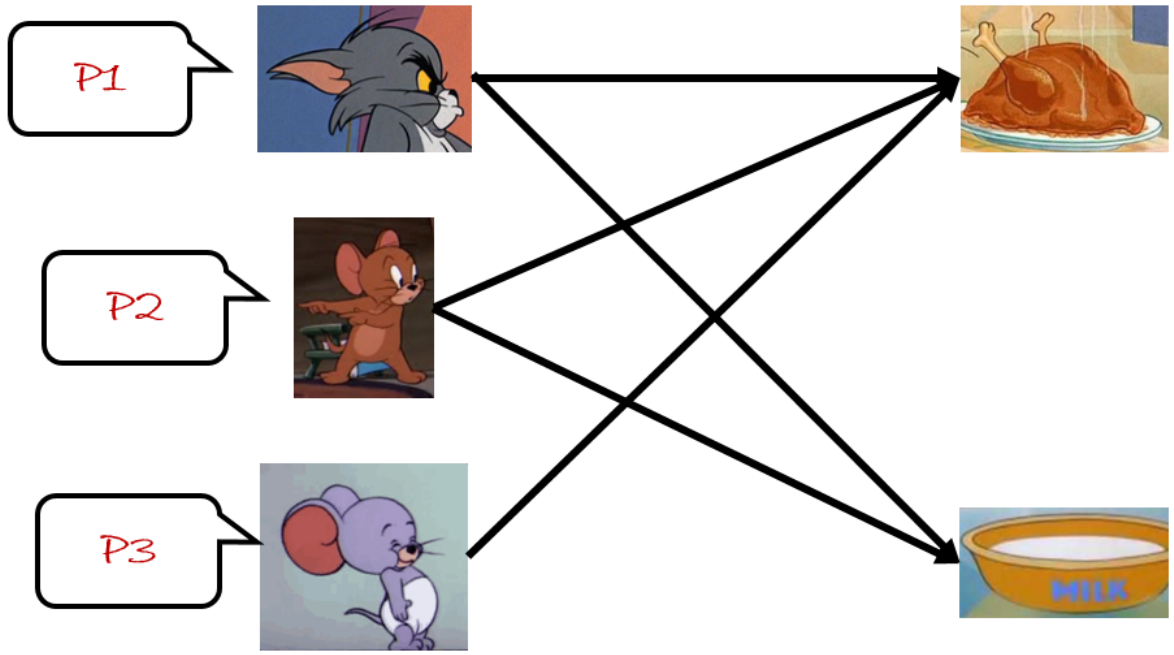
Matching Problem



Chicken cannot be shared.

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

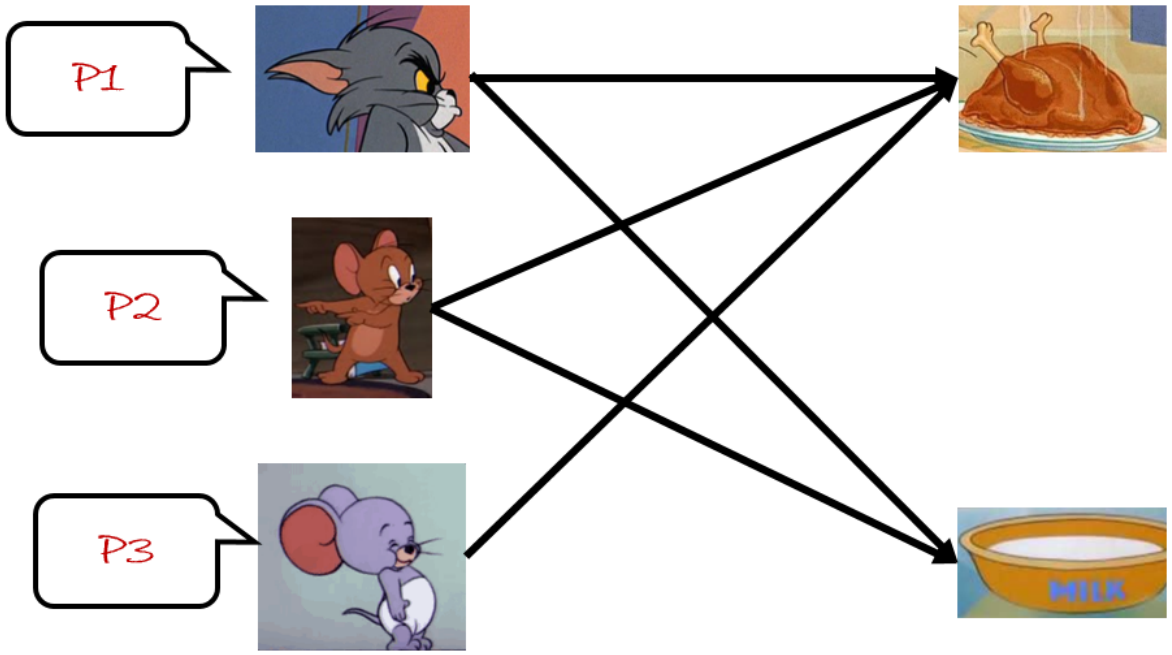
Matching Problem



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

Milk cannot be shared.

Matching Problem



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

Now convert it to F.

Matching Problem

$$
\begin{aligned}
& P3 \wedge \neg(P1 \wedge P2) \wedge \neg(P2 \wedge P3) \wedge \neg(P1 \wedge P3) \wedge \neg(\neg P1 \wedge \neg P2) \\
= {} & P3 \wedge (\neg P1 \vee \neg P2) \wedge (\neg P2 \vee \neg P3) \wedge (\neg P1 \vee \neg P3) \wedge (P1 \vee P2) \\
= {} & P3 \wedge (\neg P2 \vee \neg P3) \wedge (\neg P1 \vee \neg P2) \wedge (\neg P1 \vee \neg P3) \wedge (P1 \vee P2) \\
= {} & P3 \wedge \neg P2 \wedge (\neg P1 \vee \neg P2) \wedge (\neg P1 \vee \neg P3) \wedge (P1 \vee P2) \\
= {} & P3 \wedge \neg P2 \wedge (P1 \vee P2) \wedge (\neg P1 \vee \neg P2) \wedge (\neg P1 \vee \neg P3) \\
= {} & P3 \wedge \neg P2 \wedge P1 \wedge (\neg P1 \vee \neg P2) \wedge (\neg P1 \vee \neg P3) \\
= {} & P3 \wedge \neg P2 \wedge P1 \wedge (\neg P1 \vee \neg P3) \wedge (\neg P1 \vee \neg P2) \\
= {} & P3 \wedge \neg P2 \wedge P1 \wedge \neg P3 \wedge (\neg P1 \vee \neg P2) \\
= {} & P3 \wedge \neg P3 \wedge \neg P2 \wedge P1 \wedge (\neg P1 \vee \neg P2) \\
= {} & F
\end{aligned}
$$

It is a special proposition which is always false

Tautology（重言式/永真式）

DEFINITION

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

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

The previous problem is to prove the formula is a tautology.

Contradiction（矛盾式/永假式）

DEFINITION

Contradiction is a proposition which is always false under any interpretation.

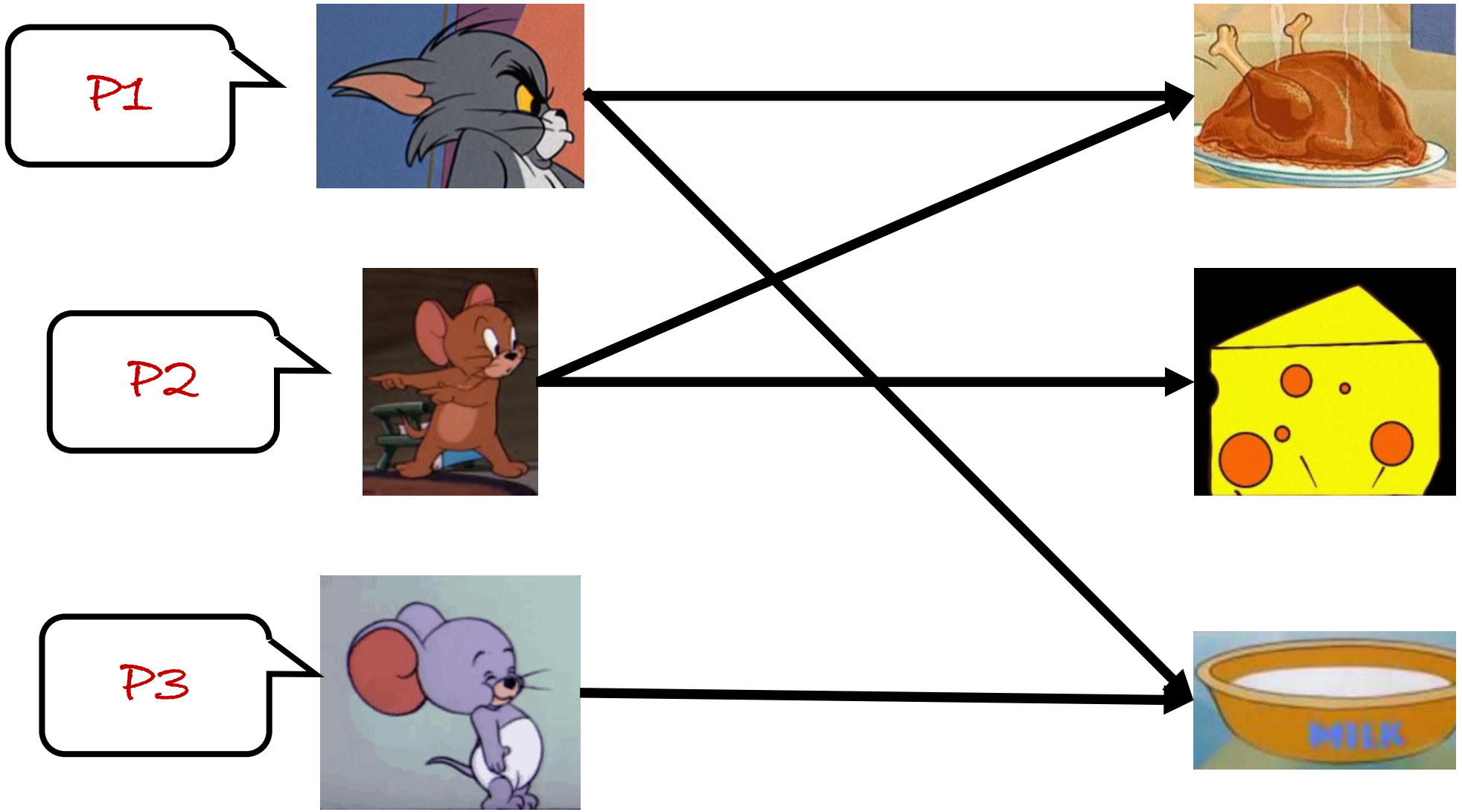
$$P3 \wedge \neg(P1 \wedge P2) \wedge \neg(P2 \wedge P3) \wedge \neg(P1 \wedge P3) \wedge \neg(\neg P1 \wedge \neg P2)$$

The previous problem is to prove the formula is a contradiction.

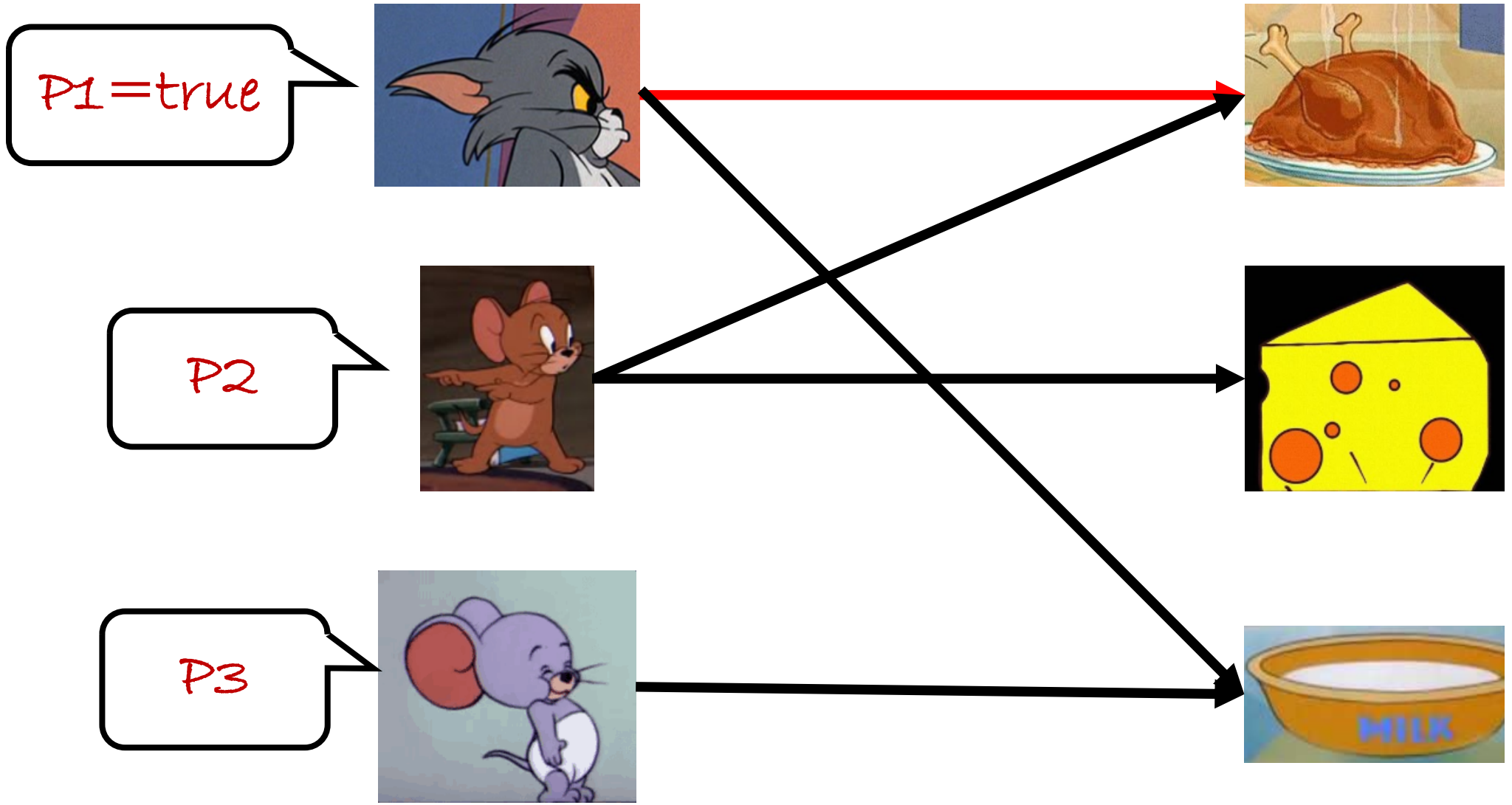
Example

$P \vee \neg P$	tautology
$P \wedge \neg P$	contradiction
$P \wedge F$	contradiction
$P \vee T$	tautology
$\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$	tautology

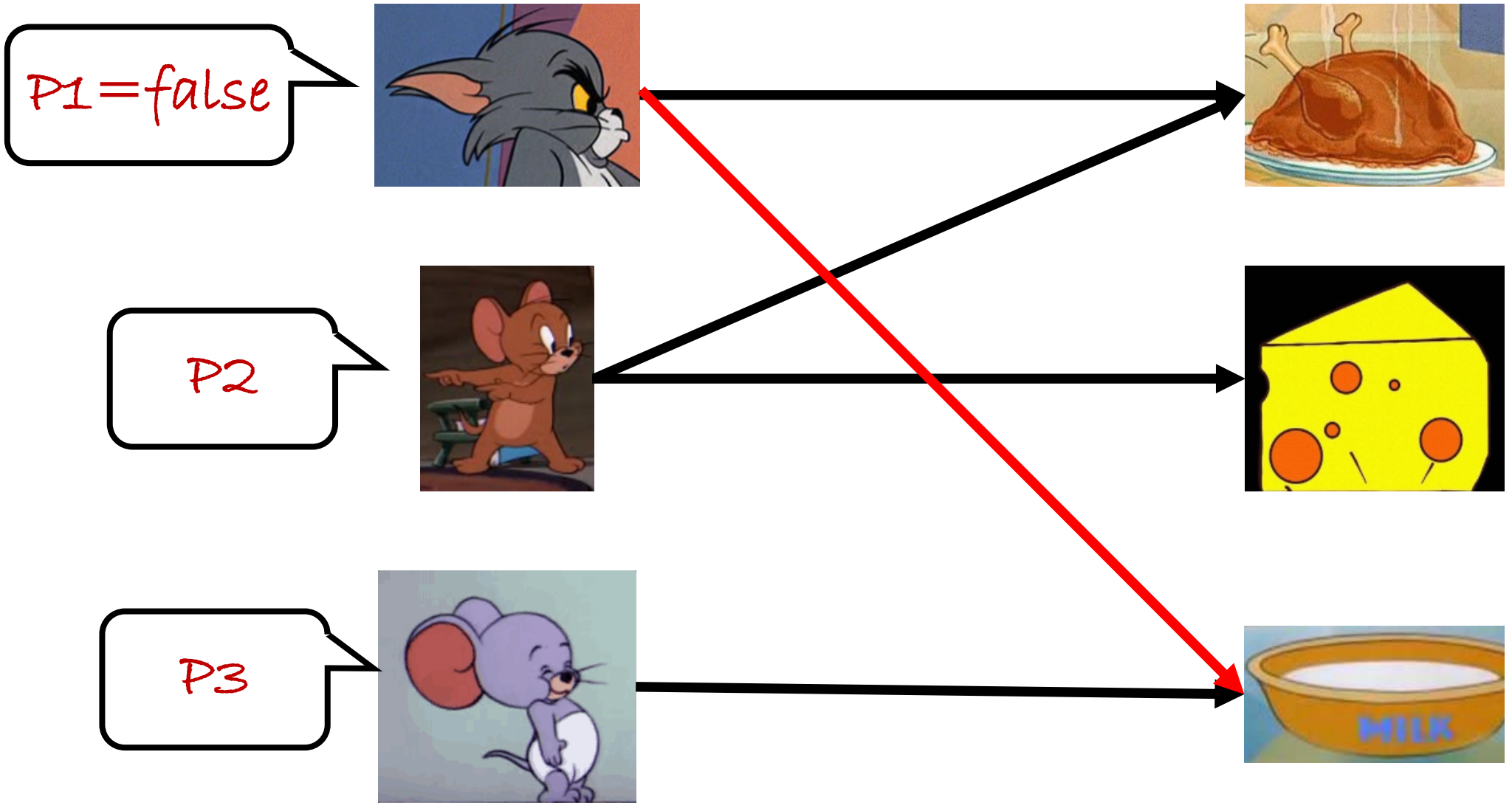
Matching Problem II



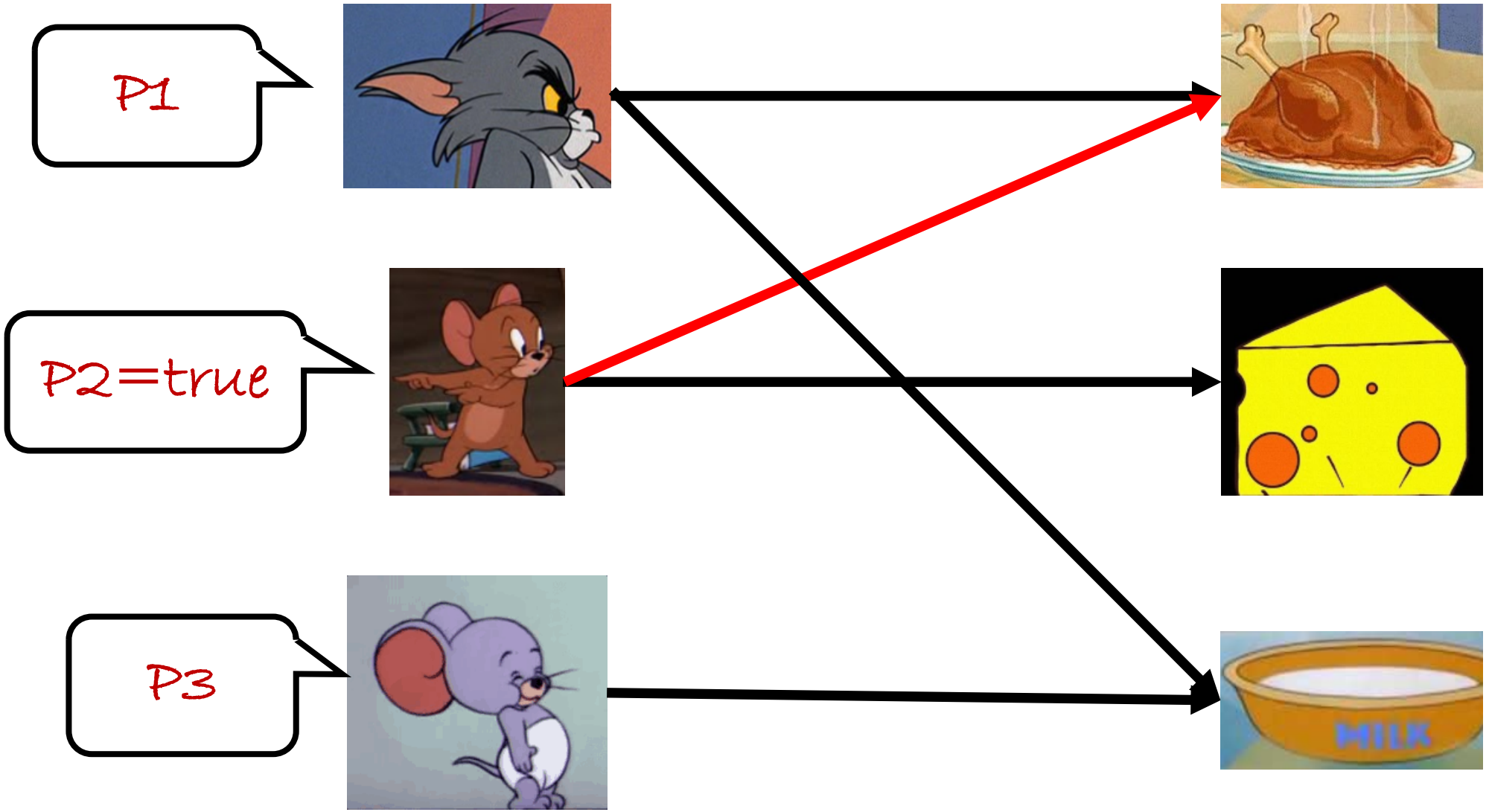
Matching Problem II



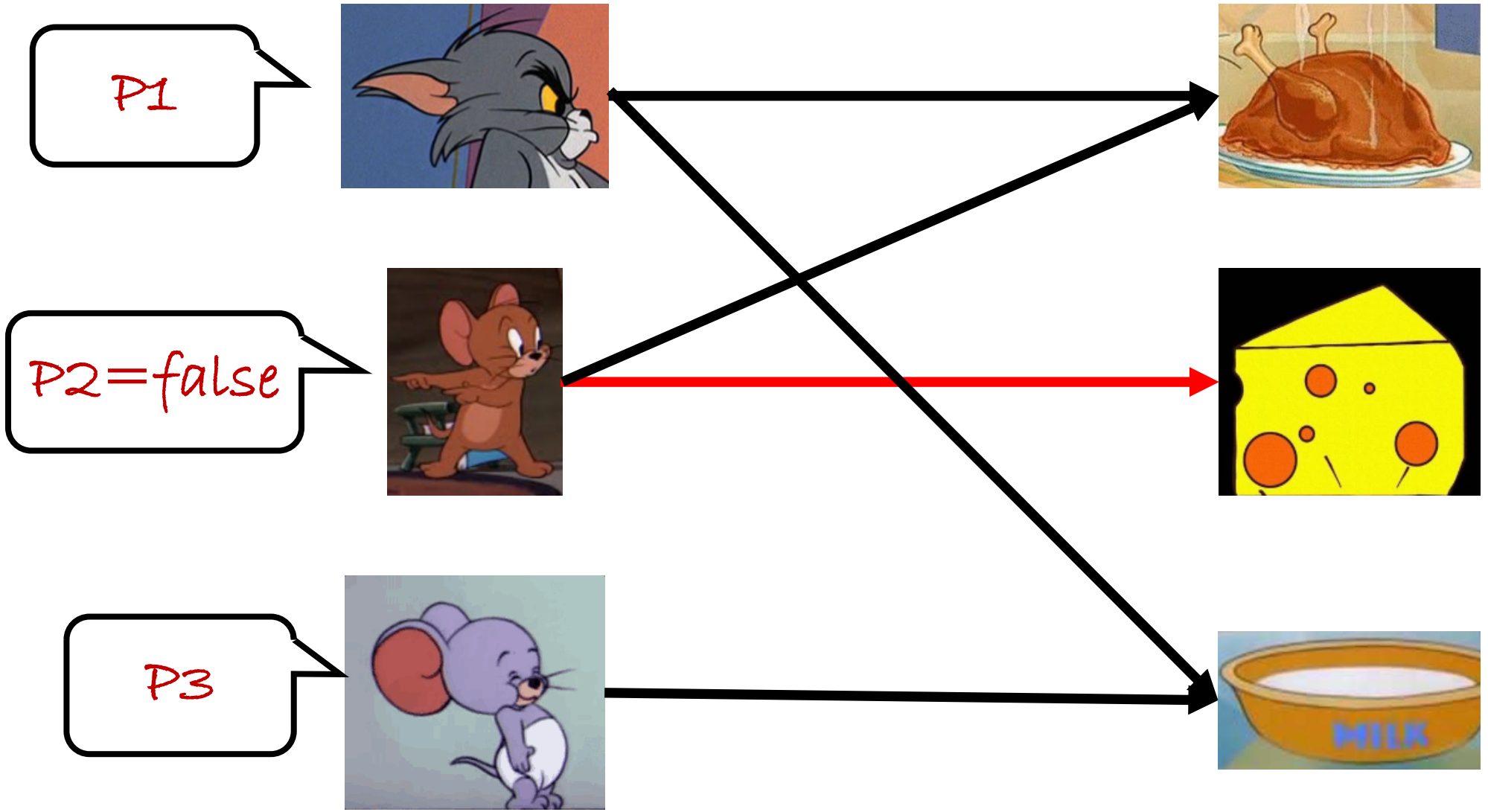
Matching Problem II



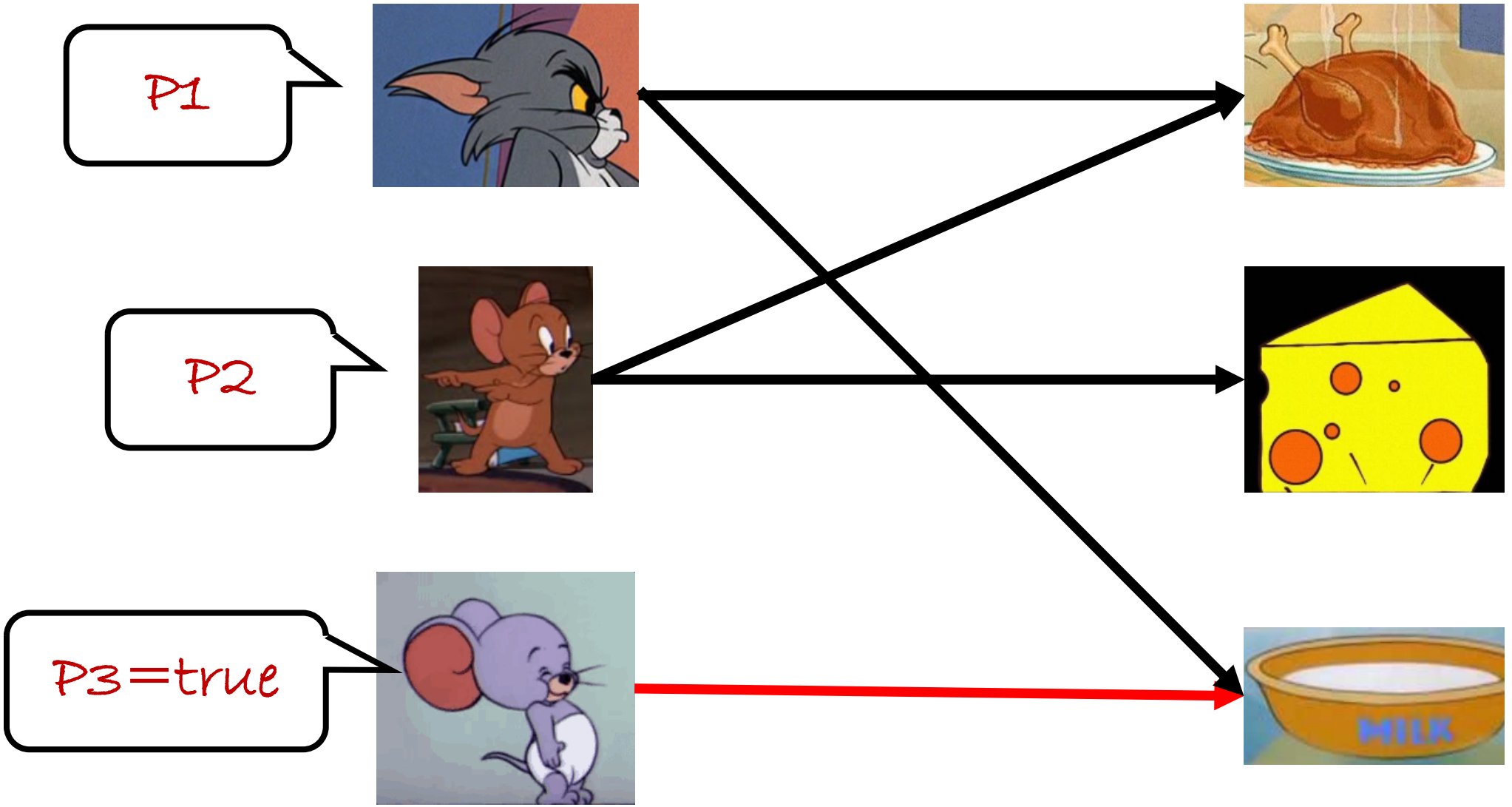
Matching Problem II



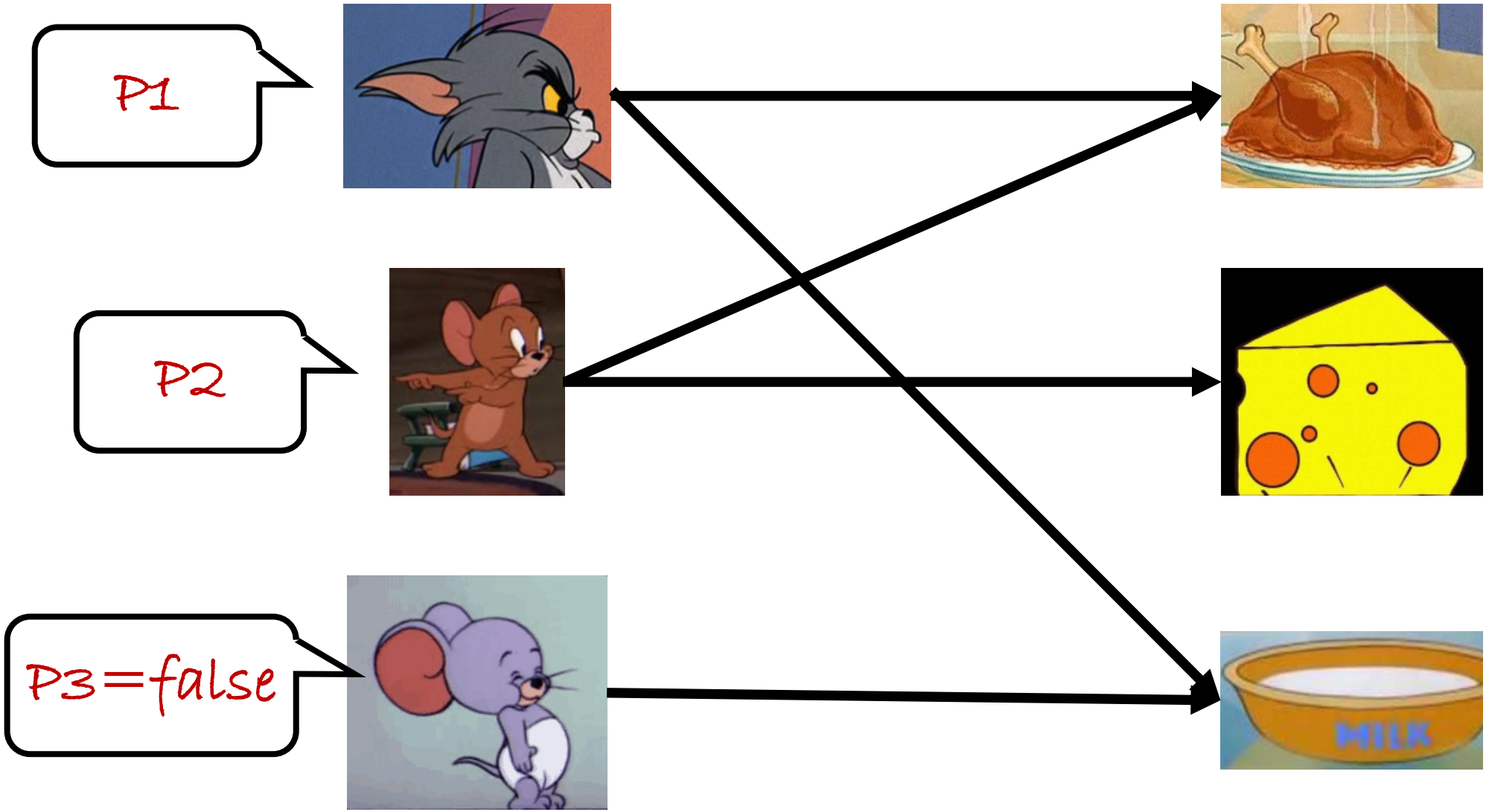
Matching Problem II



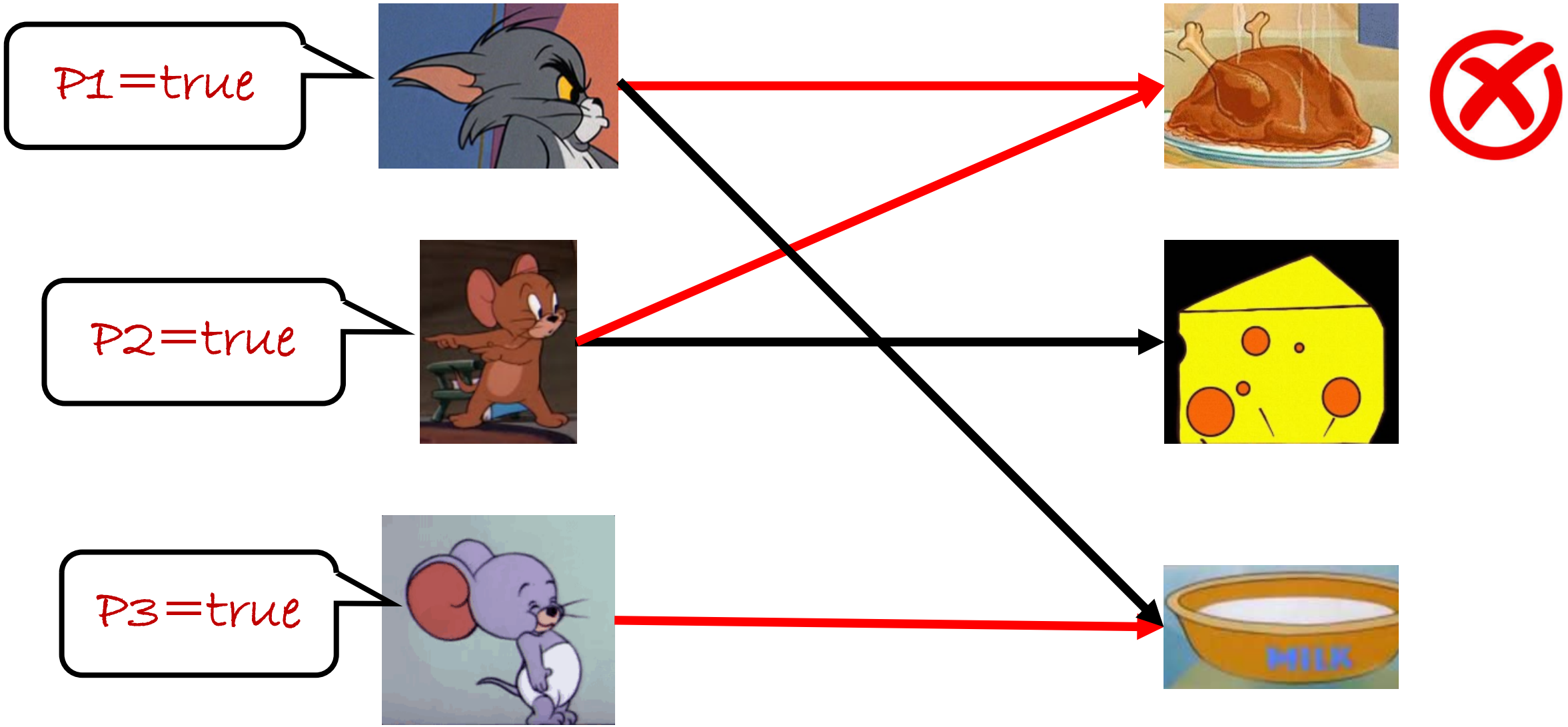
Matching Problem II



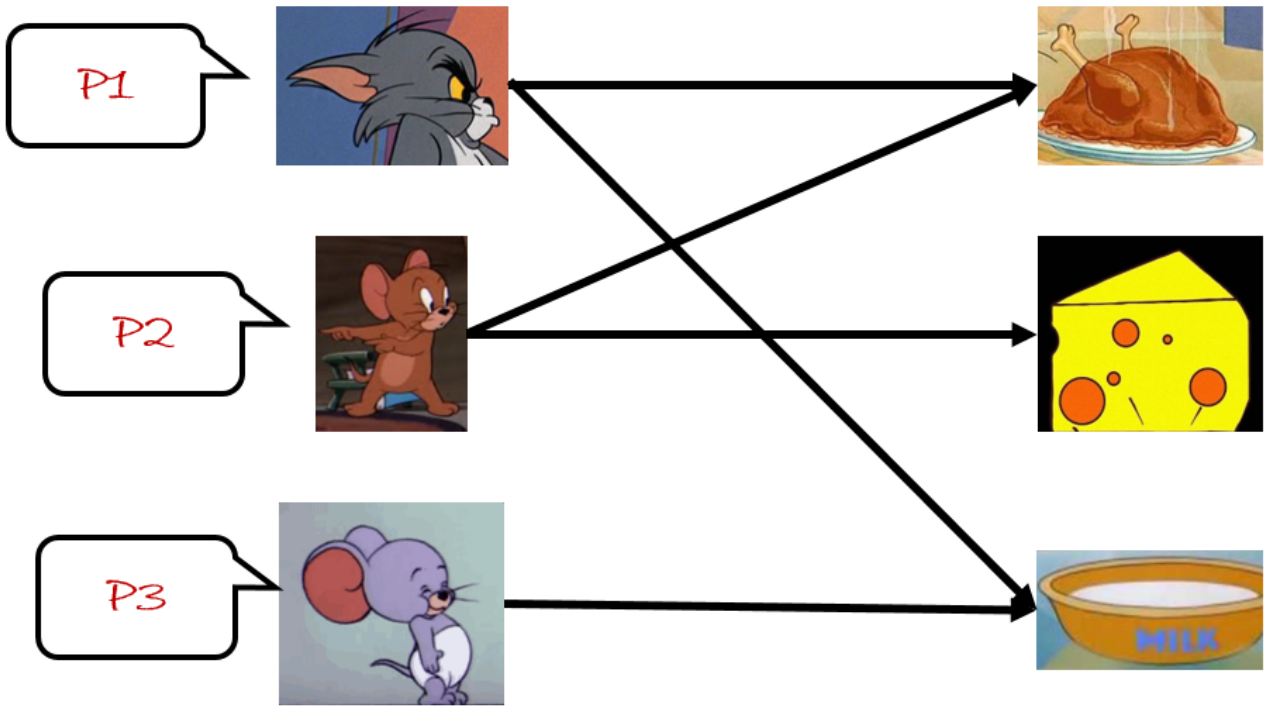
Matching Problem II



Matching Problem II

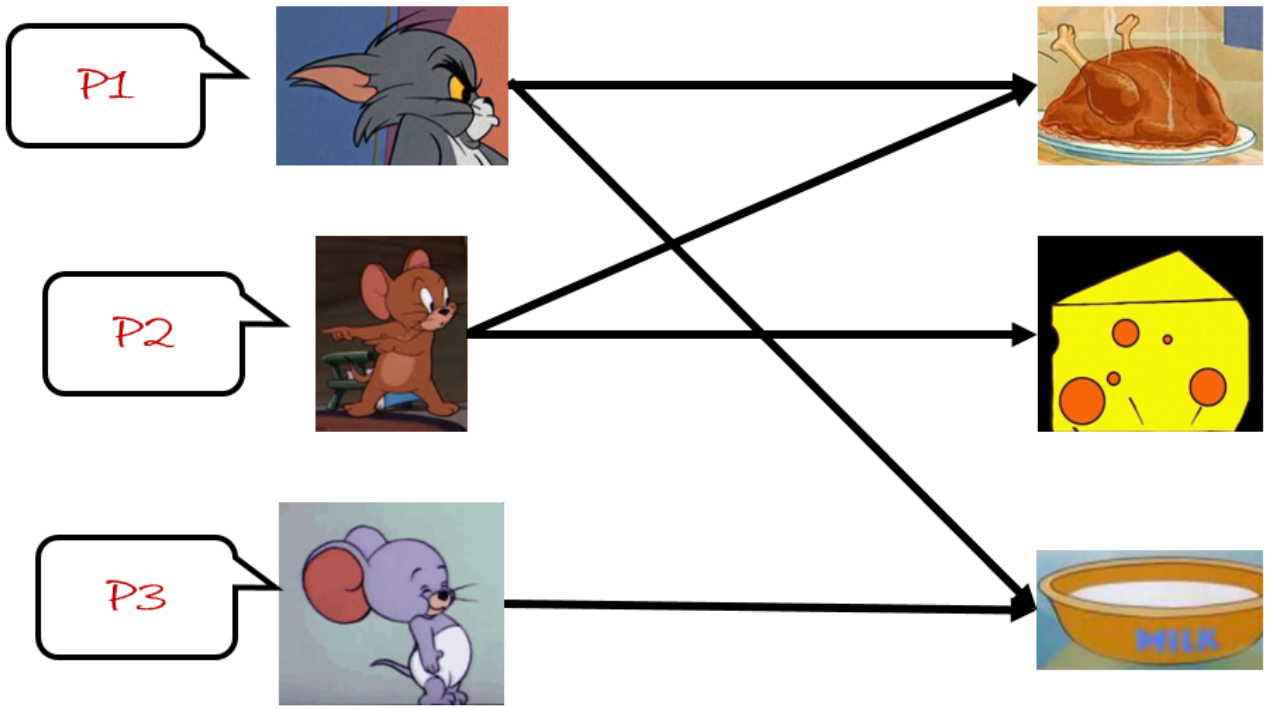


Matching Problem II



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

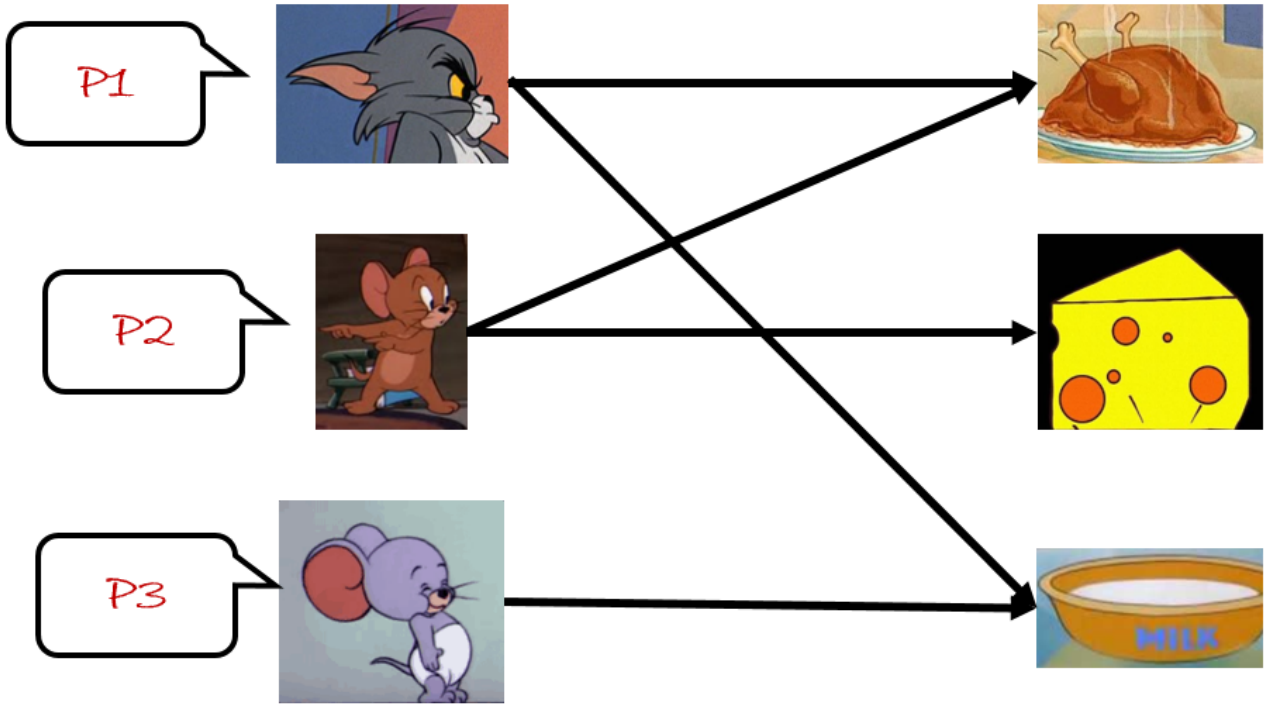
Matching Problem II



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

Nibbles must drink milk.

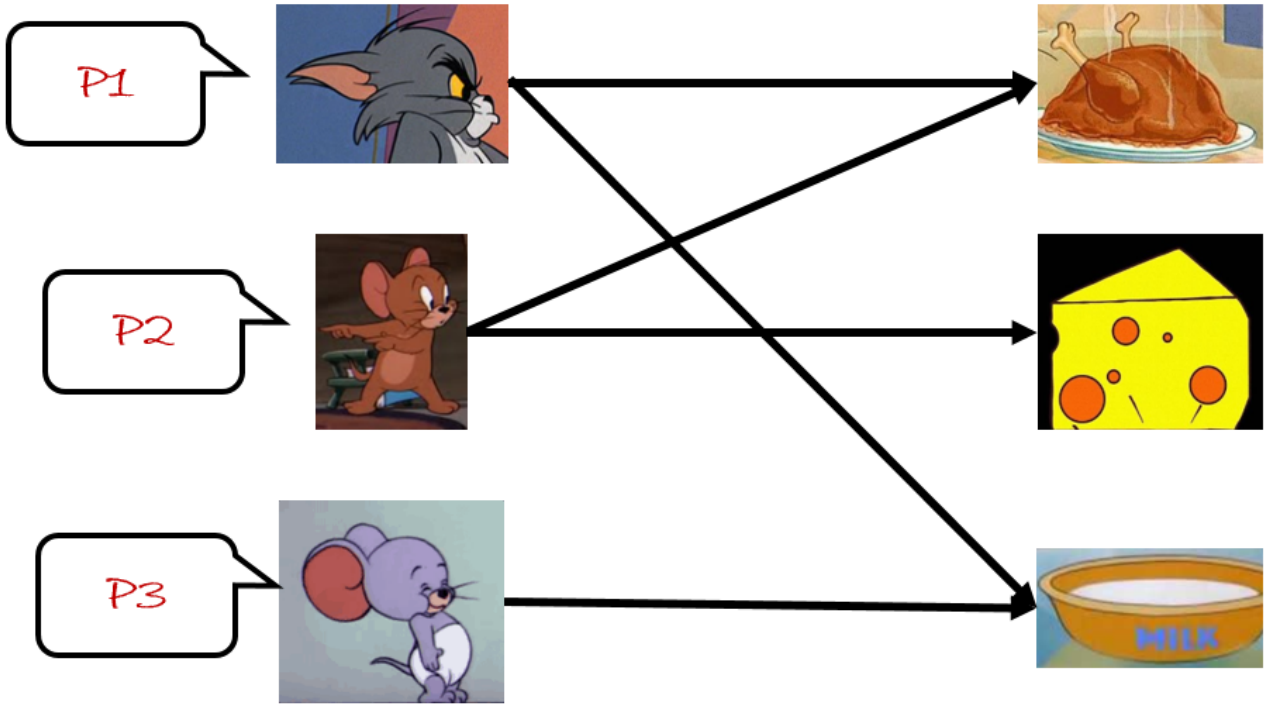
Matching Problem II



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

Tom and Jerry cannot eat chicken at the same time.

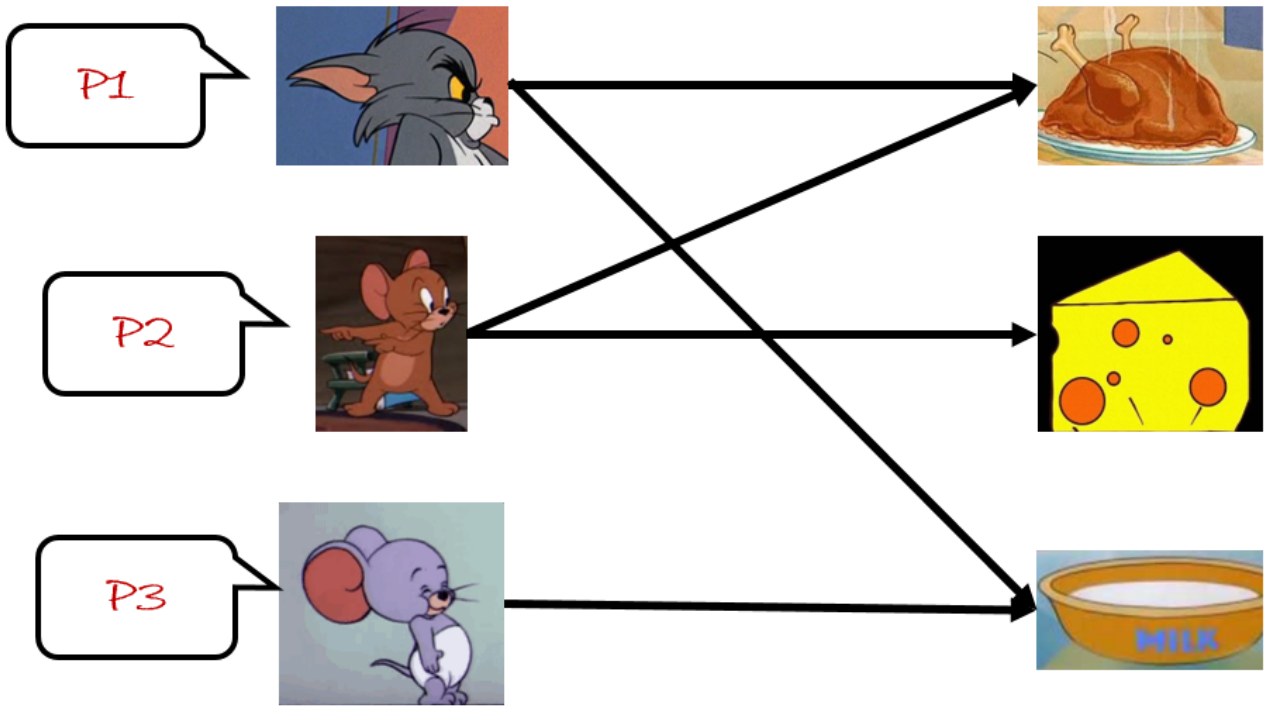
Matching Problem II



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

Tom and Nibbles cannot drink milk at the same time.

Matching Problem II



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

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

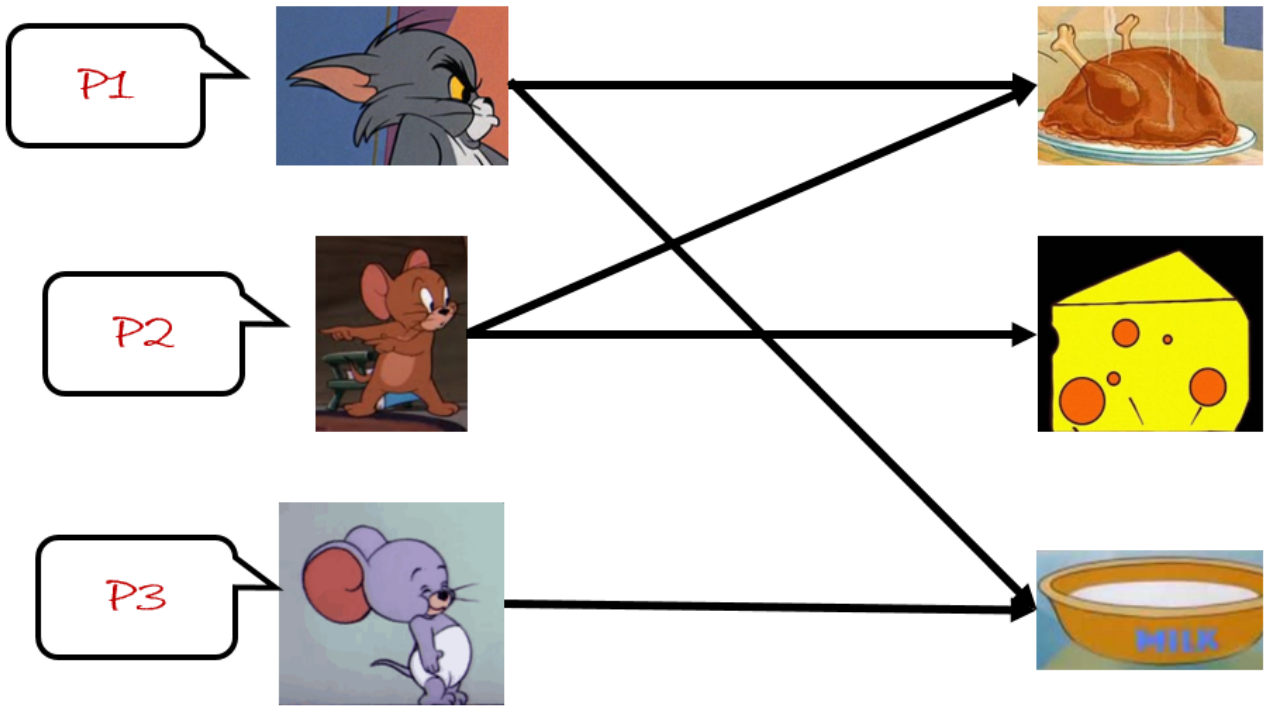
$$= P3 \wedge (P1 \vee \neg P3) \wedge (\neg P1 \vee \neg P2)$$

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

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

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) If P is unsatisfiable, P is a contradiction

4) If P is not a contradiction, P is satisfiable

Satisfiability（可满足性）

RELATION

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

2) If P is unsatisfiable, 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
$(P \wedge \neg P) \rightarrow (P \wedge \neg P)$	tautology
$A \wedge (A \rightarrow B) \rightarrow B$	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).

$(\neg P)$

$(P \wedge Q)$

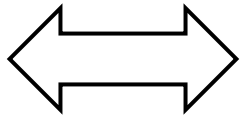
$(P \vee Q)$

$(P \rightarrow Q)$

$(P \leftrightarrow Q)$

WFF

Recognize well-formed formulas

Inductive Definition (WWF)  Recursive Function

Recursive Function

Operation for base elements.

Operation for elements built by elements-building operations.

Recursion（递归）

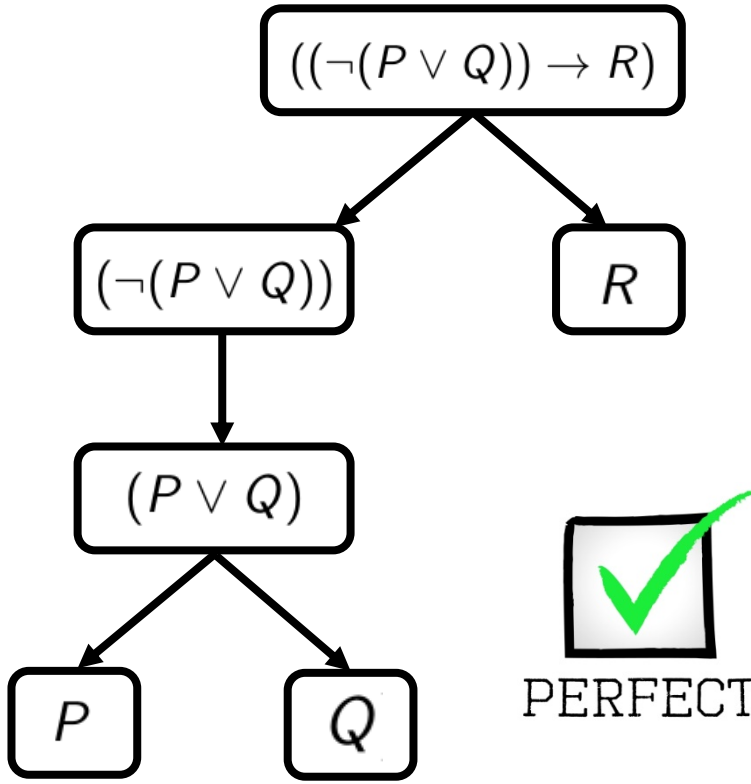
Fibonacci sequence function

- 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)

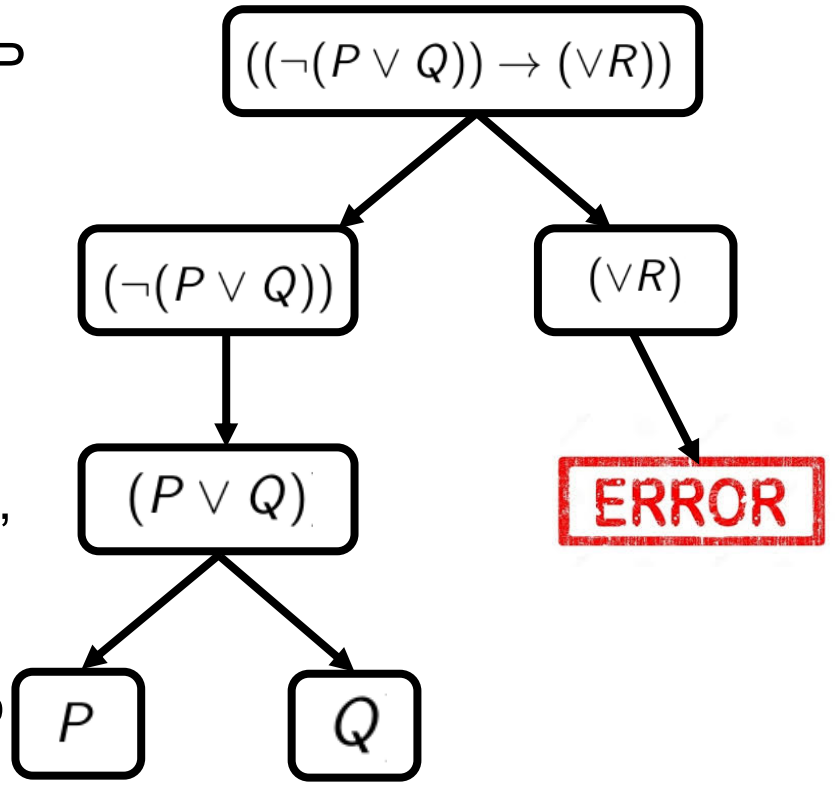
0. Begin with an initial tree T containing a single node labeled with P
1. If all leaves of T are labeled with propositional variables, return true
2. Select a leaf labeled with an expression f which is not a propositional variable
3. If f does not begin with (, return *false*
4. If f = (¬Q), then add a child to the leaf labeled by f, label it with Q, and goto 1
5. 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)

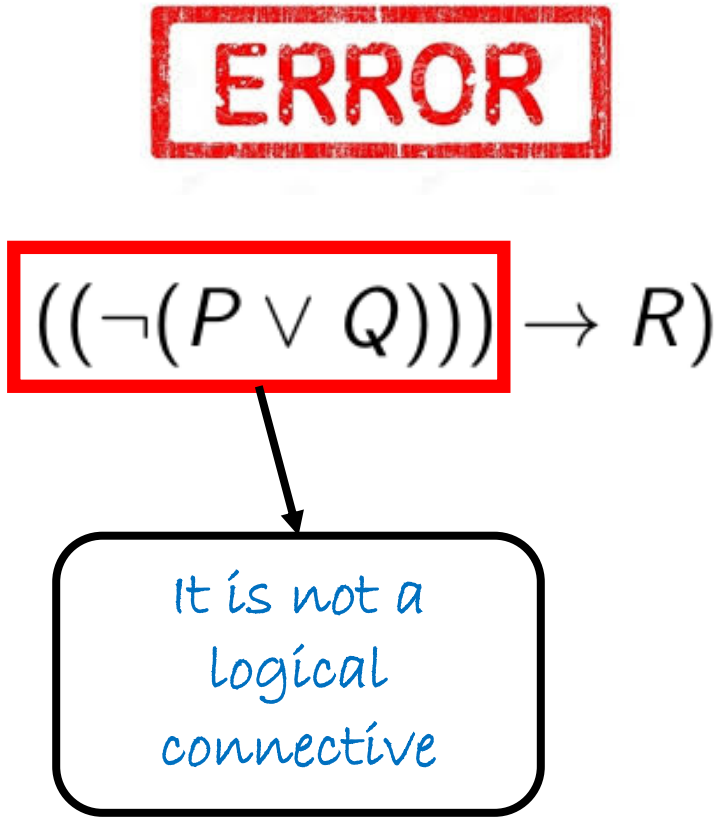
0. Begin with an initial tree T containing a single node labeled with P
1. If all leaves of T are labeled with propositional variables, return true
2. Select a leaf labeled with an expression f which is not a propositional variable
3. If f does not begin with (, return *false*
4. If f = (¬Q), then add a child to the leaf labeled by f, label it with Q, and goto 1
5. 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*
4. If f = (¬Q), then add a child to the leaf labeled by f, label it with Q, and goto 1
5. 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)

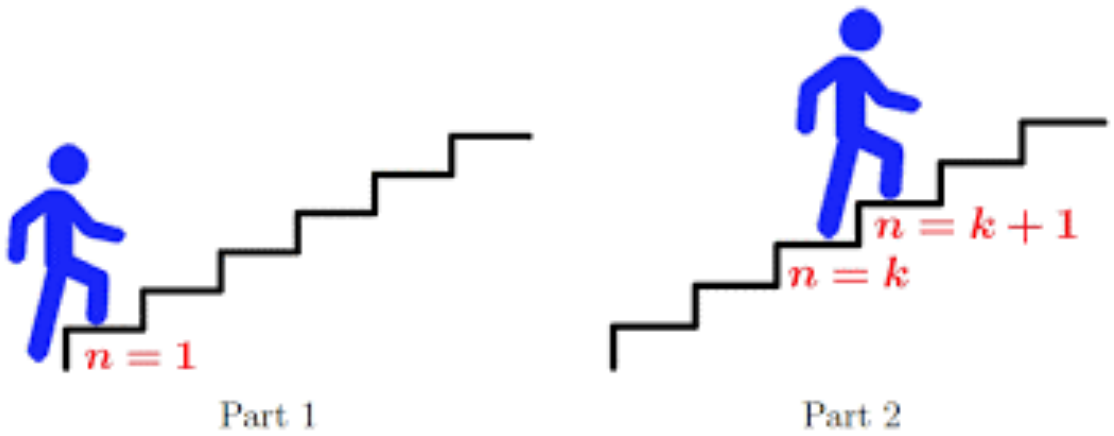
0. Begin with an initial tree T containing a single node labeled with P
1. If all leaves of T are labeled with propositional variables, return true
2. Select a leaf labeled with an expression f which is not a propositional variable
3. If f does not begin with (, return *false*
4. If f = (¬Q), then add a child to the leaf labeled by f, lab and goto 1
5. **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

How to prove it?

Induction（归纳法）

Steps of induction include

- base Case : prove the property holds for base elements
- inductive case : prove the property holds for elements built by elements-building operations according to induction hypothesis



PROOF BY INDUCTION

Let L(A) be the number of left parentheses, and R(A) be the number of right parentheses in a formula A. We will prove this by induction by showing that L(A) = R(A) for single proposition A, and if L(A) = R(A) and L(B) = R(B), then L((A ⊙ B)) = R((A ⊙ B)) where ⊙ is one of {∧, ∨,→,↔}, and L((¬A)) = R((¬A)).

First, for a single propositional A, there are no left and right parentheses, thus L(A) = R(A) = 0.

Second, let's assume L(A) = R(A) and L(B) = R(B),

For "Not" Case: L((¬A)) = 1 + L(P), R((¬A)) = 1 + R(A), thus L((¬A)) = R((¬A)).

For "And" Case: L((A ∧ B)) = 1 + L(A) + L(B), R((A ∧ B)) = 1 + R(A) + R(B),
thus L((P ∧ Q)) = R((P ∧ Q)).

For "Or" Case: L((A ∨ B)) = 1 + L(A) + L(B), R((A ∨ B)) = 1 + R(A) + R(B),
thus L((P ∨ Q)) = R((P ∨ Q)).

For "Implication" Case: …

For "Biconditional connective" Case: …

Therefore, the theorem is true. ■

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

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

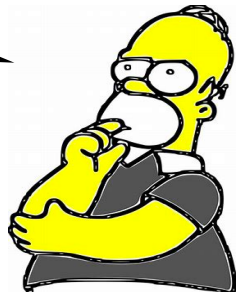
In computer science, the satisfiability problem (abbreviated SATISFIABILITY or SAT) is the problem of determining if there exists an interpretation that satisfies a given formula.

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

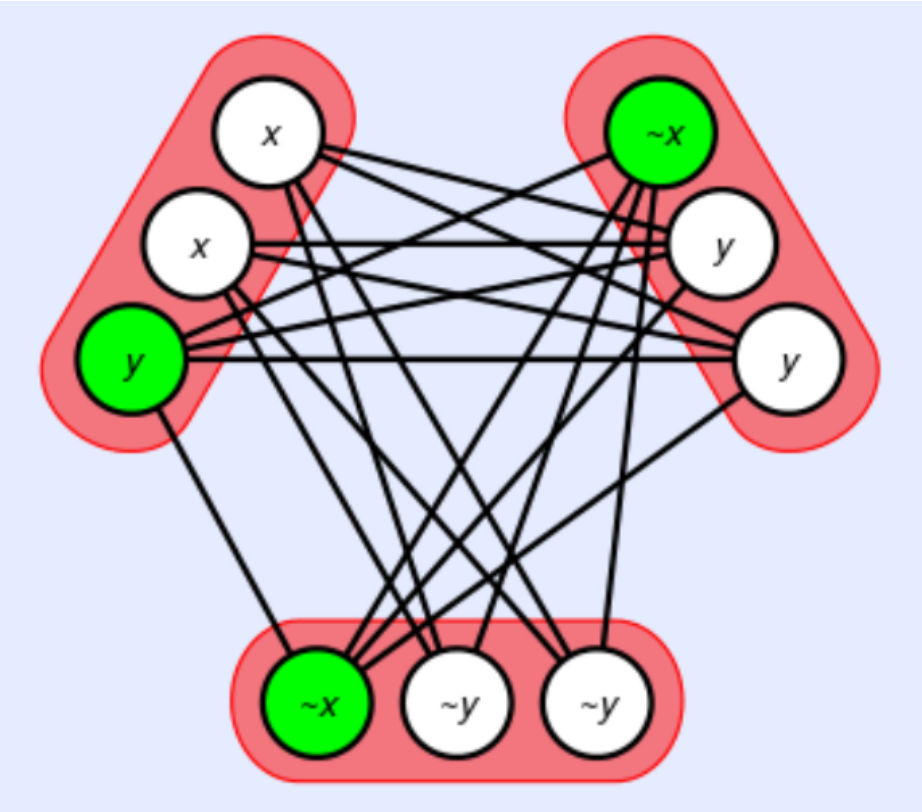
Stephen Cook

Leonid Levin

Is there a way to speed up the algorithm?



SAT solver can do it!



http://www.satcompetition.org/