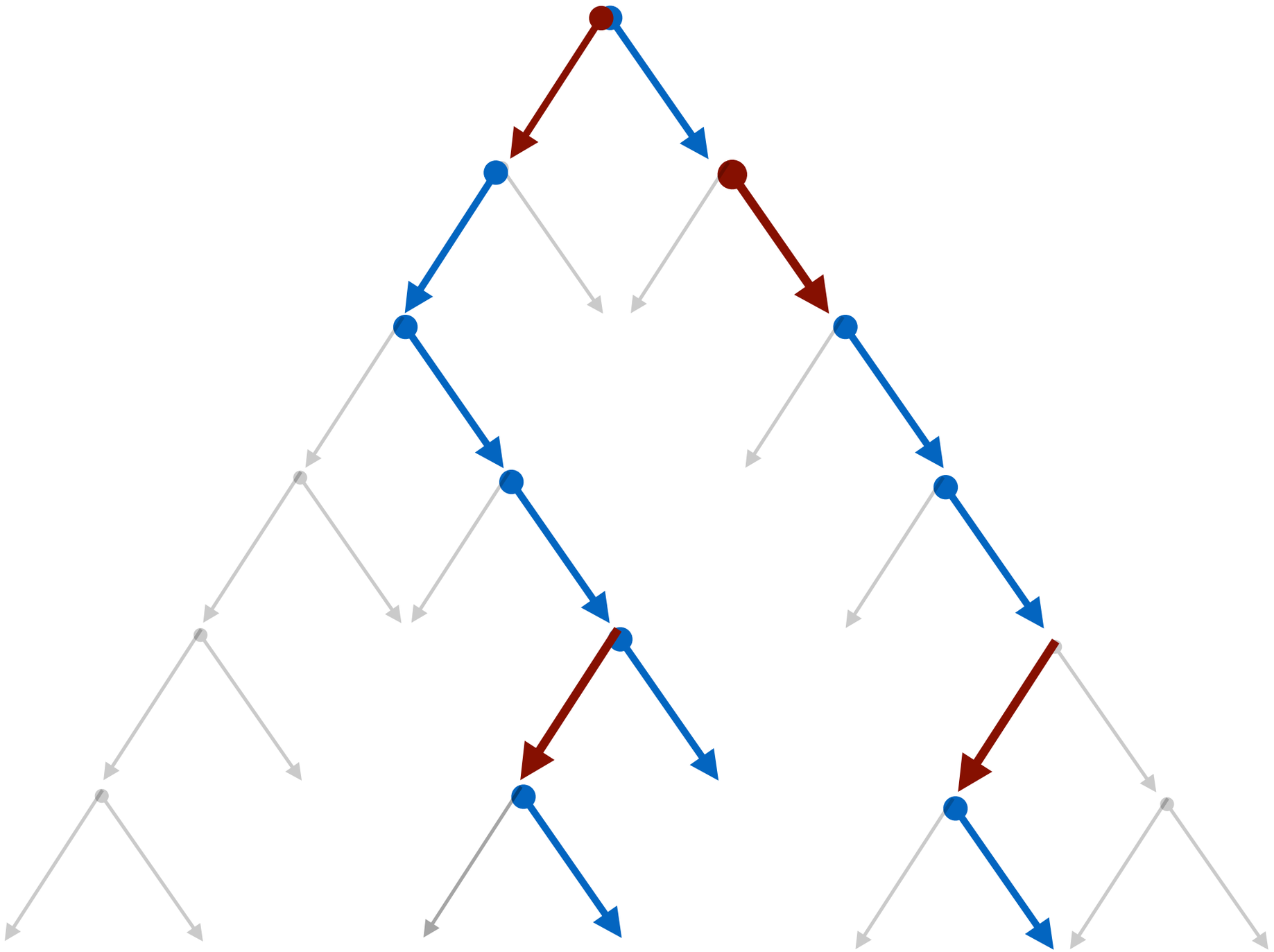
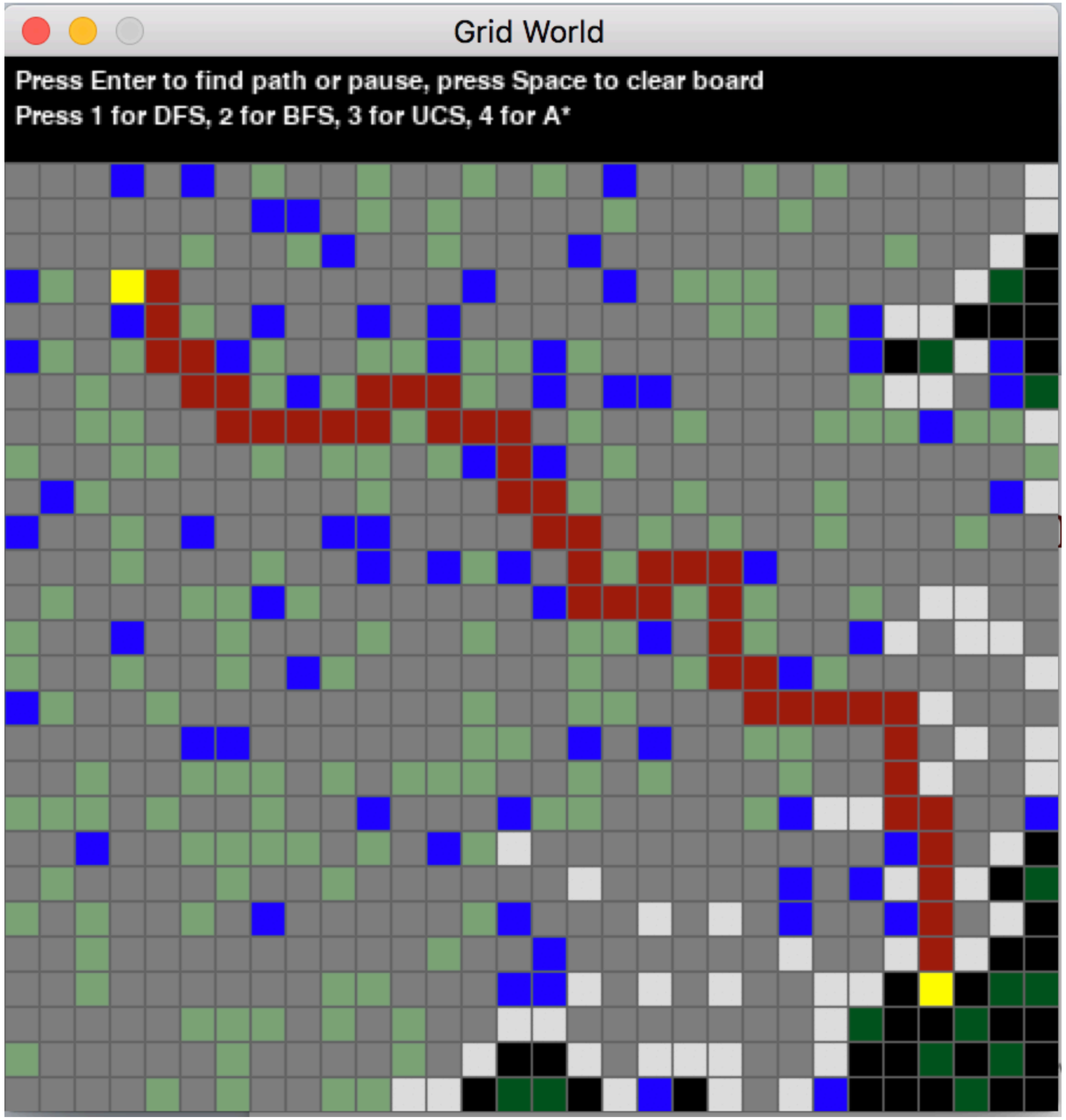
Combinatorial Search (SAT and ILP)

Sicun Gao

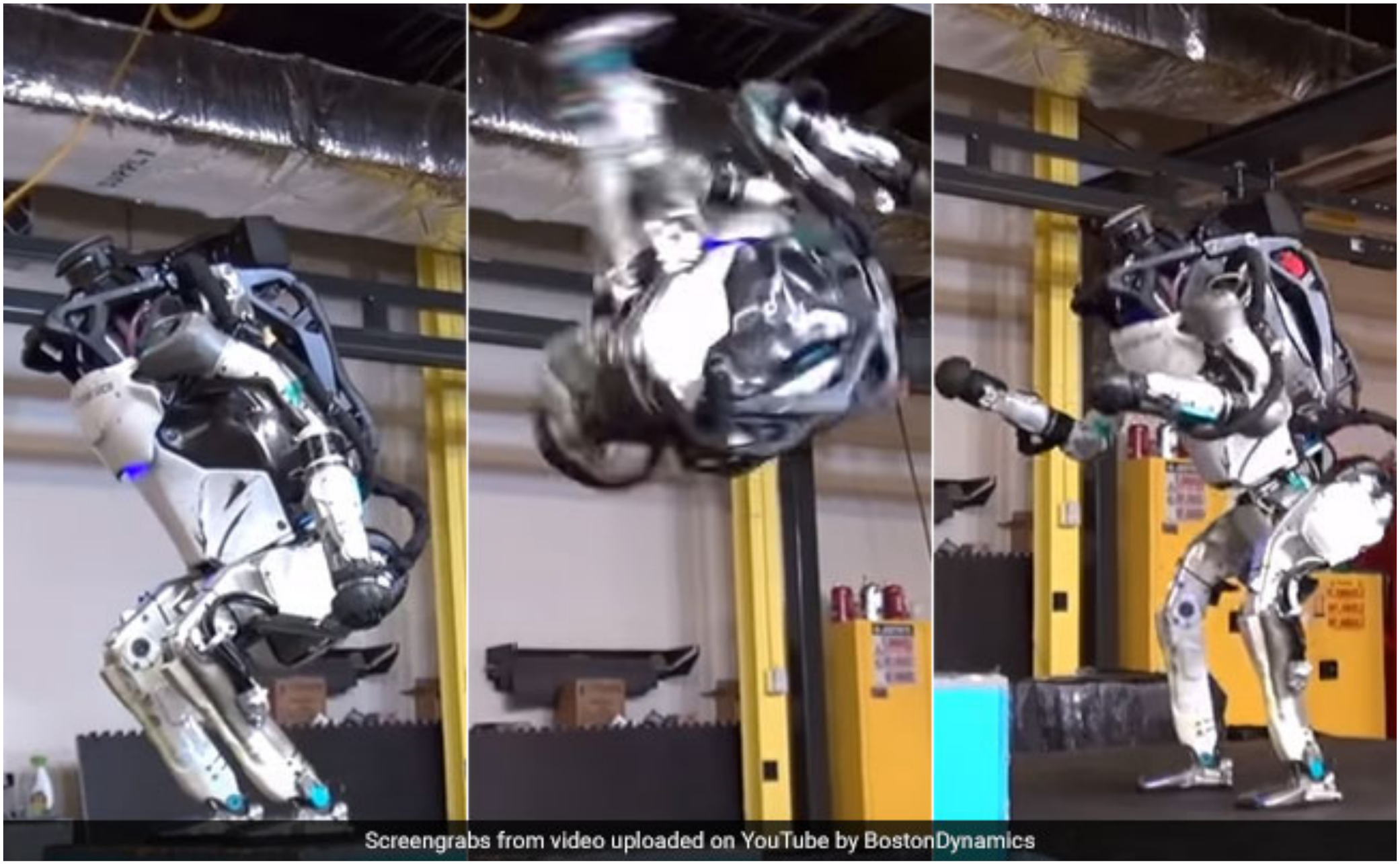
Constraint Solving



What to do for very large dimensions?

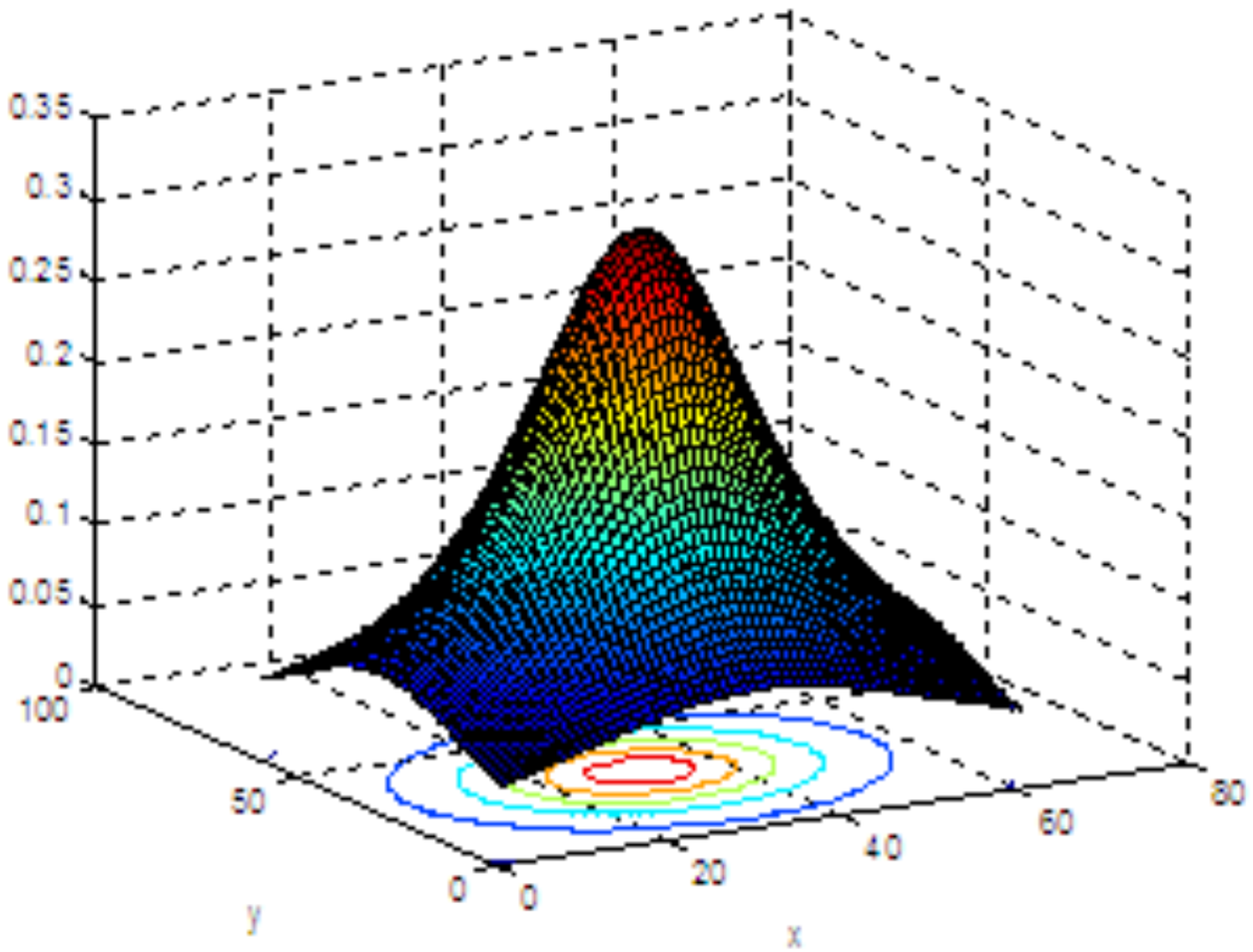


Or in an infinite space of states?



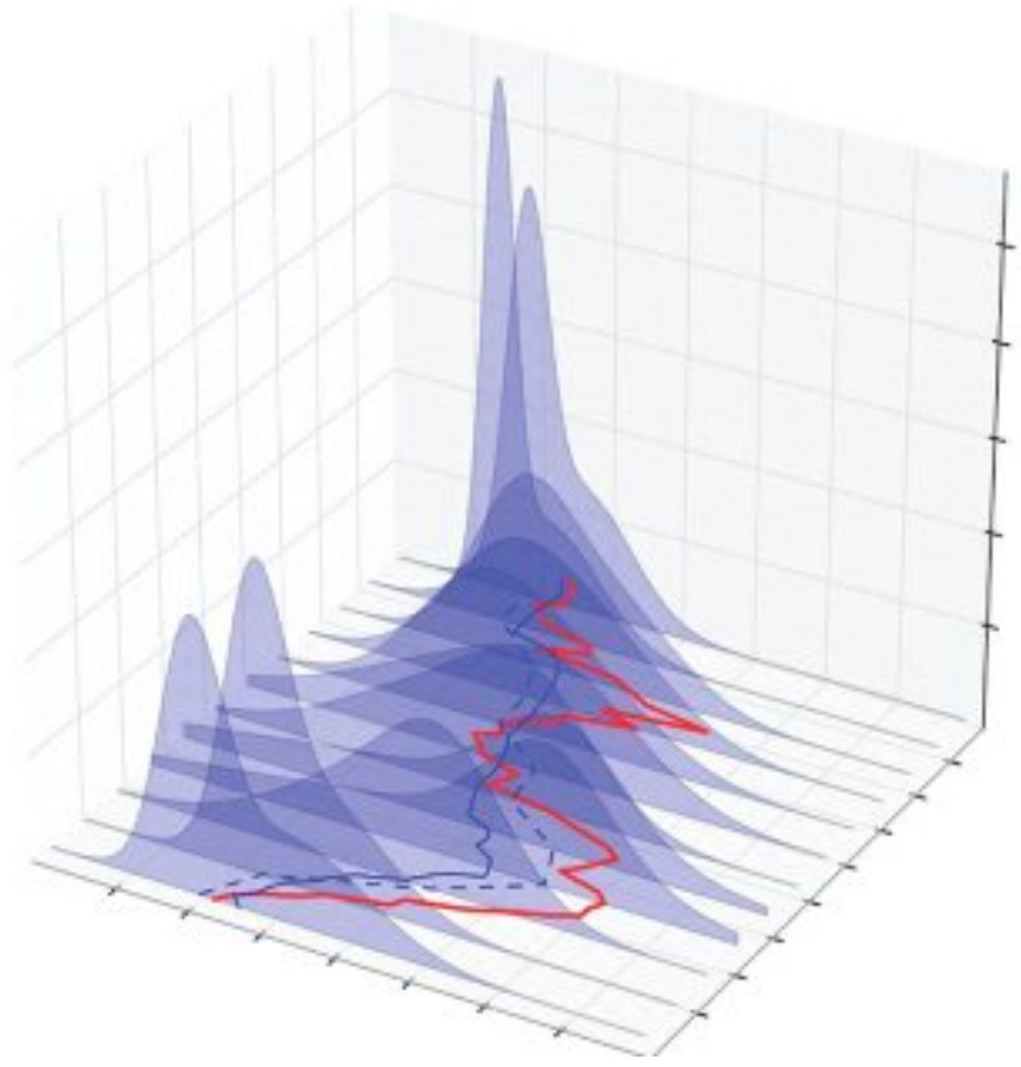
Screengrabs from video uploaded on YouTube by BostonDynamics

Need an Abstract Perspective Now



- Potential solutions form a space
- Search is about moving in the space and zoom into some solution.
- The legal moves are defined by constraints on the variables.

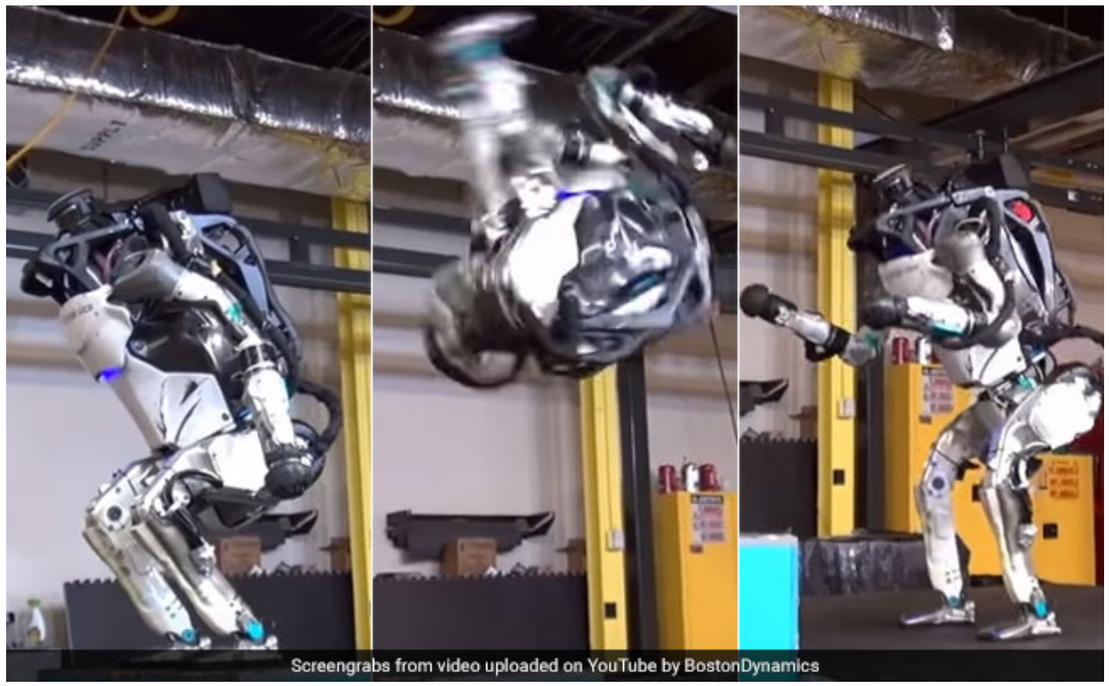
Need an Abstract Perspective Now

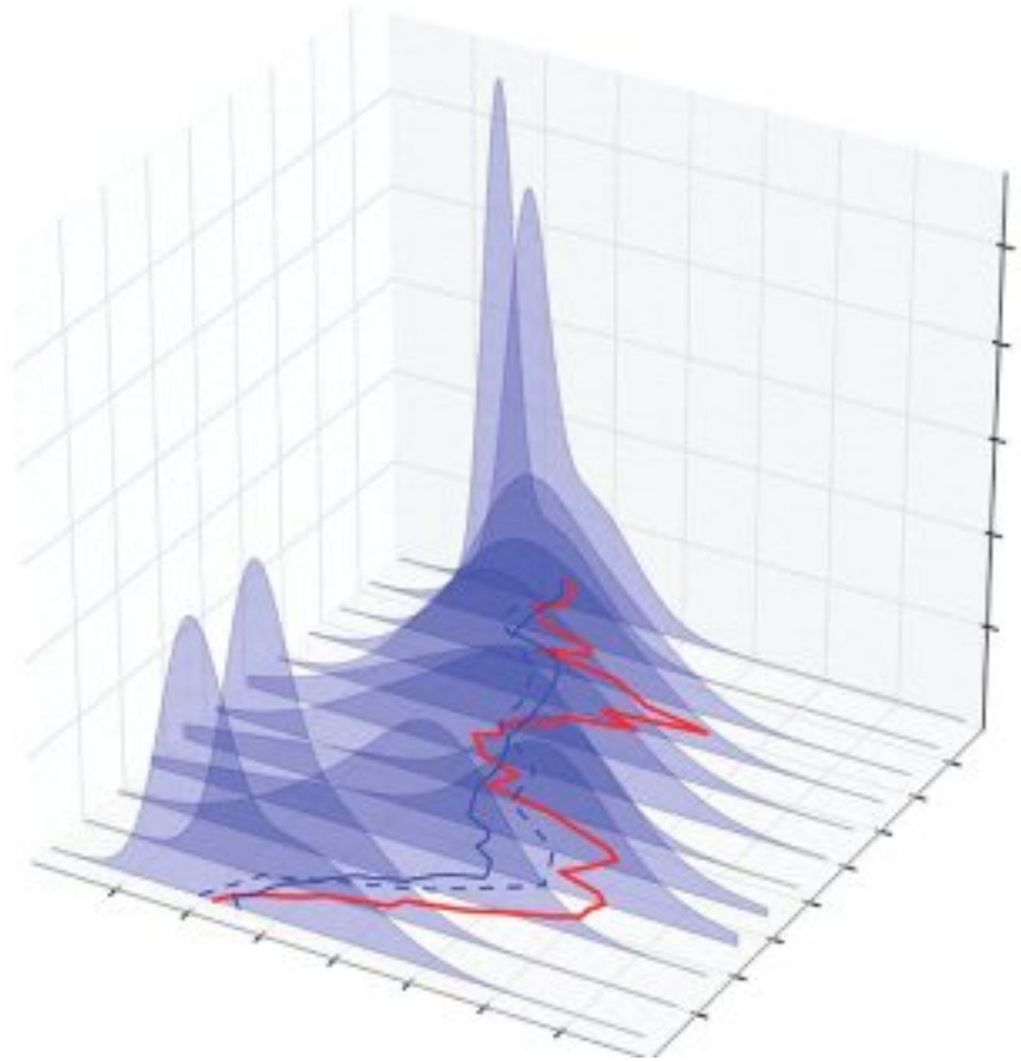


Space of solutions: $\mathbb{N} \times (\mathbb{N} \times \mathbb{N}) \times \mathbb{R}^+$

steps coordinates cost

Need an Abstract Perspective Now





Space of solutions: $\mathbb{N} \times (\mathbb{R} \times \cdots \times \mathbb{R}) \times \mathbb{R}^+$

steps coordinates cost

Constraint Satisfaction Problems

- Variables: $X = \{x_1, ..., x_n\}$
- Domains: $D = D_1 \times \cdots \times D_n$
- Constraints: $C = \{c_1, ..., c_m\}$

  Each constraint defines a subset of D.

Constraint Satisfaction Problems

- Variables: $X = \{x_1, ..., x_n\}$
- Domains: $D = D_1 \times \cdots \times D_n$
- Constraints: $C = \{c_1, ..., c_m\}$

  Each constraint defines a subset of D.

Goal: Find an assignment of X that satisfies all C.

$$\sigma : X \to D$$

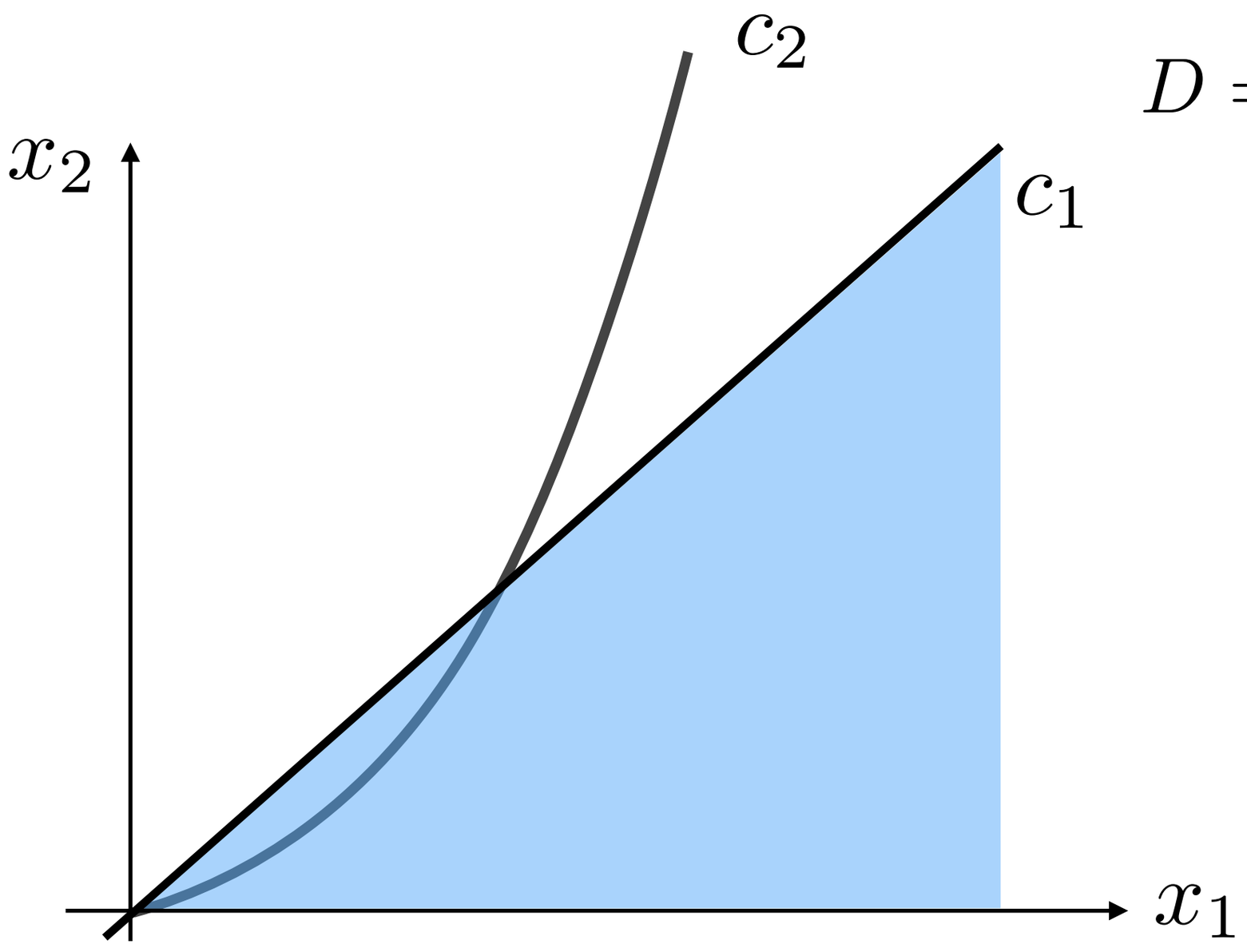
Example

- Variables: $X = \{x_1, x_2\}$
- Domains: $D = [0, 10] \times [0, 10]$
- Constraints: $C = \{c_1 : x_1 > x_2, c_2 : x_2 \geq x_1^2\}$

Example

$$X = \{x_1, x_2\} \quad C = \{c_1 : x_1 > x_2, c_2 : x_2 \geq x_1^2\}$$

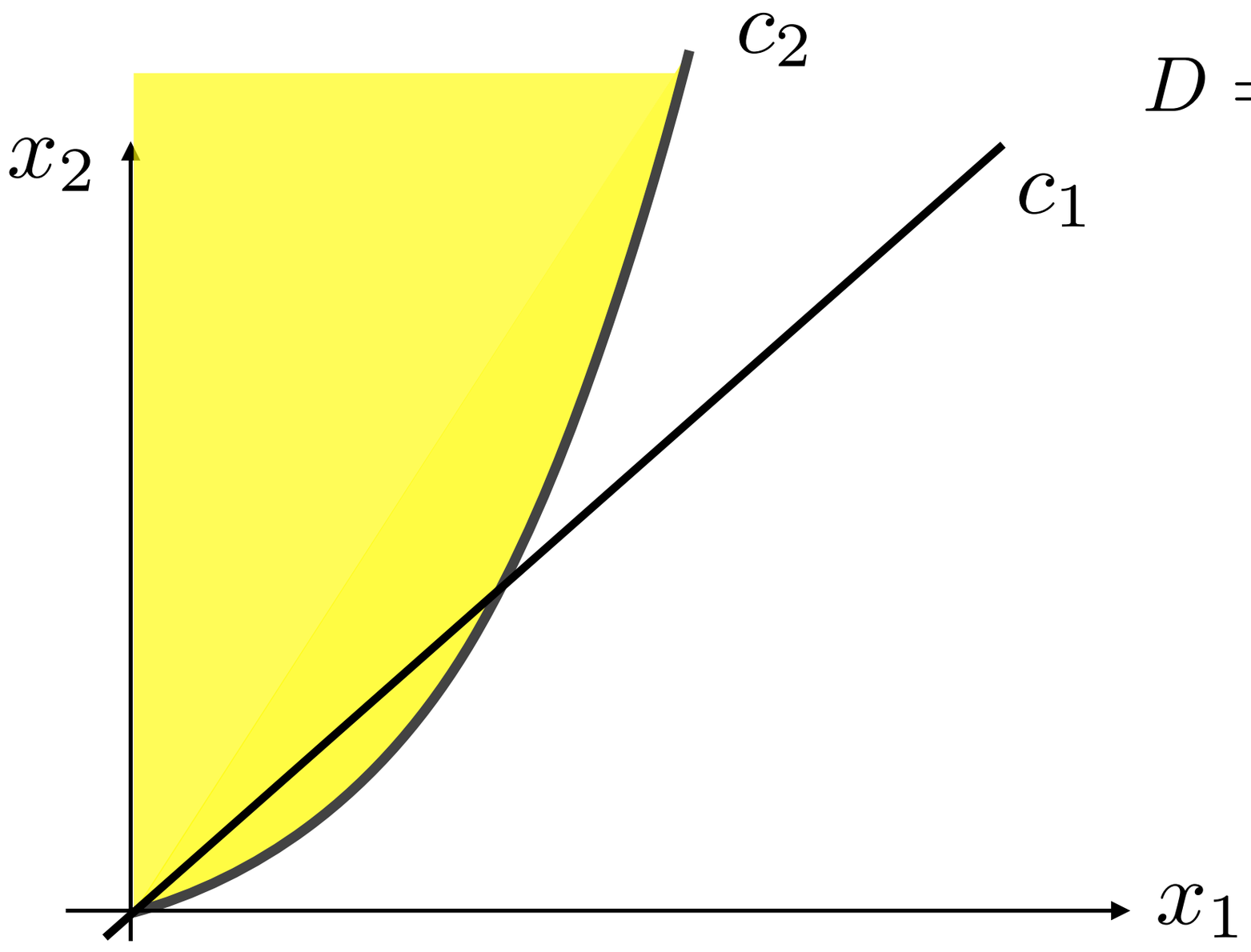
$$D = [0, 10] \times [0, 10]$$



Example

$$X = \{x_1, x_2\} \quad C = \{c_1 : x_1 > x_2, c_2 : x_2 \geq x_1^2\}$$

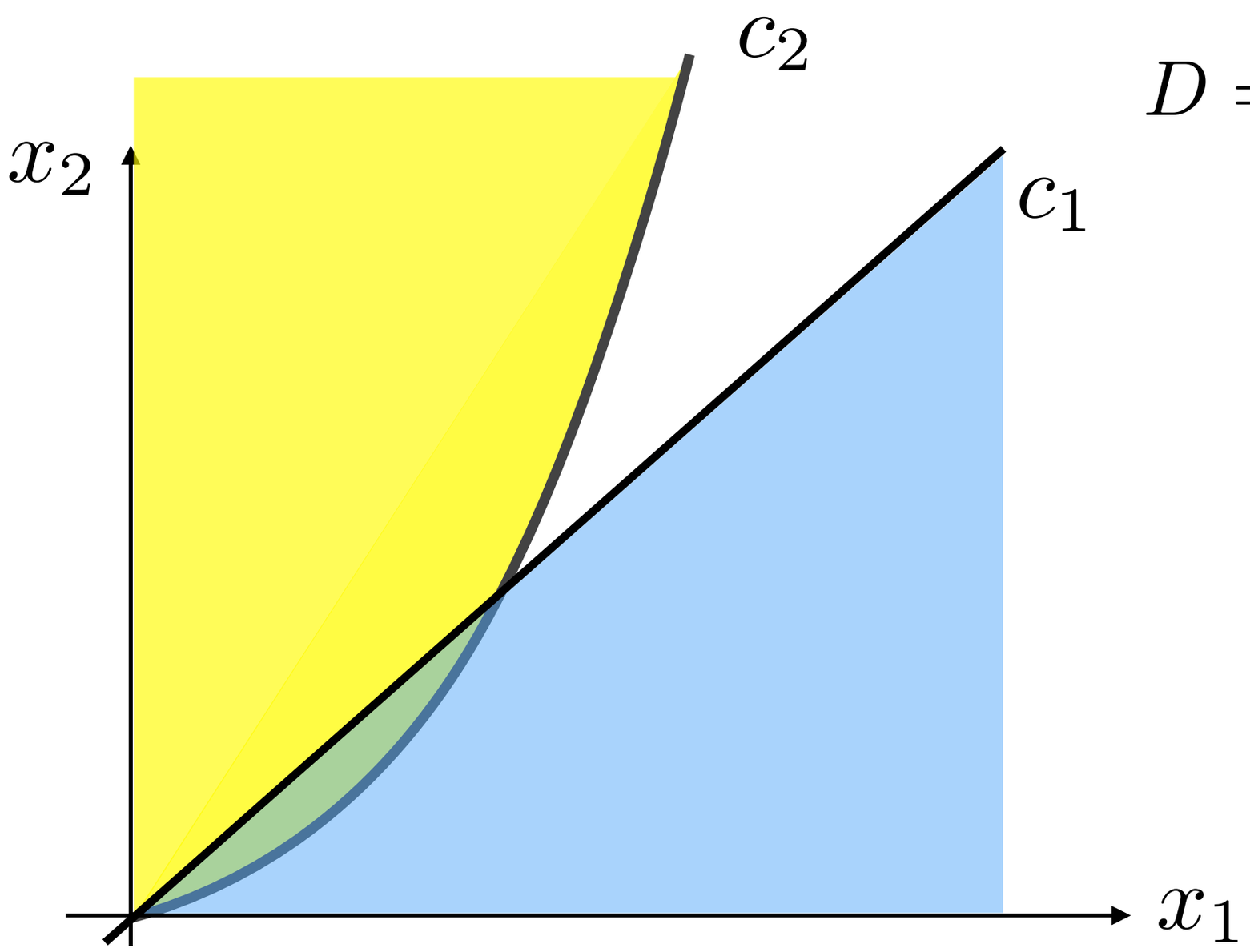
$$D = [0, 10] \times [0, 10]$$



Example

$$X = \{x_1, x_2\} \quad C = \{c_1 : x_1 > x_2, c_2 : x_2 \geq x_1^2\}$$

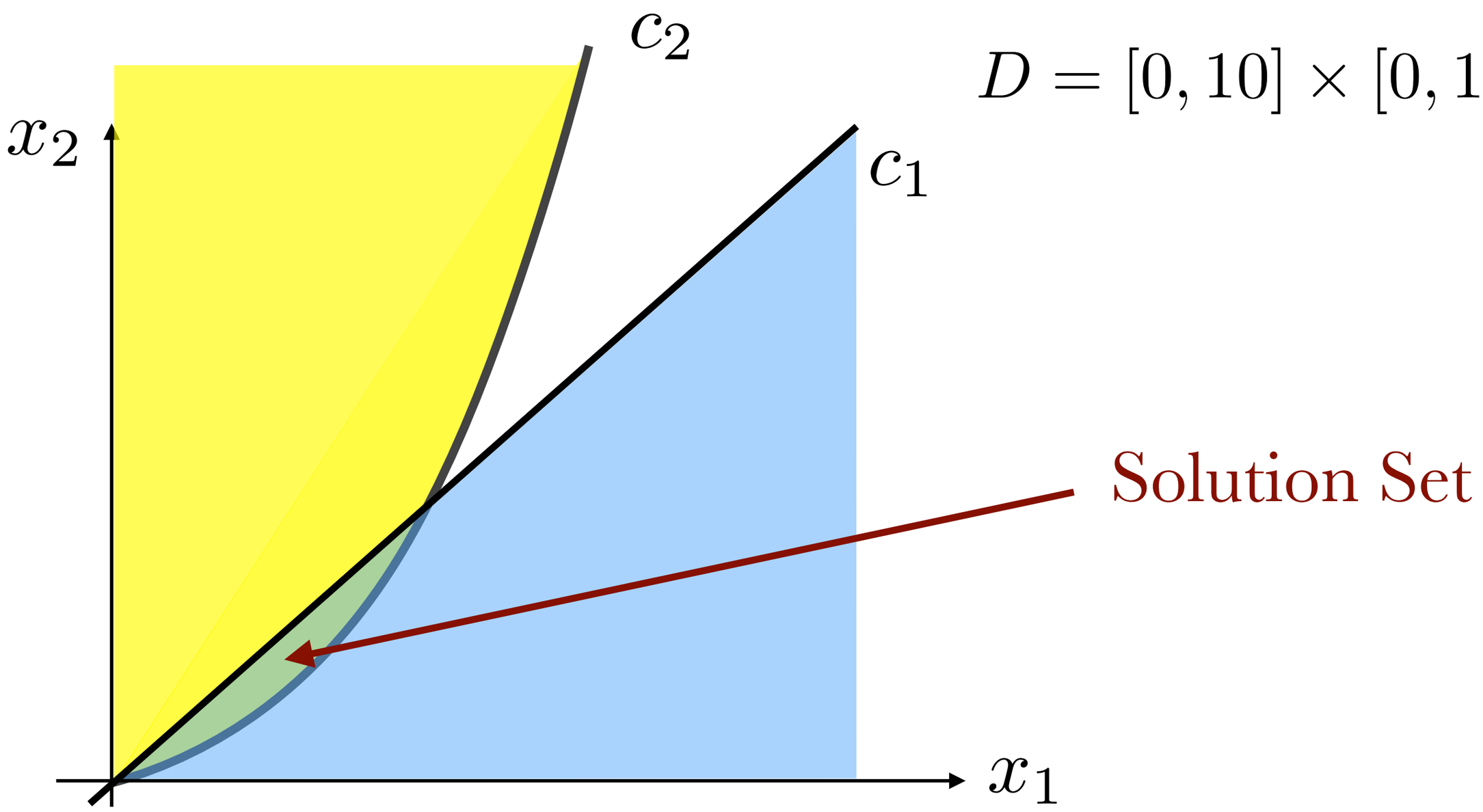
$$D = [0, 10] \times [0, 10]$$



Example

$$X = \{x_1, x_2\} \quad C = \{c_1 : x_1 > x_2, c_2 : x_2 \geq x_1^2\}$$

$$D = [0, 10] \times [0, 10]$$



Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

$$X = \{x_{(1,1)}, \ldots, x_{(9,9)}\}$$

$$D = \{1, \ldots, 9\}^{81}$$

$$C = ?$$

σ :

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

How do you solve Sudoku?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Make a Decision

5	3			7				
6			1	9	5			
	9	8		**4**			6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

What is the reasoning?

5	3			7				
6			1	9	5			
	9	8		4	X		6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Its original domain D is {1,…,9}
- Because X has to be different all the other numbers in its row, applying consistency, its domain is reduced to {1,2,3,5,7}

What is the reasoning?

5	3			7				
6			1	9	5			
	9	8		4	X		6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Its current domain D is {1,2,3,5,7}
- Because X has to be different all the other numbers in its column, applying consistency, its domain is reduced to {1,2,7}

What is the reasoning?

5	3			7				
6			1	9	5			
	9	8		4	X		6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

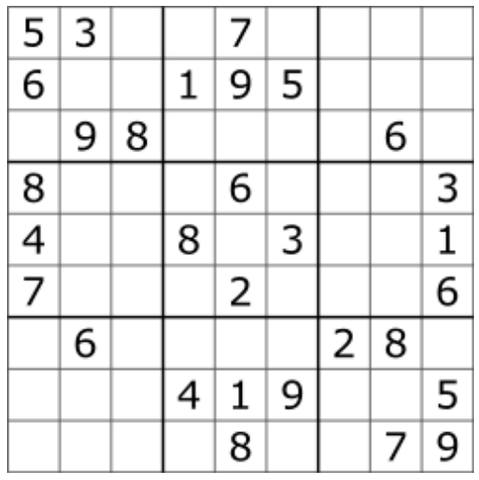
- Its current domain D is {1,2,7}
- Because X has to be different all the other numbers in its box, applying consistency, its domain is reduced to {2}

What is the reasoning?

5	3			7				
6			1	9	5			
	9	8		4	2		6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Its current domain D is {1,2,7}
- Because X has to be different all the other numbers in its box, applying consistency, its domain is reduced to {2}

Key Idea: Arc Consistency



x_1 is arc-consistent with x_2 in the domain of $D_1 \times D_2$ with respect to $C(x_1, x_2)$

if for any $a_1 \in D_1$ there exists $a_2 \in D_2$ such that $(a_1, a_2) \in C^D$

Key Idea: Arc Consistency

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

x_1 is arc-consistent with x_2 in the domain of $D_1 \times D_2$ with respect to $C(x_1, x_2)$ if for any $a_1 \in D_1$ there exists $a_2 \in D_2$ such that $(a_1, a_2) \in C^D$

We can then prune out large parts of the domains that are not arc-consistent.

What is the reasoning?

5	3			7				
6			1	9	5			
1	9	8	**3**	**4**	**2**	**5**	6	**7**
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Just by applying the same reasoning, we can fill in the row.
- This is why you feels that these steps are “easy”

(translation: polynomial-time)

Arc-Consistency gives us a lot, but not all

5	3			7				
6			1	9	5			
1	9	8	**3**	**4**	**2**	**5**	6	**7**
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Let's keep making decisions and apply consistency.

Arc-Consistency gives us a lot, but not all

5	3	1	6	7	8	4	9	1
6			1	9	5			
1	9	8	3	4	2	5	6	7
8			5	6				3
4			8		3			1
7			9	2				6
	6		7			2	8	
			4	1	9			5
			2	8			7	9

- The hard part of Sudoku will show up if we keep going.
- **Conflict!**

Arc-Consistency gives us a lot, but not all

5	3	1	6	7	8	4	9	1
6			1	9	5			
1	9	8	3	4	2	5	6	7
8			5	6				3
4			8		3			1
7			9	2				6
	6		7			2	8	
			4	1	9			5
			2	8			7	9

- **Conflict!**
- Now we need to erase some values.
- **Backtracking**

What do we need to do in backtracking?

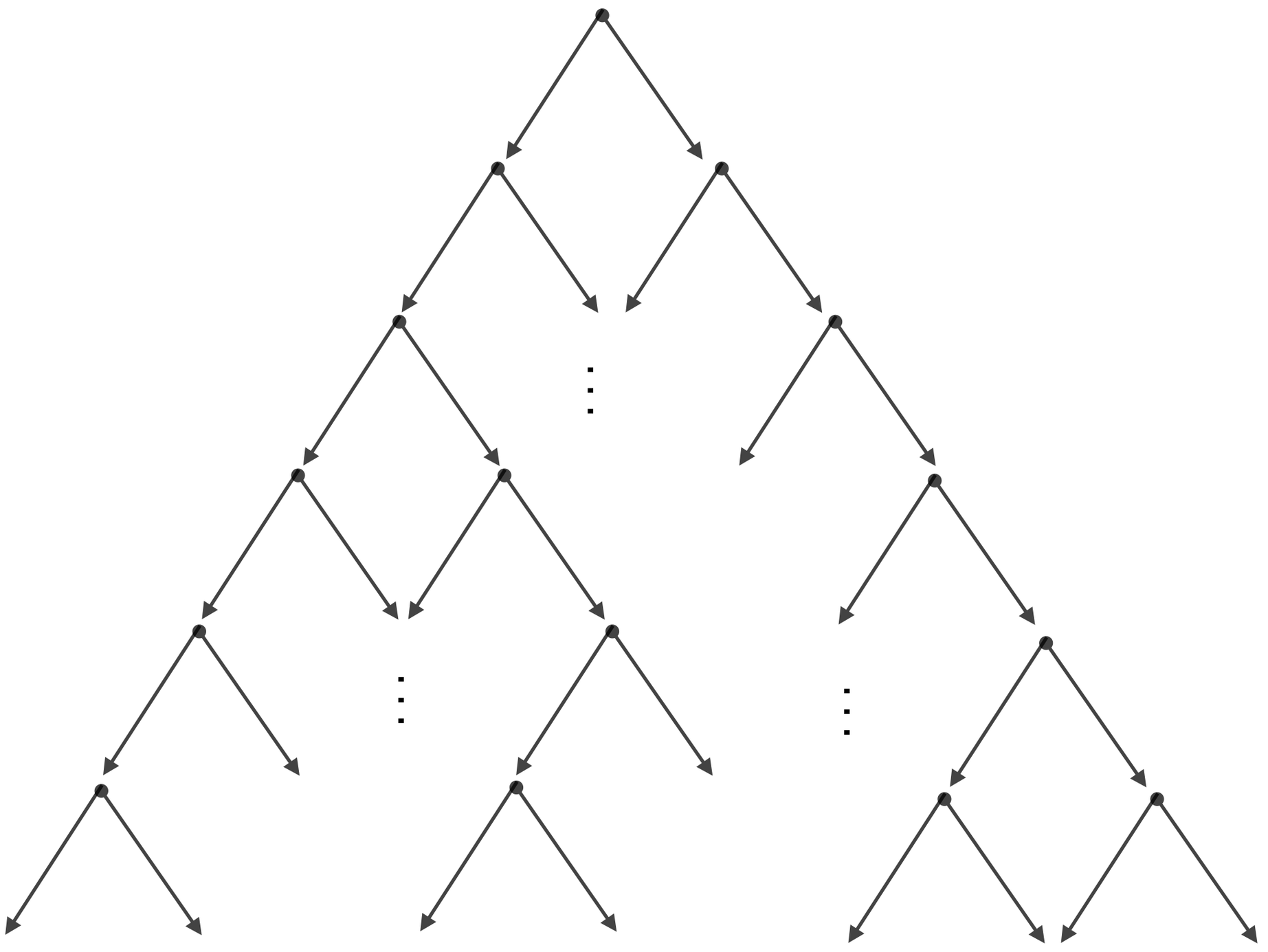
5	3	4	6	7	8	4		
6			1	9	5			
1	9	8	3	4	2	5	6	7
8			5	6				3
4			8		3			1
7			9	2				6
	6		7			2	8	
			4	1	9			5
			2	8			7	9

- Give up some pruning.
- Change some decision.
- Backtracking makes the whole thing complicated.
(translation: exponential time)

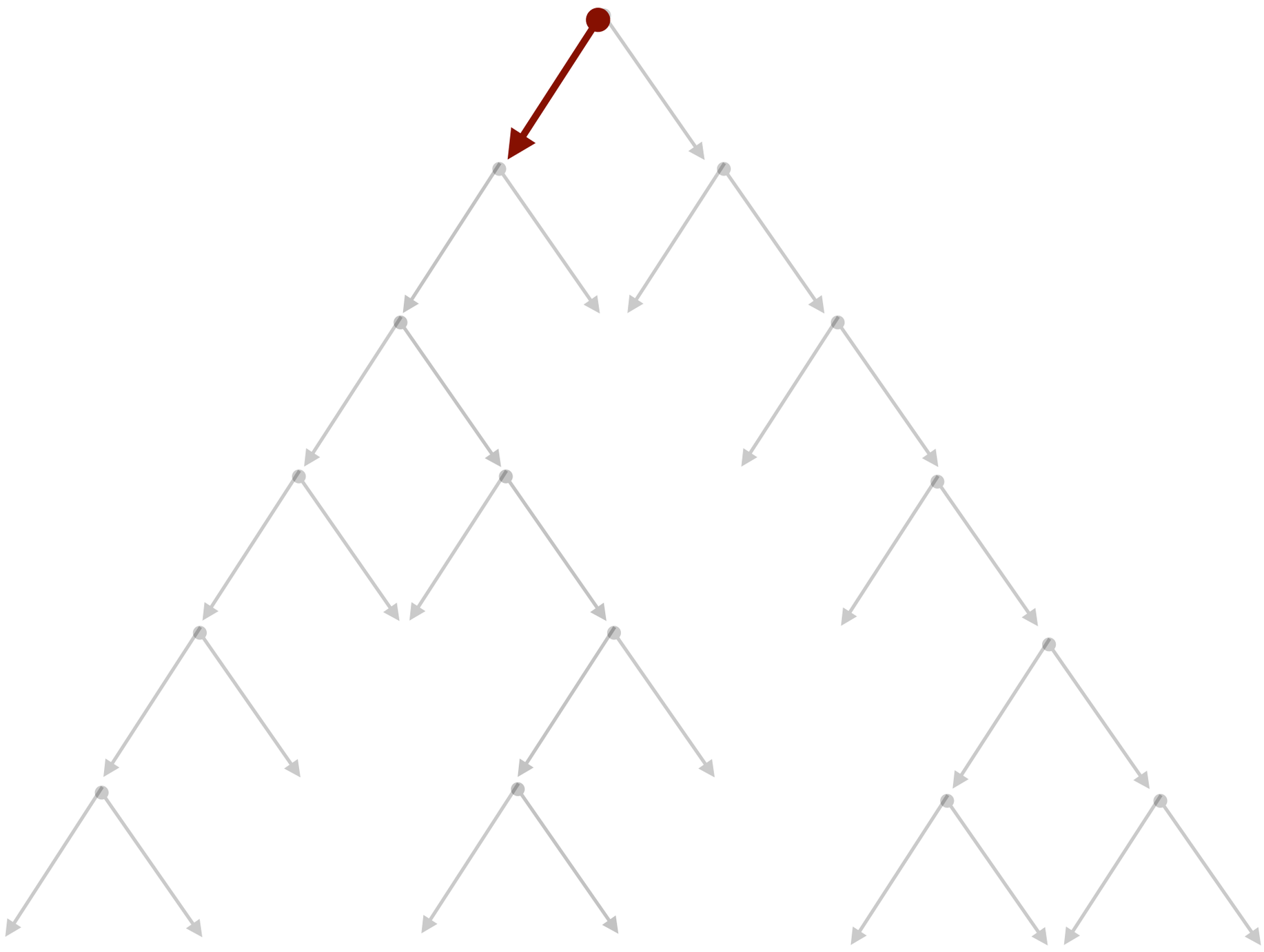
Constraint Solving via Backtracking Search

- Make decisions. (Guess)
- For each decision, apply arc-consistency to "propagate" the decision to its logical consequences. (Propagate, no guessing)
- When you reach a conflict, backtrack, and make different decisions. (Change some guess)

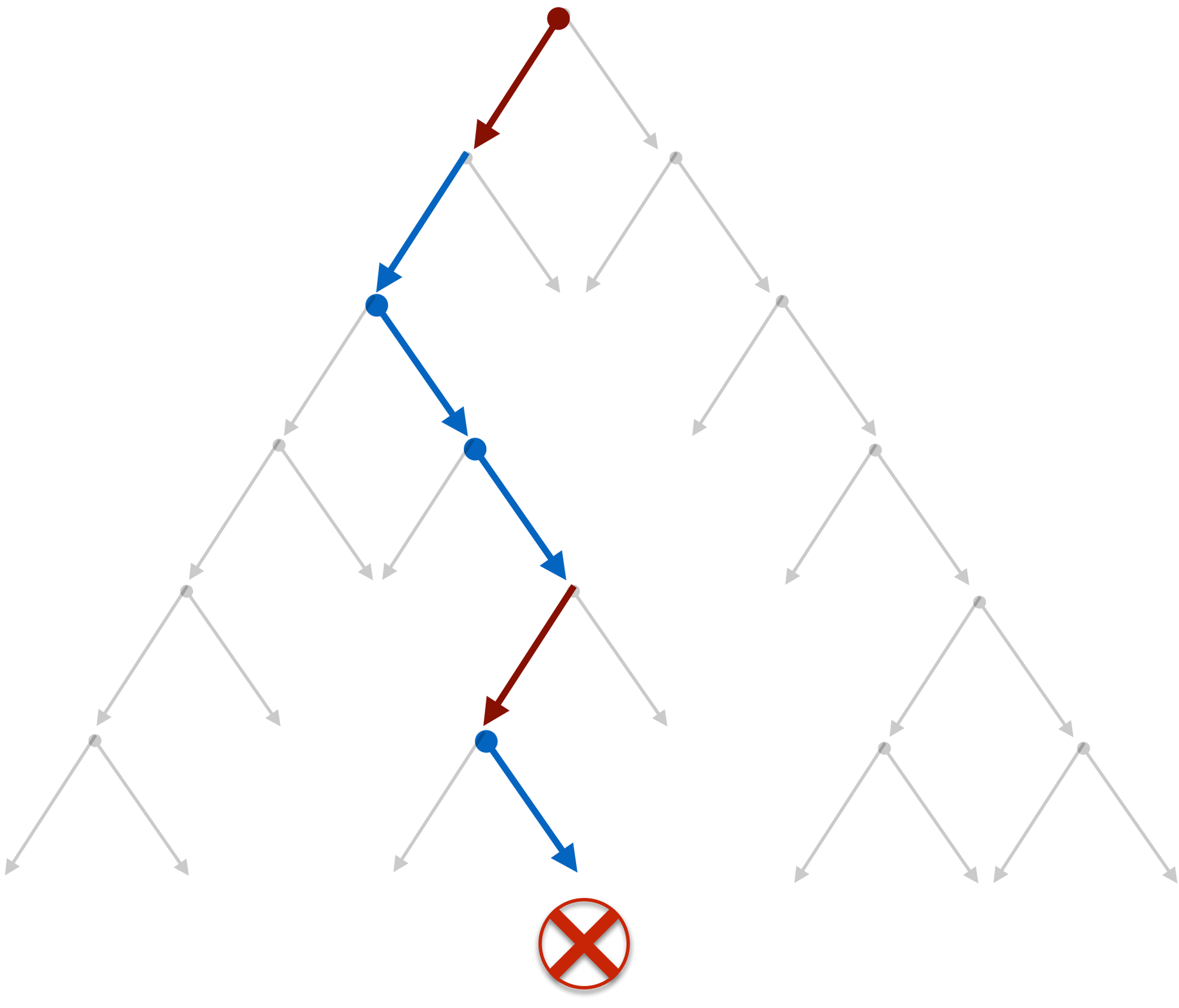
Potentially huge decision tree spanning the space



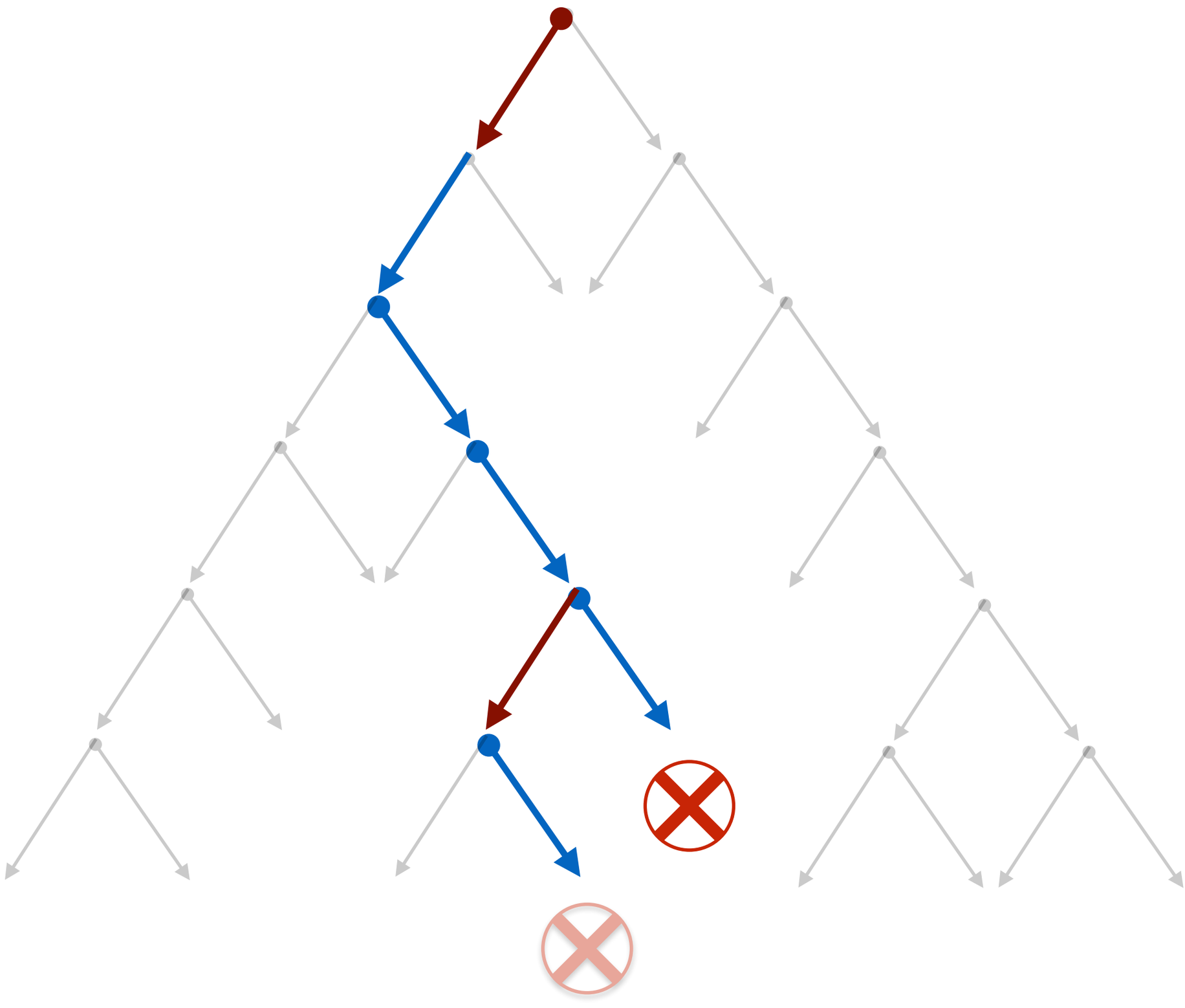
Alternate between guessing …



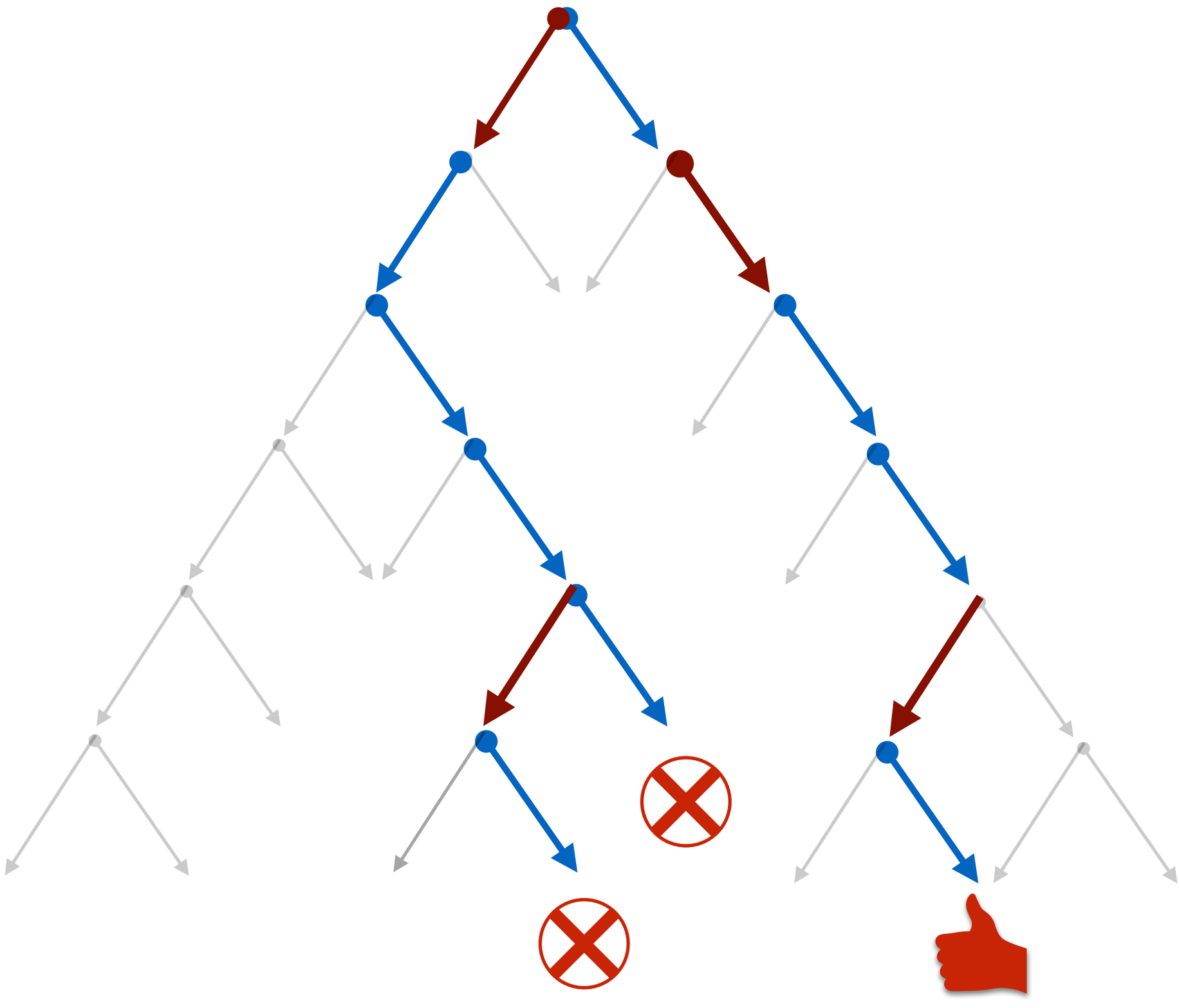
Alternate between guessing, propagating, …



Alternate between guessing, propagating, and backtracking



Eventually find a good answer



Backtracking Search

function BACKTRACKING-SEARCH(csp) **returns** a solution, or failure
  return BACKTRACK($\{\}$, csp)

function SEARCH ($assignment$, csp) **returns** a solution, or failure
  if $assignment$ is complete **then return** $assignment$
  $var \leftarrow$ SELECT-UNASSIGNED-VARIABLE(csp)
for each $value$ **in** ORDER-DOMAIN-VALUES(var, $assignment$, csp) **do**
  if $value$ is consistent with $assignment$ **then**
    add $\{var = value\}$ to $assignment$
    $inferences \leftarrow$ INFERENCE(csp, var, $value$)
    if $inferences \neq failure$ **then**
      add $inferences$ to $assignment$
      $result \leftarrow$ SEARCH($assignment$, csp)
      if $result \neq failure$ **then**
        return $result$
  remove $\{var = value\}$ and $inferences$ from $assignment$
return $failure$

Discrete Combinatorial Problems: Languages

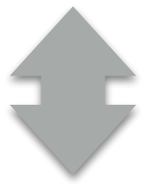
- Boolean Satisfiability (SAT)
- Constraint Satisfaction Problems (CSP)
- Integer Linear Programming (ILP)

- Other NP-complete formulations are mostly just theoretical interest

Discrete Combinatorial Problems: Languages

- SAT can be encoded as 0-1 ILP

$$q \leftrightarrow (p_1 \wedge p_2)$$



$$y \geq x_1 + x_2 - 1,\ y \leq x_1,\ y \leq x_2,\ 0 \leq y \leq 1$$

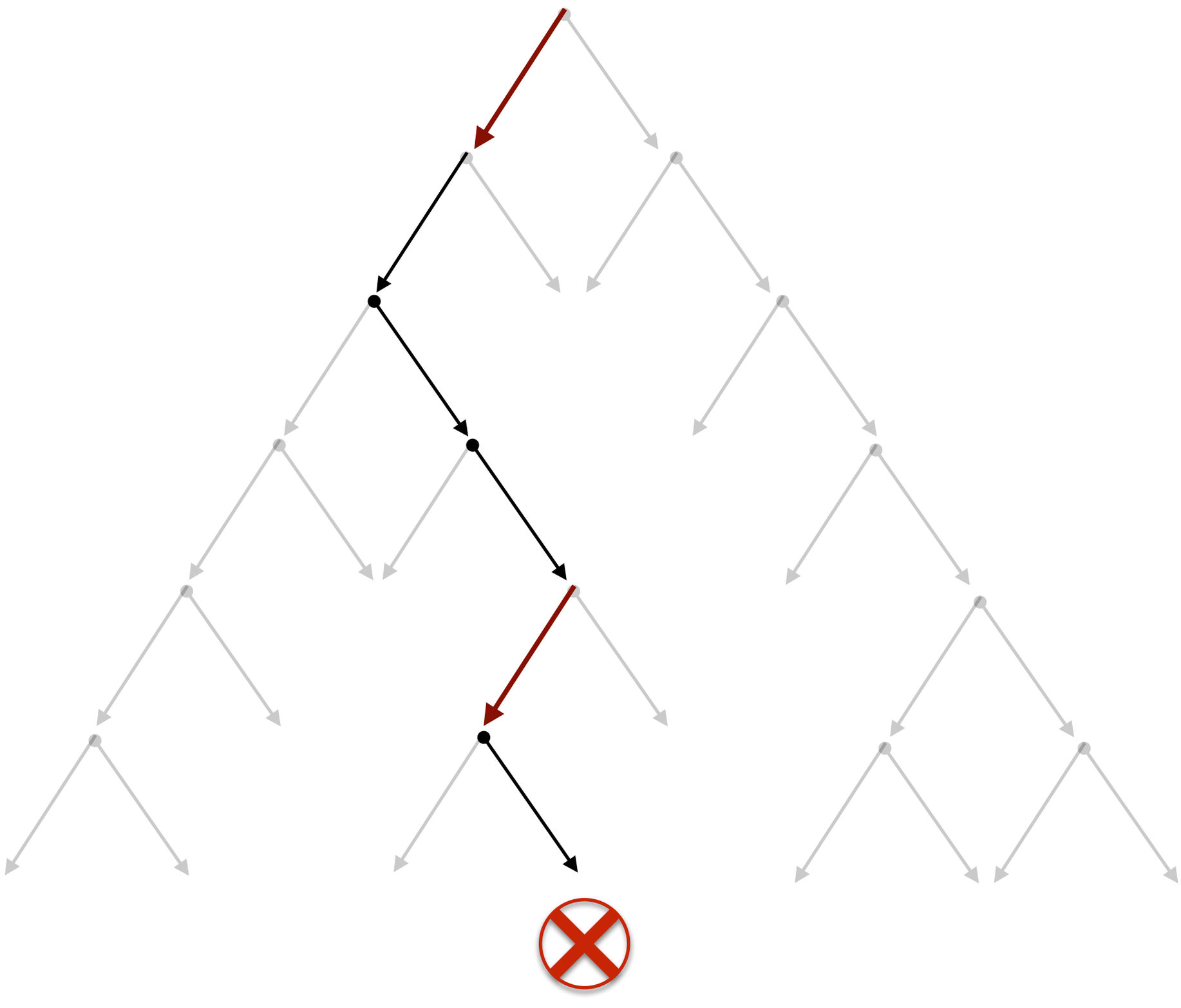
Discrete Combinatorial Problems: Languages

- ILP can be encoded as SAT
  - Use binary variables to bit-blast integers
  - Single-objective optimization reducible to feasibility in log(D)

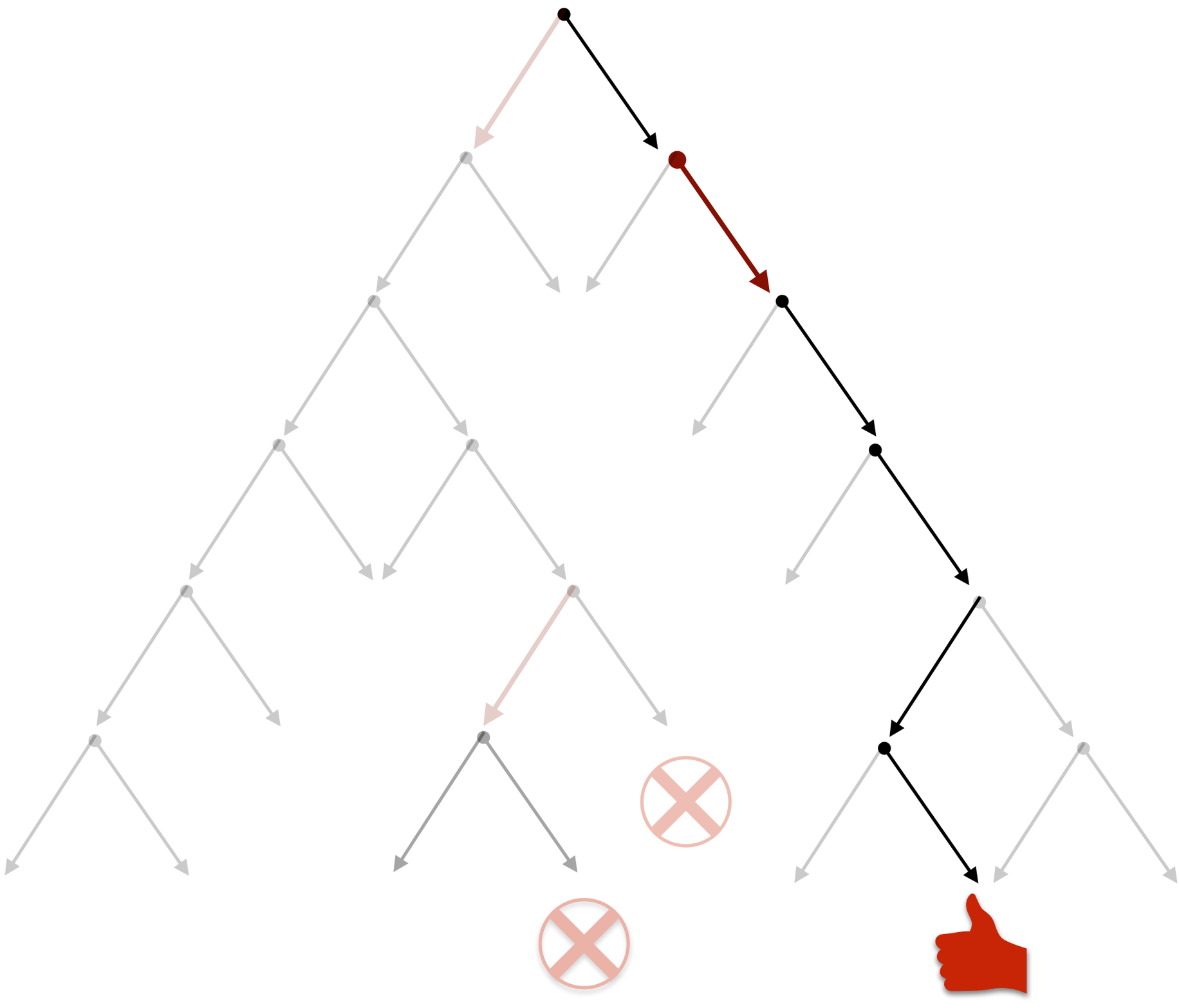
min f  f<c and f>c/2?

Potentially huge decision tree spanning the space

Most branches don’t work



Eventually find some good answer



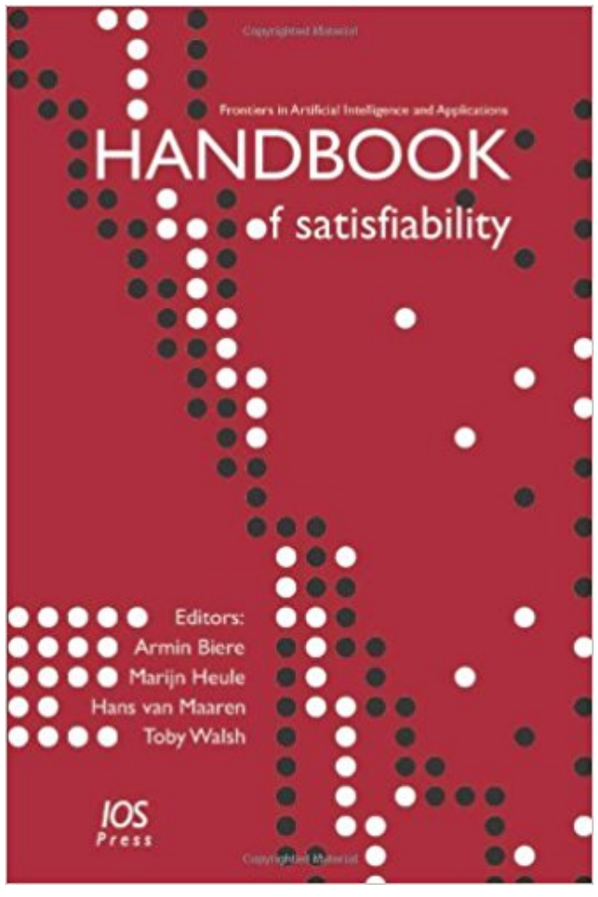
Discrete Combinatorial Problems: Languages

- The most popular question:

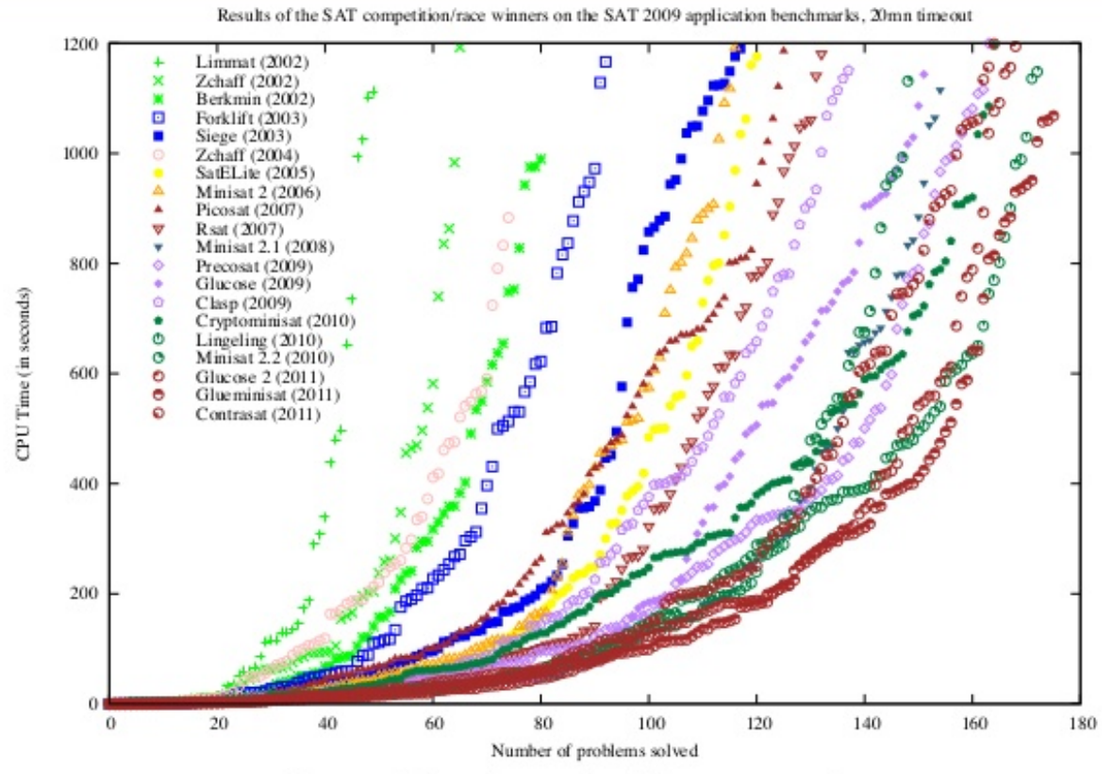
  Should we use SAT or ILP?

- No good answer
- Different styles of search
- Usable geometric structure: ILP
- Very large number of vars: SAT

Much progress in SAT since the 90s



Progress of SAT Solvers (shown by [Simon 2011])

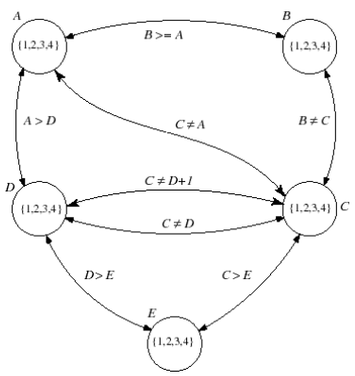


Cactus Plot shown by [Simon 2011]

Propositional Logic

Syntax:

$$\phi := p \mid \neg\phi \mid \phi \wedge \phi$$



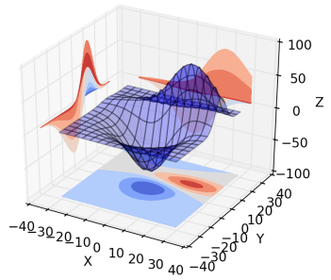
Semantics:

$$\sigma : \Phi \to \{0, 1\}$$

$$\sigma(p) \in \{0, 1\}$$

$$\sigma(\neg\phi) = 1 - \sigma(\phi)$$

$$\sigma(\phi_1 \wedge \phi_2) = \sigma(\phi_1) \cdot \sigma(\phi_2)$$



Syntax (Formulas)

Propositional Variables

$$P = \{p_1, p_2, ...\}$$

For any $p \in P$, p is a formula.

Negation

If ϕ is a formula, then $\neg\phi$ is a formula.

Syntax (Formulas)

Propositional Variables

$$P = \{p_1, p_2, ...\}$$

For any $p \in P$, p is a formula.

Negation

If ϕ is a formula, then $\neg\phi$ is a formula.

$$p, \neg p, \neg\neg p, ...$$

Syntax (Formulas)

Propositional Variables

For any $p \in P$, p is a formula.

Negation

If ϕ is a formula, then $\neg\phi$ is a formula.

Conjunction

If ϕ_1 and ϕ_2 are formulas, then $\phi_1 \wedge \phi_2$ is a formula.

Syntax (Formulas)

Propositional Variables

For any $p \in P$, p is a formula.

Negation

If ϕ is a formula, then $\neg\phi$ is a formula.

Conjunction

If ϕ_1 and ϕ_2 are formulas, then $\phi_1 \wedge \phi_2$ is a formula.

$$\neg(\neg p_1 \wedge (\neg p_2 \wedge p_3))$$

Syntax (Syntactic Sugar)

Disjunction?

Syntax (Syntactic Sugar)

Disjunction?

$$p_1 \vee p_2 := \neg(\neg p_1 \wedge \neg p_2)$$

Syntax (Syntactic Sugar)

Disjunction?

$$p_1 \vee p_2 := \neg(\neg p_1 \wedge \neg p_2)$$

Implication?

Syntax (Syntactic Sugar)

Disjunction?

$$p_1 \vee p_2 := \neg(\neg p_1 \wedge \neg p_2)$$

Implication?

$$p_1 \to p_2 := \neg p_1 \vee p_2$$

Syntax (Syntactic Sugar)

Disjunction? $p_1 \vee p_2 := \neg(\neg p_1 \wedge \neg p_2)$

Implication? $p_1 \to p_2 := \neg p_1 \vee p_2$

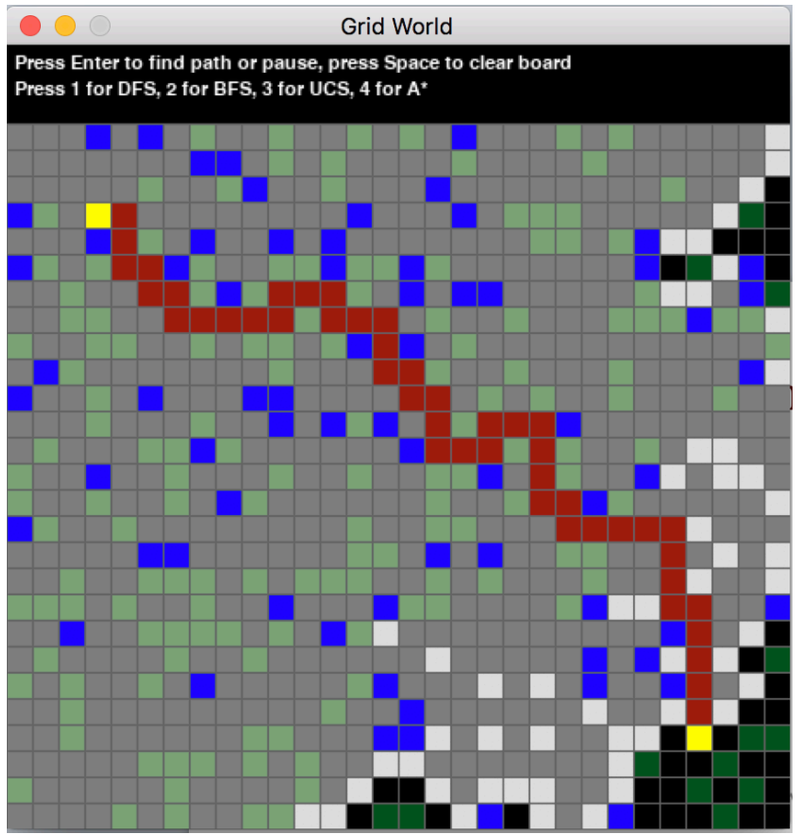
Examples:

$$(p_1 \to p_2) \to (\neg p_2 \to \neg p_1)$$

$$p_1 \to (p_2 \to p_3) \to ((p_1 \to p_2) \to (p_1 \to p_3))$$

Example: How to define trajectories?

$$\mathsf{start}(x_0) \wedge \bigwedge_{i=0}^{k} \bigvee_{j=0}^{m_i} \mathsf{next}(x_i, x_{i+1}) \wedge \mathsf{goal}(x_{k+1})$$

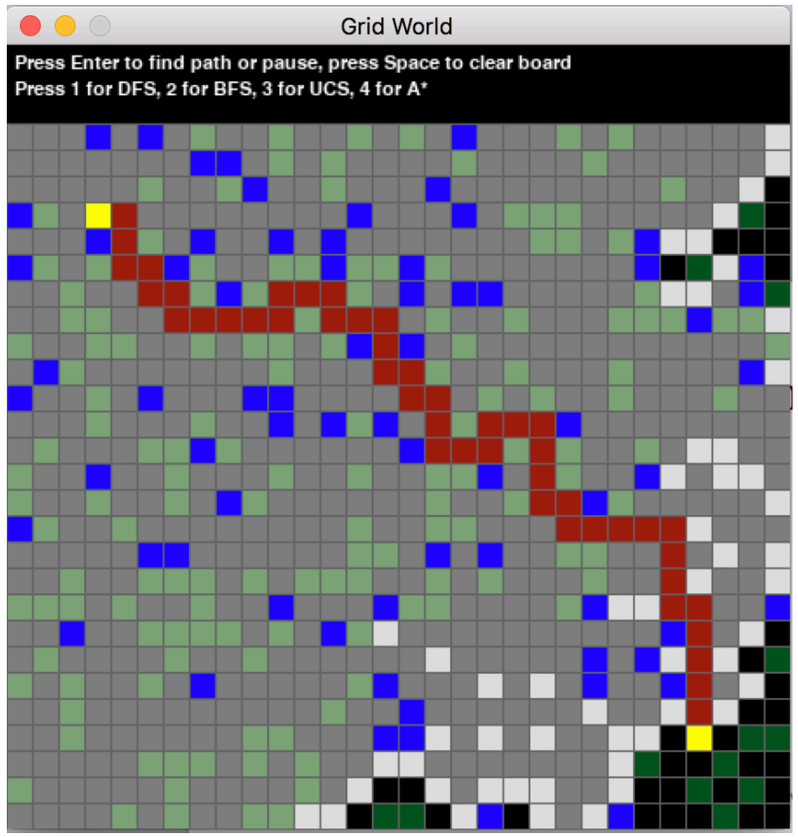


Each x is a vector of binary variables.

Each "predicate" is a formula over these variables.

Example: How to define trajectories?

$$\mathsf{start}(x_0) \wedge \bigwedge_{i=0}^{k} \bigvee_{j=0}^{m_i} \mathsf{next}(x_i, x_{i+1}) \wedge \mathsf{goal}(x_{k+1})$$



Each x is a vector of binary variables.

Each "predicate" is a formula over these variables.

Size of formula grows linearly in the number of dimensions.

Semantics $\sigma : \Phi \to \{0, 1\}$

Variable assignments:

$$\text{For any } p \in P, \sigma(p) \in \{0, 1\}$$

Composition rules:

$$\sigma(\neg\phi) := 1 - \sigma(\phi)$$

$$\sigma(\phi_1 \wedge \phi_2) := \sigma(\phi_1) \cdot \sigma(\phi_2)$$

Semantics $\sigma : \Phi \to \{0, 1\}$

Disjunction?

$$\sigma(\phi_1 \vee \phi_2) = \sigma(\neg(\neg\phi_1 \wedge \neg\phi_2))$$
$$= 1 - (1 - \sigma(\phi_1))(1 - \sigma(\phi_2))$$

Semantics $\sigma : \Phi \to \{0, 1\}$

Disjunction?

$$\sigma(\phi_1 \vee \phi_2) = \sigma(\neg(\neg\phi_1 \wedge \neg\phi_2))$$
$$= 1 - (1 - \sigma(\phi_1))(1 - \sigma(\phi_2))$$

When $\sigma(\phi_1) = 1, \sigma(\phi_1 \vee \phi_2) = 1$

When $\sigma(\phi_2) = 1, \sigma(\phi_1 \vee \phi_2) = 1$

When $\sigma(\phi_1) = 0$ and $\sigma(\phi_2) = 0, \sigma(\phi_1 \vee \phi_2) = 0$

Semantics $\sigma : \Phi \to \{0, 1\}$

Disjunction?

$$\sigma(\phi_1 \vee \phi_2) = \sigma(\neg(\neg\phi_1 \wedge \neg\phi_2))$$
$$= 1 - (1 - \sigma(\phi_1))(1 - \sigma(\phi_2))$$

Implication?

$$\sigma(\phi_1 \to \phi_2) = \sigma(\neg\phi_1 \vee \phi_2)$$

Satisfiability

Given any propositional logic formula

$$\phi \in \Phi$$

We say

$$\phi \text{ is satisfiable}$$

if there exists an assignment function

$$\sigma \text{ such that } \sigma(\phi) = 1$$

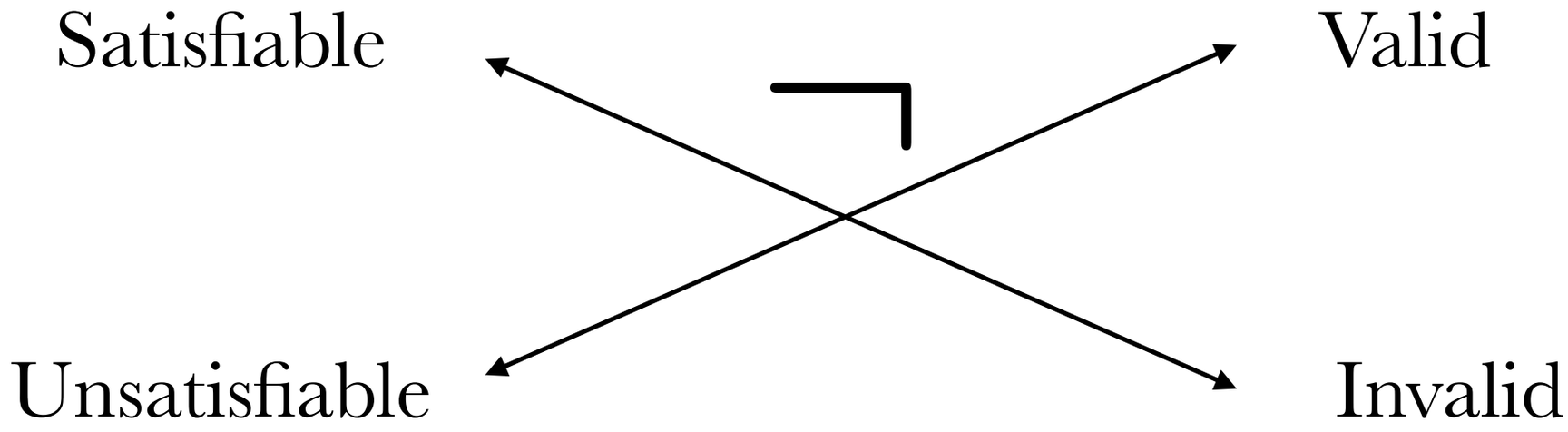
Satisfiability vs Validity

We say a logic formula is valid if it is true under any assignment.

A logic formula is valid if and only if its negation is unsatisfiable.

Satisfiability vs Validity

A logic formula is valid if and only if its negation is unsatisfiable.



A theorem is a valid logic formula.

THE SAT PROBLEM

Design an algorithm **A**, such that given any propositional logic formula

$$\phi \in \Phi$$

A correctly returns whether

$$\phi \text{ is satisfiable or not.}$$

SAT: Getting Started

Basic idea is simple:

- Write down formulas as some constraint system.
- Each constraint enforces some consistency
- Make decisions, do propagation, backtrack, etc.

SAT: Getting Started

Basic idea is simple:

- Write down formulas as some constraint system.
- Each constraint enforces some consistency
- Make decisions, do propagation, backtrack, etc.

SAT diverged from generic constraint solving because all these tasks can be done in very efficient ways.

Key definition: CNF

Conjunctive normal form:

$$(p_1 \vee p_2 \vee p_3)$$
$$\wedge(p_3 \vee \neg p_1)$$
$$\wedge(p_1 \vee \neg p_2 \vee p_4)$$

Key definition: CNF

Conjunctive normal form:

$$(p_1 \vee p_2 \vee p_3)$$
$$\wedge(p_3 \vee \neg p_1)$$
$$\wedge(p_1 \vee \neg p_2 \vee p_4)$$

Constraints!

Key definition: CNF

The conjunctive normal form allows us to think about SAT from a constraint solving perspective.

- Variables: $P = \{p_1, ..., p_n\}$
- Domains: $D = \{0, 1\}^n$
- Constraints: $C = \{c_1, ..., c_m\}$

Each constraint is a small disjunction.

Key definition: CNF

Every propositional logic formula can be turned in to a CNF in linear time, with additional variables.

- Variables: $P = \{p_1, ..., p_n\}$
- Domains: $D = \{0, 1\}^n$
- Constraints: $C = \{c_1, ..., c_m\}$

Each constraint is a small disjunction.

Putting formulas in CNF (Tseitin's Method)

$$\phi := ((p \vee q) \wedge r) \rightarrow (\neg s)$$

Introduce new variables for each subformula

$$x_1 \leftrightarrow \neg s$$

$$x_2 \leftrightarrow p \vee q$$

$$x_3 \leftrightarrow x_2 \wedge r$$

$$x_4 \leftrightarrow x_3 \rightarrow x_1$$

Putting formulas in CNF

$$\phi := ((p \vee q) \wedge r) \rightarrow (\neg s)$$

$$x_1 \leftrightarrow \neg s$$
$$x_2 \leftrightarrow p \vee q$$
$$x_3 \leftrightarrow x_2 \wedge r$$
$$x_4 \leftrightarrow x_3 \rightarrow x_1$$

Rewrite the original formula into a big conjunction

$$x_4 \wedge (x_4 \leftrightarrow x_3 \rightarrow x_1) \wedge (x_3 \leftrightarrow x_2 \wedge r) \wedge (x_2 \leftrightarrow p \vee q) \wedge (x_1 \leftrightarrow \neg s)$$

Each of these conjunctive clauses can be CNF

$$x_2 \leftrightarrow p \vee q \equiv (\neg x_2 \vee p \vee q) \wedge (\neg p \vee x_2) \wedge (\neg q \vee x_2)$$

Key definition: CNF

Dimacs Format

```
p cnf 59056 323700
1 2 0
1 3 0
1 4 0
1 -5 0
1 6 0
1 -7 0
1 -8 0
1 -9 0
1 -10 0
-2 -3 -4 5 -6 7 8 9 10 -1 0
-11 -12 -13 14 0
-14 11 0
-14 12 0
-14 13 0
```

CNF significantly simplifies representation of SAT problems, and it is key to many algorithmic ideas.

Disjunctive Normal Form (DNF)

Dual of CNF: disjunctions of conjunctions

$$(p_1 \wedge p_2 \wedge p_3)$$
$$\vee (p_3 \wedge \neg p_1)$$
$$\vee (p_1 \wedge \neg p_2 \wedge p_4)$$

CNF vs. DNF

SAT is NP-complete if the input formulas are in CNF.

SAT is in linear time if the input formulas are in DNF!

Disjunctive Normal Form (DNF)

Why linear time?
Every disjunct gives you a satisfying assignment.

$$(p_1 \wedge p_2 \wedge p_3)$$

$$\vee (p_3 \wedge \neg p_1)$$

$$\vee (p_1 \wedge \neg p_2 \wedge p_4)$$

If you have the DNF, you actually know all the solutions.

CNF vs. DNF

SAT is NP-complete if the input formulas are in CNF.

SAT is in linear time if the input formulas are in DNF!

Naturally, if P is not NP, there does not exist any procedure that puts propositional formulas into an equivalent DNF in polynomial time.

Basis of Modern SAT Solving

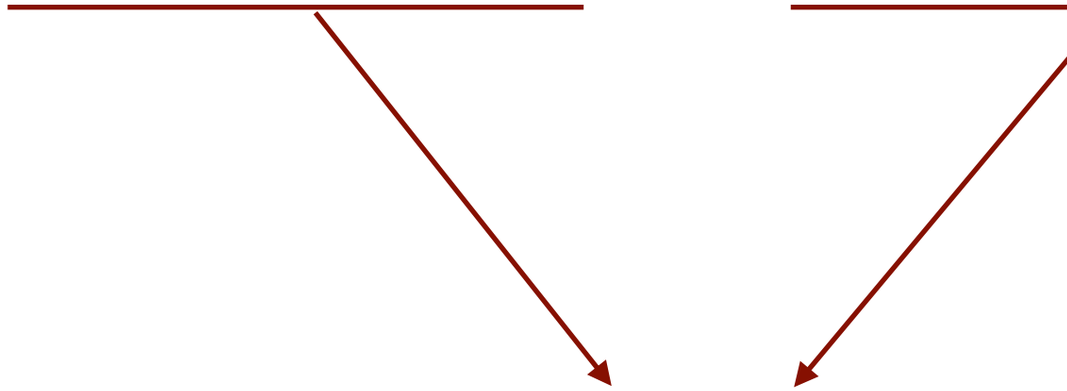
The basic algorithm is named DPLL.

It is just a constraint solving loop, specially designed for logical rules.

- make decisions
- propagate based on consistency
- backtrack when needed

DPLL: Terminology

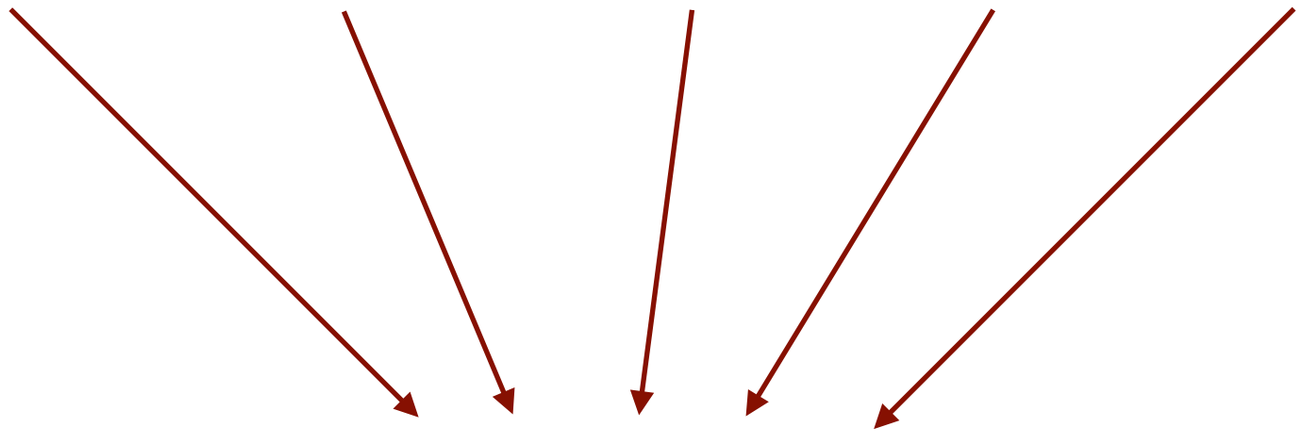
$$(p_1 \vee \neg p_2) \wedge (p_3 \vee \neg p_1 \vee p_2) \wedge \cdots$$



Clauses

DPLL: Terminology

$$(p_1 \vee \neg p_2) \wedge (p_3 \vee \neg p_1 \vee p_2) \wedge \cdots$$



Literals

DPLL: Terminology

$$(p_1 \vee \boxed{\neg p_2}) \wedge (p_3 \vee \neg p_1 \vee \boxed{p_2}) \wedge \cdots$$

Negative literal Positive literal

Literals

DPLL: Terminology

$$(p_1 \vee \boxed{\neg p_2}) \wedge (p_3 \vee \neg p_1 \vee \boxed{p_2}) \wedge \cdots$$

$$\{p_1, \neg p_2\}, \{p_3, \neg p_1, p_2\}, \cdots$$

DPLL: Terminology

$$(p_1 \vee \boxed{\neg p_2}) \wedge (p_3 \vee \neg p_1 \vee \boxed{p_2}) \wedge \cdots$$

$$\{p_1, \neg p_2\}, \{p_3, \neg p_1, p_2\}, \cdots$$

A formula is just a set of clauses.
A clauses is just a set of literals.

DPLL: Getting Started

$$\{p_1, \neg p_2\}, \{p_3, \neg p_1, p_2\}$$

Pure literal rule:

If a variable only occur in one polarity in all clauses, then eliminate the variable and all the clauses that contain it.

DPLL: Getting Started

$$\{p_1, \neg p_2\}, \{p_3, \neg p_1, p_2\}$$

Pure literal rule:

If a variable only occur in one polarity in all clauses, then eliminate the variable and all the clauses that contain it.

DPLL: Getting Started

Pure literal rule:

If a variable only occur in one polarity in all clauses, then eliminate the variable and all the clauses that contain it.

Why?

DPLL: Getting Started

Pure literal rule:

If a variable only occur in one polarity in all clauses, then eliminate the variable and all the clauses that contain it.

If a variable appear in one polarity, then we can set the corresponding literal to 1, and all the clauses that contain it are satisfied.

DPLL: Getting Started

Singleton rule:

If a clause only contains one literal, the corresponding variable has to be assigned a value that makes the literal true.

DPLL: Getting Started

Singleton rule:

If a clause only contains one literal, the corresponding variable has to be assigned a value that makes the literal true.

Why?

DPLL: High-Level Picture

Take a formula in CNF as input:

- Simplify using pure literal and singletons, and initialize the assignment function accordingly.
- Then, loop between the following steps:
  - (Decision) Pick an unassigned variable and pick a value. Push the decision to a stack.
  - (Unit Propagation) Propagate the decision to other variables. If all variables are assigned, return.
  - (Backtracking) If a conflict is found, backtrack and invert a previous decision on the stack. If the stack becomes empty, return.

DPLL: High-Level Picture

Take a formula in CNF as input:

- Simplify using pure literal and singletons, and initialize the assignment function accordingly.
- Then, loop between the following steps:
  - (Decision) Pick an unassigned variable and pick a value. Push the decision to a stack.
  - (Unit Propagation) Propagate the decision to other variables. If all variables are assigned, return.
  - (Backtracking) If a conflict is found, backtrack and invert a previous decision on the stack. If the stack becomes empty, return.

DPLL: Unit Propagation

Unit Clause:

A clause of length *n*, where *n-1* literals have been assigned to false, and only one literal is unassigned.

DPLL: Unit Propagation

Unit Clause:
A clause of length *n*, where *n-1* literals have been assigned to false, and only one literal is unassigned.

$$\{p_3, \neg p_1, p_2\}$$

$$\sigma(p_1) = 1$$

Unit clause?

DPLL: Unit Propagation

Unit Clause:
A clause of length *n*, where *n-1* literals have been assigned to false, and only one literal is unassigned.

$$\{p_3, \neg p_1, p_2\}$$

$$\sigma(p_1) = 1 \quad \sigma(p_2) = 1$$

Unit clause?

DPLL: Unit Propagation

Unit Clause:
A clause of length *n*, where *n-1* literals have been assigned to false, and only one literal is unassigned.

$$\{p_3, \neg p_1, p_2\}$$

$$\sigma(p_1) = 1 \quad \sigma(p_2) = 0$$

Unit clause?

DPLL: Unit Propagation

Unit Propagation:

Take a unit clause, and simply set the unassigned literal to true.

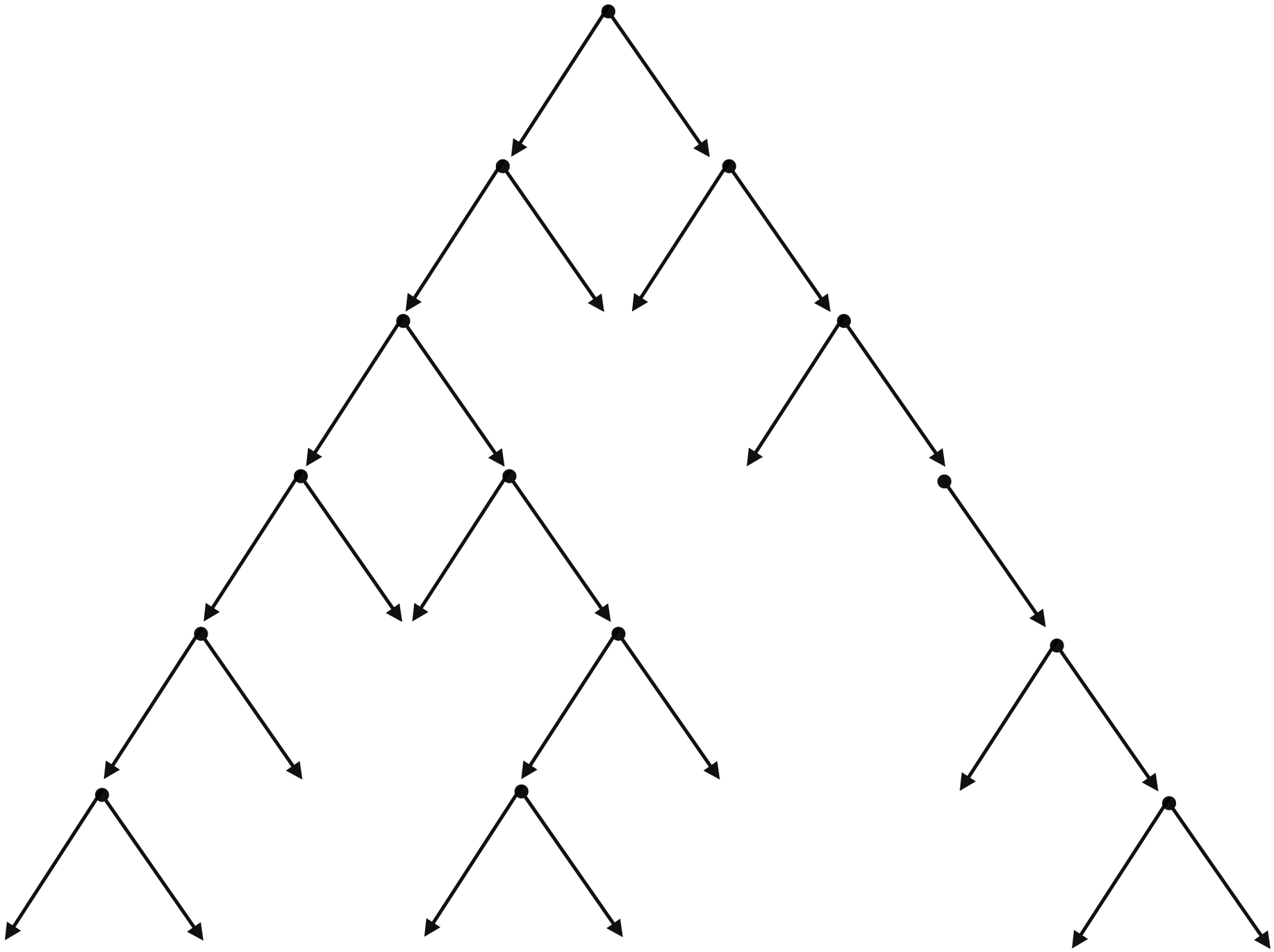
DPLL: Unit Propagation

Unit Propagation:
Take a unit clause, and simply set the unassigned literal to true.

$$\{p_3, \neg p_1, p_2\}$$

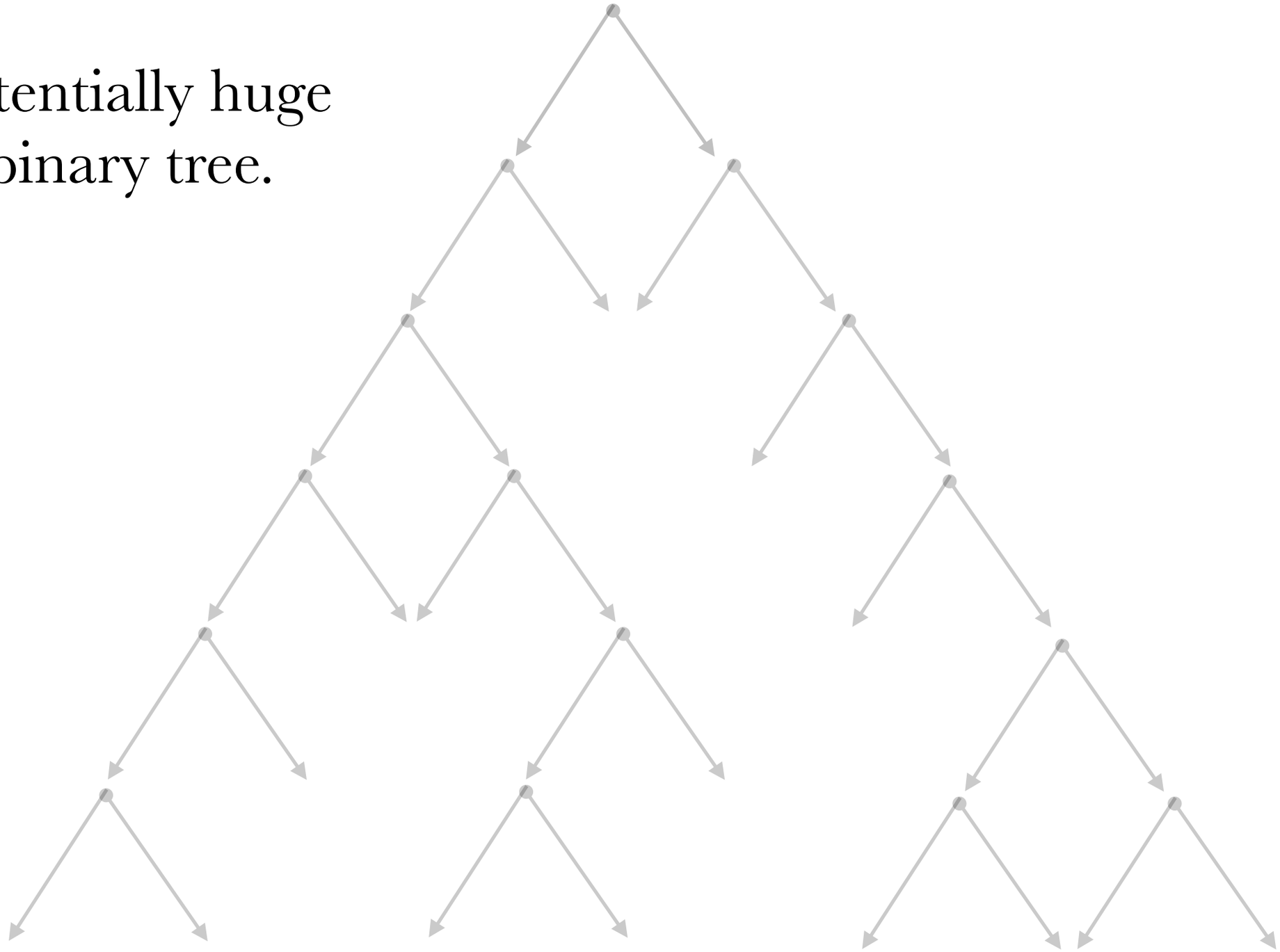
$$\sigma(p_1) = 1 \quad \sigma(p_2) = 0$$

$$\sigma(p_3) = 1$$



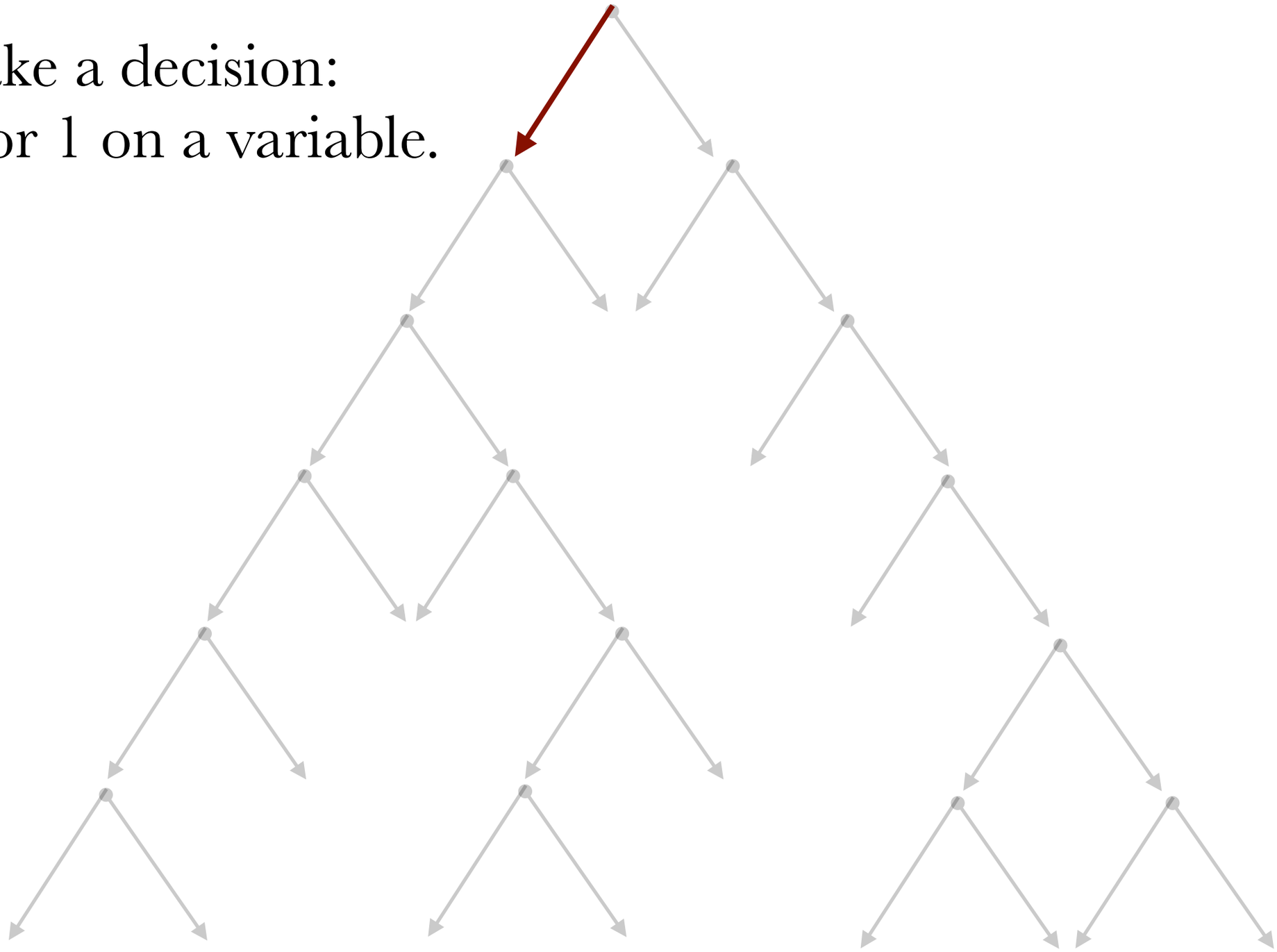
DPLL: Full Algorithm

Potentially huge
binary tree.



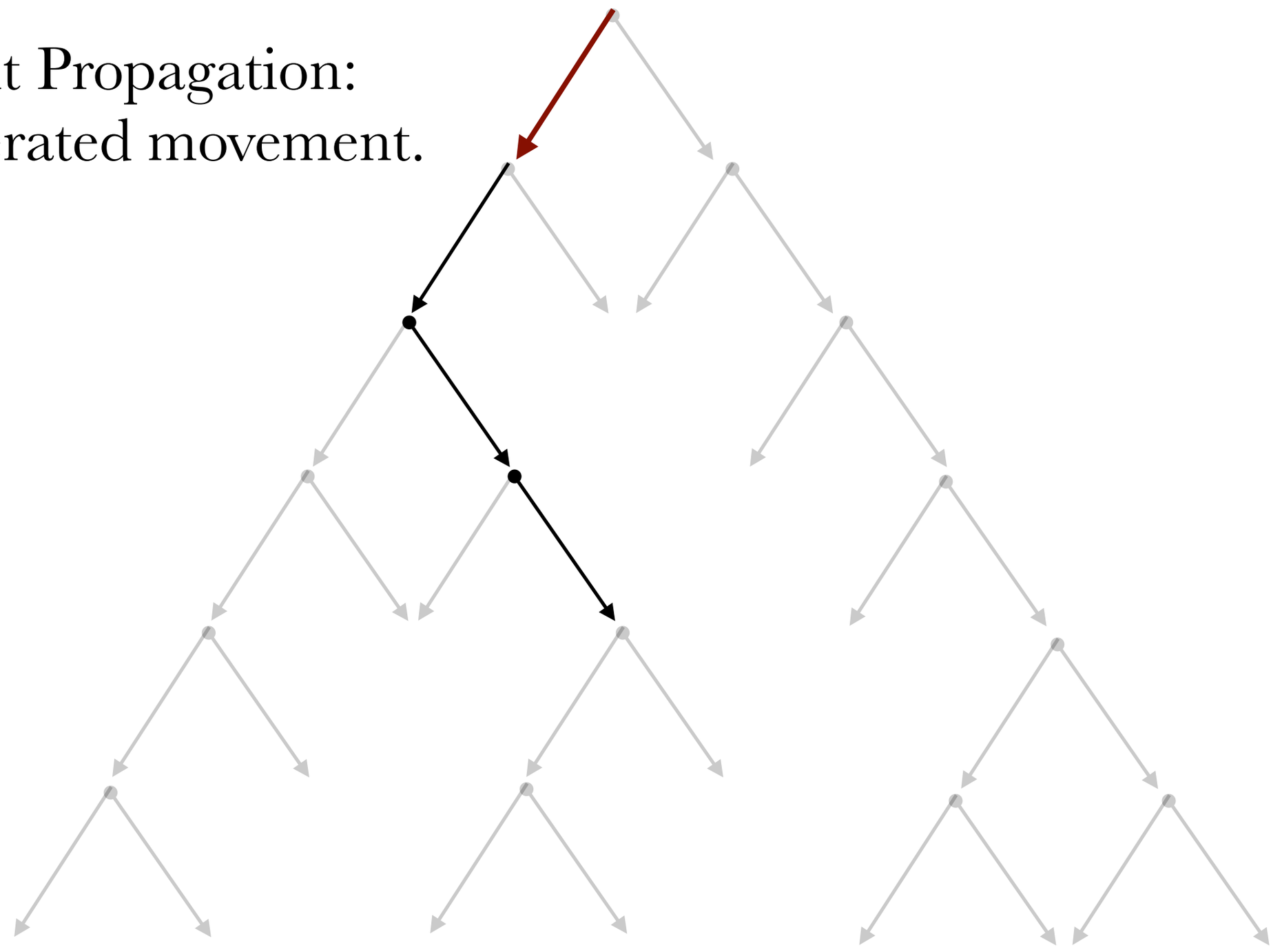
DPLL: Full Algorithm

Make a decision:
Pick 0 or 1 on a variable.



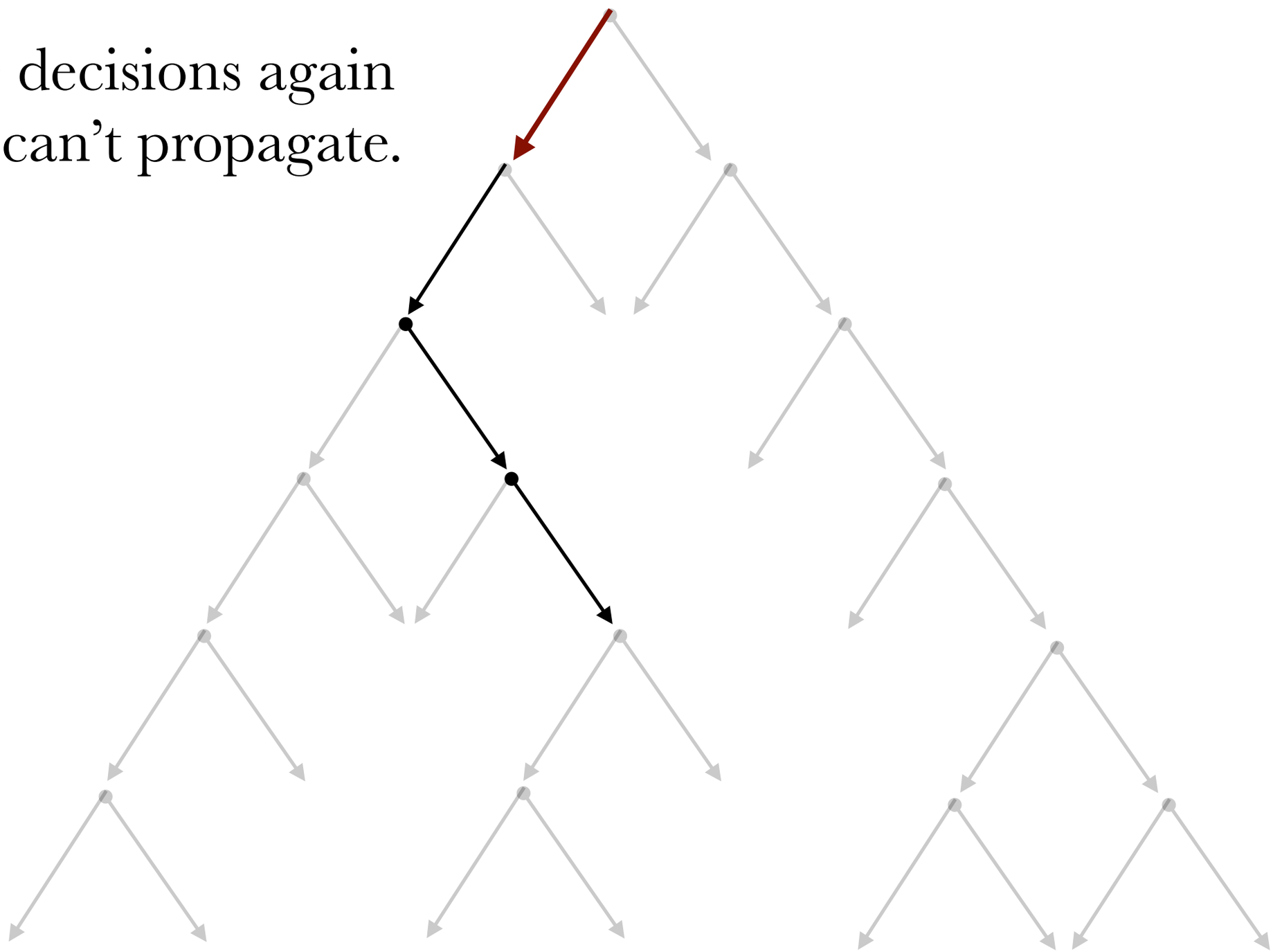
DPLL: Full Algorithm

Unit Propagation:
Accelerated movement.



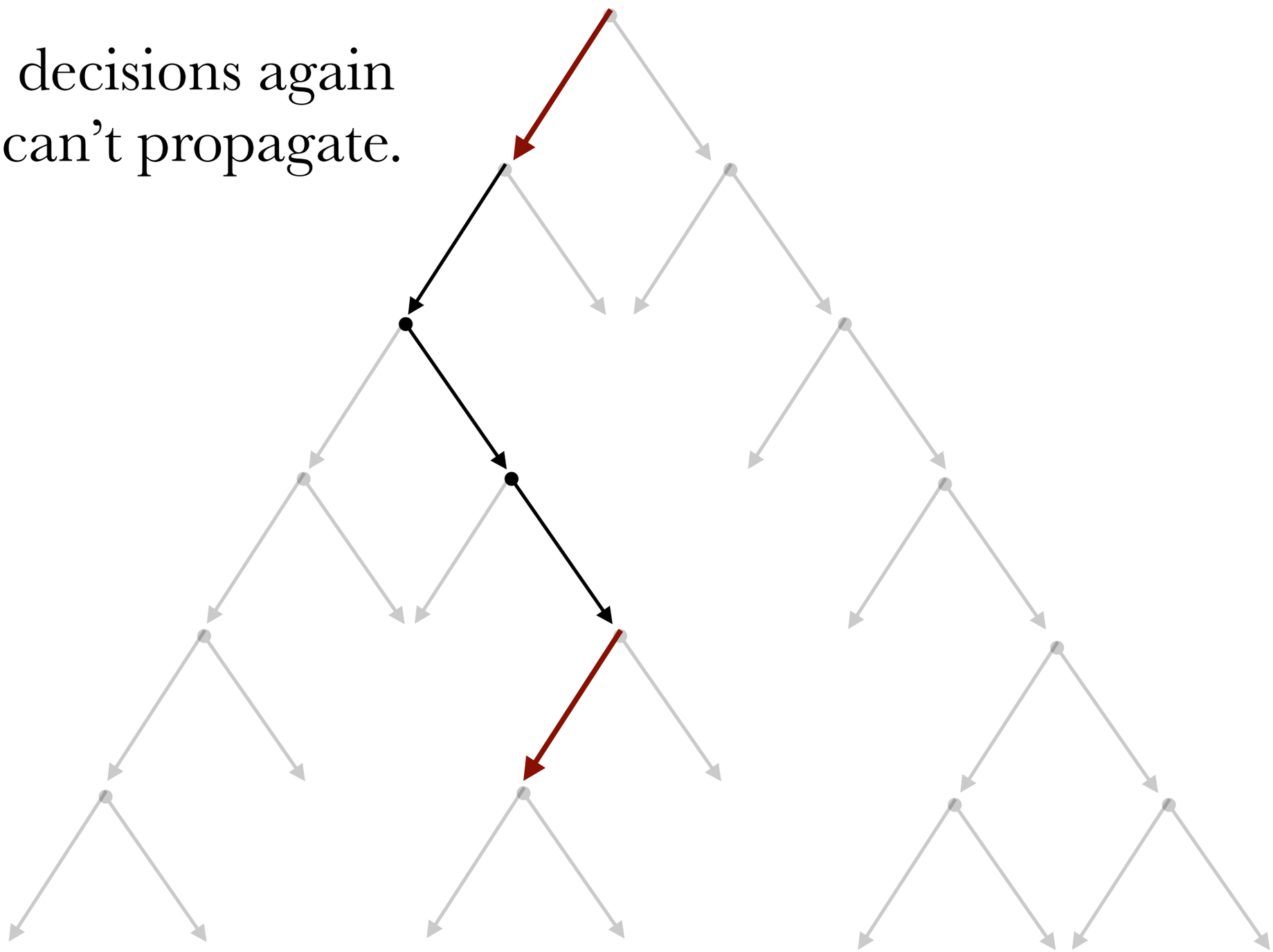
DPLL: Full Algorithm

Make decisions again
when can't propagate.



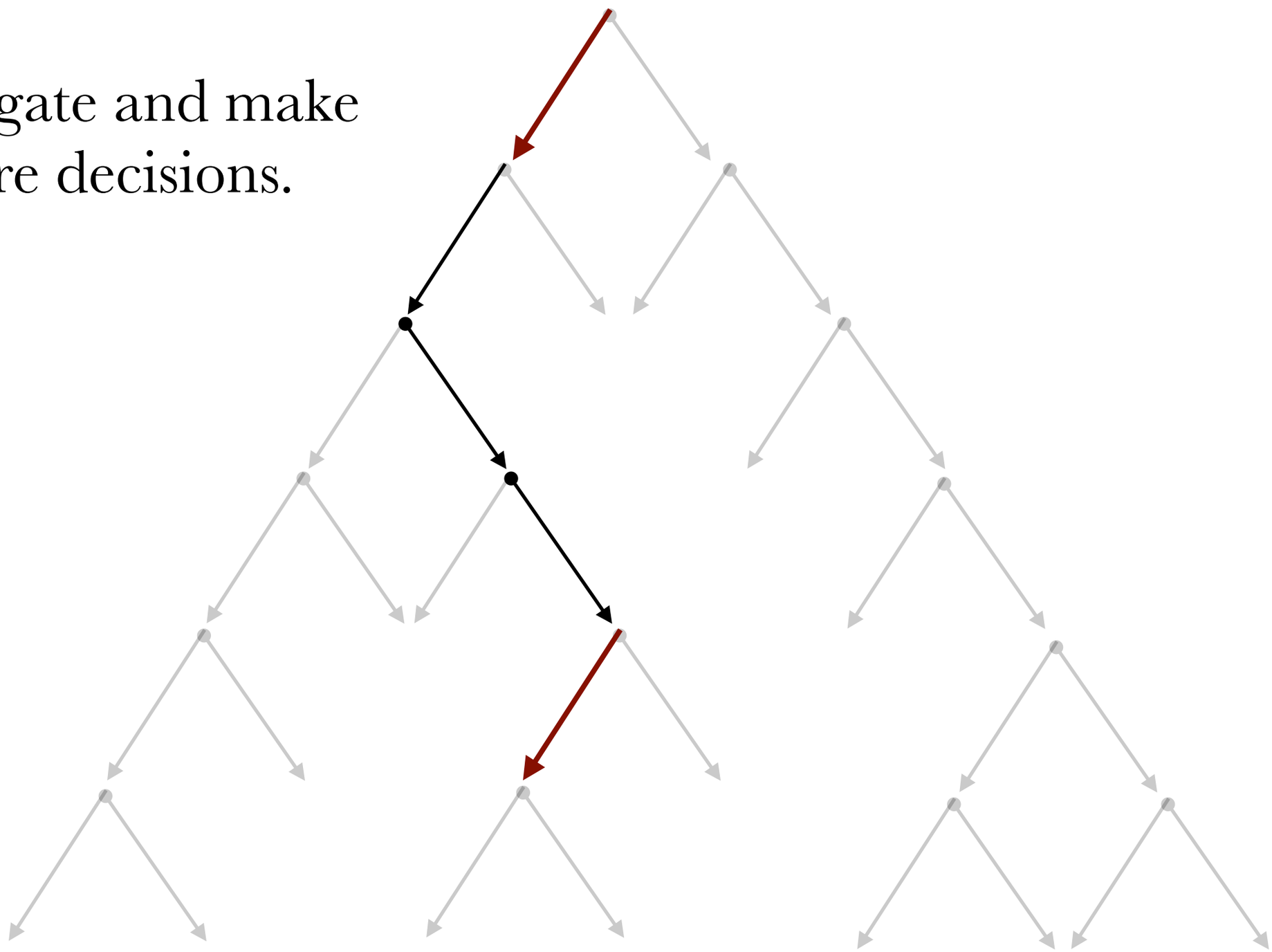
DPLL: Full Algorithm

Make decisions again
when can't propagate.



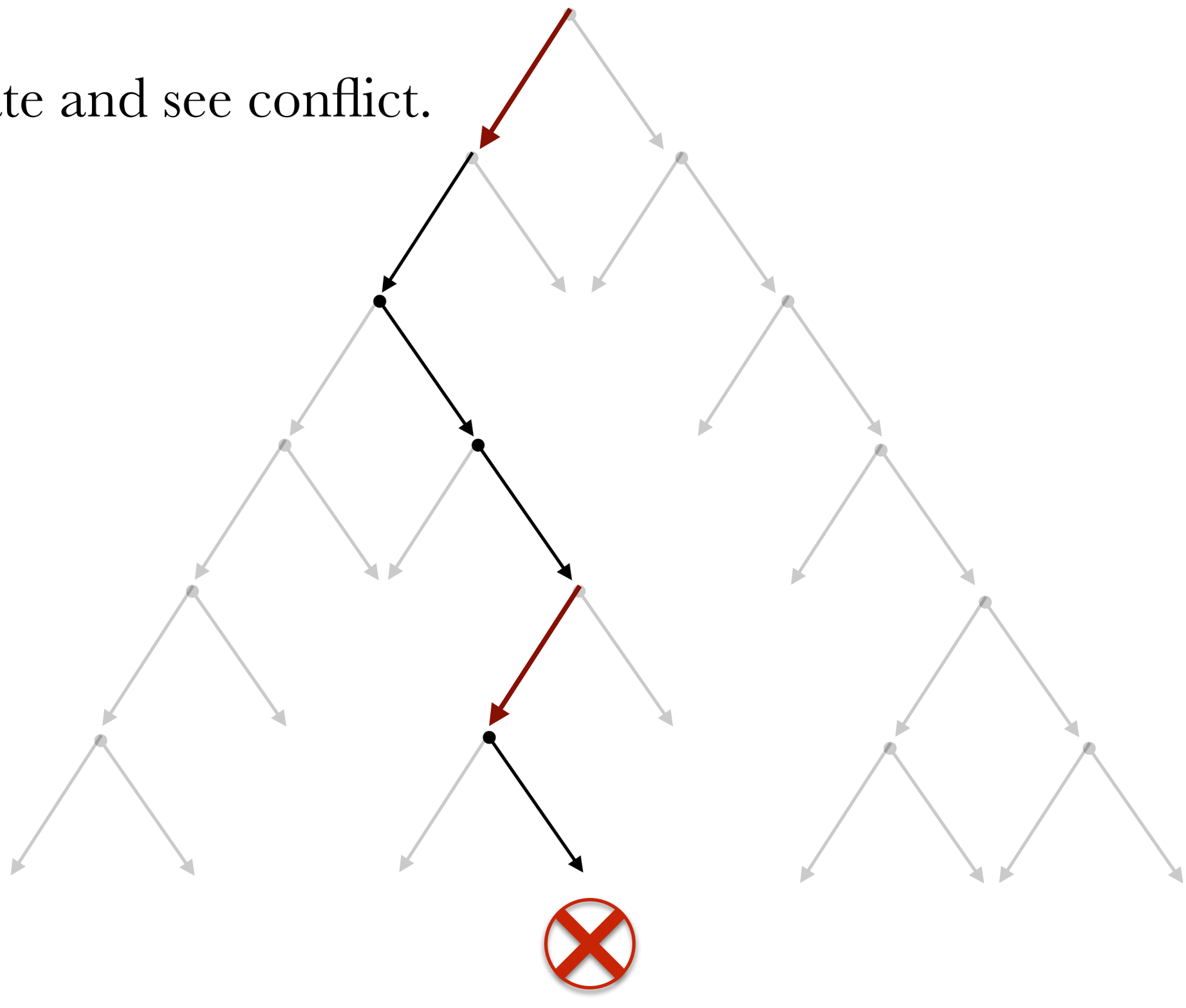
DPLL: Full Algorithm

Propagate and make more decisions.



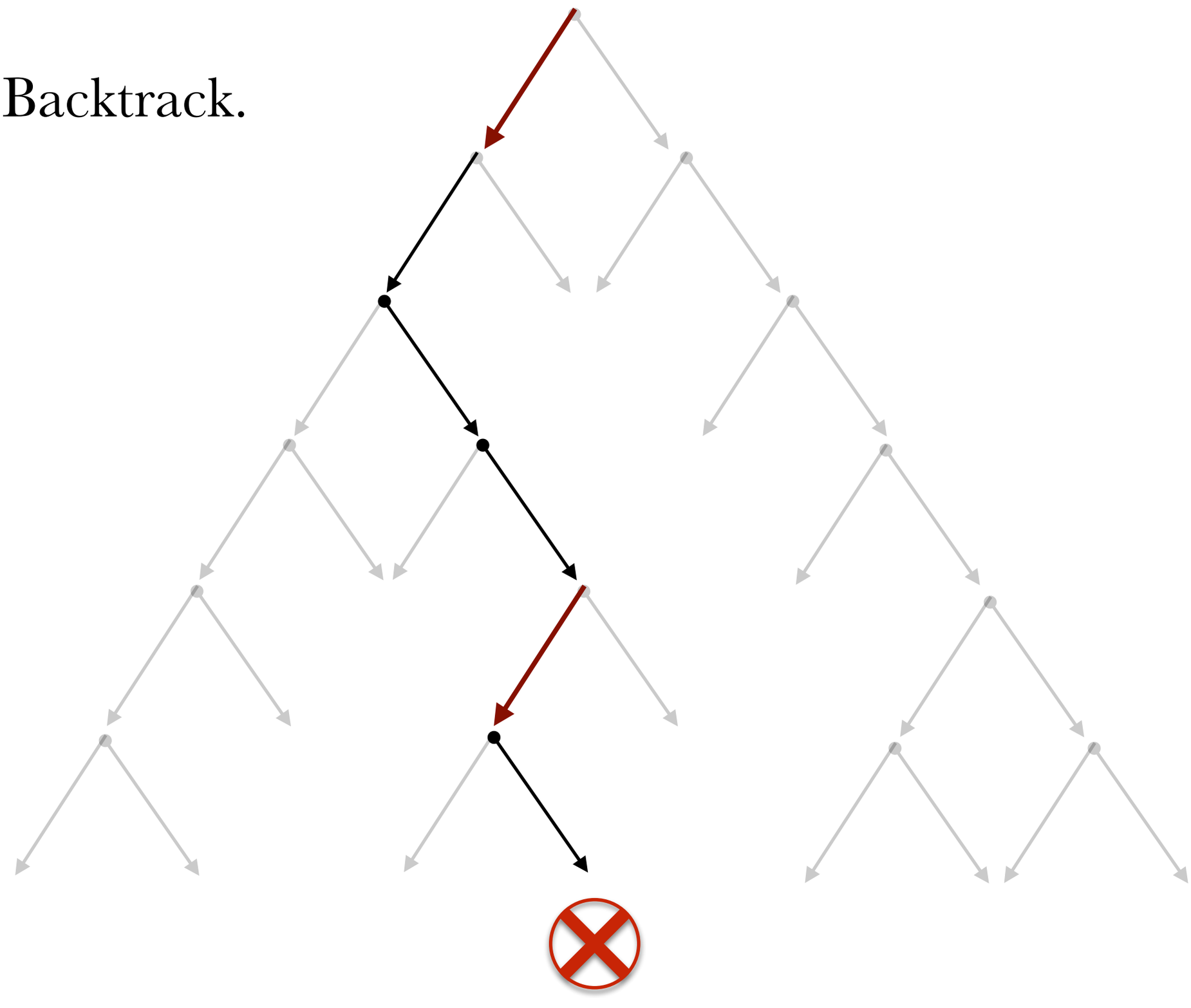
DPLL: Full Algorithm

Propagate and see conflict.



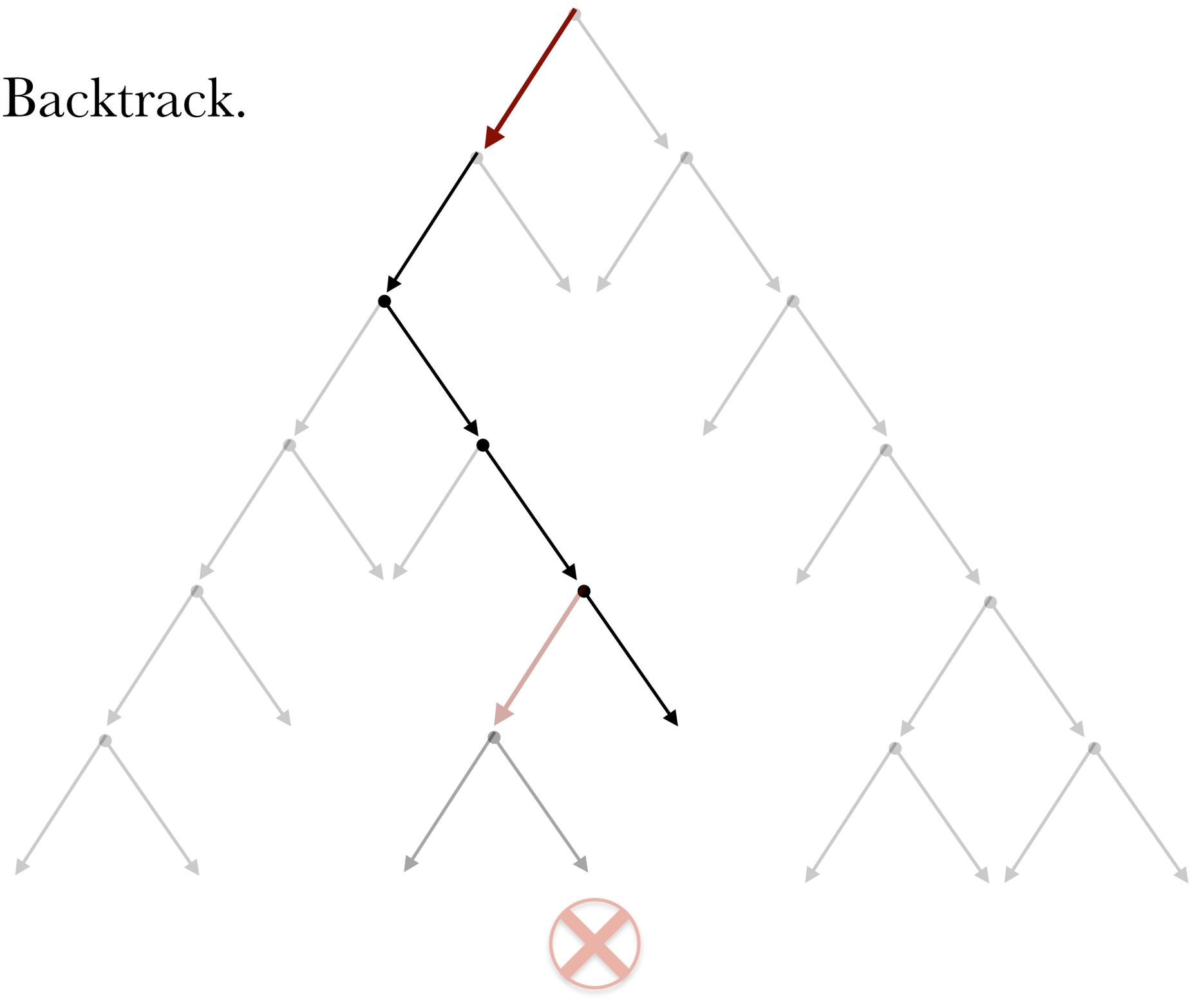
DPLL: Full Algorithm

Backtrack.



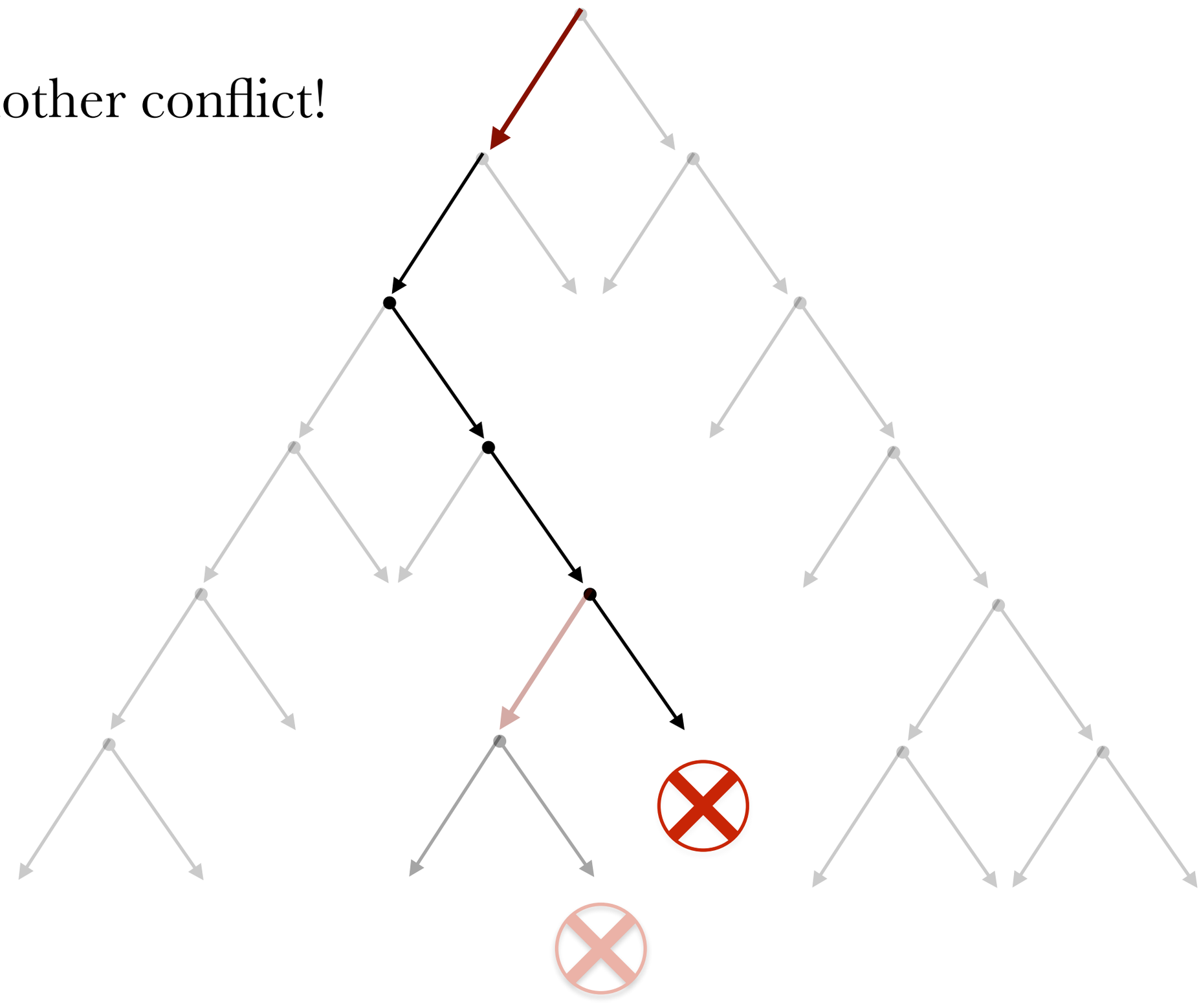
DPLL: Full Algorithm

Backtrack.



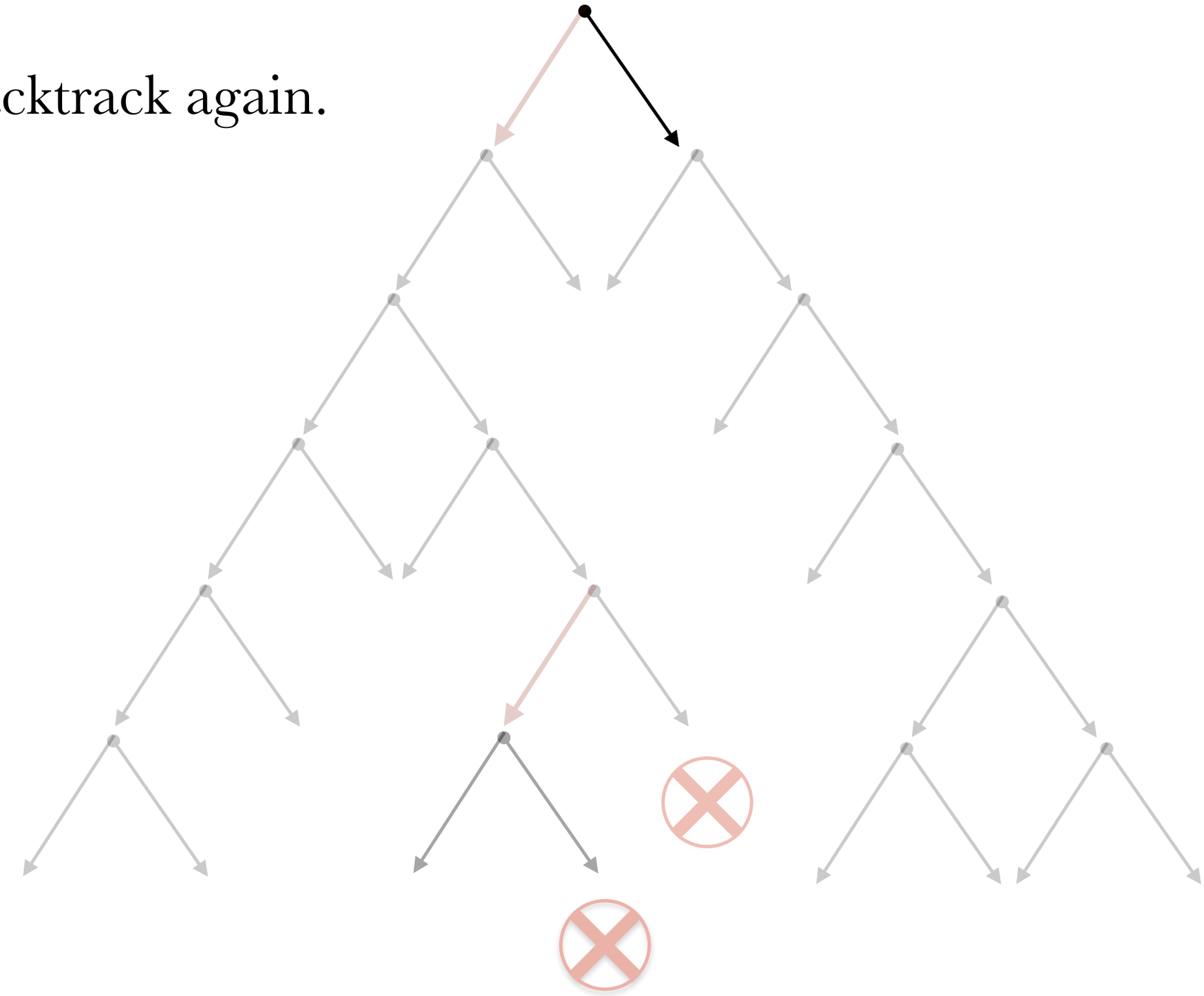
DPLL: Full Algorithm

Another conflict!



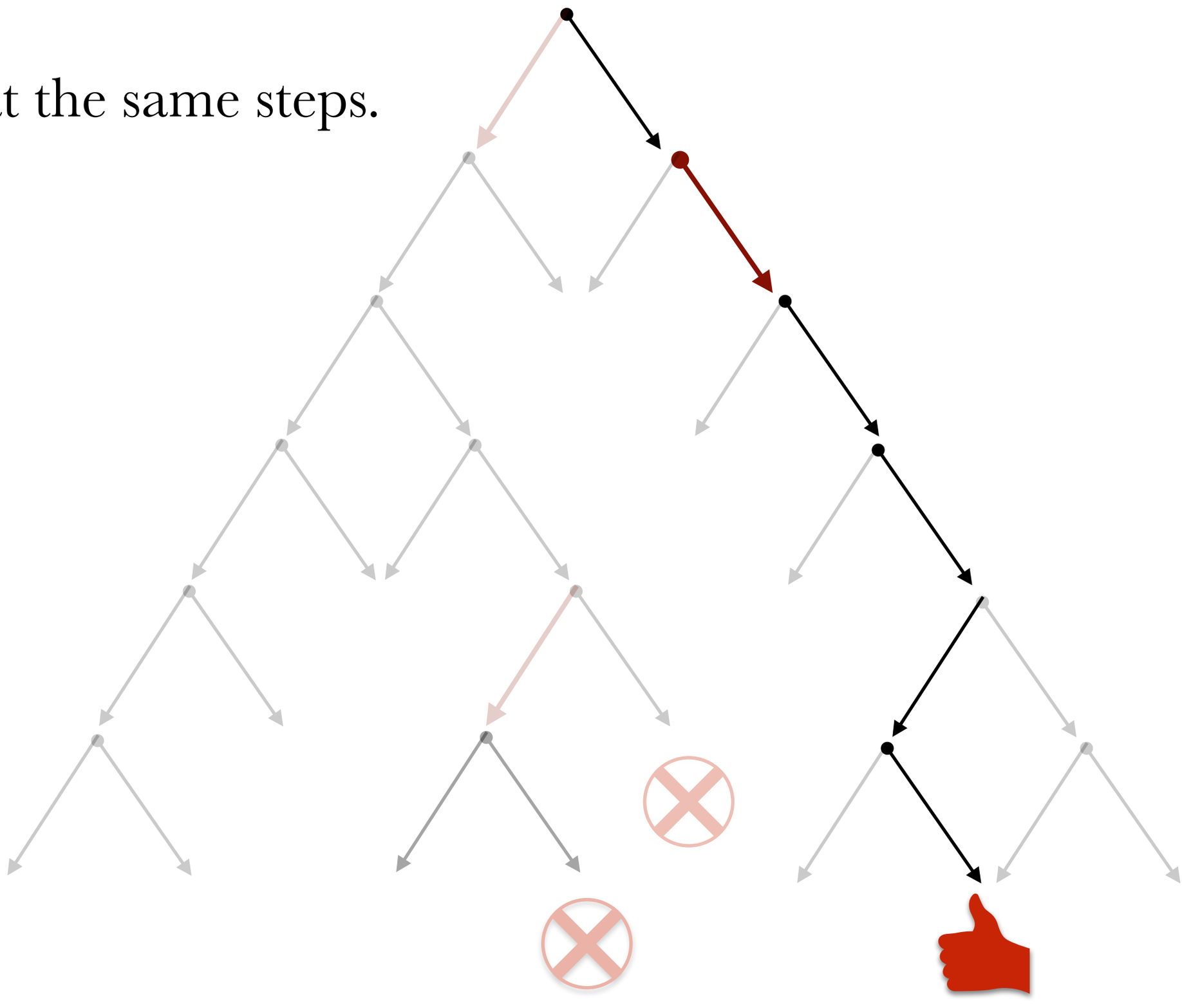
DPLL: Full Algorithm

Backtrack again.



DPLL: Full Algorithm

Repeat the same steps.



DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

What do we do first?

- Pure literals? $\neg p_5$
- Singletons? $\{\neg p_4\}$

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

After applying the rules:

- Pure literals? Eliminated.

- Singletons? Assigned.

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

After applying the rules:

- Pure literals? Eliminated.

- Singletons? Assigned.

$$\sigma(p_4) = 0$$

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\sigma(p_4) = 0$

What's next?

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\sigma(p_4) = 0$

What's next?

No unit clause to use.

Make a decision.

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\sigma(p_4) = 0$

What's next?

Make a decision.

$\sigma(p_1) = 0$

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\sigma(p_4) = 0$

What's next?

Make a decision.

$\sigma(p_1) = 0$

Also, record the decision.

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

Assignment so far:

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

Decision stack:

$$\Delta = \{\neg p_1\}$$

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\Delta = \{\neg p_1\}$

$\sigma(p_4) = 0$

$\sigma(p_1) = 0$

Now: Propagate!

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

Now: Propagate!

Unit Clauses?

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

Now: Propagate!

Unit Clauses?

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

Now: Propagate!

$$\sigma(p_2) = 0$$

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, \neg p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\Delta = \{\neg p_1\}$

$\sigma(p_4) = 0$

$\sigma(p_1) = 0$

Now: Propagate!

$\sigma(p_2) = 0$

$\sigma(p_3) = 0$

DPLL: Example

$$\{p_1, \neg p_2\}$$

$$\{p_3, \neg p_1, p_2\}$$

$$\{\neg p_3, p_2, p_4\}$$

$$\{\neg p_4\}$$

$$\{\neg p_5\}$$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

$$\sigma(p_2) = 0$$

$$\sigma(p_3) = 0$$

Everything variable gets assigned.
No conflicts. Done.

DPLL: Example

$$\{p_1, \neg p_2\}$$

$$\{p_3, p_1, p_2\}$$

$$\{\neg p_3, p_2, p_4\}$$

$$\{\neg p_4\}$$

$$\{\neg p_5\}$$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

Let's change one clause.

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

Unit Clauses?

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

Propagate!

$$\sigma(p_2) = 0$$

DPLL: Example

$$\{p_1, \neg p_2\}$$

$$\boxed{\{p_3, p_1, p_2\}}$$

$$\{\neg p_3, p_2, p_4\}$$

$$\{\neg p_4\}$$

$$\{\neg p_5\}$$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

$$\sigma(p_2) = 0$$

Now: Propagate!

$$\sigma(p_3) = 1$$

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\Delta = \{\neg p_1\}$

$\sigma(p_4) = 0$

$\sigma(p_1) = 0$

$\sigma(p_2) = 0$

$\sigma(p_3) = 1$

Conflict!

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

$$\sigma(p_2) = 0$$

$$\sigma(p_3) = 1$$

The latest propagation sequence

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 0$$

$$\sigma(p_2) = 0$$

$$\sigma(p_3) = 1$$

We now have to flip the last decision.

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\Delta = \{\neg p_1\}$

$\sigma(p_4) = 0$

$\sigma(p_1) = 0$

We now have to flip the last decision.

DPLL: Example

$$\{p_1, \neg p_2\}$$

$$\{p_3, p_1, p_2\}$$

$$\{\neg p_3, p_2, p_4\}$$

$$\{\neg p_4\}$$

$$\{\neg p_5\}$$

$$\Delta = \{\neg p_1\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 1$$

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\Delta = \{\}$

$\sigma(p_4) = 0$

$\sigma(p_1) = 1$

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\Delta = \{\}$

$\sigma(p_4) = 0$

$\sigma(p_1) = 1$

Need to make a new decision.

$\sigma(p_2) = 0$

DPLL: Example

$$\{p_1, \neg p_2\}$$

$$\{p_3, p_1, p_2\}$$

$$\{\neg p_3, p_2, p_4\}$$

$$\{\neg p_4\}$$

$$\{\neg p_5\}$$

$$\Delta = \{\neg p_2\}$$

$$\sigma(p_4) = 0$$

$$\sigma(p_1) = 1$$

$$\sigma(p_2) = 0$$

Propagate:

$$\sigma(p_3) = 0$$

DPLL: Example

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2, p_4\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$\Delta = \{\neg p_2\}$

$\sigma(p_4) = 0$

$\sigma(p_1) = 1$

$\sigma(p_2) = 0$

$\sigma(p_3) = 0$

Every variable gets assigned.
No conflicts. Done.

DPLL: Full Algorithm

Algorithm 2 DPLL Search

```
1:  function DPLL(φ)
2:      σ ← ∅                                          ▷ Assignment mapping
3:      Δ ← ∅                                          ▷ Decision variable stack
4:      φ ← PureLiteralElimination(φ)
5:      Γ ←Singletons(φ)                               ▷ Singleton clauses are the initial active clauses
6:      while true do
7:          σ, Γ ←UnitPropagate(φ, σ, Γ)
8:          if ⊥ ∉ σ then
9:              if AllAssigned(σ) then
10:                 return SATISFIABLE
11:             else
12:                 ⟨p, b⟩ ← MakeDecision(φ, σ)
13:                 Δ ← Δ ∪ {⟨p, b⟩}
14:                 σ ← σ ∪ {⟨p, b⟩}
15:                 Γ ← Γ ∪ {γ(p)}
16:             end if
17:         else
18:             if Δ == ∅ then
19:                 return UNSATISFIABLE
20:             else
21:                 ⟨p, b⟩ ← Δ.pop_back()
22:                 σ ← Backtrack(p, b)
23:                 σ ← σ ∪ {⟨p, ¬b⟩}
24:                 Γ ← Γ ∪ {γ(p)}
25:             end if
26:         end if
27:     end while
28: end function
```

DPLL: Full Algorithm

```
6:  while true do
7:      σ, Γ ←UnitPropagate(φ, σ, Γ)
8:      if ⊥ ∉ σ then
9:          if AllAssigned(σ) then
10:             return SATISFIABLE
11:         else
12:             ⟨p, b⟩ ← MakeDecision(φ, σ)
13:             Δ ← Δ ∪ {⟨p, b⟩}
14:             σ ← σ ∪ {⟨p, b⟩}
15:             Γ ← Γ ∪ {γ(p)}
16:         end if
17:     else
18:         if Δ == ∅ then
19:             return UNSATISFIABLE
20:         else
21:             ⟨p, b⟩ ← Δ.pop_back()
22:             σ ← Backtrack(p, b)
23:             σ ← σ ∪ {⟨p, ¬b⟩}
24:             Γ ← Γ ∪ {γ(p)}
25:         end if
26:     end if
27: end while
```

Unit Propagation

DPLL: Full Algorithm

```
6:   while true do
7:       σ, Γ ←UnitPropagate(φ, σ, Γ)
8:       if ⊥ ∉ σ then
9:           if AllAssigned(σ) then
10:              return SATISFIABLE
11:          else
12:              ⟨p, b⟩ ← MakeDecision(φ, σ)
13:              Δ ← Δ ∪ {⟨p, b⟩}
14:              σ ← σ ∪ {⟨p, b⟩}
15:              Γ ← Γ ∪ {γ(p)}
16:          end if
17:      else
18:          if Δ == ∅ then
19:              return UNSATISFIABLE
20:          else
21:              ⟨p, b⟩ ← Δ.pop_back()
22:              σ ← Backtrack(p, b)
23:              σ ← σ ∪ {⟨p, ¬b⟩}
24:              Γ ← Γ ∪ {γ(p)}
25:          end if
26:      end if
27:  end while
```

Check for conflicts

DPLL: Full Algorithm

```
6:   while true do
7:       σ, Γ ←UnitPropagate(φ, σ, Γ)
8:       if ⊥ ∉ σ then
9:           if AllAssigned(σ) then
10:              return SATISFIABLE
11:          else
12:              ⟨p, b⟩ ← MakeDecision(φ, σ)
13:              Δ ← Δ ∪ {⟨p, b⟩}
14:              σ ← σ ∪ {⟨p, b⟩}
15:              Γ ← Γ ∪ {γ(p)}
16:          end if
17:      else
18:          if Δ == ∅ then
19:              return UNSATISFIABLE
20:          else
21:              ⟨p, b⟩ ← Δ.pop_back()
22:              σ ← Backtrack(p, b)
23:              σ ← σ ∪ {⟨p, ¬b⟩}
24:              Γ ← Γ ∪ {γ(p)}
25:          end if
26:      end if
27:  end while
```

No conflict:
Make a new decision

DPLL: Full Algorithm

```
6:   while true do
7:       σ, Γ ←UnitPropagate(φ, σ, Γ)
8:       if ⊥ ∉ σ then
9:           if AllAssigned(σ) then
10:              return SATISFIABLE
11:          else
12:              ⟨p, b⟩ ← MakeDecision(φ, σ)
13:              Δ ← Δ ∪ {⟨p, b⟩}
14:              σ ← σ ∪ {⟨p, b⟩}
15:              Γ ← Γ ∪ {γ(p)}
16:          end if
17:      else
18:          if Δ == ∅ then
19:              return UNSATISFIABLE
20:          else
21:              ⟨p, b⟩ ← Δ.pop_back()
22:              σ ← Backtrack(p, b)
23:              σ ← σ ∪ {⟨p, ¬b⟩}
24:              Γ ← Γ ∪ {γ(p)}
25:          end if
26:      end if
27:  end while
```

Put on decision stack

DPLL: Full Algorithm

```
6:   while true do
7:       σ, Γ ←UnitPropagate(φ, σ, Γ)
8:       if ⊥ ∉ σ then
9:           if AllAssigned(σ) then
10:              return SATISFIABLE
11:          else
12:              ⟨p, b⟩ ← MakeDecision(φ, σ)
13:              Δ ← Δ ∪ {⟨p, b⟩}
14:              σ ← σ ∪ {⟨p, b⟩}
15:              Γ ← Γ ∪ {γ(p)}
16:          end if
17:      else
18:          if Δ == ∅ then
19:              return UNSATISFIABLE
20:          else
21:              ⟨p, b⟩ ← Δ.pop_back()
22:              σ ← Backtrack(p, b)
23:              σ ← σ ∪ {⟨p, ¬b⟩}
24:              Γ ← Γ ∪ {γ(p)}
25:          end if
26:      end if
27:  end while
```

Update relevant clauses

DPLL: Full Algorithm

```
6:  while true do
7:      σ, Γ ←UnitPropagate(φ, σ, Γ)
8:      if ⊥ ∉ σ then
9:          if AllAssigned(σ) then
10:             return SATISFIABLE
11:         else
12:             ⟨p, b⟩ ← MakeDecision(φ, σ)
13:             Δ ← Δ ∪ {⟨p, b⟩}
14:             σ ← σ ∪ {⟨p, b⟩}
15:             Γ ← Γ ∪ {γ(p)}
16:         end if
17:     else
18:         if Δ == ∅ then
19:             return UNSATISFIABLE
20:         else
21:             ⟨p, b⟩ ← Δ.pop_back()
22:             σ ← Backtrack(p, b)
23:             σ ← σ ∪ {⟨p, ¬b⟩}
24:             Γ ← Γ ∪ {γ(p)}
25:         end if
26:     end if
27: end while
```

Found conflict

DPLL: Full Algorithm

```
6:   while true do
7:       σ, Γ ←UnitPropagate(φ, σ, Γ)
8:       if ⊥ ∉ σ then
9:           if AllAssigned(σ) then
10:              return SATISFIABLE
11:          else
12:              ⟨p, b⟩ ← MakeDecision(φ, σ)
13:              Δ ← Δ ∪ {⟨p, b⟩}
14:              σ ← σ ∪ {⟨p, b⟩}
15:              Γ ← Γ ∪ {γ(p)}
16:          end if
17:      else
18:          if Δ == ∅ then
19:              return UNSATISFIABLE
20:          else
21:              ⟨p, b⟩ ← Δ.pop_back()
22:              σ ← Backtrack(p, b)
23:              σ ← σ ∪ {⟨p, ¬b⟩}
24:              Γ ← Γ ∪ {γ(p)}
25:          end if
26:      end if
27:  end while
```

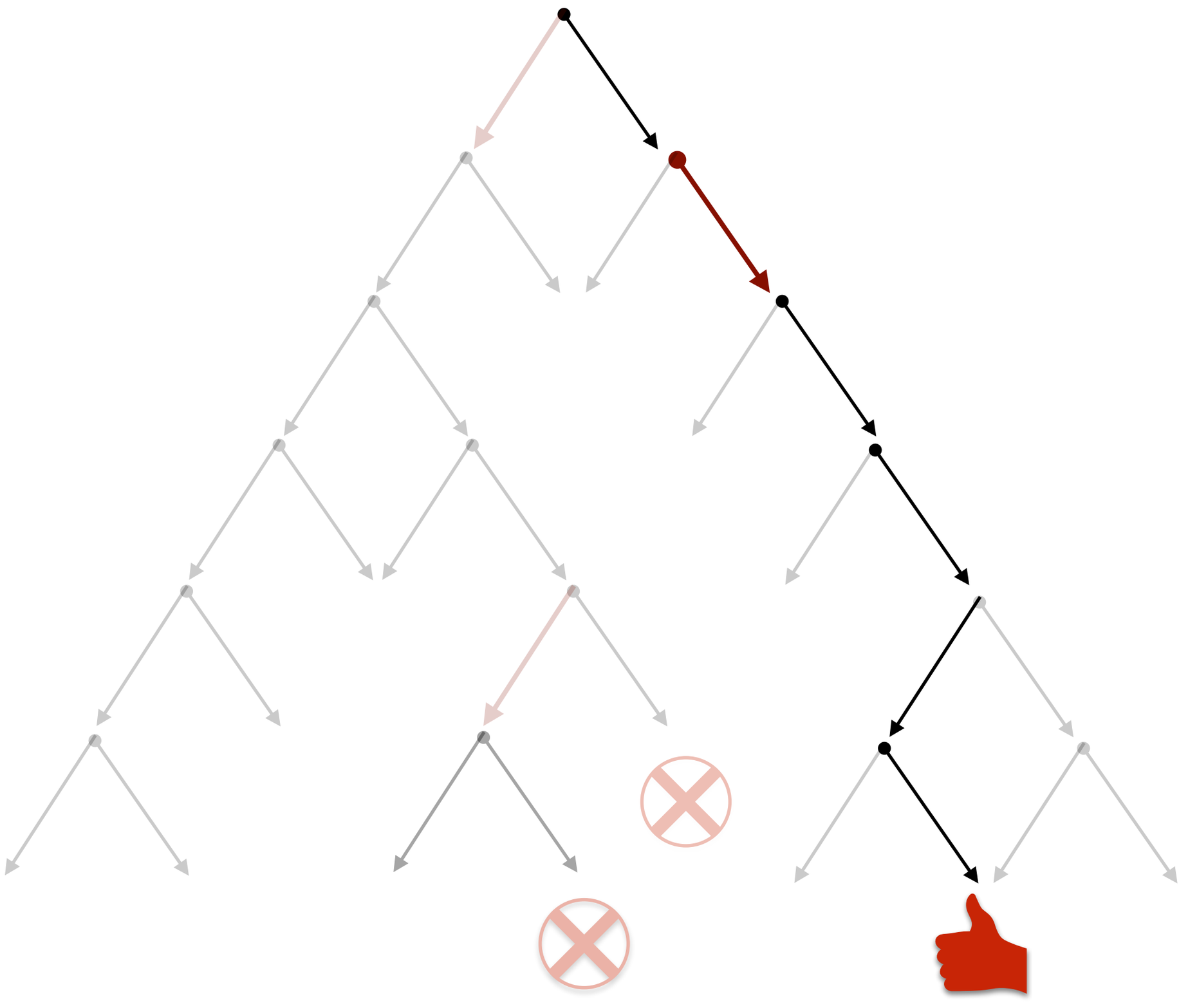
If no decision was made, then there is no solution.

DPLL: Full Algorithm

```
6:  while true do
7:      σ, Γ ←UnitPropagate(φ, σ, Γ)
8:      if ⊥ ∉ σ then
9:          if AllAssigned(σ) then
10:             return SATISFIABLE
11:         else
12:             ⟨p, b⟩ ← MakeDecision(φ, σ)
13:             Δ ← Δ ∪ {⟨p, b⟩}
14:             σ ← σ ∪ {⟨p, b⟩}
15:             Γ ← Γ ∪ {γ(p)}
16:         end if
17:     else
18:         if Δ == ∅ then
19:             return UNSATISFIABLE
20:         else
21:             ⟨p, b⟩ ← Δ.pop_back()
22:             σ ← Backtrack(p, b)
23:             σ ← σ ∪ {⟨p, ¬b⟩}
24:             Γ ← Γ ∪ {γ(p)}
25:         end if
26:     end if
27: end while
```

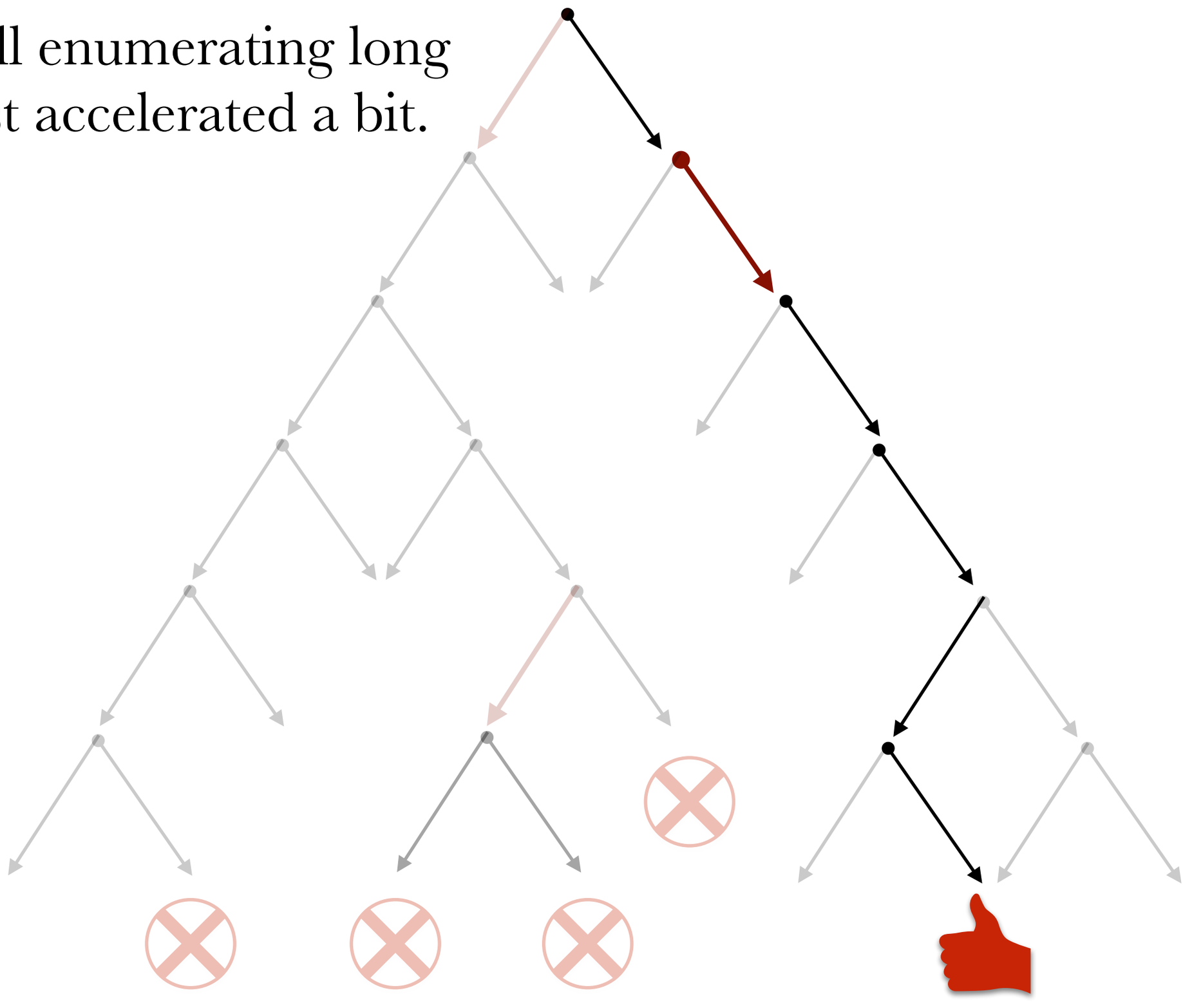
If the conflict came from a decision, backtrack

However, DPLL is still naive.



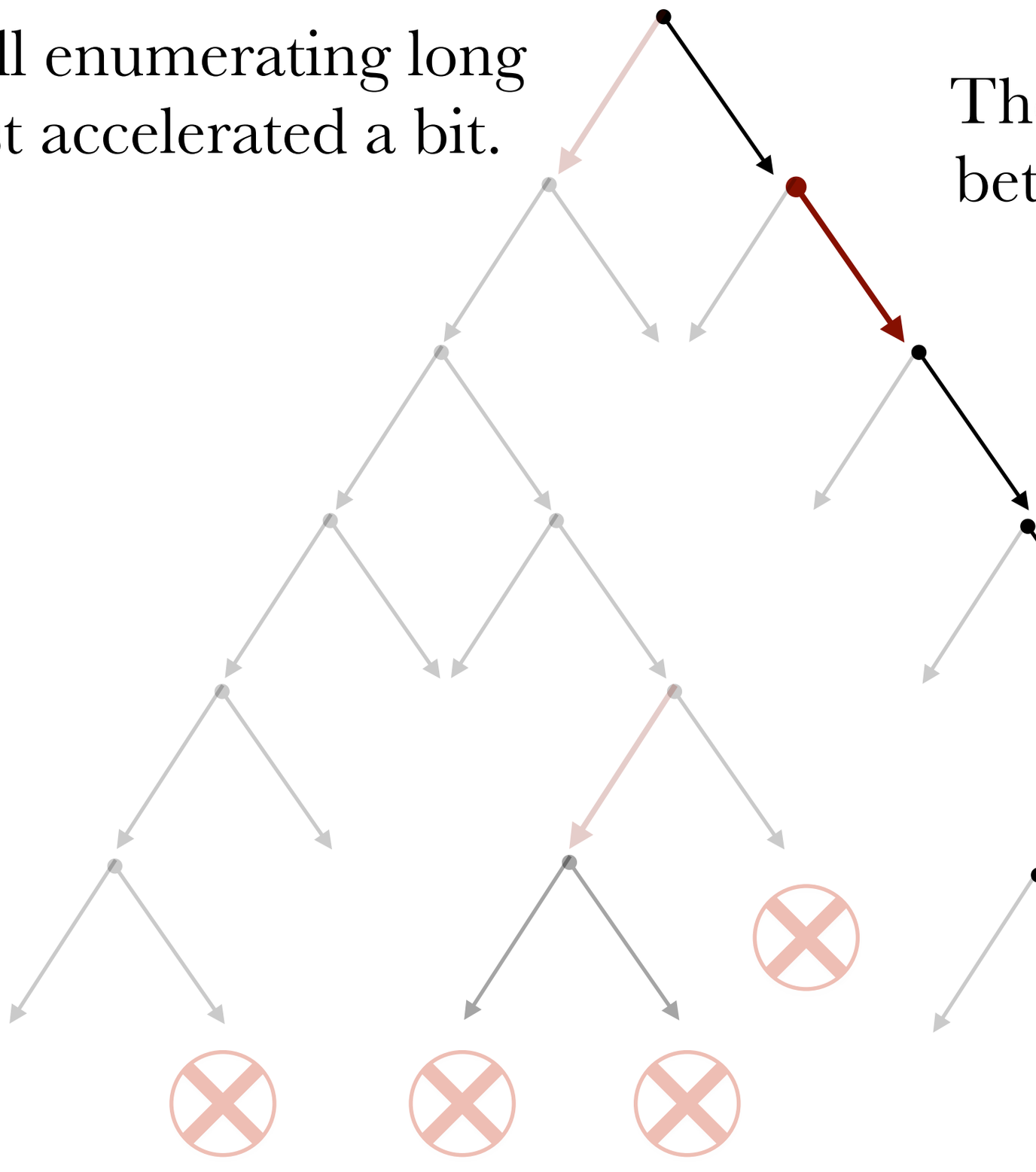
However, DPLL is still naive.

We are still enumerating long paths, just accelerated a bit.



However, DPLL is still naive.

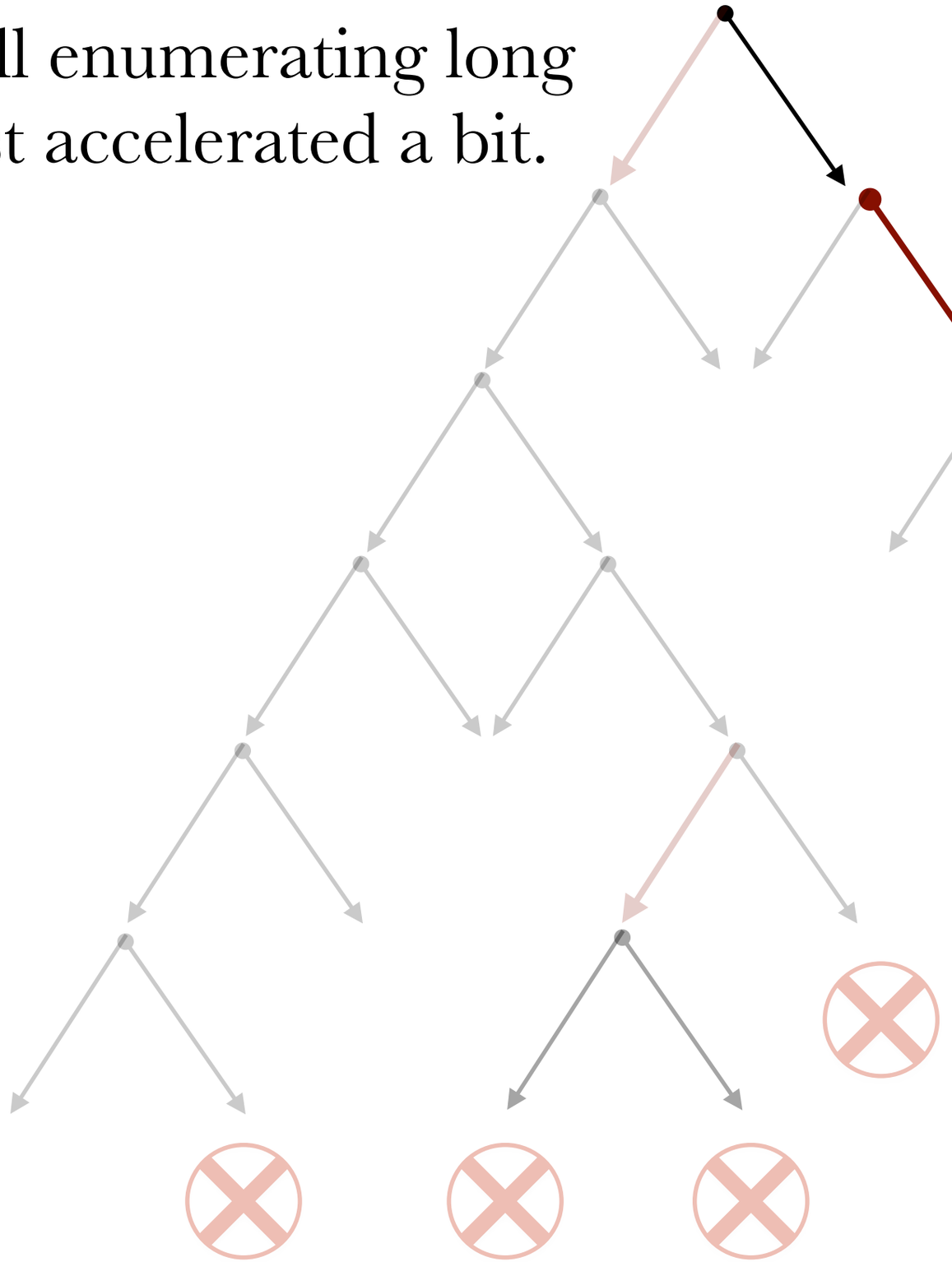
We are still enumerating long paths, just accelerated a bit.



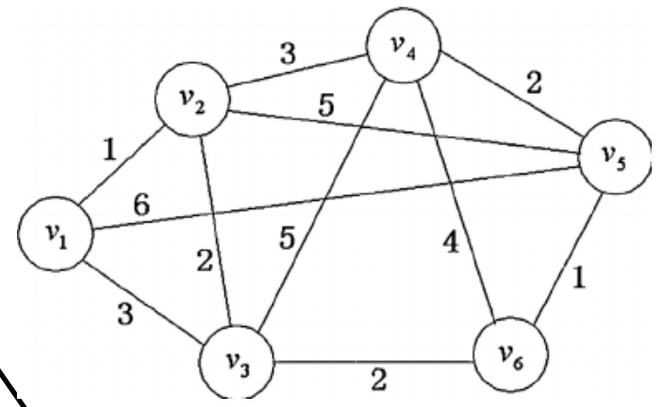
The tight connections between variables are not utilized.

However, DPLL is still naive.

We are still enumerating long paths, just accelerated a bit.



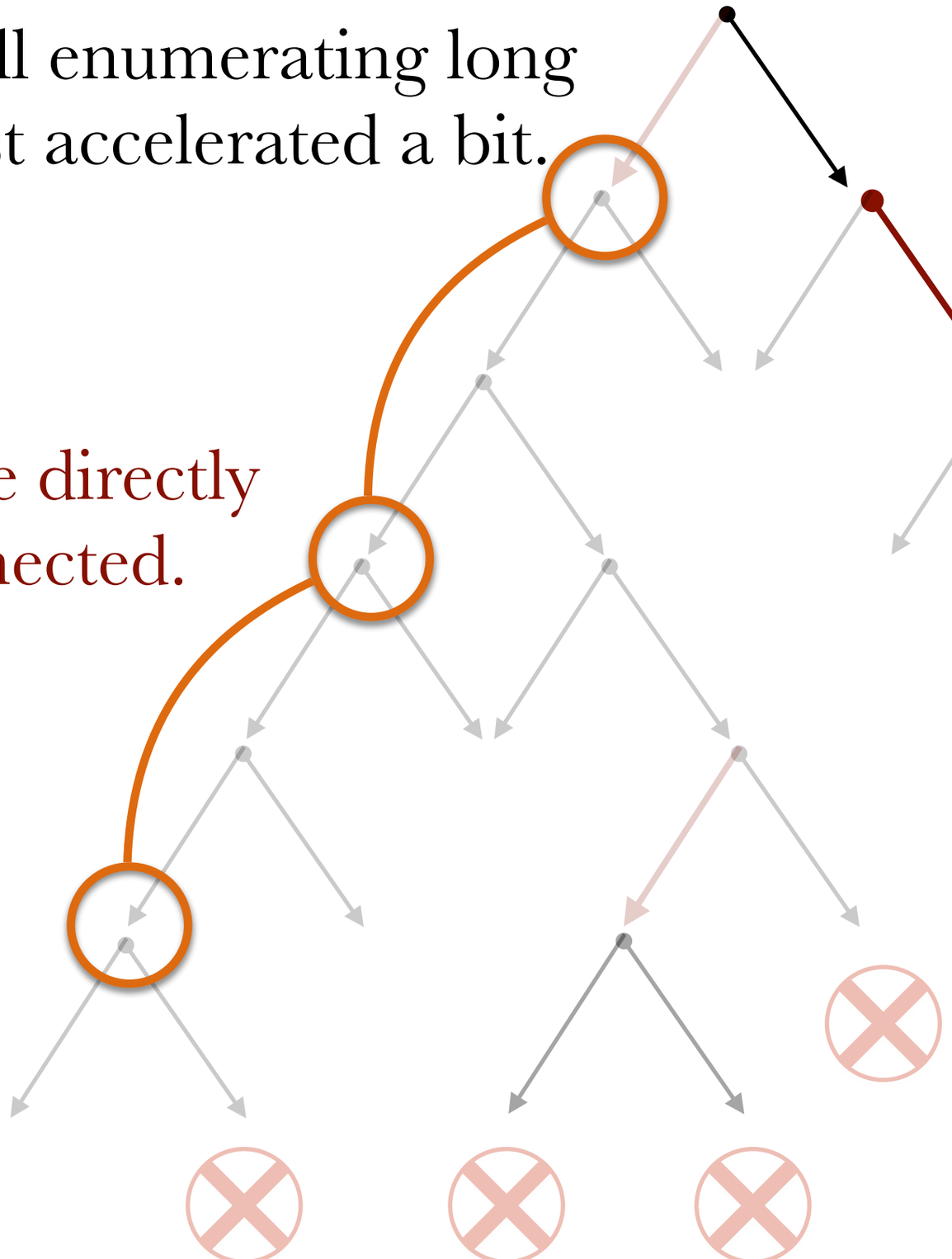
The tight connections between variables are not utilized.



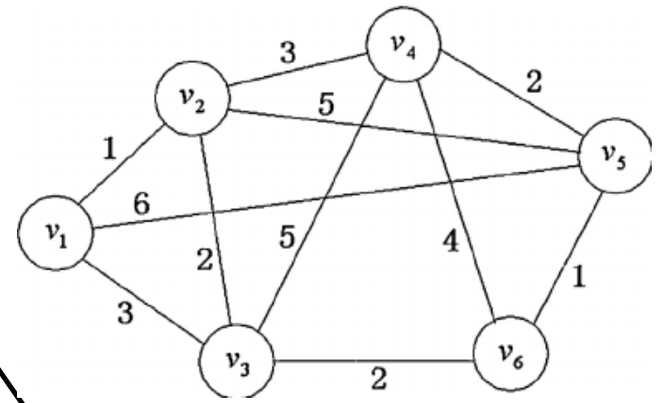
However, DPLL is still naive.

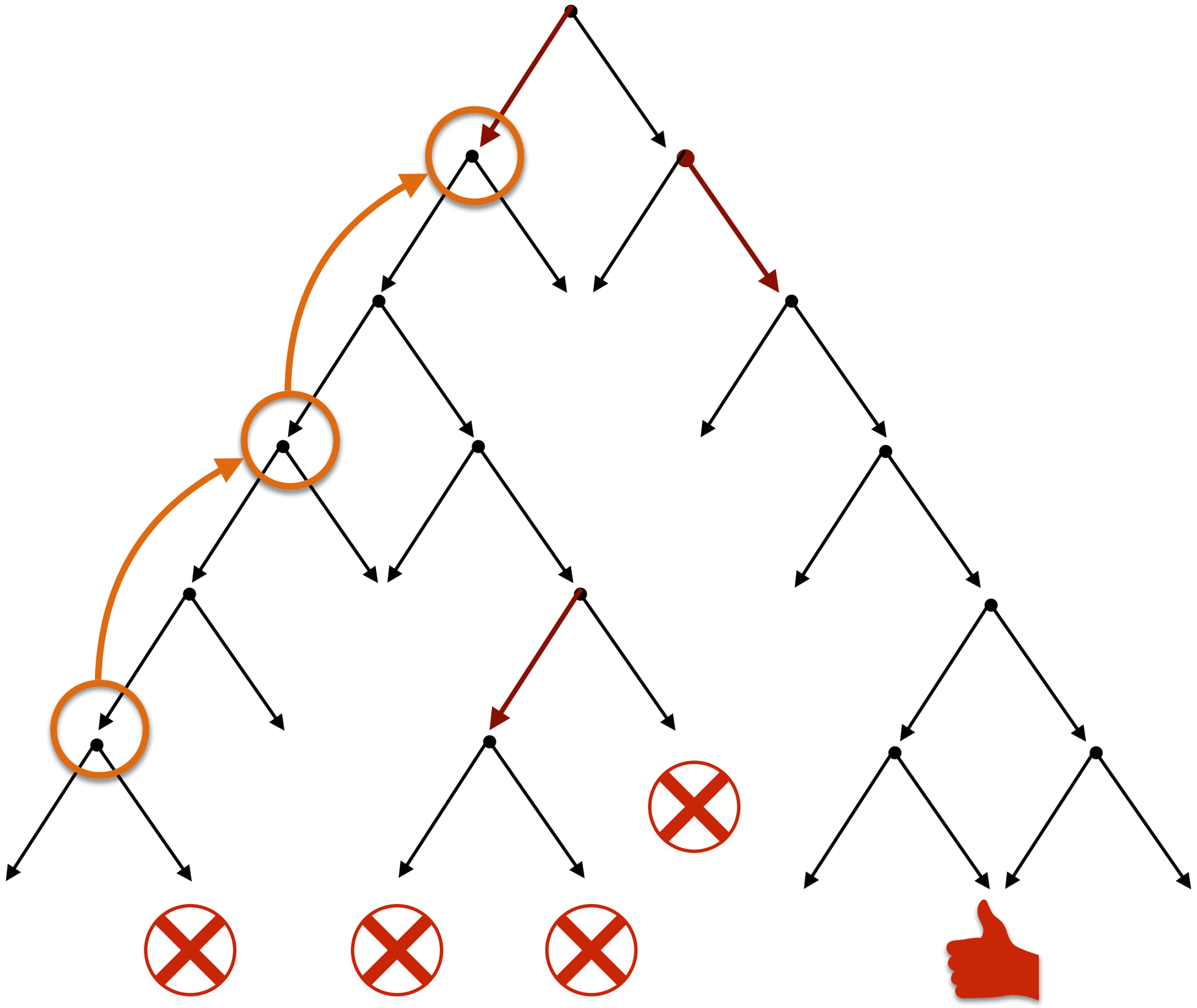
We are still enumerating long paths, just accelerated a bit.

May be directly connected.



The tight connections between variables are not utilized.





However, DPLL is still naive.

CDCL

Conflict-Driven Clause-Learning Search

CDCL: Basic Ideas

We should exploit the constraints among variables to improve the following aspects:

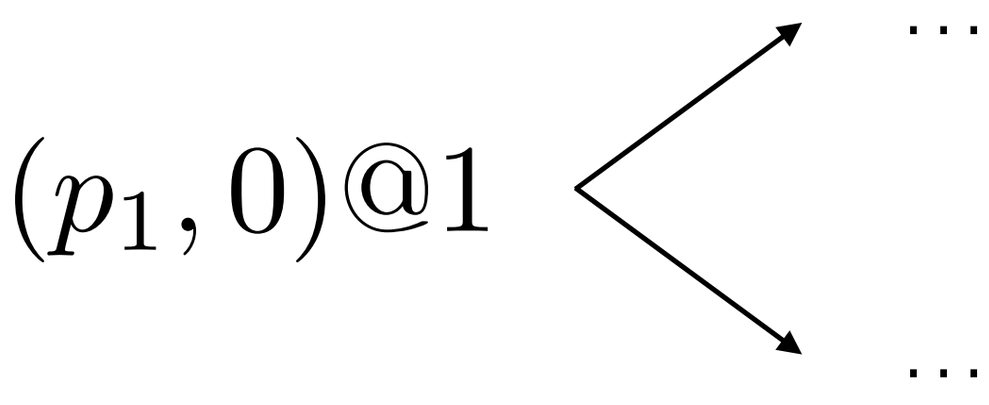
- We should keep track of what combinations of variables lead to conflicts and avoid them.

- Instead of backtracking to the most recent decision, we should "backjump" and skip much larger parts of the tree.

Key Concept: Implication Graph

Each decision has a "decision level". Unit propagation does not increase decision levels.

When conflict occur, we analyze what levels they came from, and figure out what constraints were responsible for that.



Key Concept: Implication Graph

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$(p_4, 0)@0$

Key Concept: Implication Graph

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$(p_4, 0)@0$

$(p_1, 0)@1$

Key Concept: Implication Graph

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

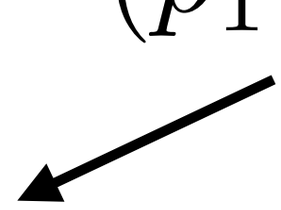
$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$\{\neg p_5\}$

$(p_4, 0)@0$

$(p_1, 0)@1$



$(p_2, 0)@1$

Key Concept: Implication Graph

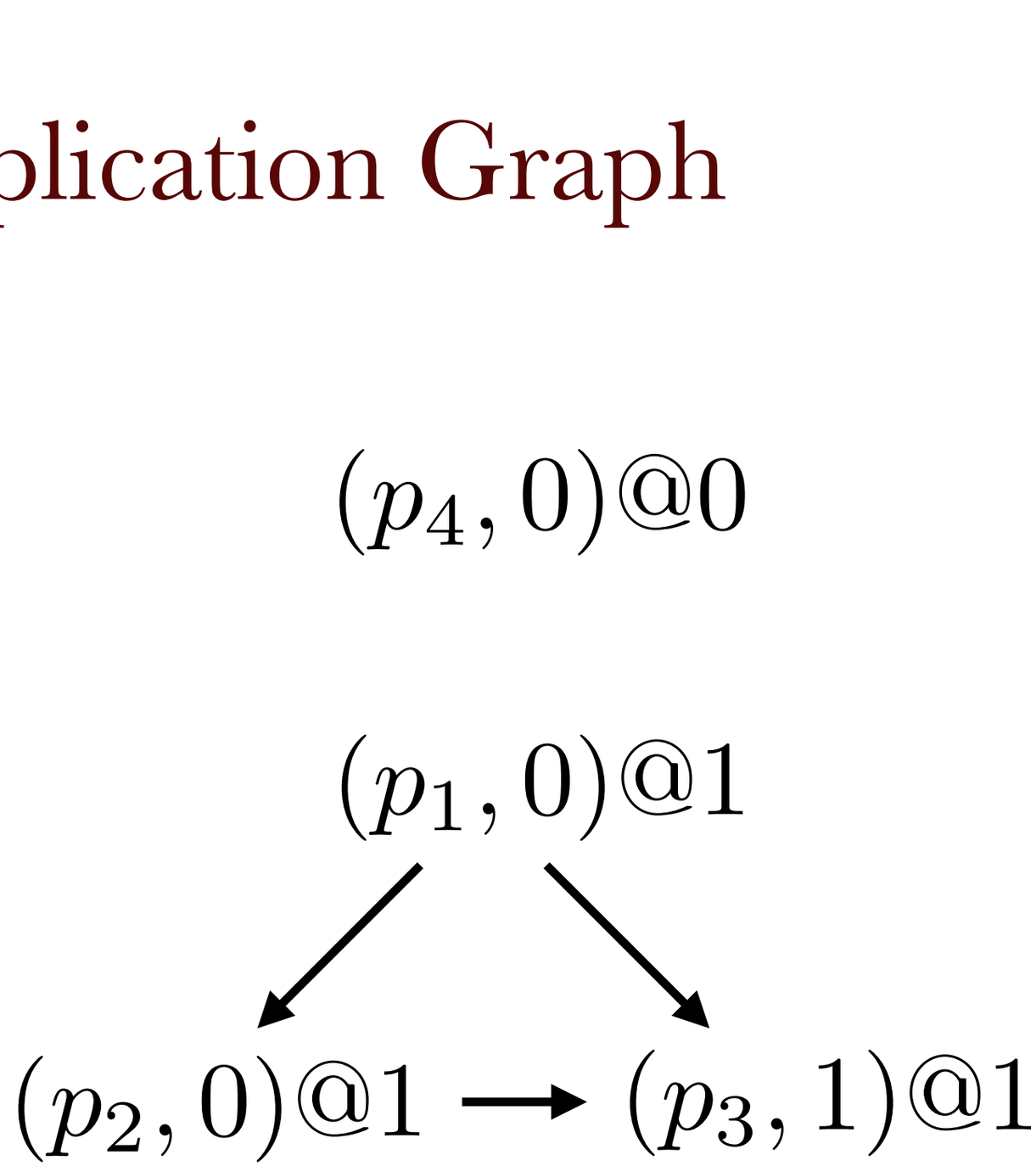
$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$\{\neg p_5\}$



Key Concept: Implication Graph

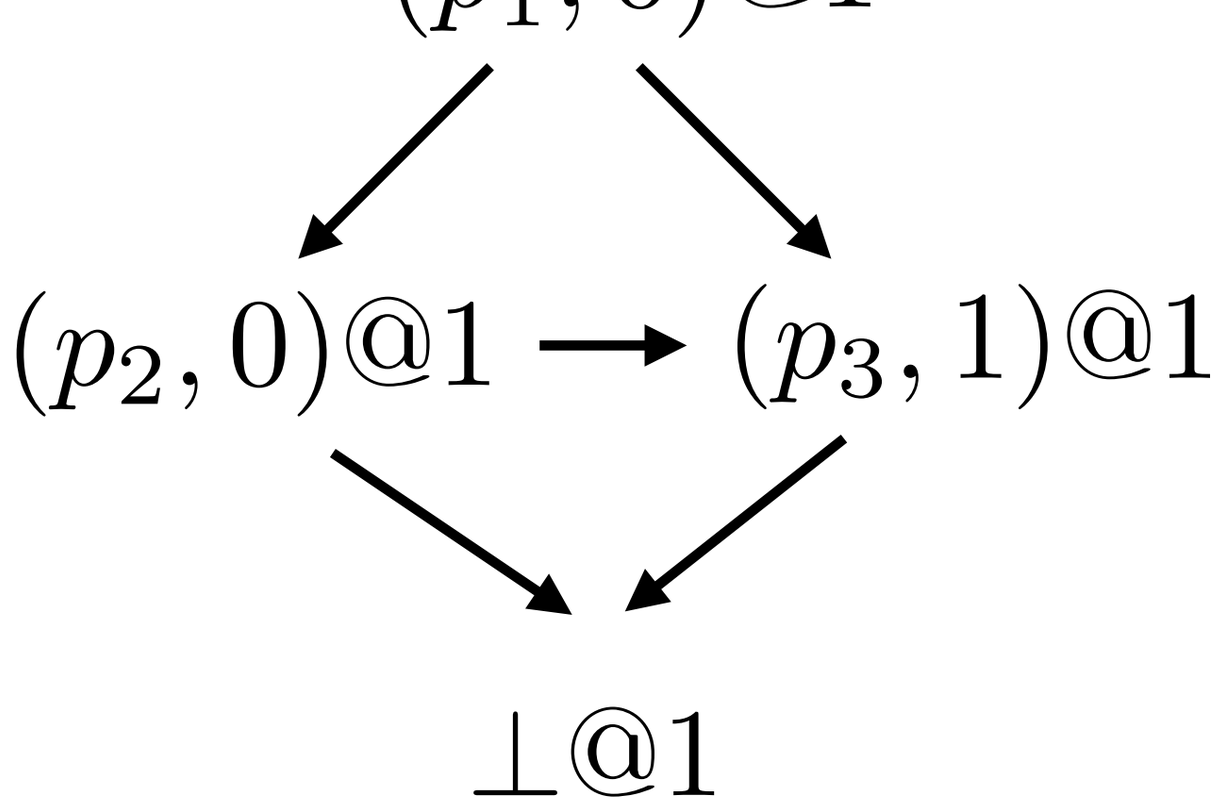
$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$\{\neg p_5\}$



How do we explain the conflict?

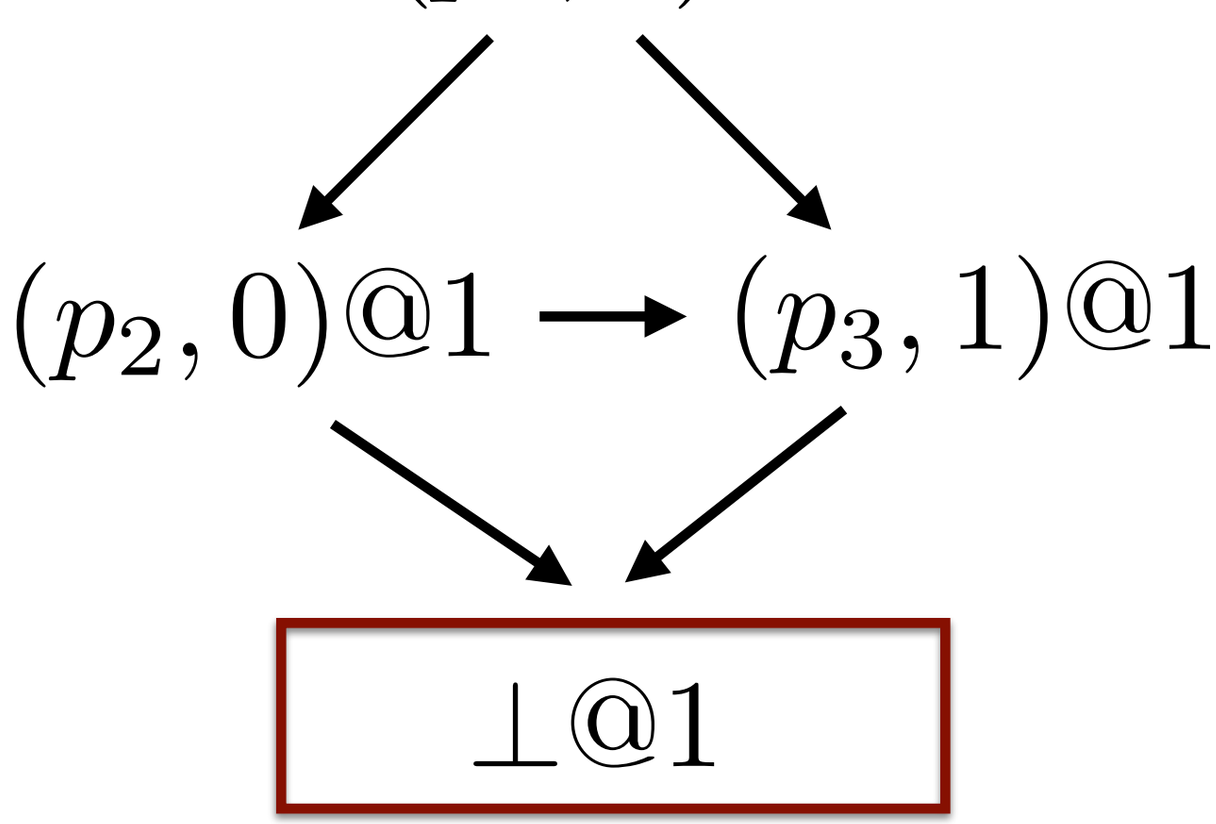
$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$\{\neg p_5\}$



How do we explain the conflict?

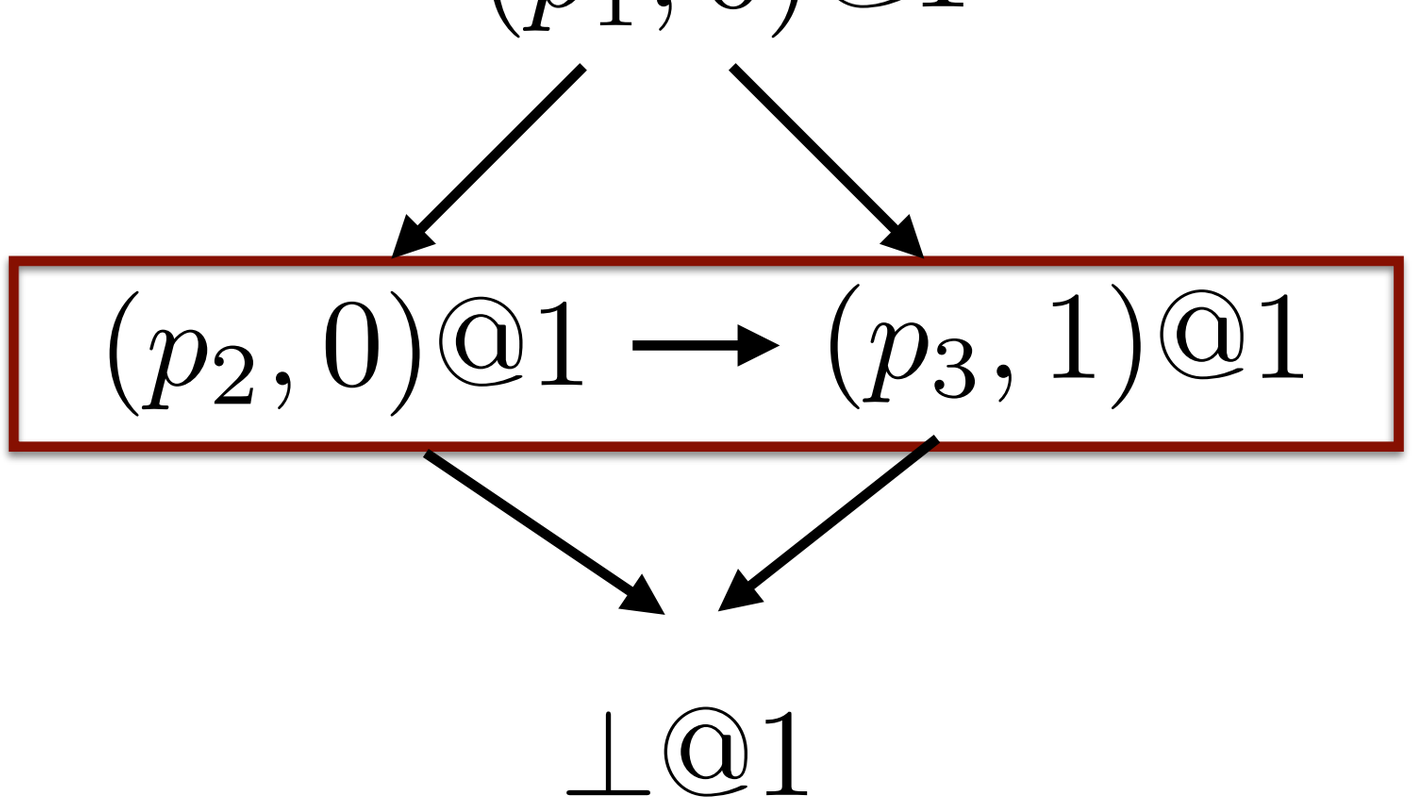
$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$\{\neg p_5\}$



How do we explain the conflict?

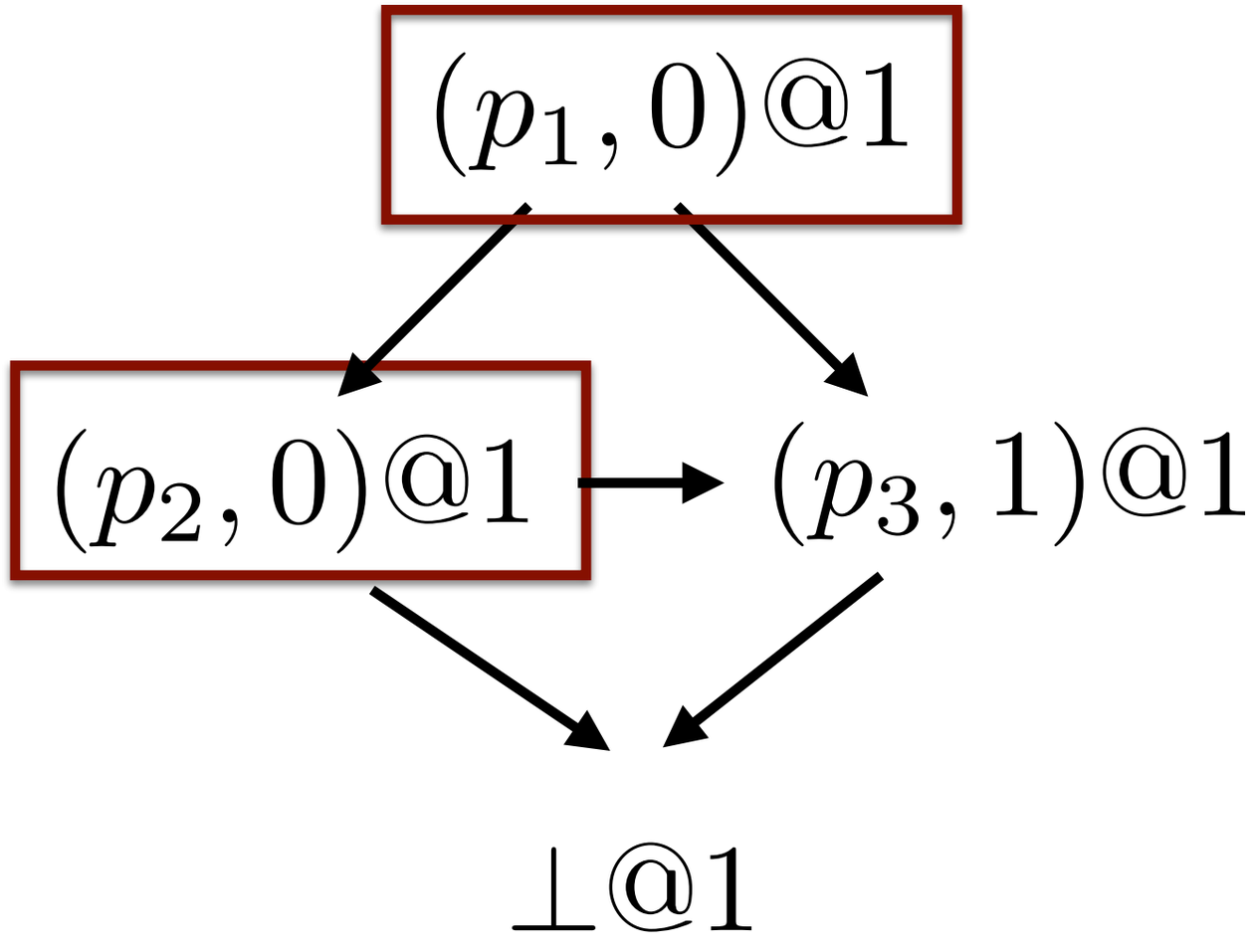
$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$\{\neg p_5\}$



How do we explain the conflict?

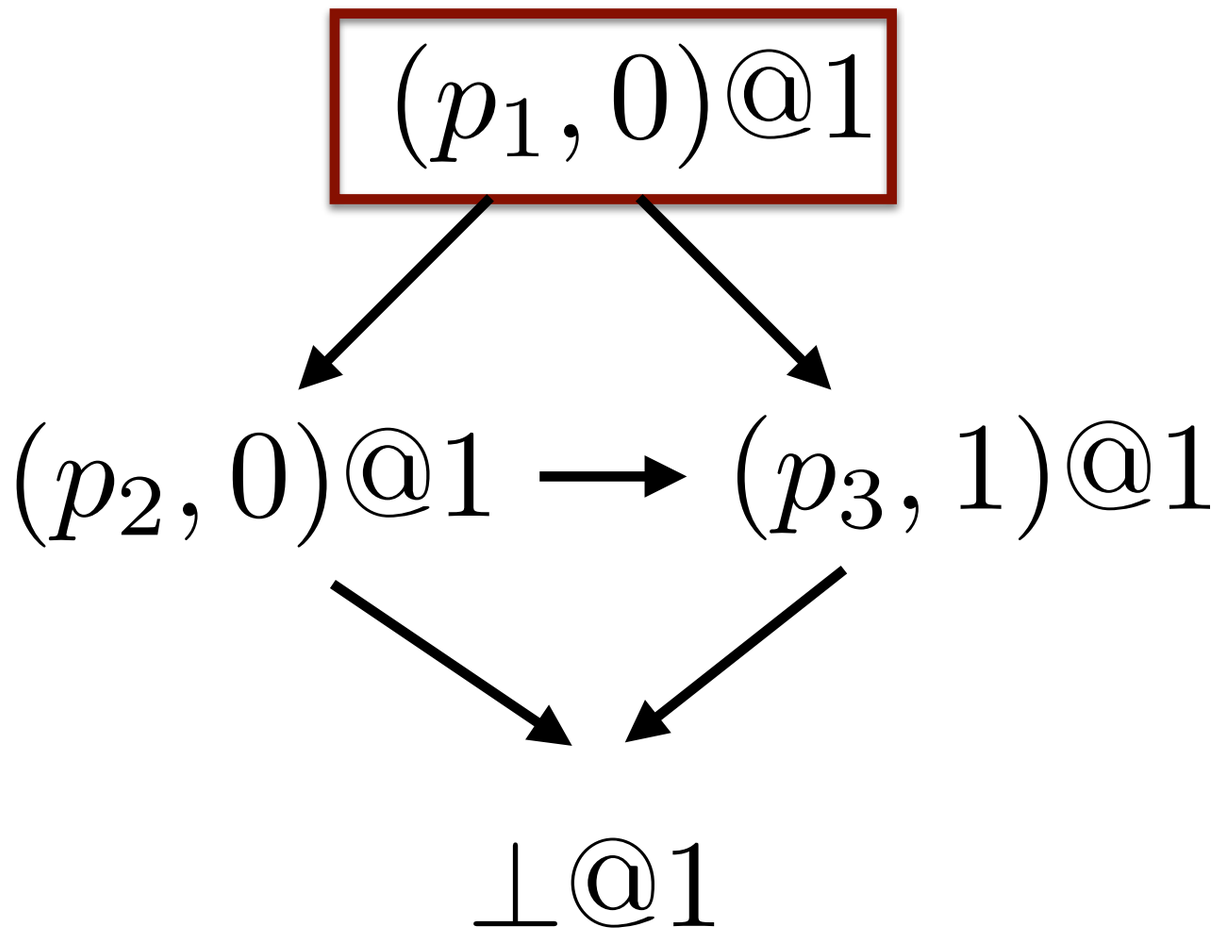
$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$\{\neg p_5\}$



We can learn a lot from a conflict

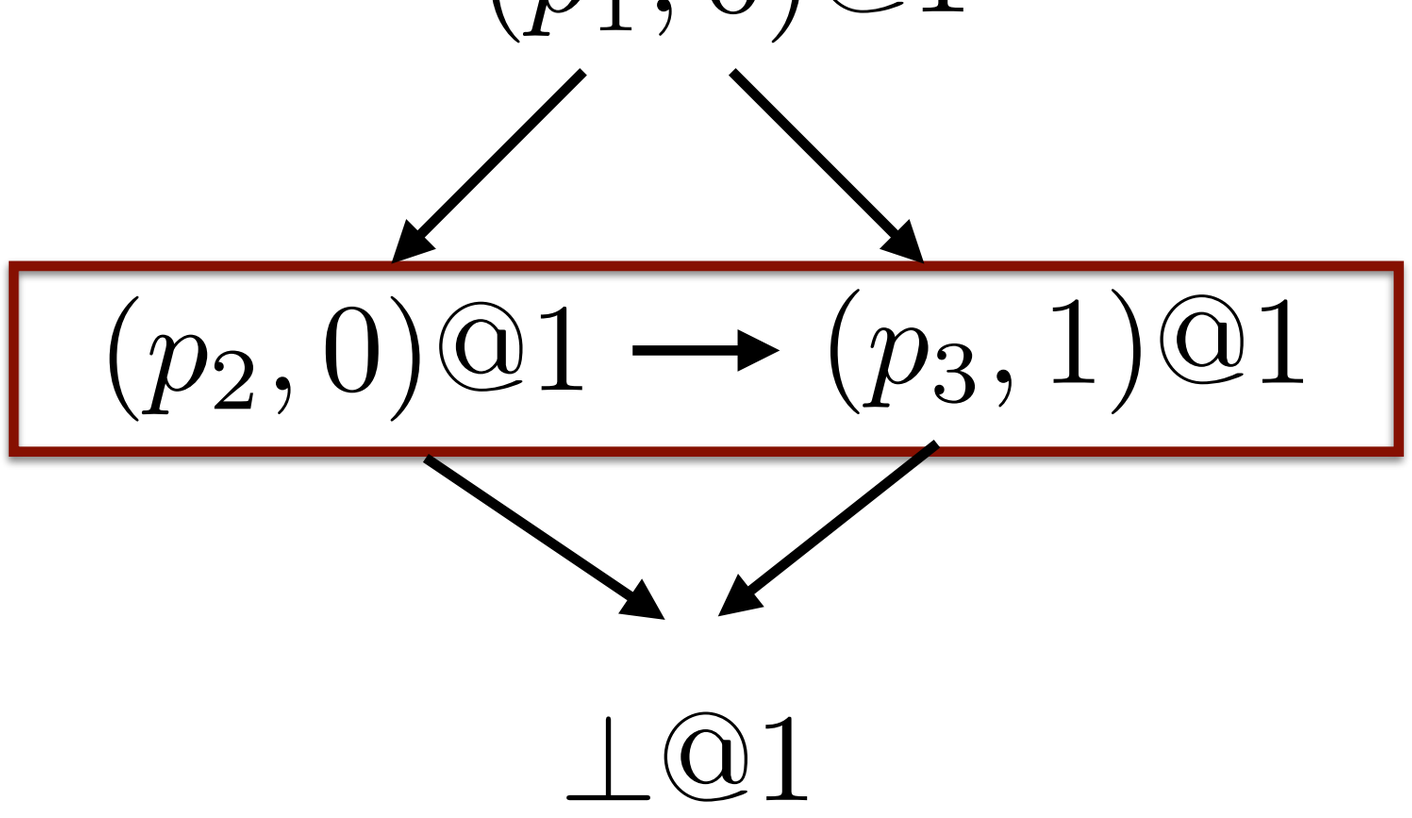
$\neg(\neg p_2 \wedge p_3)$

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$



We can learn a lot from a conflict

$\neg(\neg p_2 \wedge p_3)$

$\neg(\neg p_2 \wedge \neg p_1)$

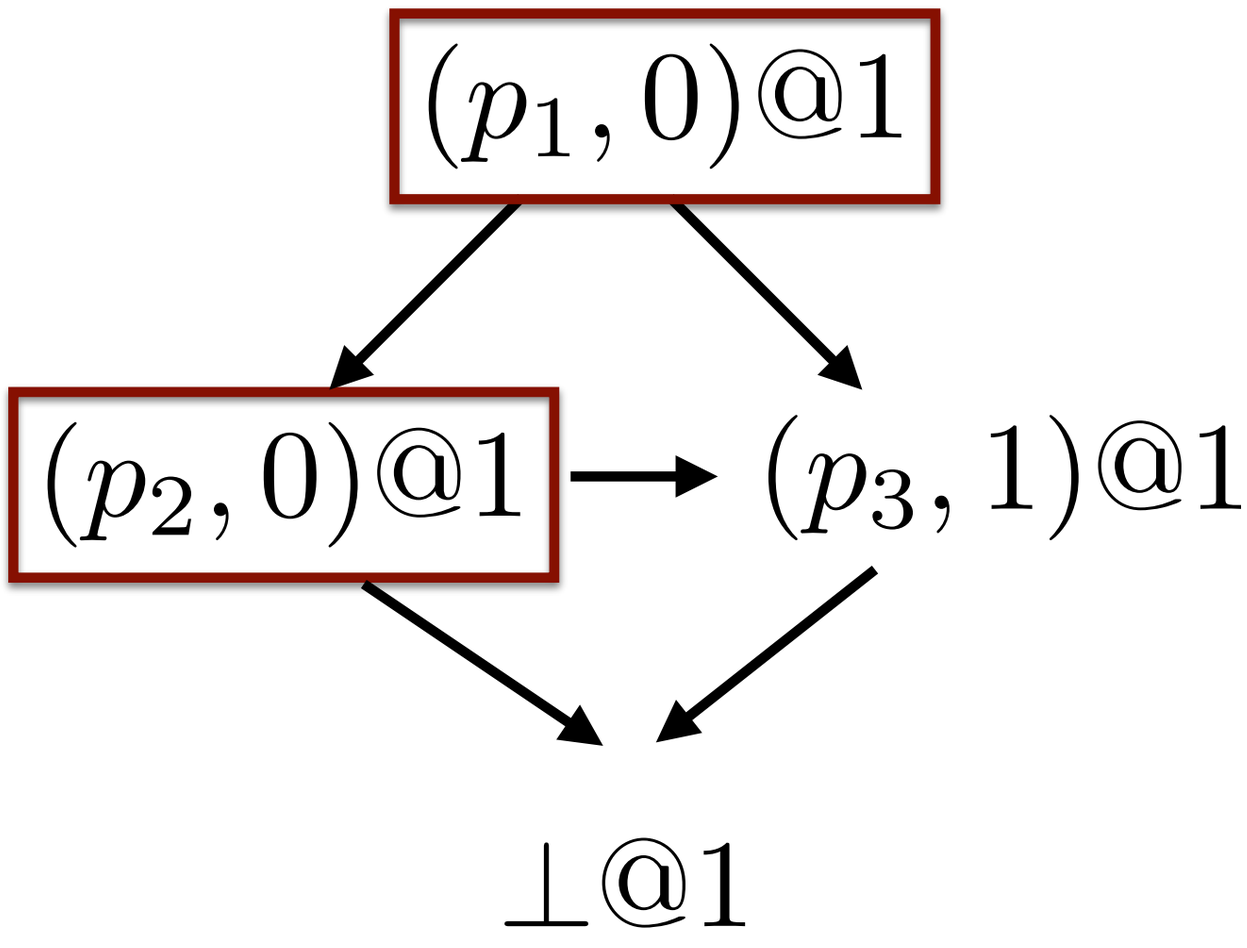
$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$(p_4, 0)@0$



We can learn a lot from a conflict

$\neg(\neg p_2 \wedge p_3)$

$\neg(\neg p_2 \wedge \neg p_1)$

$\neg(\neg p_1)$

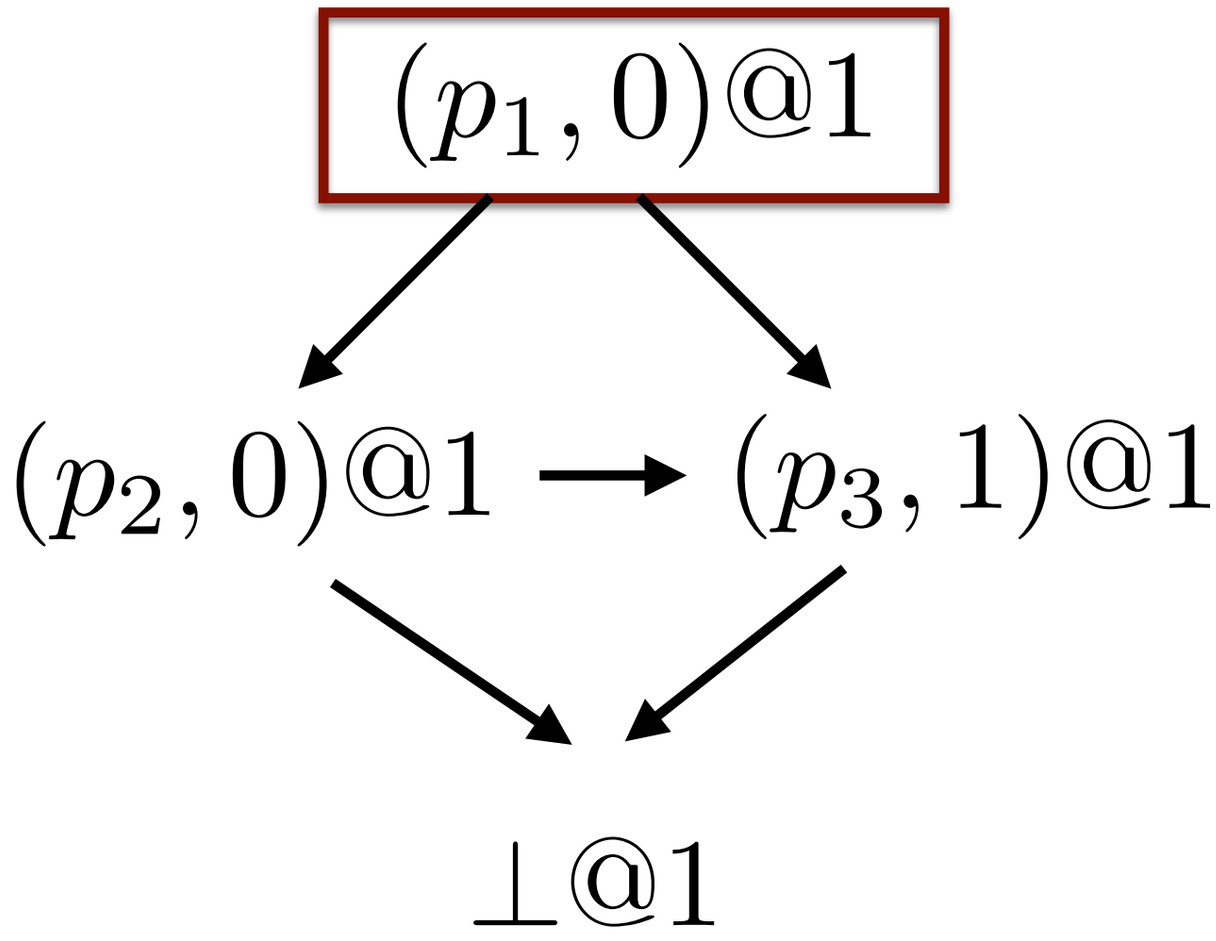
$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$(p_4, 0)@0$



CNF helps: Everything we learn is a clause!

$\neg(\neg p_2 \wedge \neg p_3)$ $\{p_2, \neg p_3\}$ $(p_4, 0)@0$

$\neg(\neg p_2)$ $\{p_2, p_1\}$

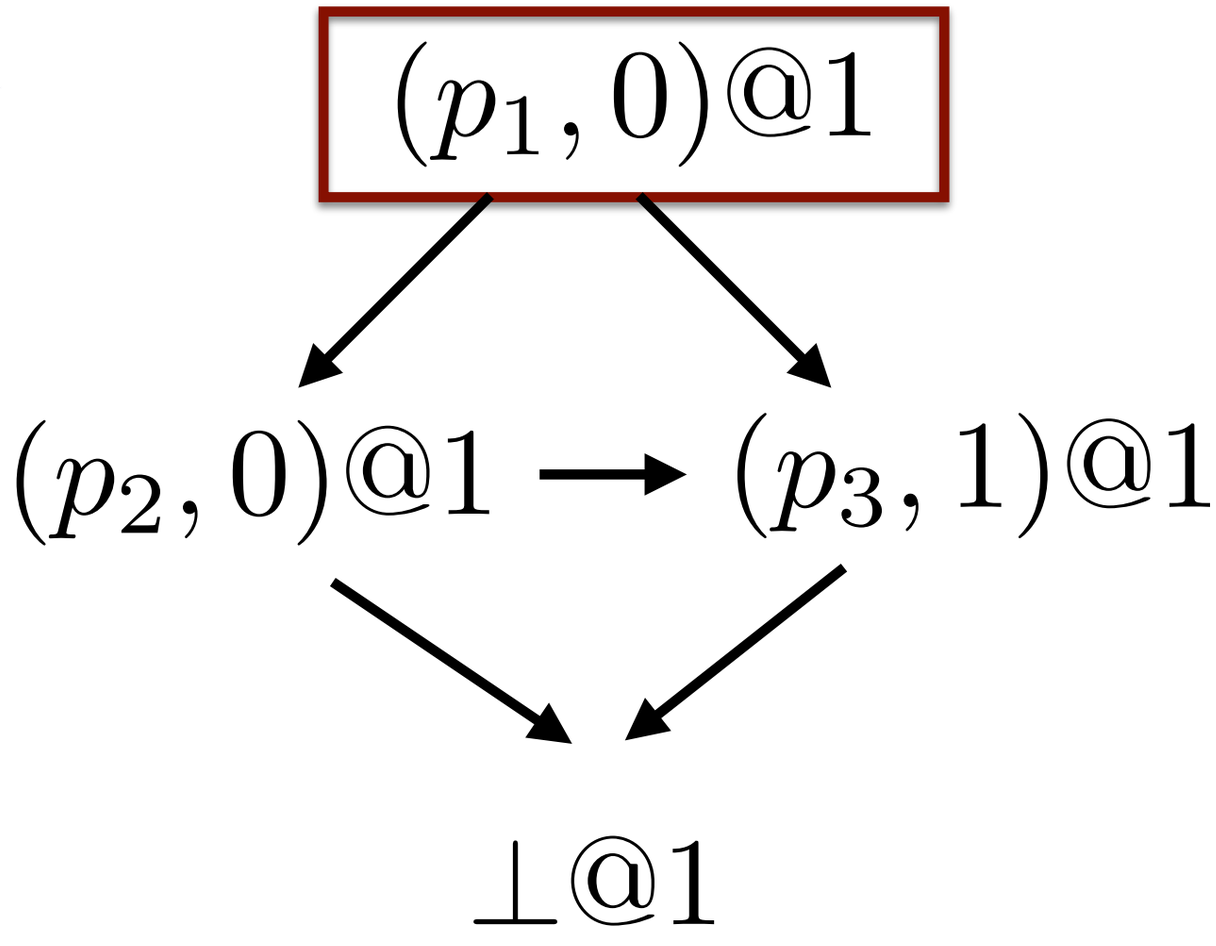
$\neg(\neg p_1)$ $\{p_1\}$

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$



The learned clauses force new values

$\{p_2, \neg p_3\}$

$\{p_2, p_1\}$

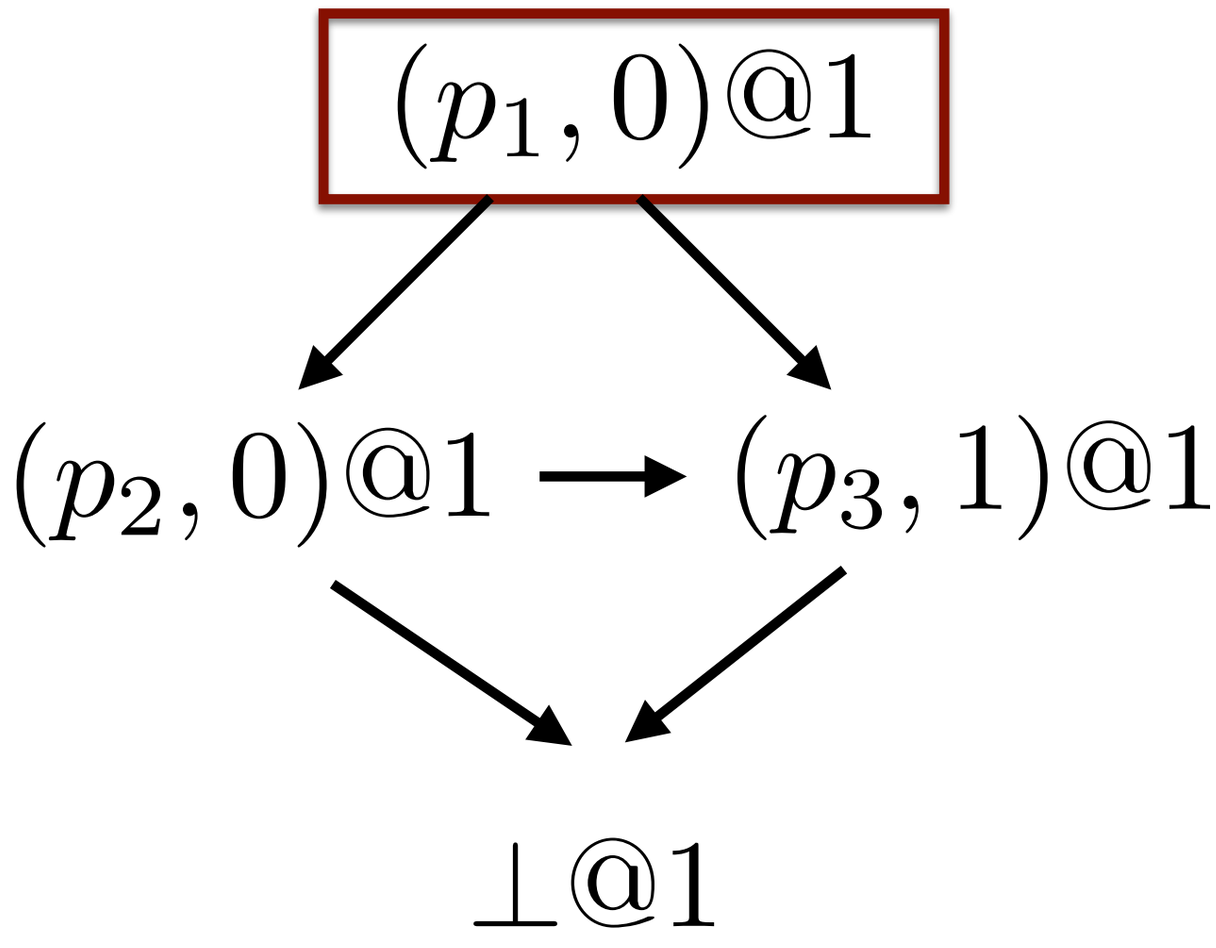
$\{p_1\}$

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$



The learned clauses force new values

$\{p_2, \neg p_3\}$

$\{p_2, p_1\}$

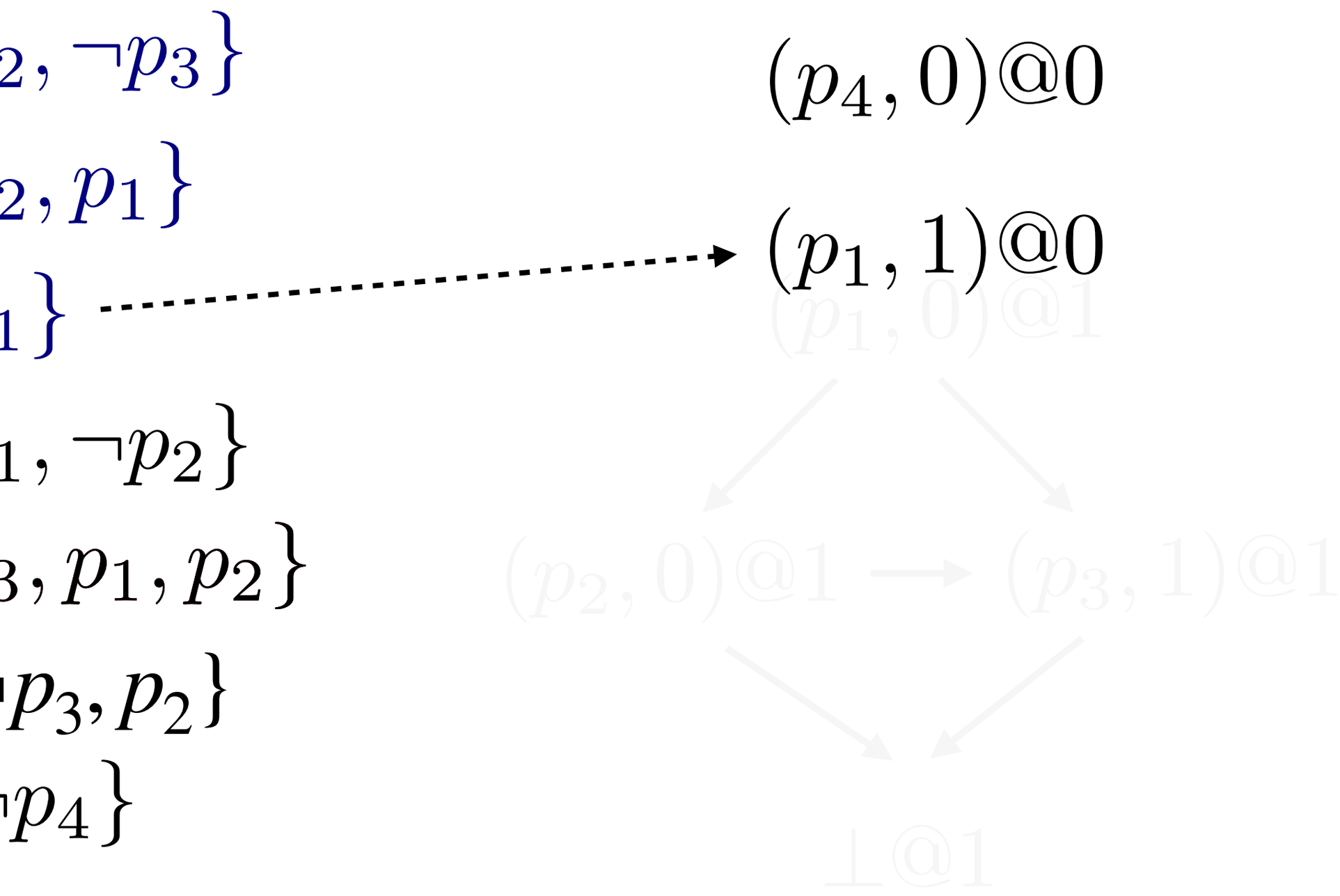
$\{p_1\}$

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$



Don't need the decision stack anymore!

$\{p_2, \neg p_3\}$

$\{p_2, p_1\}$

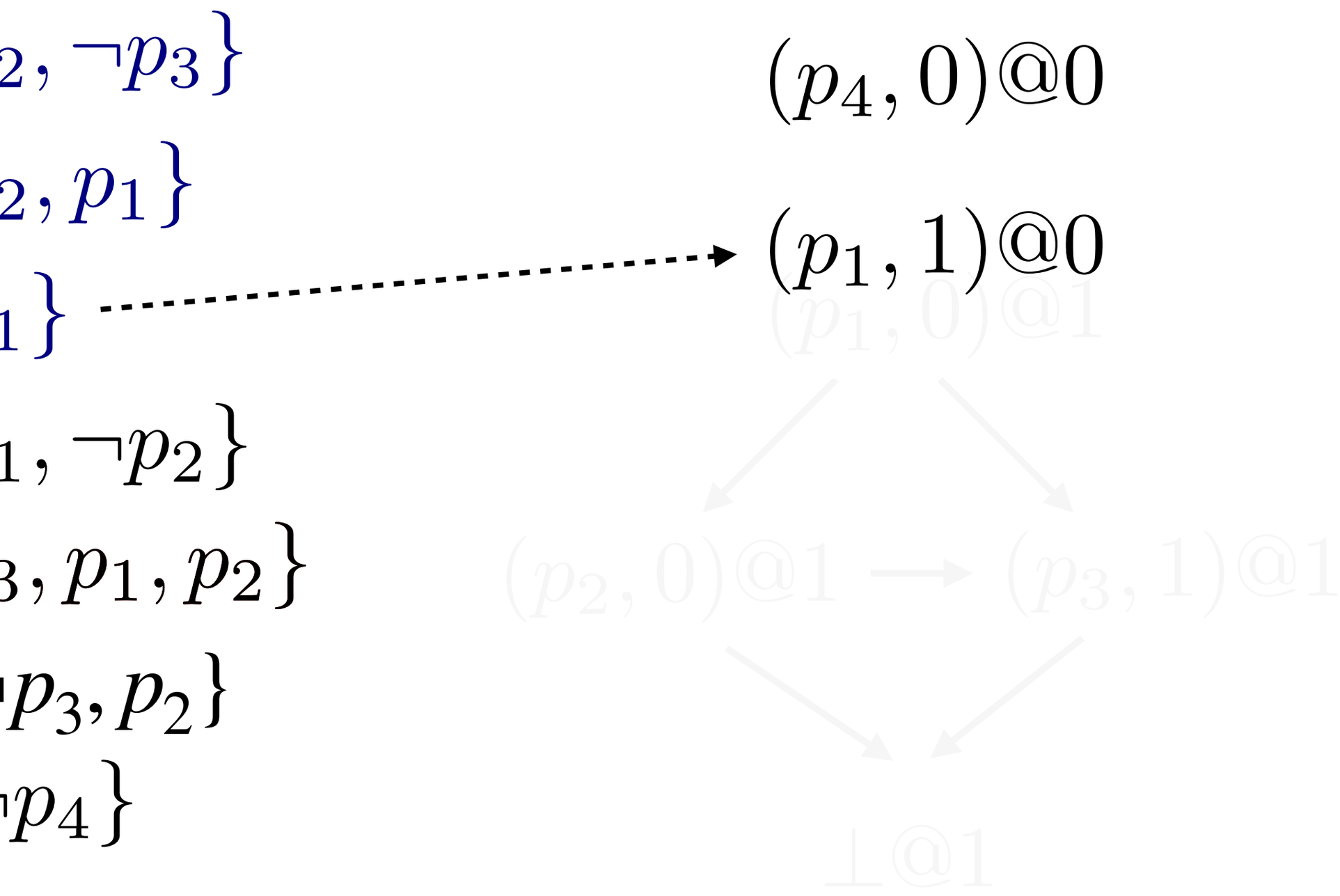
$\{p_1\}$

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$



Make decisions and propagate in the same way

$\{p_2, \neg p_3\}$

$\{p_2, p_1\}$

$\{p_1\}$

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$

$(p_4, 0)@0$

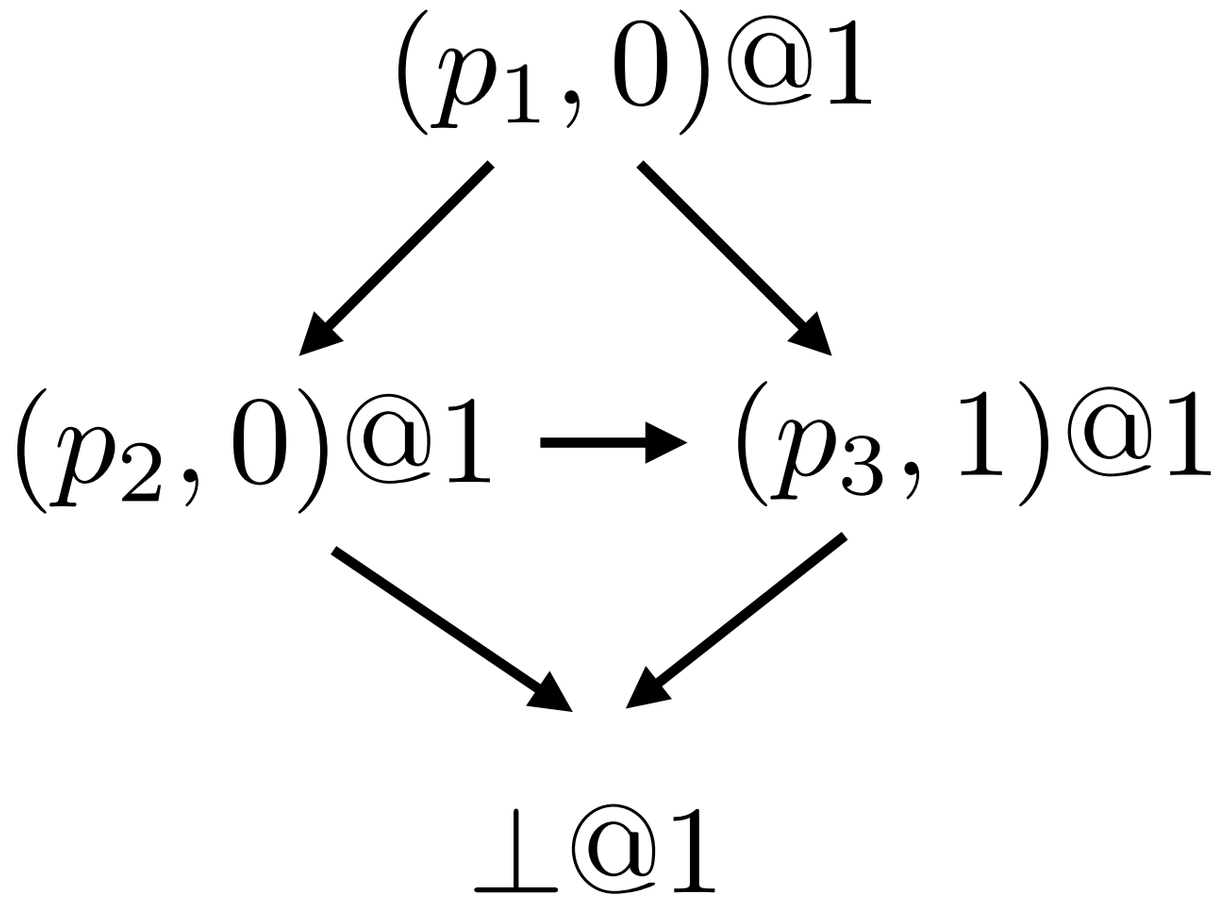
$(p_1, 1)@0$

$(p_2, 0)@1$

$(p_3, 0)@1$

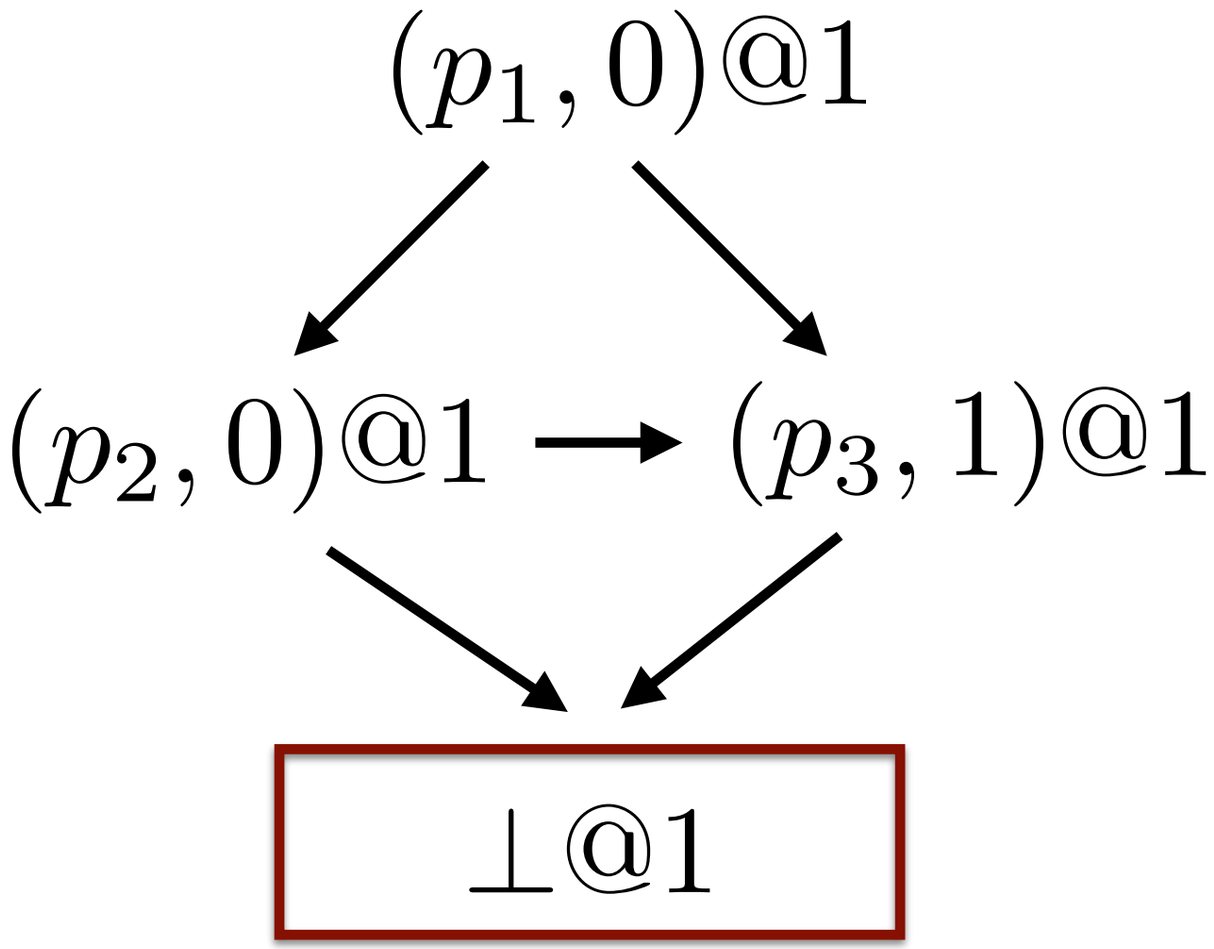
Done.

CDCL: More concepts



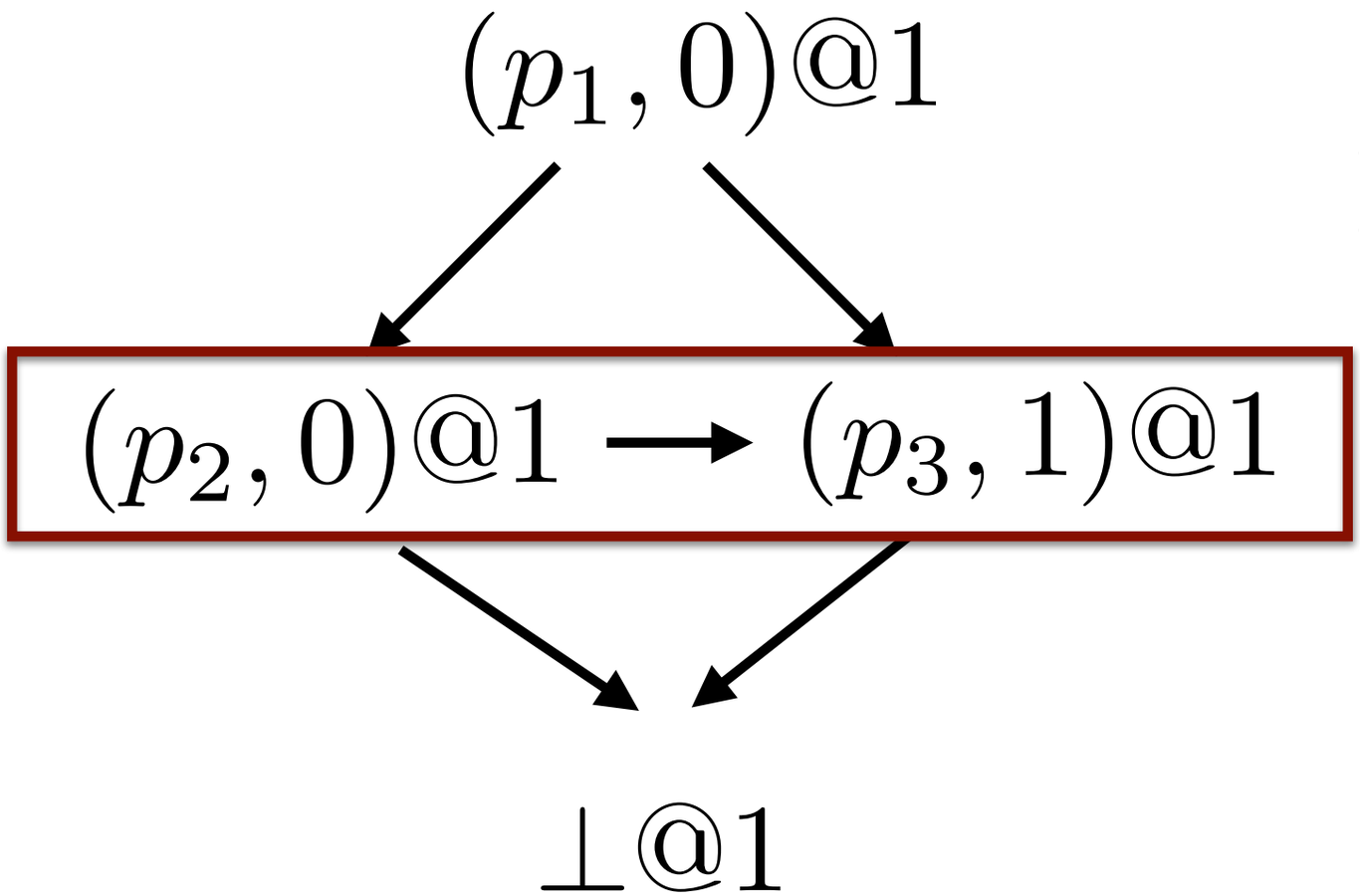
How did we learn?

CDCL: More concepts



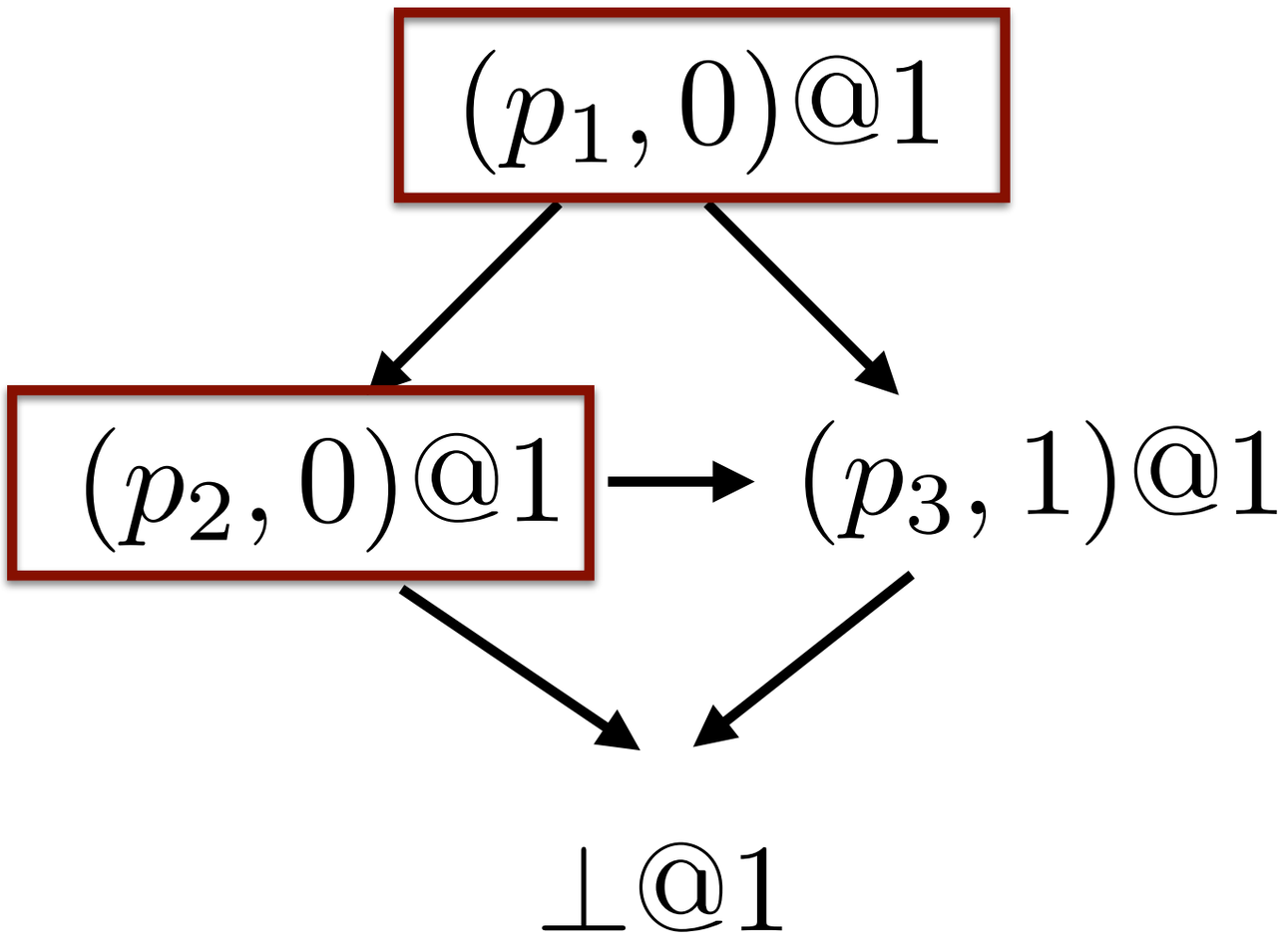
How did we learn?

CDCL: More concepts



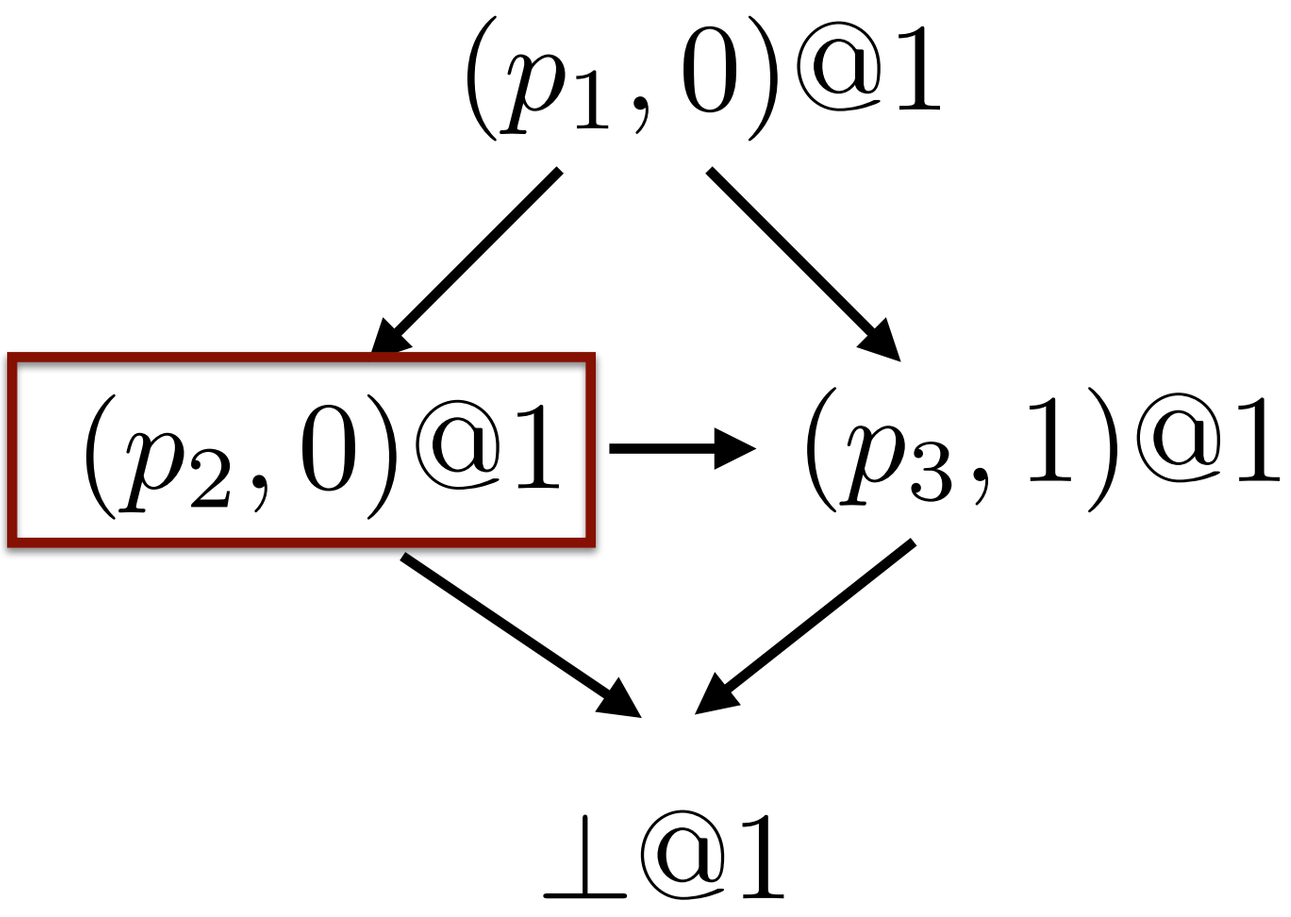
How did we learn?

CDCL: More concepts



How did we learn?

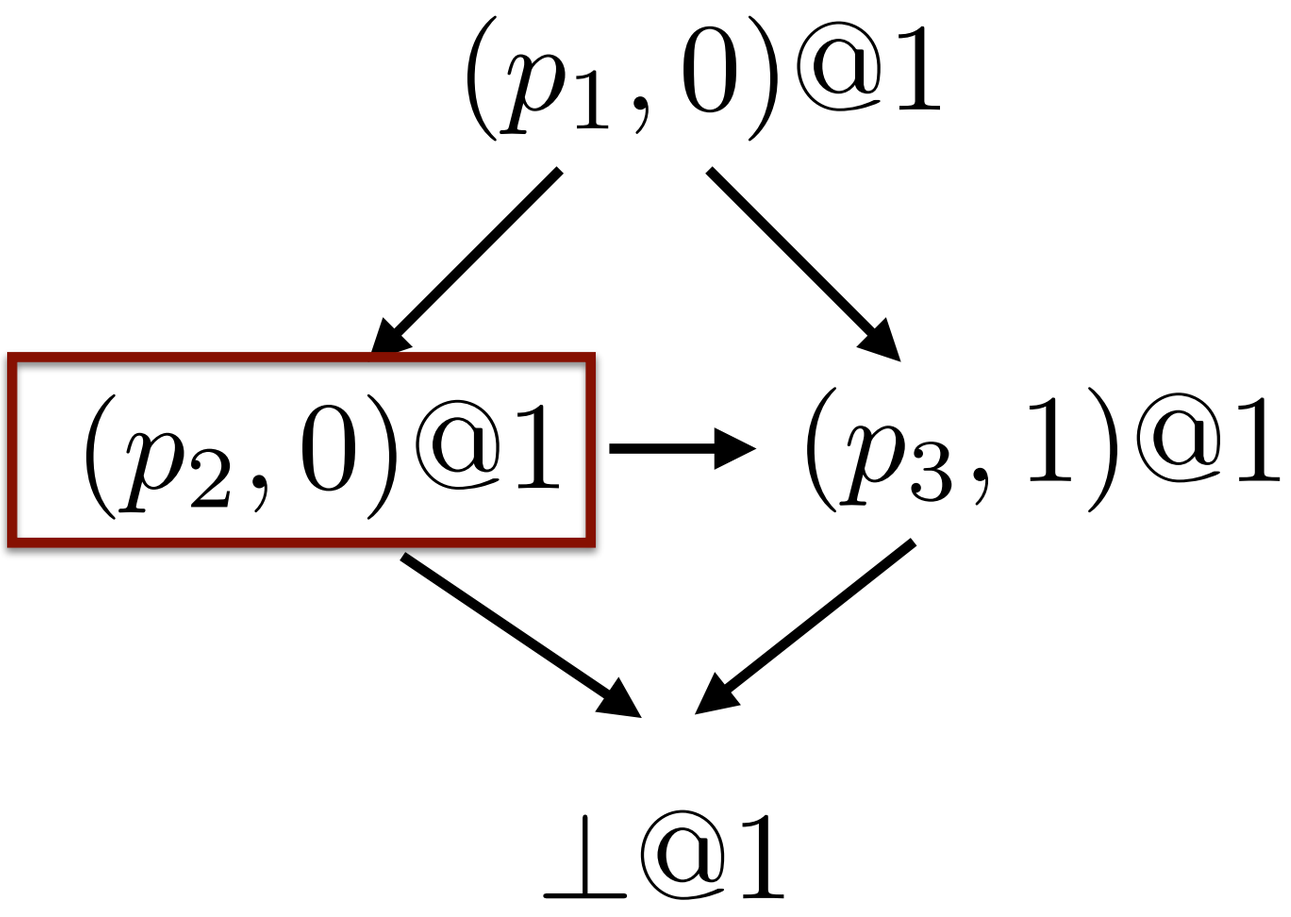
CDCL: More concepts



Why not learn this?

$$\{\neg\neg p_2\}$$

CDCL: More concepts

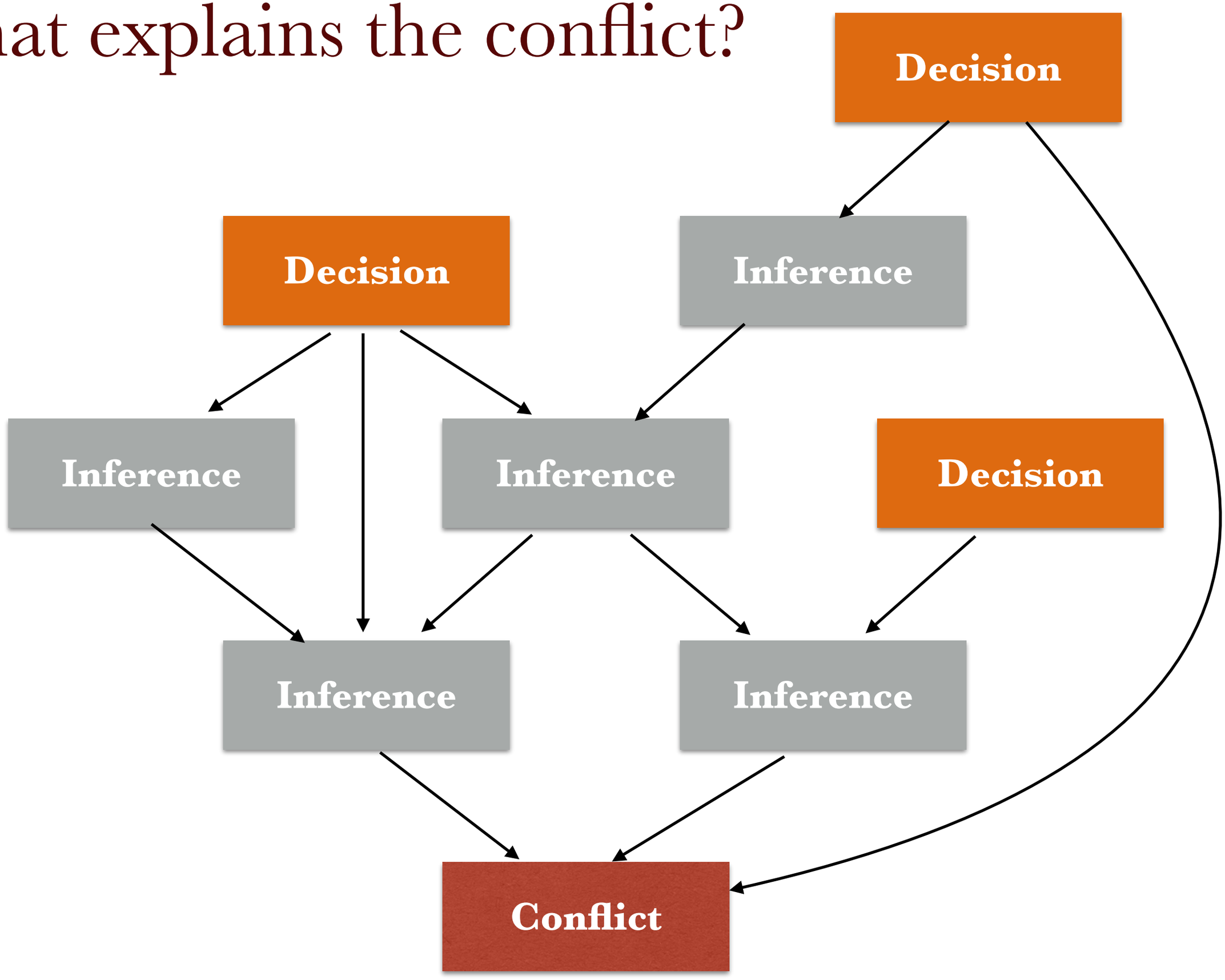


Why not learn this?

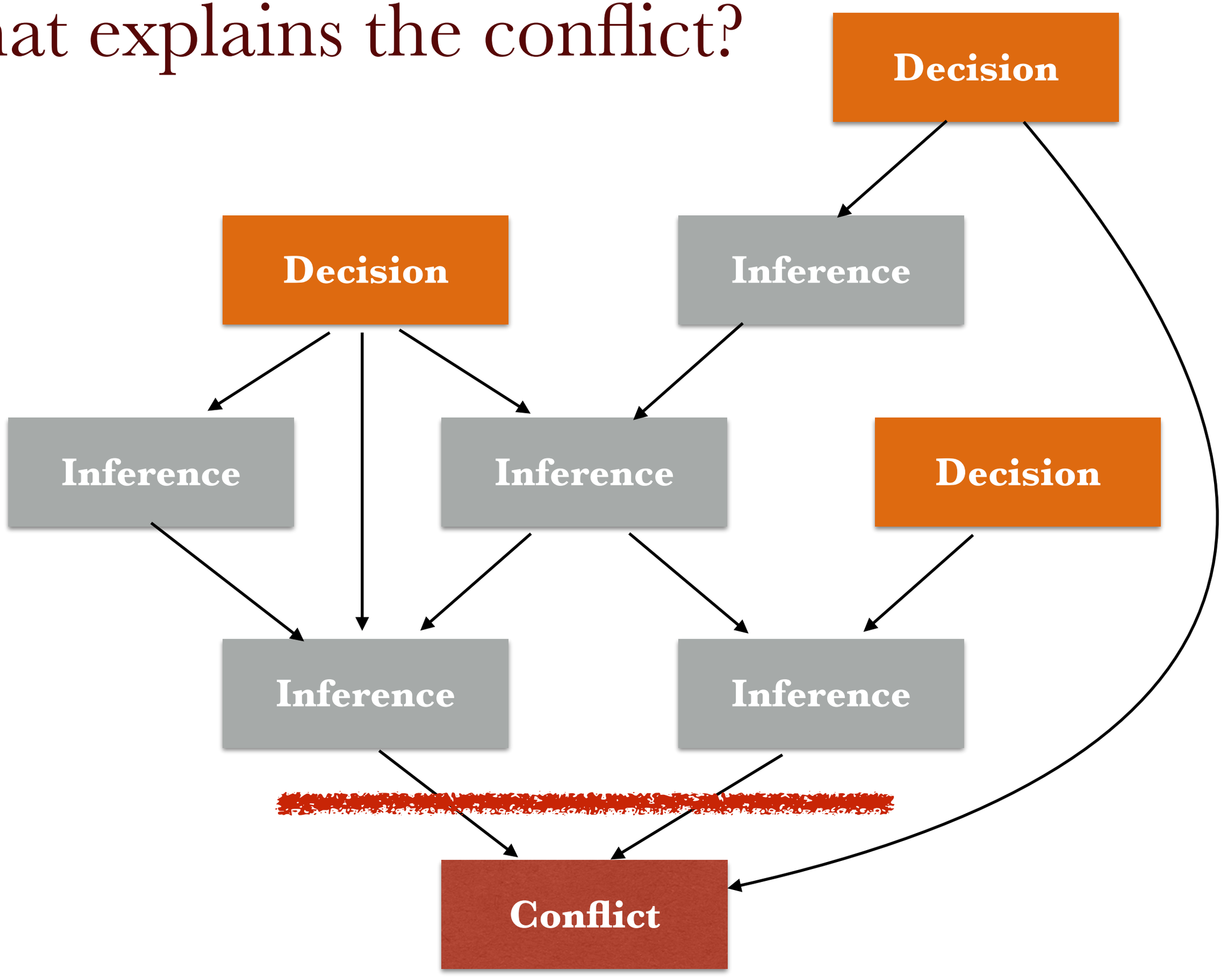
$$\{\neg p_2\}$$

It is not a complete explanation.

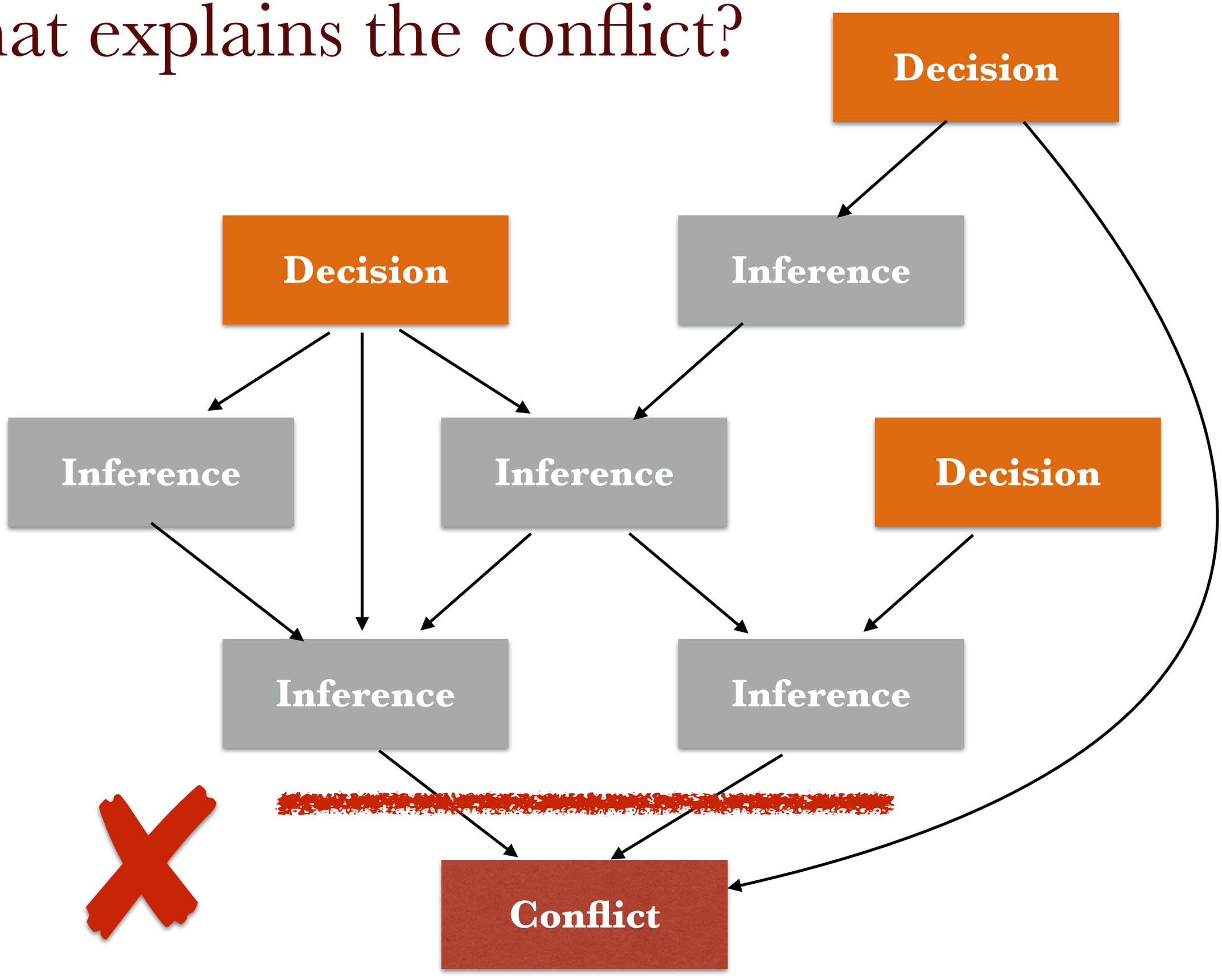
What explains the conflict?



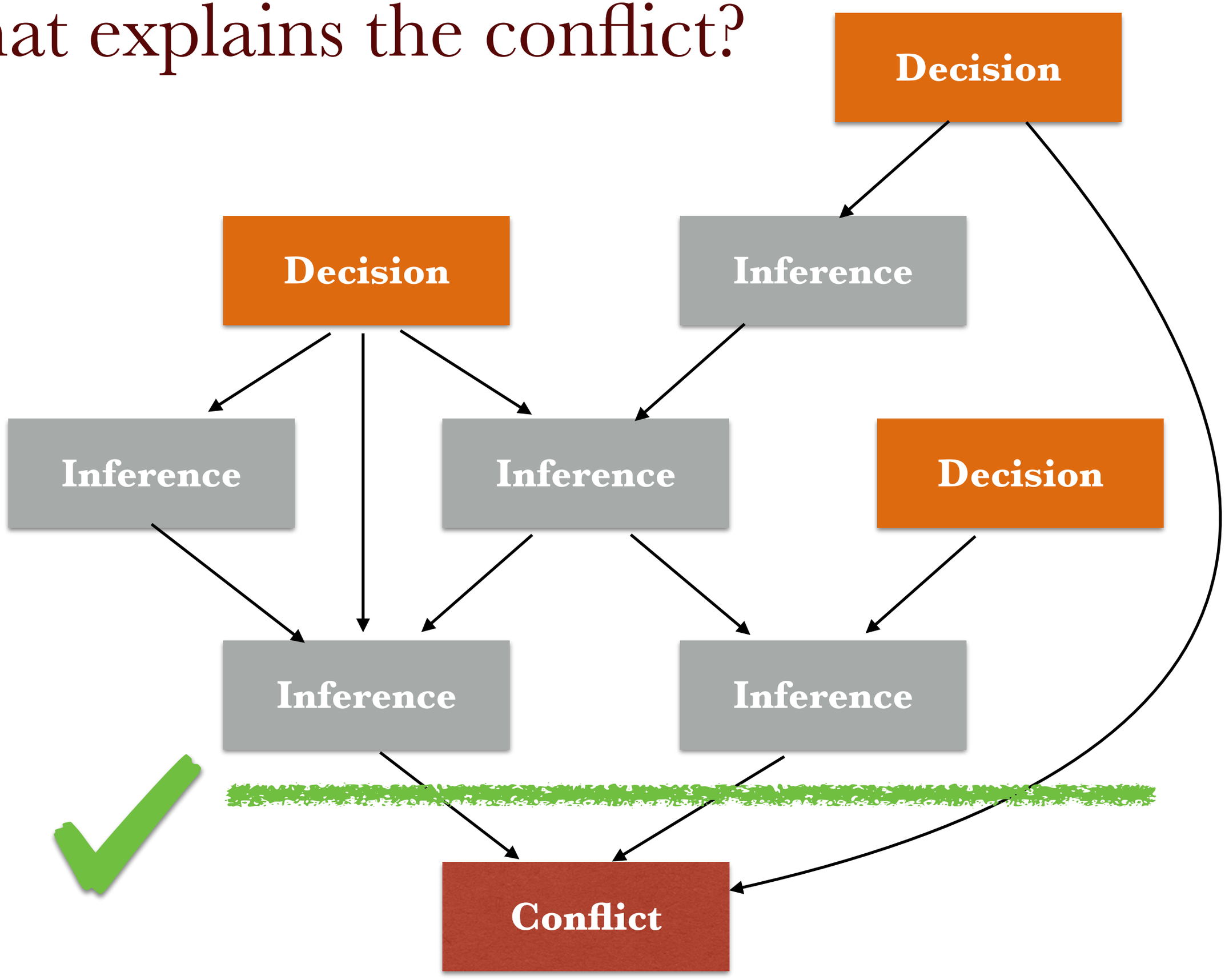
What explains the conflict?



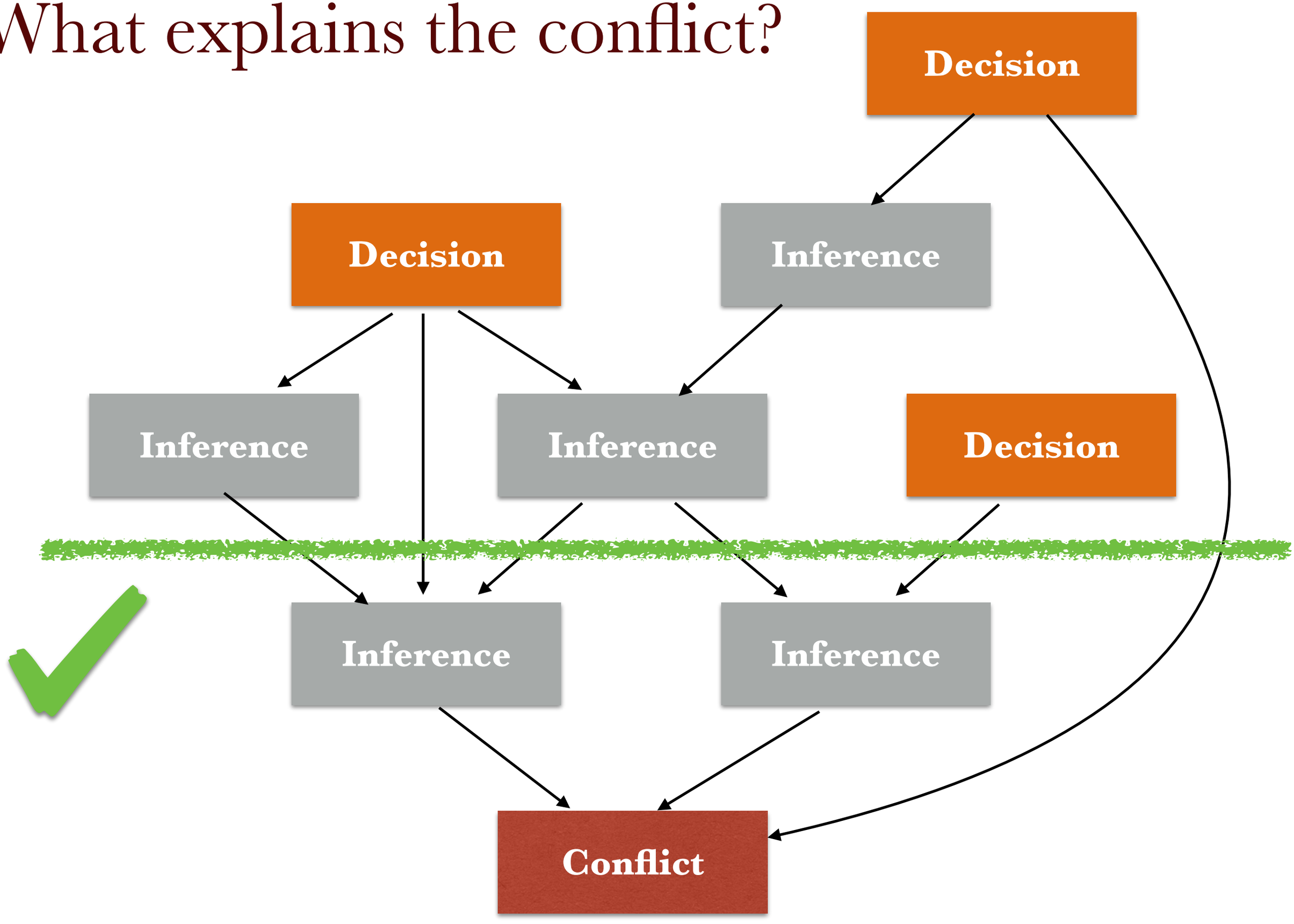
What explains the conflict?



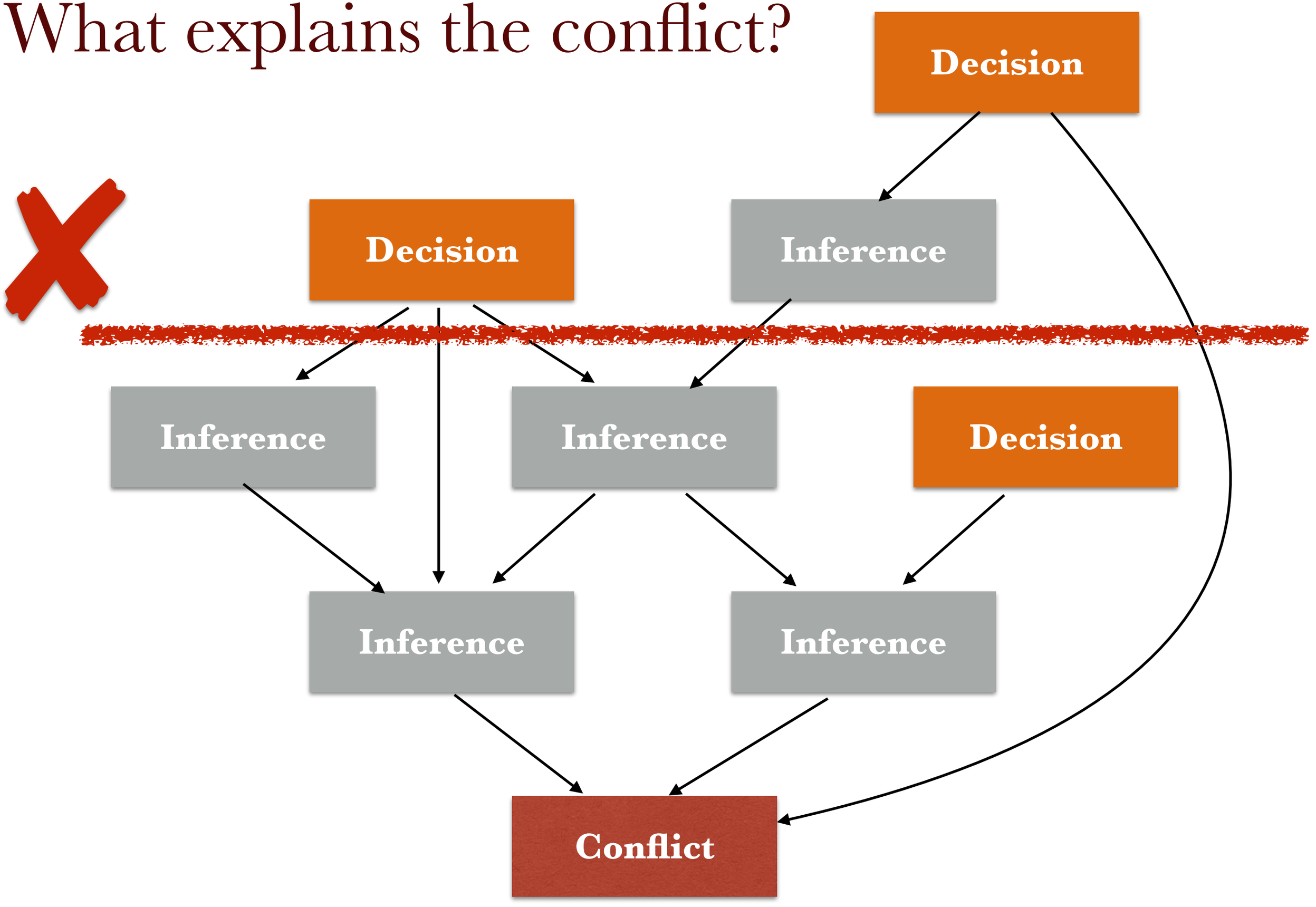
What explains the conflict?



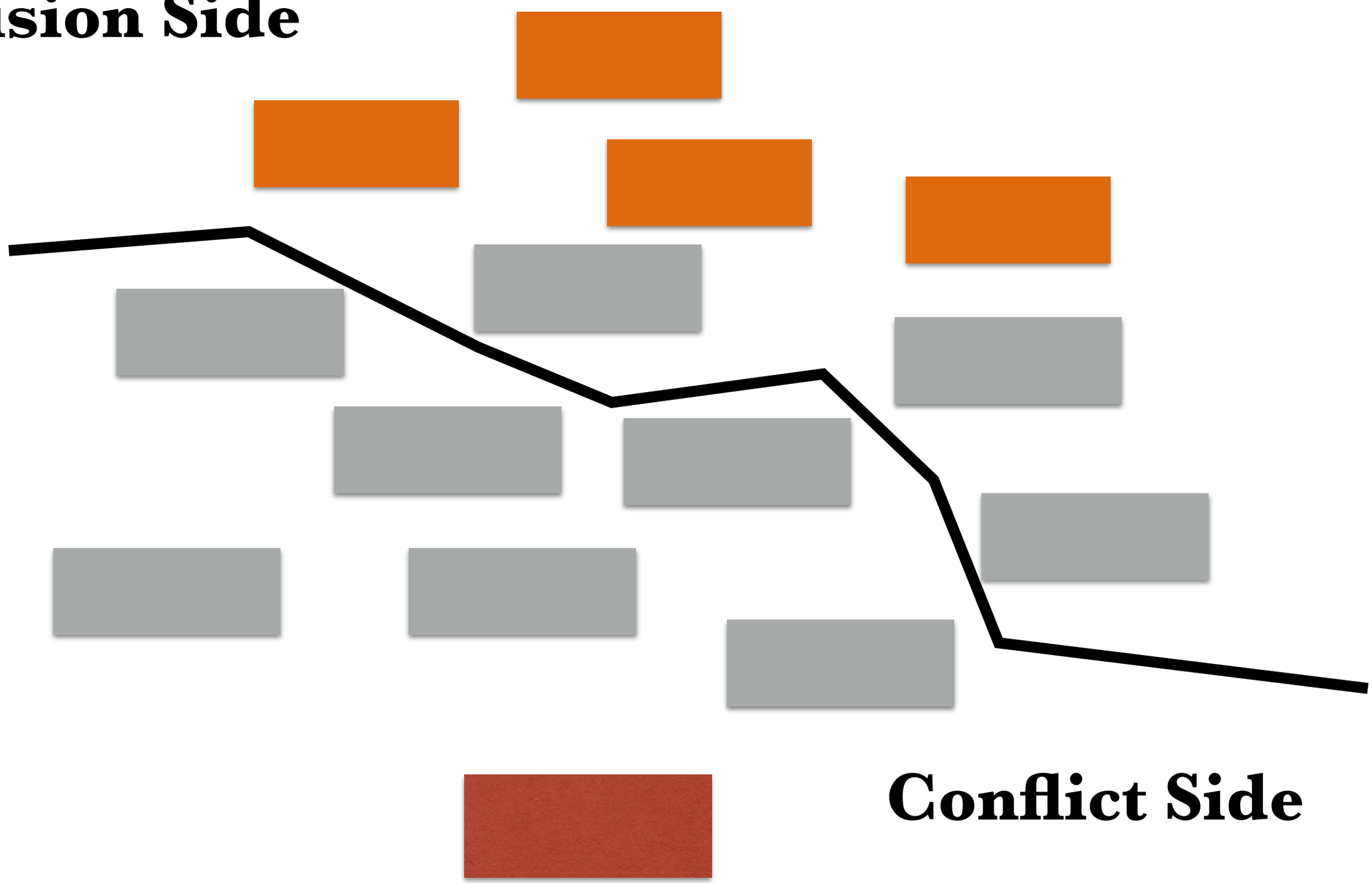
What explains the conflict?



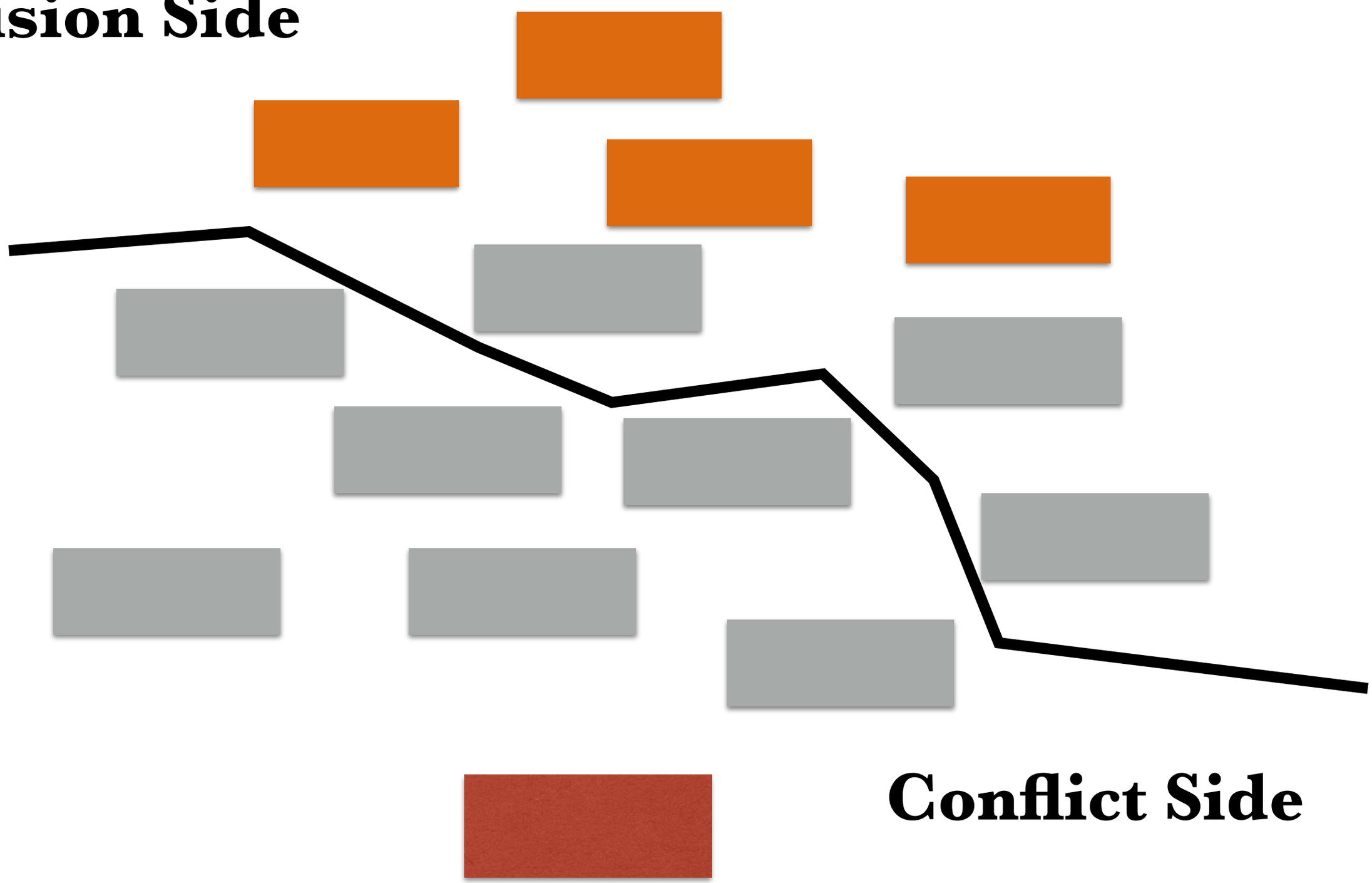
What explains the conflict?



Learn cuts of the implication graph!



Explanatory Cuts



Now how to pick among all these cuts?

$\neg(\neg p_2 \wedge \neg p_3)$ $\{p_2, \neg p_3\}$ $(p_4, 0)@0$

$\neg(\neg p_2)$ $\{p_2, p_1\}$

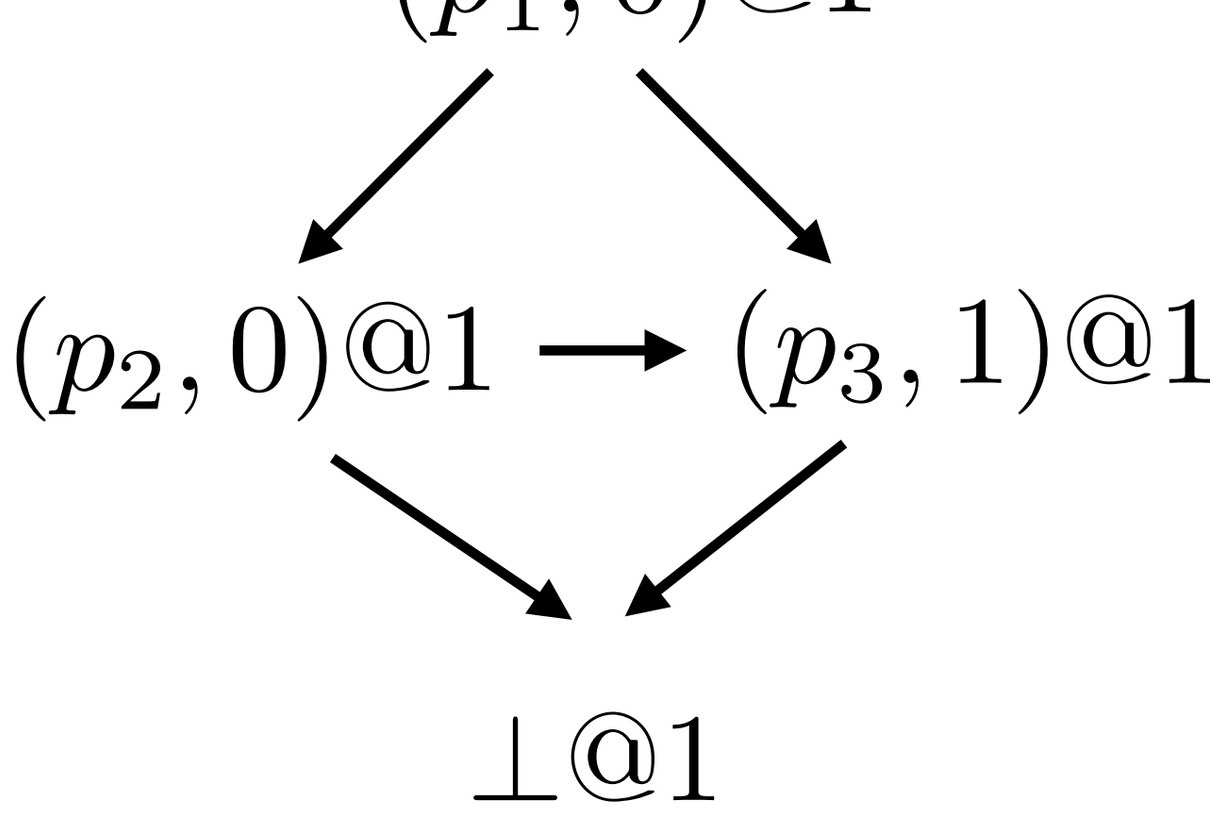
$\neg(\neg p_1)$ $\{p_1\}$ $(p_1, 0)@1$

$\{p_1, \neg p_2\}$

$\{p_3, p_1, p_2\}$

$\{\neg p_3, p_2\}$

$\{\neg p_4\}$



Unique Implication Point

$c_1: \neg x_1 \vee x_2 \vee \neg x_4$

$c_2: \neg x_1 \vee \neg x_2 \vee x_3$

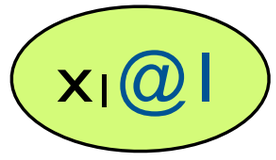
$c_3: \neg x_3 \vee \neg x_4$

$c_4: x_4 \vee x_5 \vee x_6$

$c_5: \neg x_5 \vee x_7$

$c_6: \neg x_6 \vee x_7 \vee \neg x_8$

…



Unique Implication Point

c_1 : $\neg x_1 \vee x_2 \vee \neg x_4$

c_2 : $\neg x_1 \vee \neg x_2 \vee x_3$

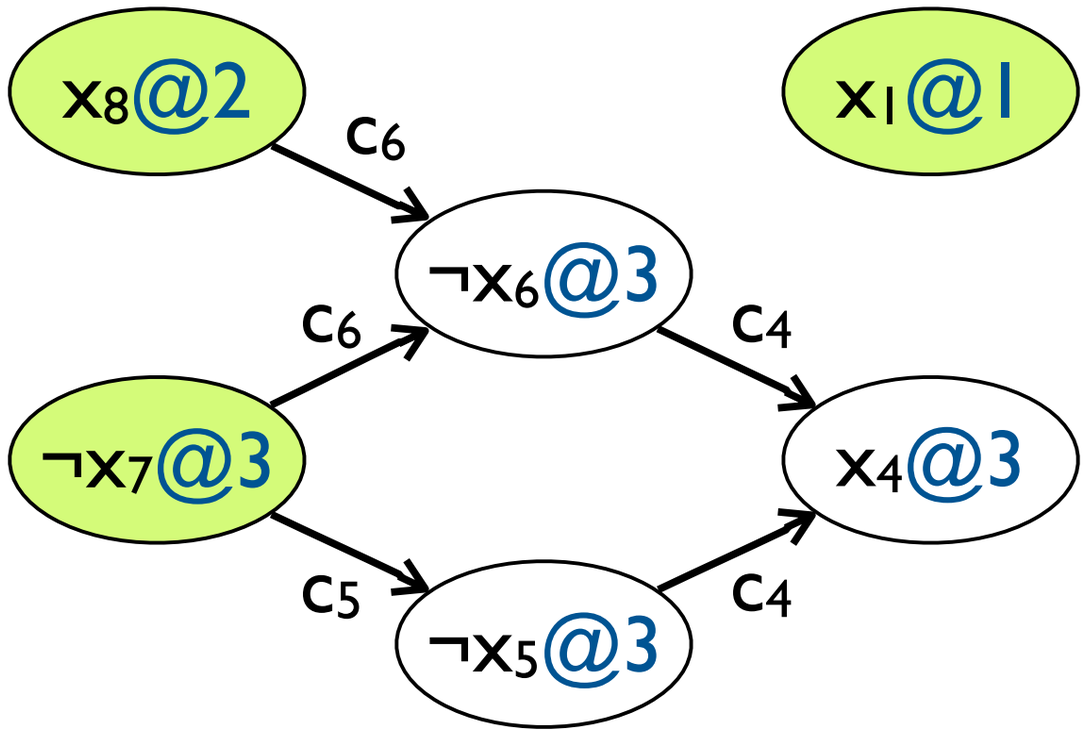
c_3 : $\neg x_3 \vee \neg x_4$

c_4 : $x_4 \vee x_5 \vee x_6$

c_5 : $\neg x_5 \vee x_7$

c_6 : $\neg x_6 \vee x_7 \vee \neg x_8$

…



Unique Implication Point

c_1 : $\neg x_1 \vee x_2 \vee \neg x_4$

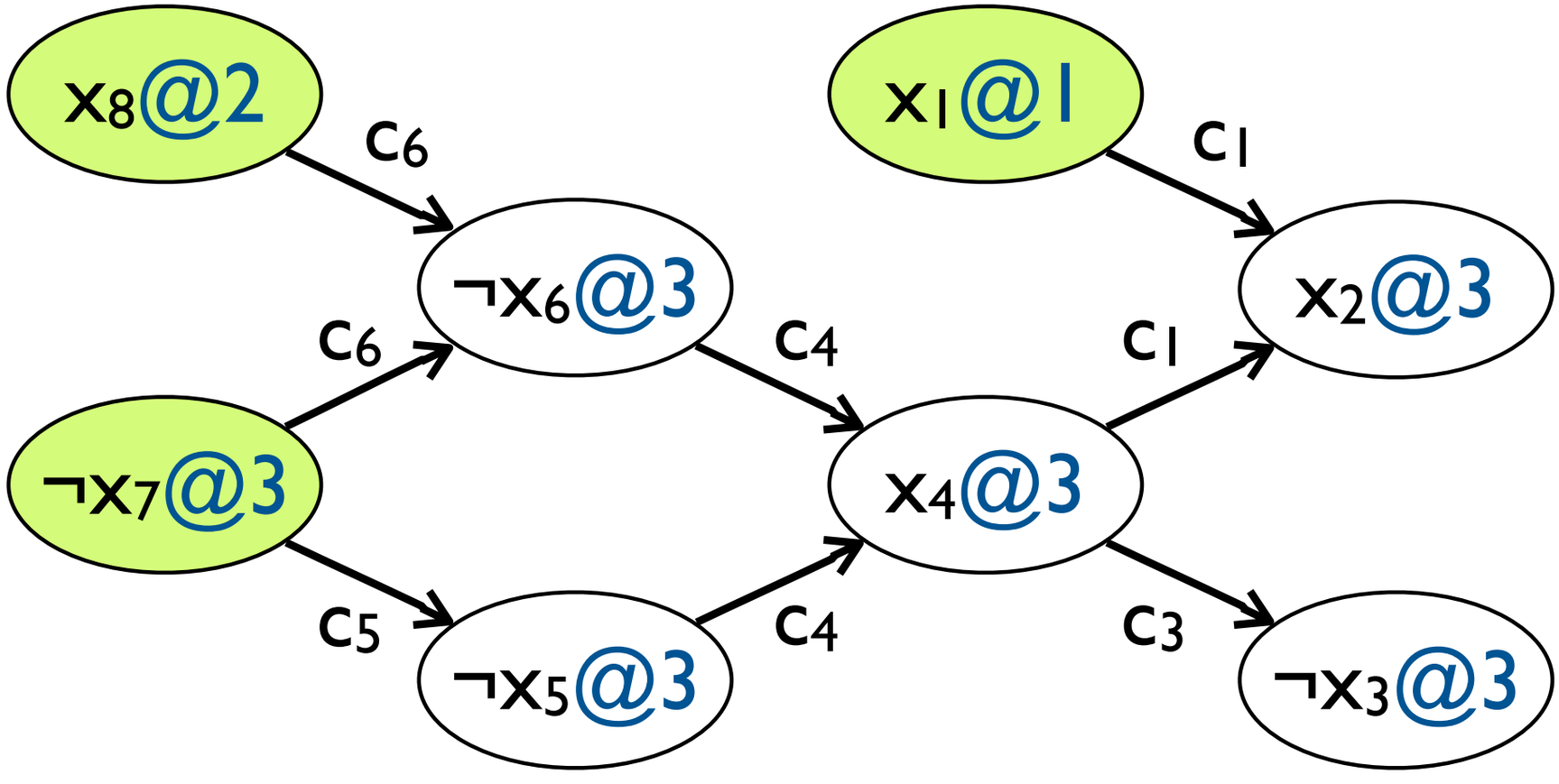
c_2 : $\neg x_1 \vee \neg x_2 \vee x_3$

c_3 : $\neg x_3 \vee \neg x_4$

c_4 : $x_4 \vee x_5 \vee x_6$

c_5 : $\neg x_5 \vee x_7$

c_6 : $\neg x_6 \vee x_7 \vee \neg x_8$



Unique Implication Point

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$

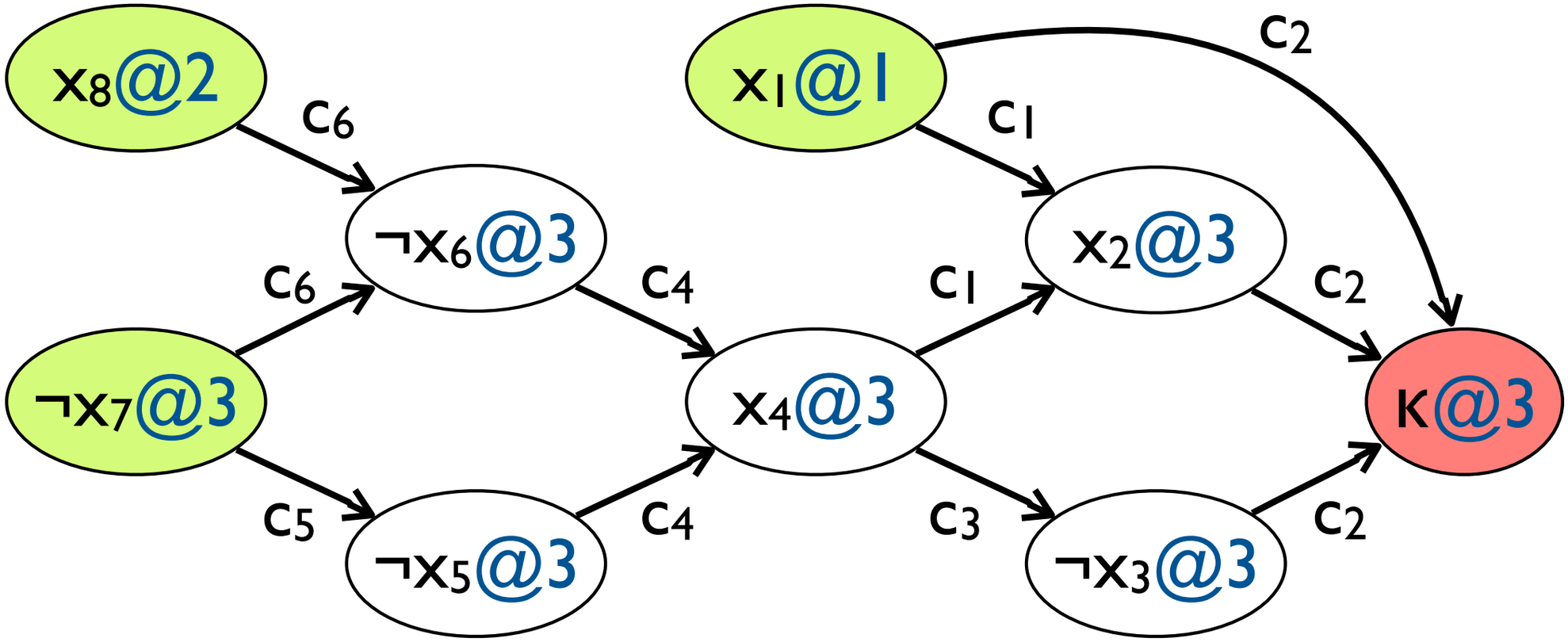
$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$

$c_3 : \neg x_3 \vee \neg x_4$

$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

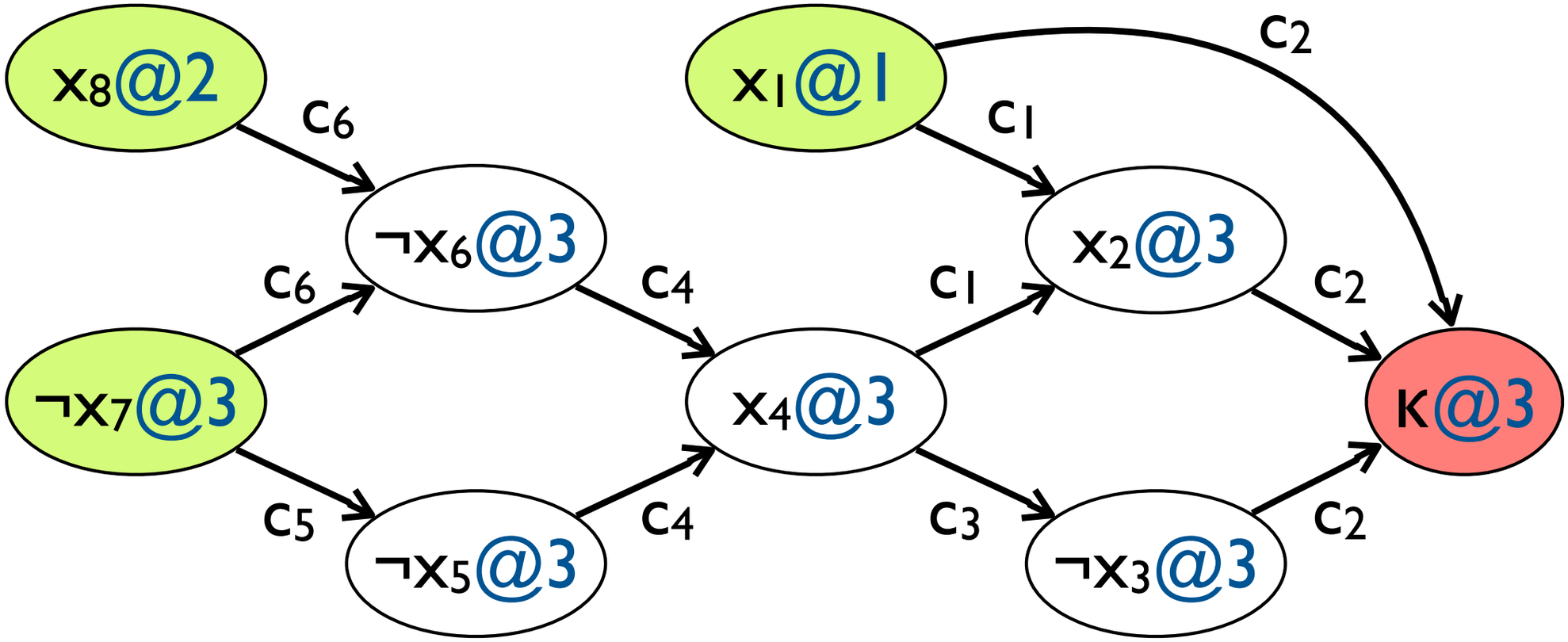
$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$



Unique Implication Point

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$
$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$
$c_3 : \neg x_3 \vee \neg x_4$
$c_4 : x_4 \vee x_5 \vee x_6$
$c_5 : \neg x_5 \vee x_7$
$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

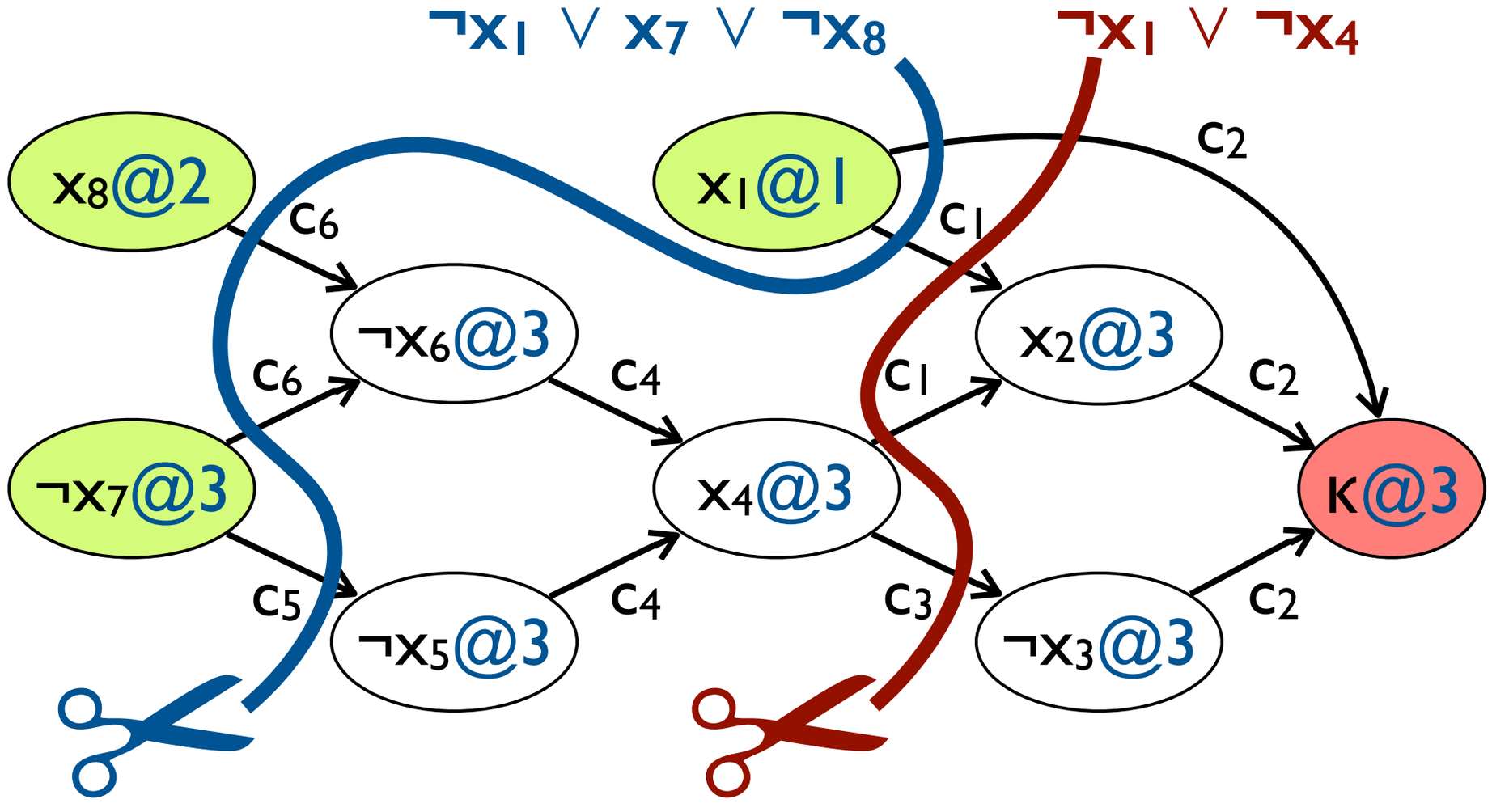
Find a cut in which the highest decision level only appears once.



Unique Implication Point

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$
$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$
$c_3 : \neg x_3 \vee \neg x_4$
$c_4 : x_4 \vee x_5 \vee x_6$
$c_5 : \neg x_5 \vee x_7$
$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

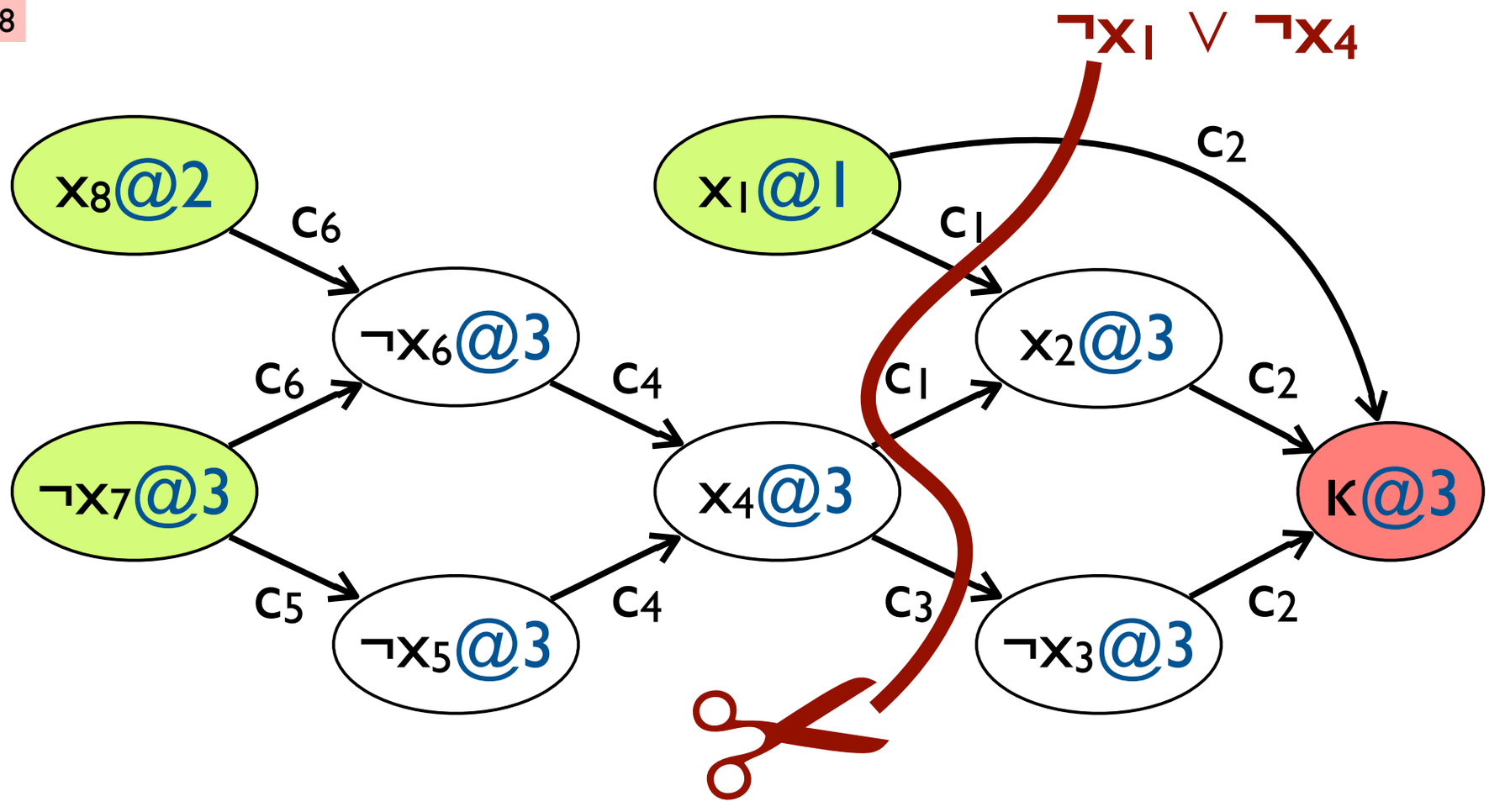
Find a cut in which the highest decision level only appears once.



First Unique Implication Point (1-UIP)

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$
$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$
$c_3 : \neg x_3 \vee \neg x_4$
$c_4 : x_4 \vee x_5 \vee x_6$
$c_5 : \neg x_5 \vee x_7$
$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

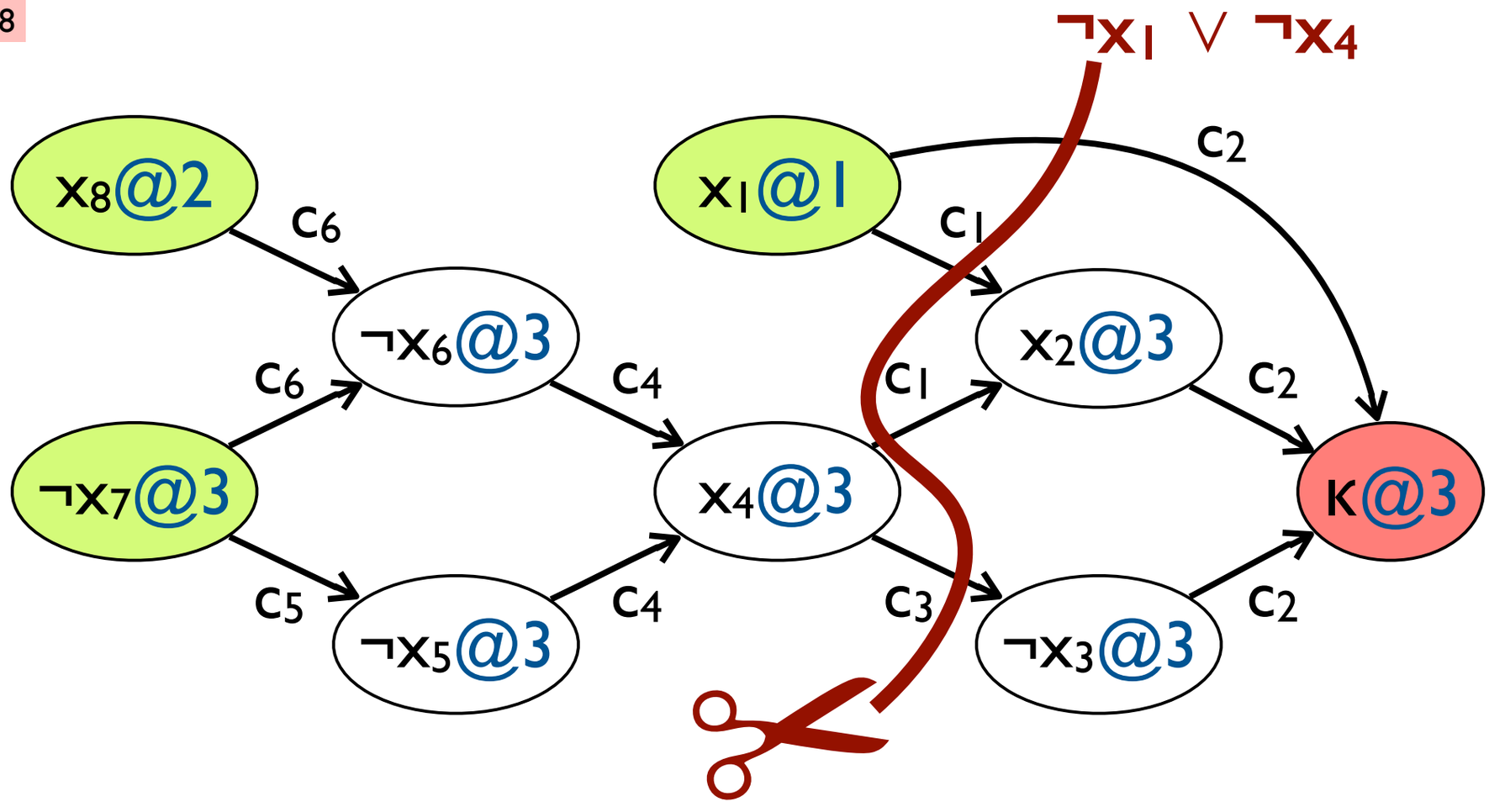
1UIP: the first cut in which the highest decision level only appears once.



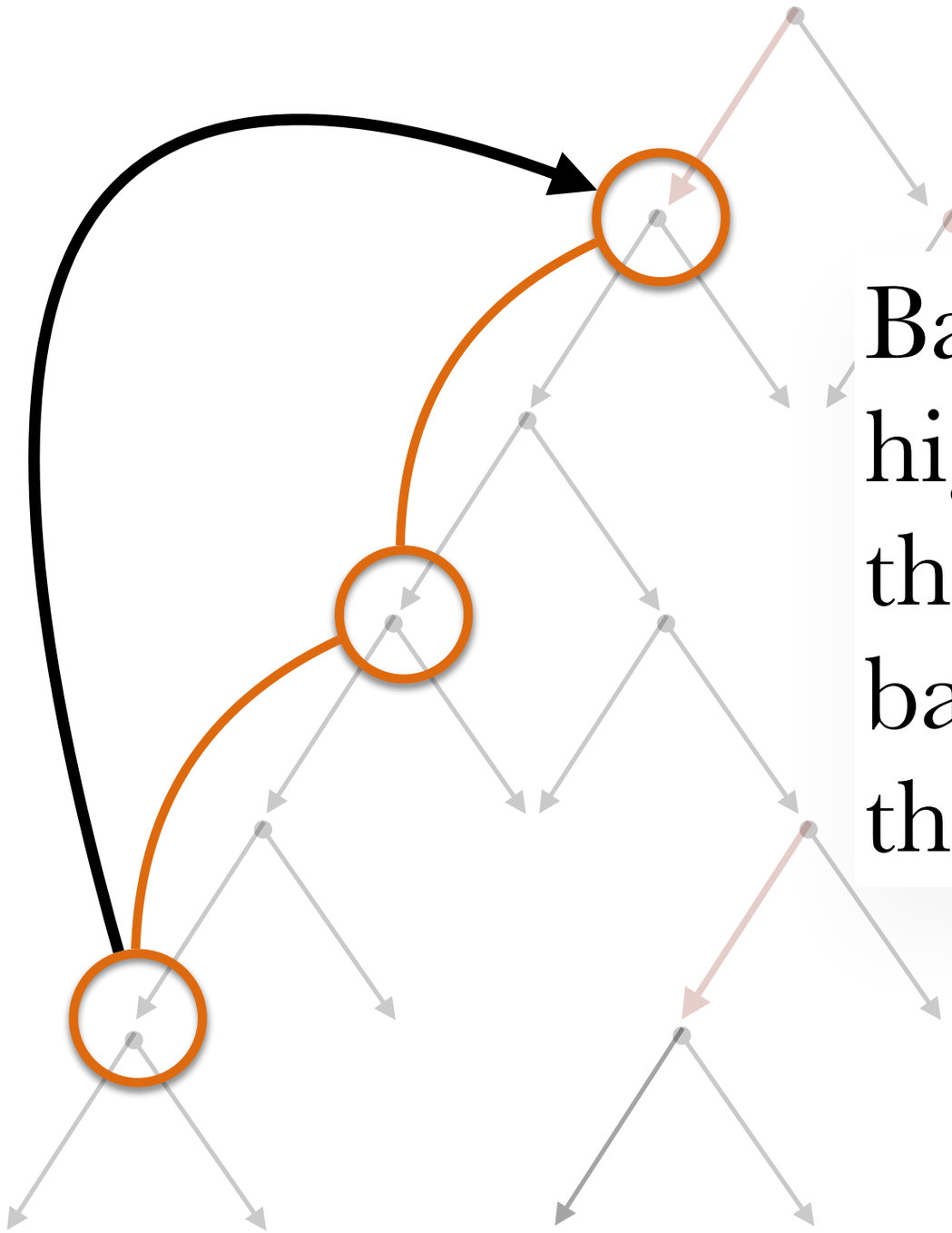
First Unique Implication Point (1-UIP)

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$
$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$
$c_3 : \neg x_3 \vee \neg x_4$
$c_4 : x_4 \vee x_5 \vee x_6$
$c_5 : \neg x_5 \vee x_7$
$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

Backjumping: Jump to the **second** decision level in the 1-UIP!



CDCL: Backjumping



Backtrack to much higher levels than the latest decision, based on analysis of the learned clause.

CDCL: Key Points

- Keep track of how inferences are made with an **implication graph**

- When conflict arises, analyze the conflict through cutting the IMP and find the **1-UIP**

- Backjump to the **2nd highest** decision level in 1-UIP and restart the propagation

With careful engineering, this algorithm can search in spaces with up to 2^1000000 states.

Algorithm 1 CDCL Search

```
1:  function CDCL(φ)
2:      σ ← ∅
3:      dl ← 0
4:      IG ← ∅
5:      ⟨σ, IG⟩ ← UnitPropagation(φ, σ, IG, dl)
6:      if ⊥ ∈ IG then
7:          return unsat
8:      end if
9:      while not AllAssigned(φ, σ) do
10:         ⟨p, b⟩ ← PickBranchingVariable(φ, σ, IG)
11:         σ ← σ ∪ {⟨p, b⟩}
12:         dl ← dl + 1
13:         IG ← ⟨V ∪ {⟨p, b, dl⟩}, E, l⟩
14:         ⟨σ, IG⟩ ← UnitPropagation(φ, σ, IG, dl)
15:         if ⊥ ∈ IG then
16:             dl ← ConflictAnalysis(φ, σ, IG)
17:             if dl < 0 then
18:                 return unsat
19:             else
20:                 ⟨σ, IG⟩ ← Backtrack(φ, σ, dl, IG)
21:             end if
22:         end if
23:     end while
24:     return sat
25: end function
```

Linear Programming

$$\min c^T x$$

s.t.

$$Ax = b$$

$$x \geq 0$$

$$x \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$$

Remember the interpretation: find the right mixture of the column vectors of A to produce b while minimizing total cost of x

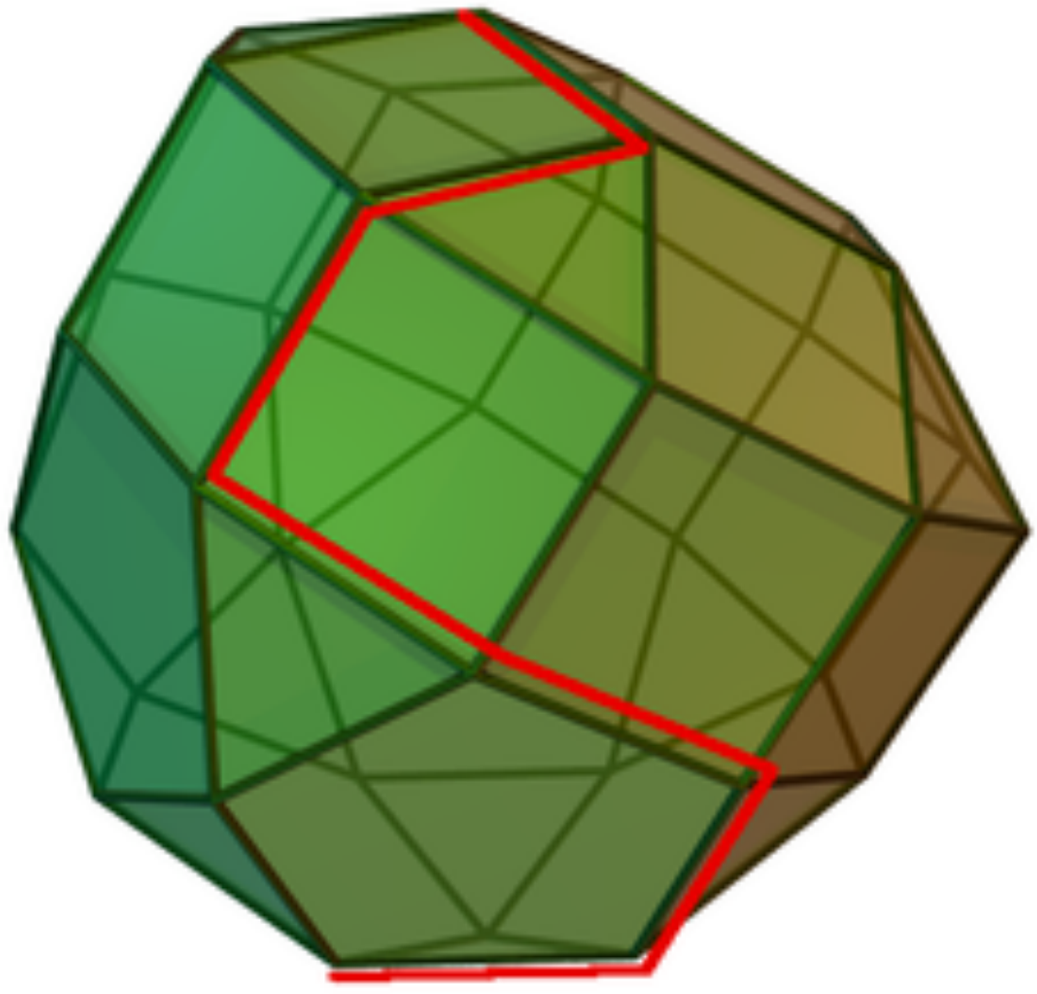
Linear Programming

$$\min c^T x$$

s.t.

$$Ax = b$$

$$x \geq 0$$



Linear Programming

$$\hat{A}_1 x_1 + \ldots + \hat{A}_m x_m + \hat{A}_{m+1} x_{m+1} + \cdots + \hat{A}_n x_n = \hat{b}$$

$$\begin{aligned}
\hat{a}_1 x_1 + \quad & \hat{a}_{1,m+1} x_{m+1} + \cdots + \hat{a}_{1,n} x_n = \hat{b}_1 \\
\hat{a}_2 x_2 + \quad & \hat{a}_{2,m+1} x_{m+1} + \cdots + \hat{a}_{2,n} x_n = \hat{b}_2 \\
& \vdots \\
\hat{a}_m x_m + \quad & \hat{a}_{m,m+1} x_{m+1} + \cdots + \hat{a}_{m,n} x_n = \hat{b}_m
\end{aligned}$$

basic columns and basic variables

Linear Programming

$$\hat{A}_1 x_1 + \ldots + \hat{A}_m x_m + \hat{A}_{m+1} x_{m+1} + \cdots + \hat{A}_n x_n = \hat{b}$$

$$\begin{aligned}
\hat{a}_1 x_1 + & \quad \hat{a}_{1,m+1} x_{m+1} + \cdots + \hat{a}_{1,n} x_n = \hat{b}_1 \\
\hat{a}_2 x_2 + & \quad \hat{a}_{2,m+1} x_{m+1} + \cdots + \hat{a}_{2,n} x_n = \hat{b}_2 \\
& \quad \vdots \\
\hat{a}_m x_m + & \quad \hat{a}_{m,m+1} x_{m+1} + \cdots + \hat{a}_{m,n} x_n = \hat{b}_m
\end{aligned}$$

nonbasic columns and nonbasic variables

Linear Programming

$$\hat{a}_1 x_1 + \qquad\qquad \hat{a}_{1,m+1} x_{m+1} + \cdots + \hat{a}_{1,n} x_n = \hat{b}_1$$

$$\hat{a}_2 x_2 + \qquad\qquad \hat{a}_{2,m+1} x_{m+1} + \cdots + \hat{a}_{2,n} x_n = \hat{b}_2$$

$$\vdots$$

$$\hat{a}_m x_m + \hat{a}_{m,m+1} x_{m+1} + \cdots + \hat{a}_{m,n} x_n = \hat{b}_m$$

$$x_i = \hat{b}_i / \hat{a}_i \qquad\qquad x_j = 0$$

$$i \in \{1,...m\} \qquad j \in \{m+1,...,n\}$$

Mixed Integer Linear Programming

$$\min c^T x + d^T y$$

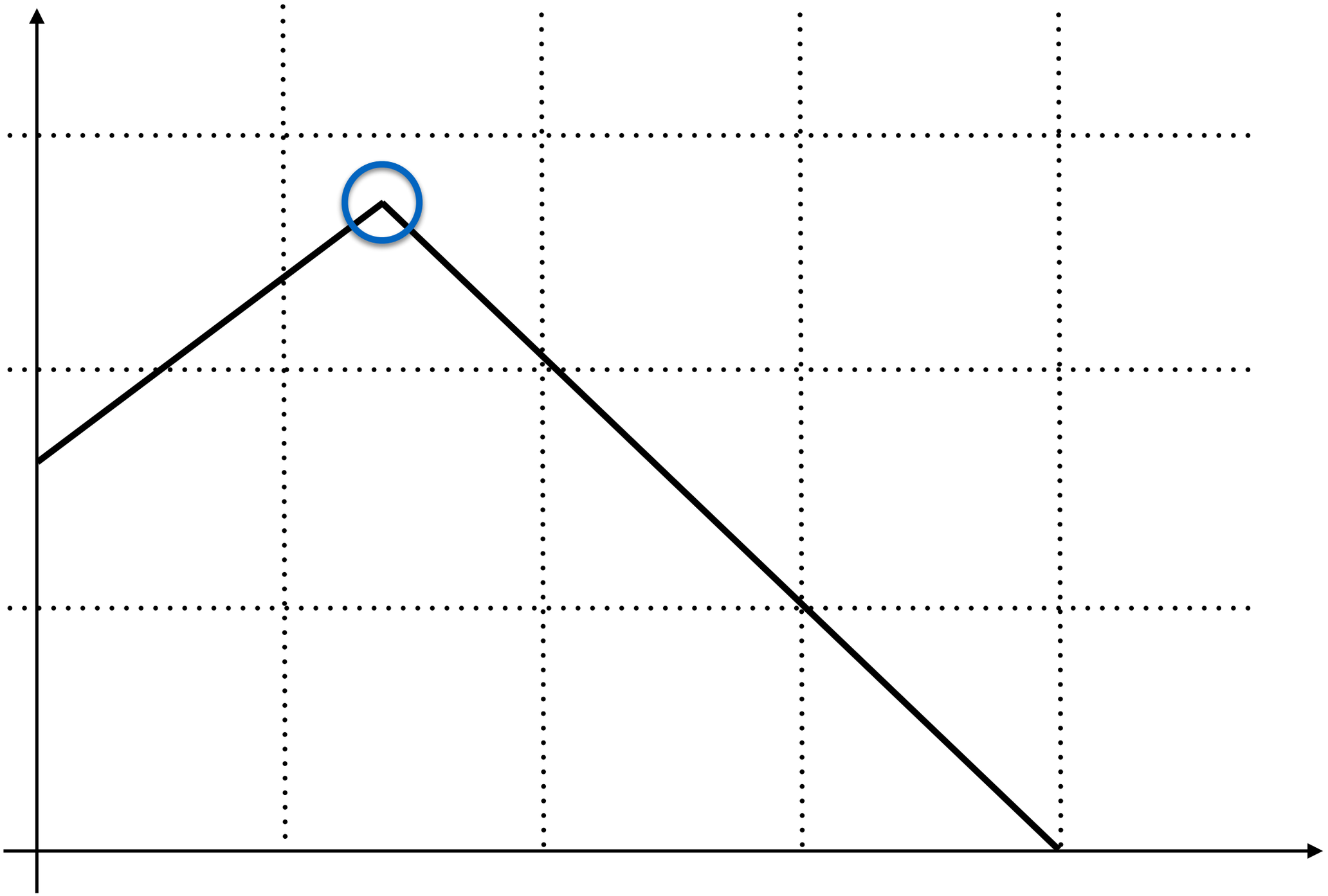
s.t.

$$Ax + By = b$$

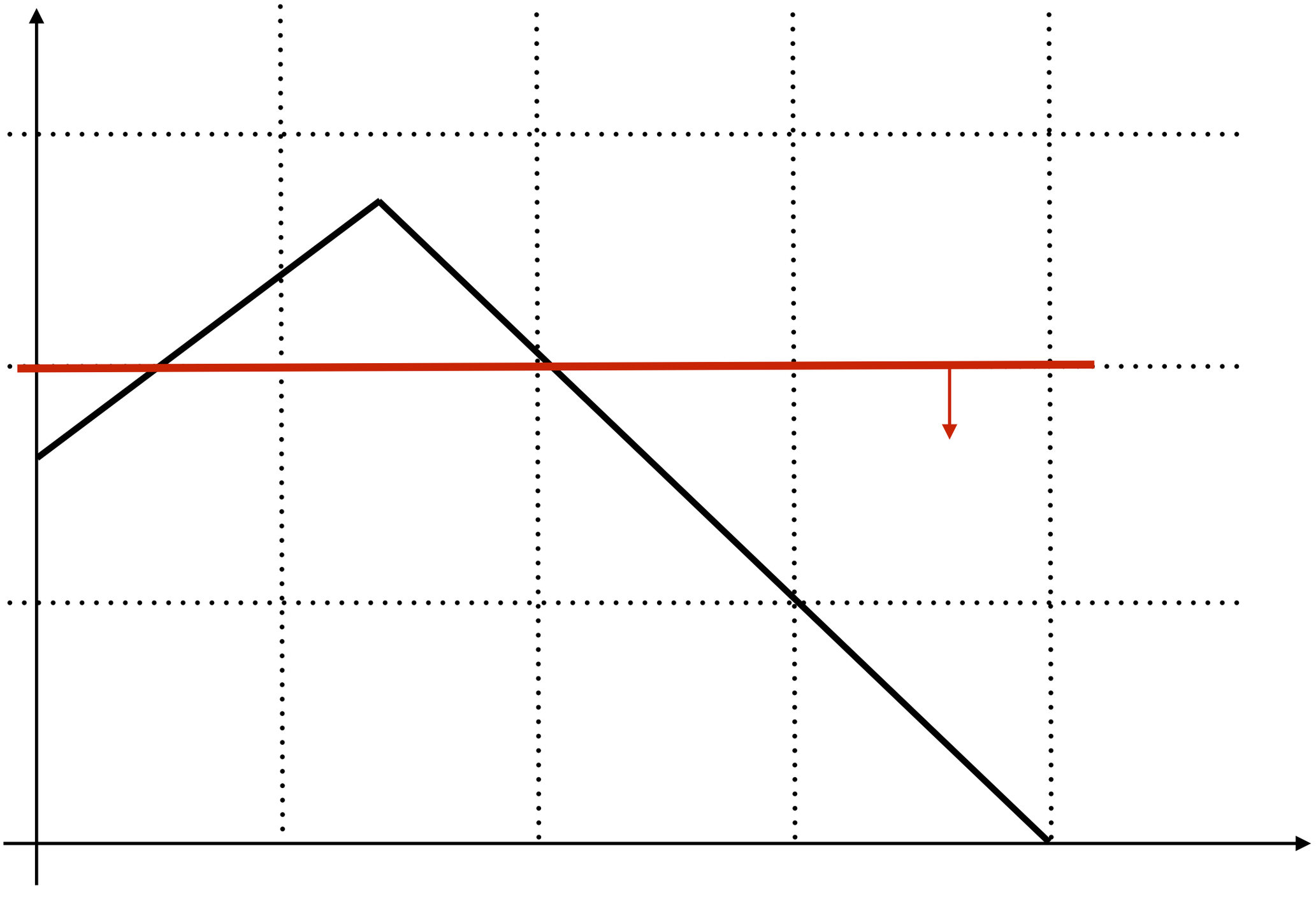
$$x, y \geq 0$$

$$x \in \mathbb{Z}^n, y \in \mathbb{R}^k, A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times k}, b \in \mathbb{R}^m$$

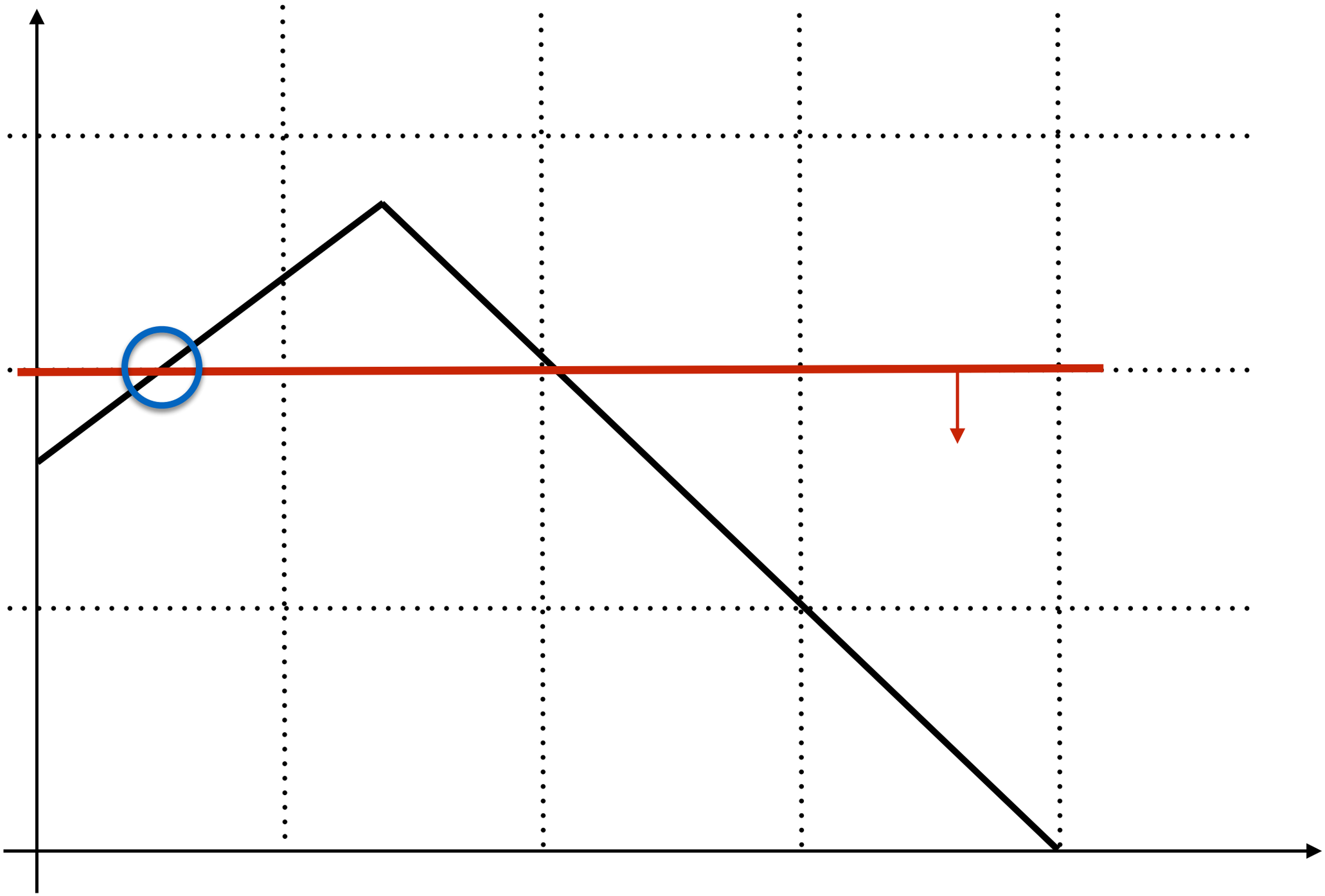
Cutting Planes



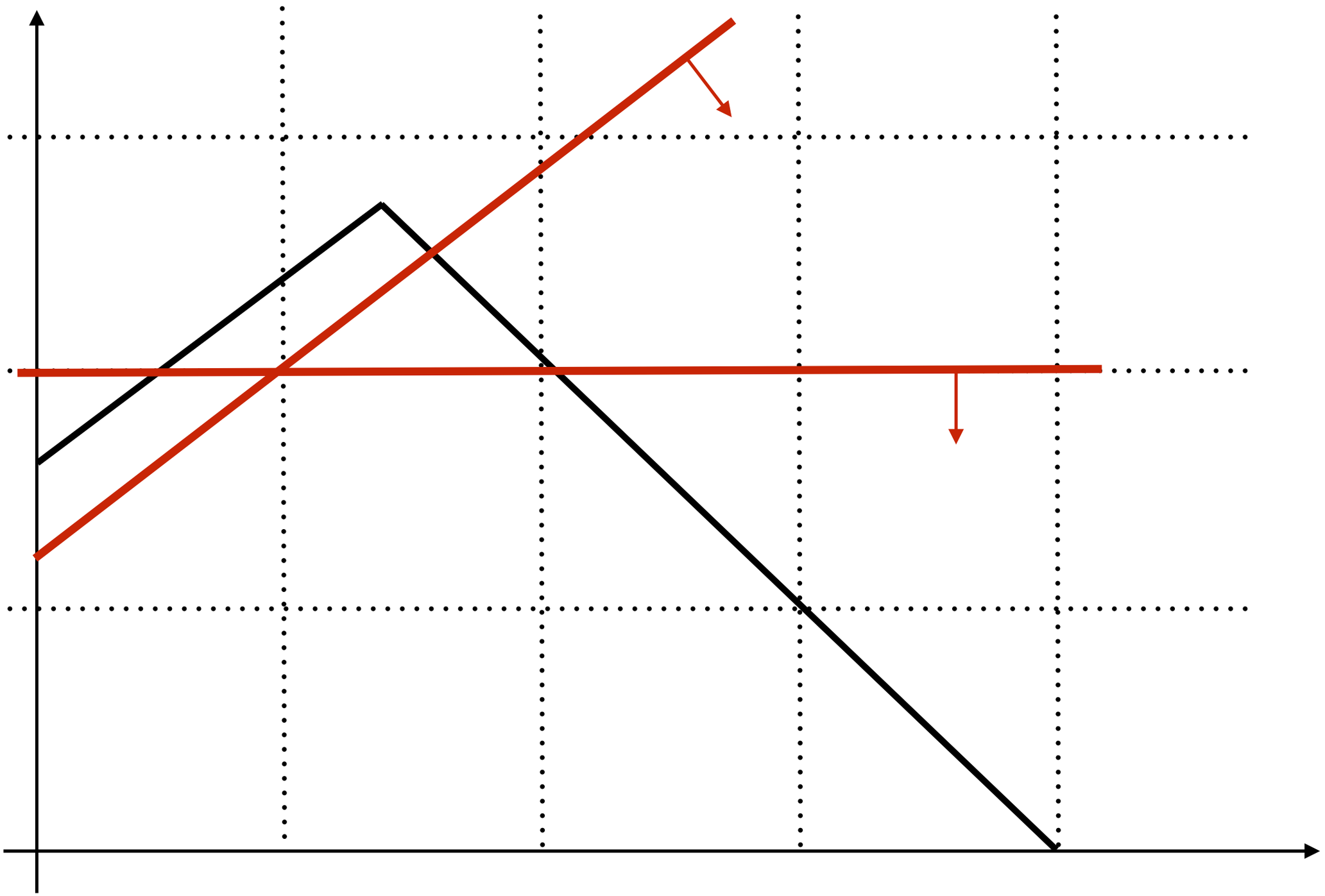
Cutting Planes



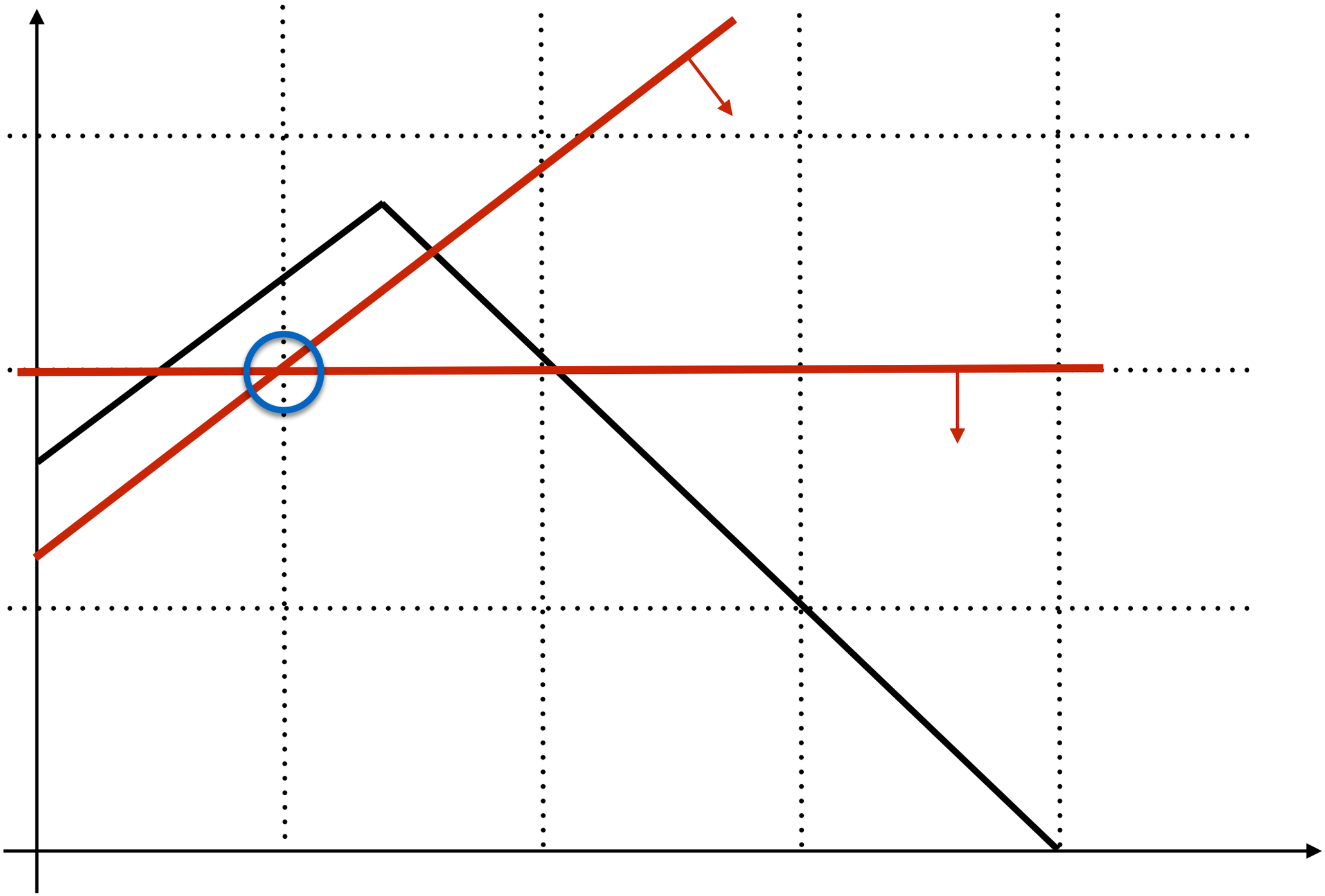
Cutting Planes



Cutting Planes



Cutting Planes



Cutting Planes

$$\hat{a}_1 x_1 + \qquad\qquad \hat{a}_{1,m+1} x_{m+1} + \cdots + \hat{a}_{1,n} x_n = \hat{b}_1$$

$$\hat{a}_2 x_2 + \qquad\qquad \hat{a}_{2,m+1} x_{m+1} + \cdots + \hat{a}_{2,n} x_n = \hat{b}_2$$

$$\vdots$$

$$\hat{a}_m x_m + \hat{a}_{m,m+1} x_{m+1} + \cdots + \hat{a}_{m,n} x_n = \hat{b}_m$$

$$x_i = \hat{b}_i / \hat{a}_i \qquad\qquad x_j = 0$$

$$i \in \{1,...m\} \qquad j \in \{m+1,...,n\}$$

Cutting Planes

Pick any row where the integer constraints are violated on its basic variable

$$\hat{a}_1 x_1 + \qquad\qquad \hat{a}_{1,m+1} x_{m+1} + \cdots + \hat{a}_{1,n} x_n = \hat{b}_1$$

$$\hat{a}_2 x_2 + \qquad\qquad \hat{a}_{2,m+1} x_{m+1} + \cdots + \hat{a}_{2,n} x_n = \hat{b}_2$$

$$\vdots$$

$$\hat{a}_m x_m + \hat{a}_{m,m+1} x_{m+1} + \cdots + \hat{a}_{m,n} x_n = \hat{b}_m$$

Cutting Planes

Pick any row where the integer constraints are violated on its basic variable

$$\hat{a}_2 x_2 + \hat{a}_{2,m+1} x_{m+1} + \cdots + \hat{a}_{2,n} x_n = \hat{b}_2$$

$$x_i + \sum_j a_{ij} x_j = b_i$$

Cutting Planes

Pick any row where the integer constraints are violated on its basic variable

$$x_i + \sum_j a_{ij} x_j = b_i$$

Because of nonnegative ranges on all variables

$$x_i + \sum_j \lfloor a_{ij} \rfloor x_j \leq x_i + \sum_j a_{ij} x_j$$

Cutting Planes

Pick any row where the integer constraints are violated on its basic variable

$$x_i + \sum_j a_{ij} x_j = b_i$$

Because of nonnegative ranges on all variables

$$x_i + \sum_j \lfloor a_{ij} \rfloor x_j \leq b_i$$

Cutting Planes

Pick any row where the integer constraints are violated on its basic variable

$$x_i + \sum_j a_{ij} x_j = b_i$$

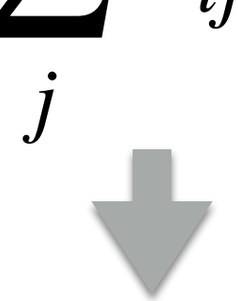
Because of nonnegative ranges and integer constraints

$$x_i + \sum_j \lfloor a_{ij} \rfloor x_j \leq \lfloor b_i \rfloor$$

Cutting Planes

Pick any row where the integer constraints are violated on its basic variable and generate an additional constraint

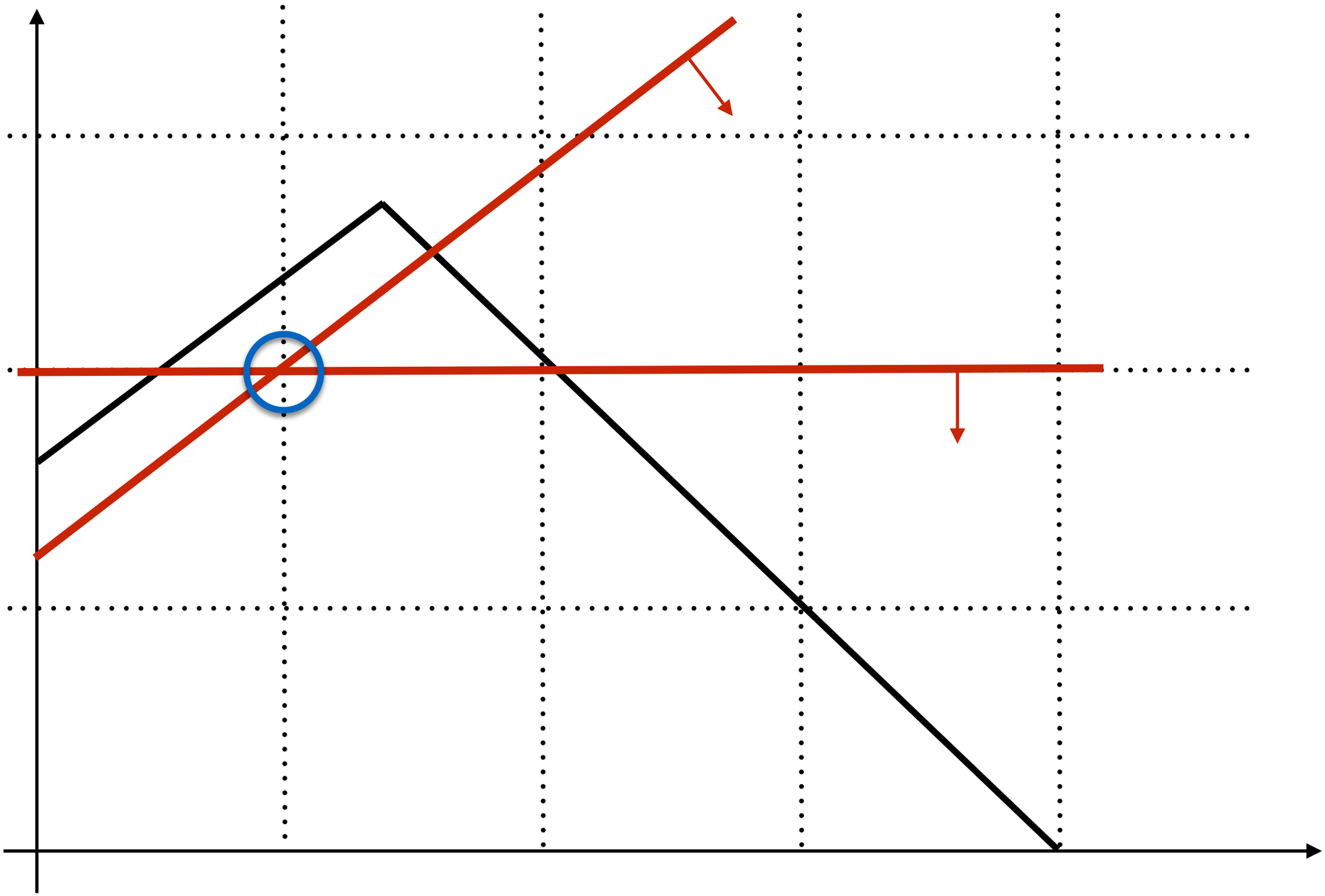
$$x_i + \sum_j a_{ij} x_j = b_i$$



$$x_i + \sum_j \lfloor a_{ij} \rfloor x_j \leq \lfloor b_i \rfloor$$

This new constraint gives us a new (Gomory) cutting plane

Cutting Planes



Example

$$\text{minimize } x_1 - 2x_2$$

s.t.

$$-4x_1 + 6x_2 \leq 9$$

$$x_1 + x_2 \leq 4$$

$$x_i \in \mathbb{N}$$

Example

$$\text{minimize } x_1 - 2x_2$$

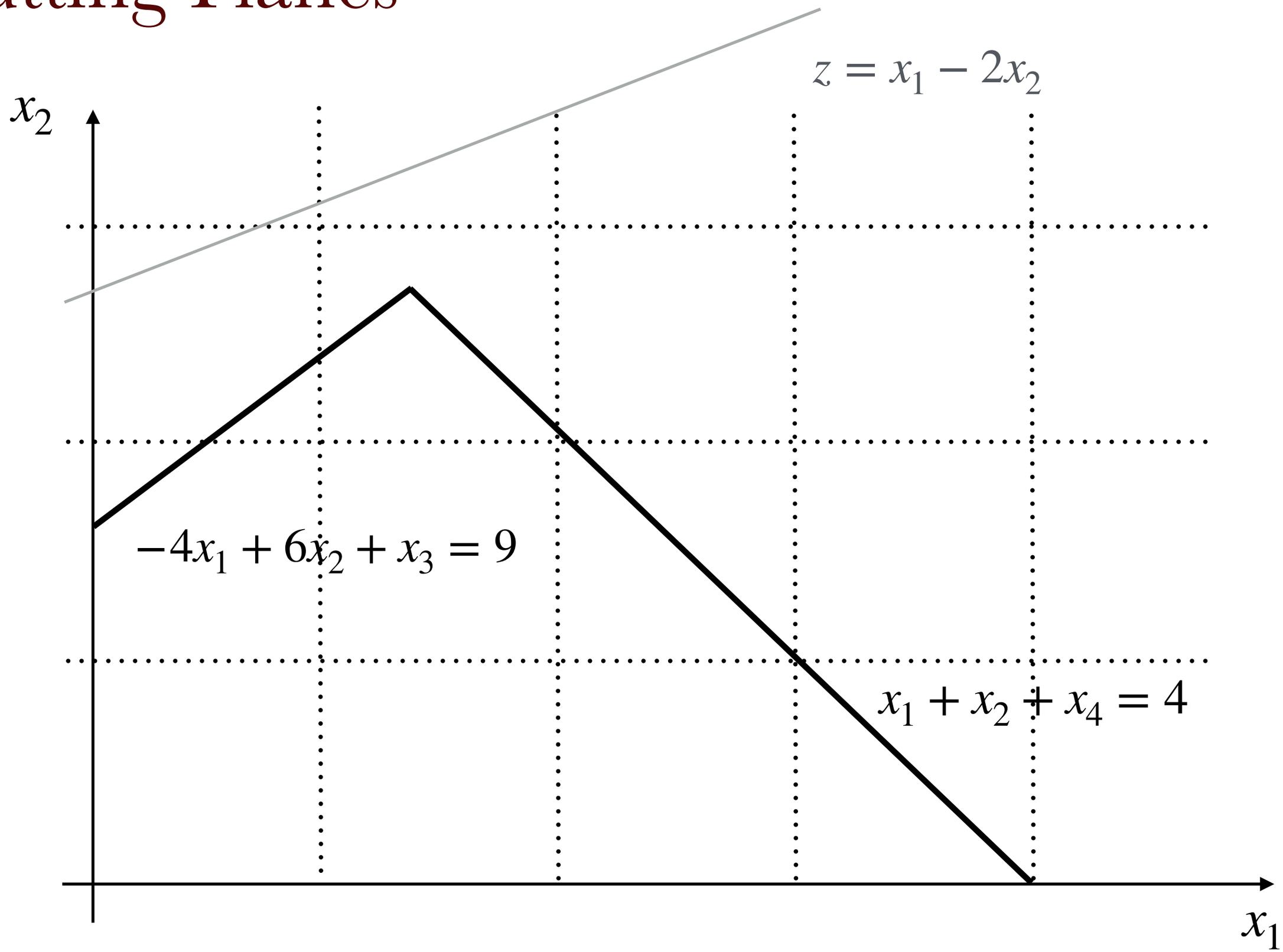
s.t.

$$-4x_1 + 6x_2 + x_3 = 9$$

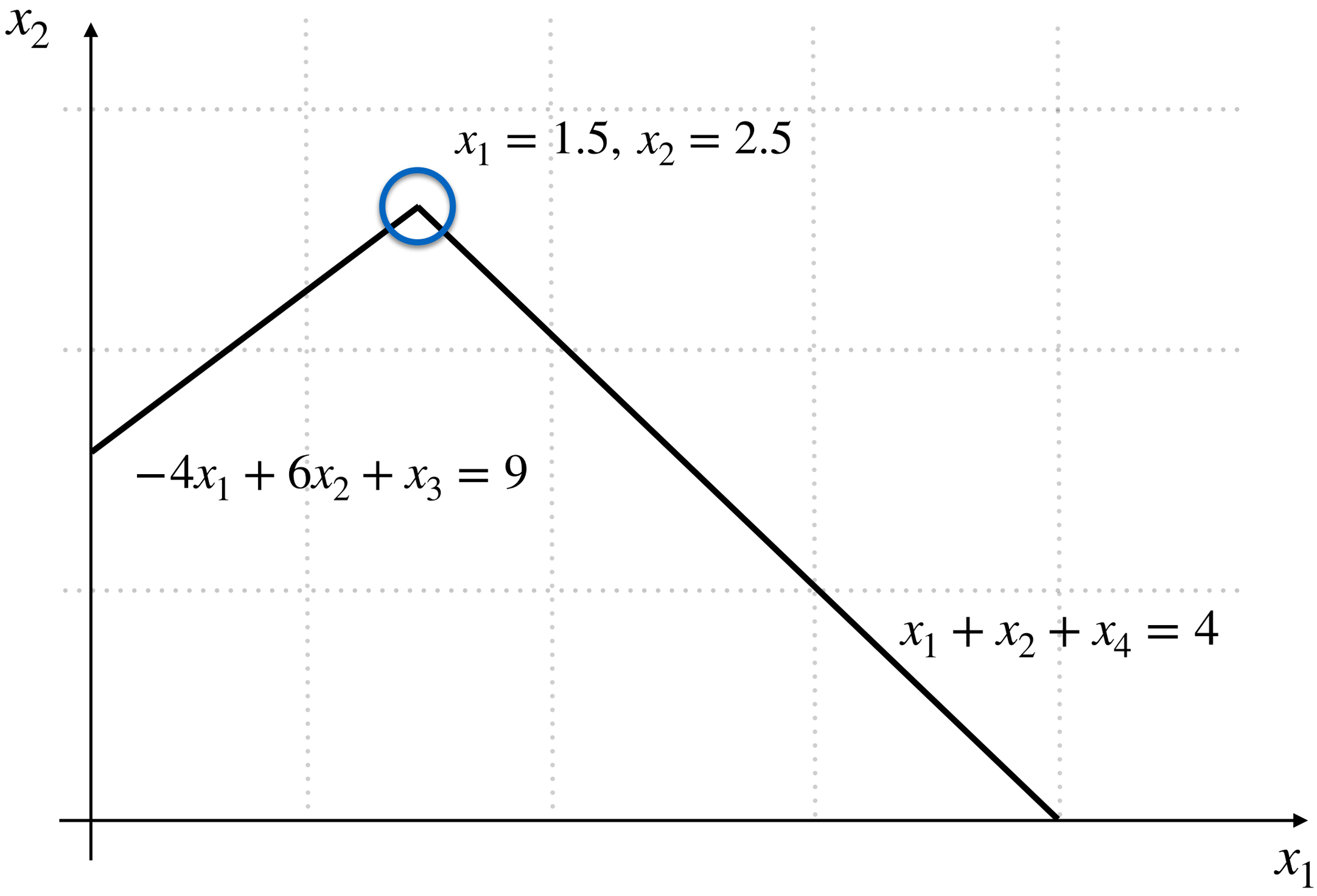
$$x_1 + x_2 + x_4 = 4$$

$$x_i \in \mathbb{N}$$

Cutting Planes



Cutting Planes



Example

Solving with simplex, we get the optimal tableau

$$x_1 - \frac{1}{10}x_3 + \frac{3}{5}x_4 = \frac{15}{10},$$

$$x_2 + \frac{1}{10}x_3 + \frac{2}{5}x_4 = \frac{25}{10}.$$

Example

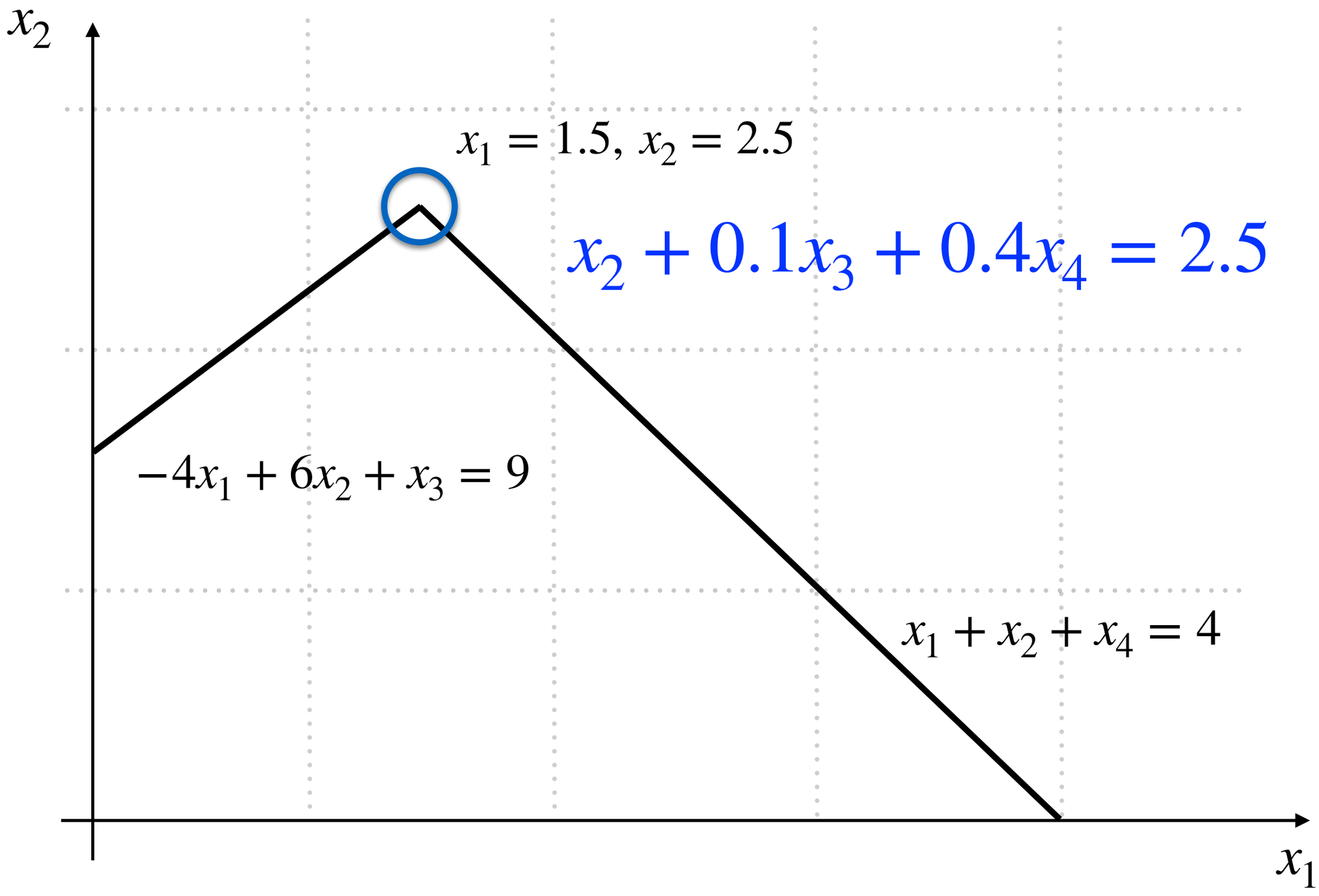
Solving with simplex, we get the optimal tableau

$$x_1 - \frac{1}{10}x_3 + \frac{3}{5}x_4 = \frac{15}{10},$$

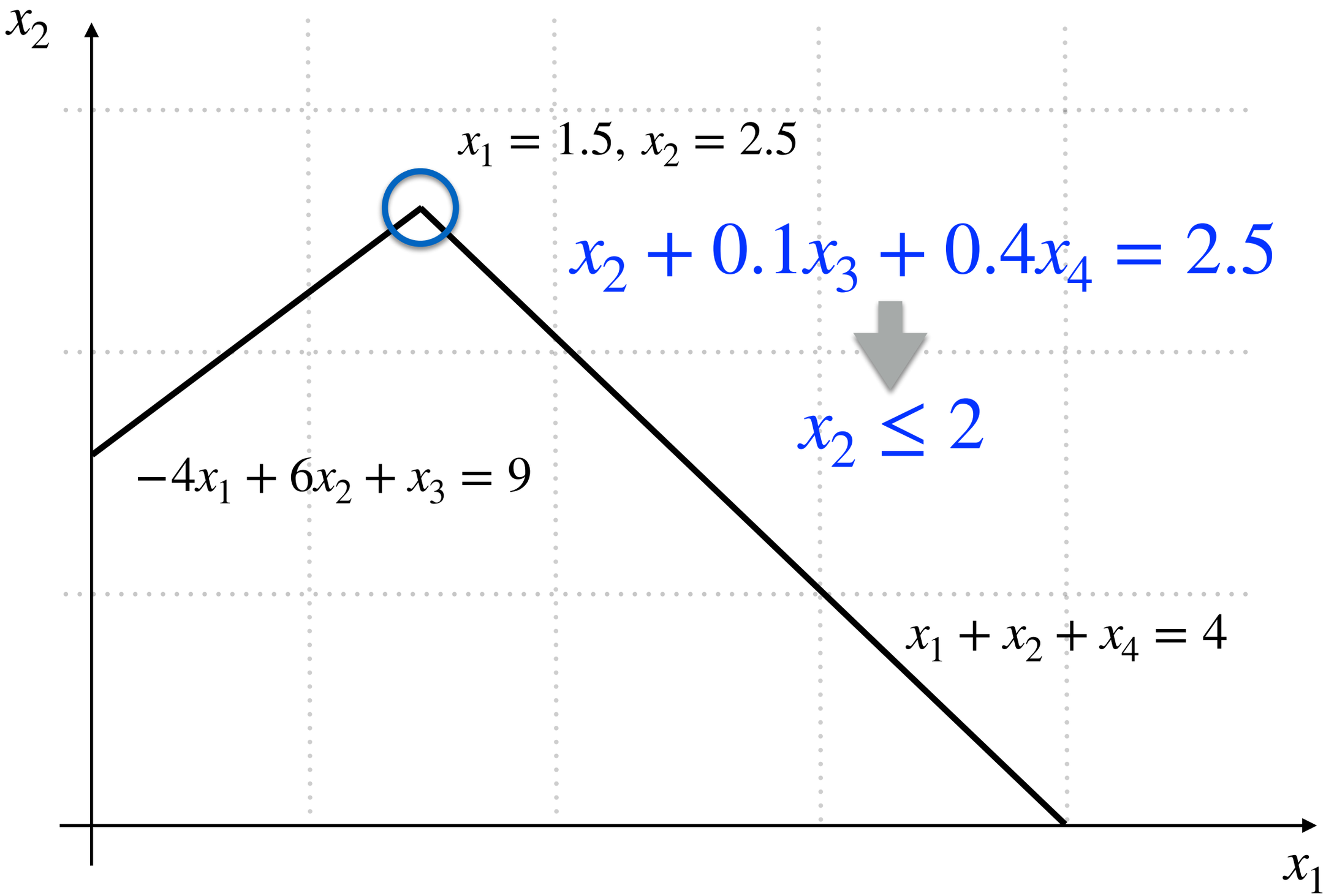
$$x_2 + \frac{1}{10}x_3 + \frac{2}{5}x_4 = \frac{25}{10}.$$

Optimal solution: $x_1 = 1.5,\ x_2 = 2.5$

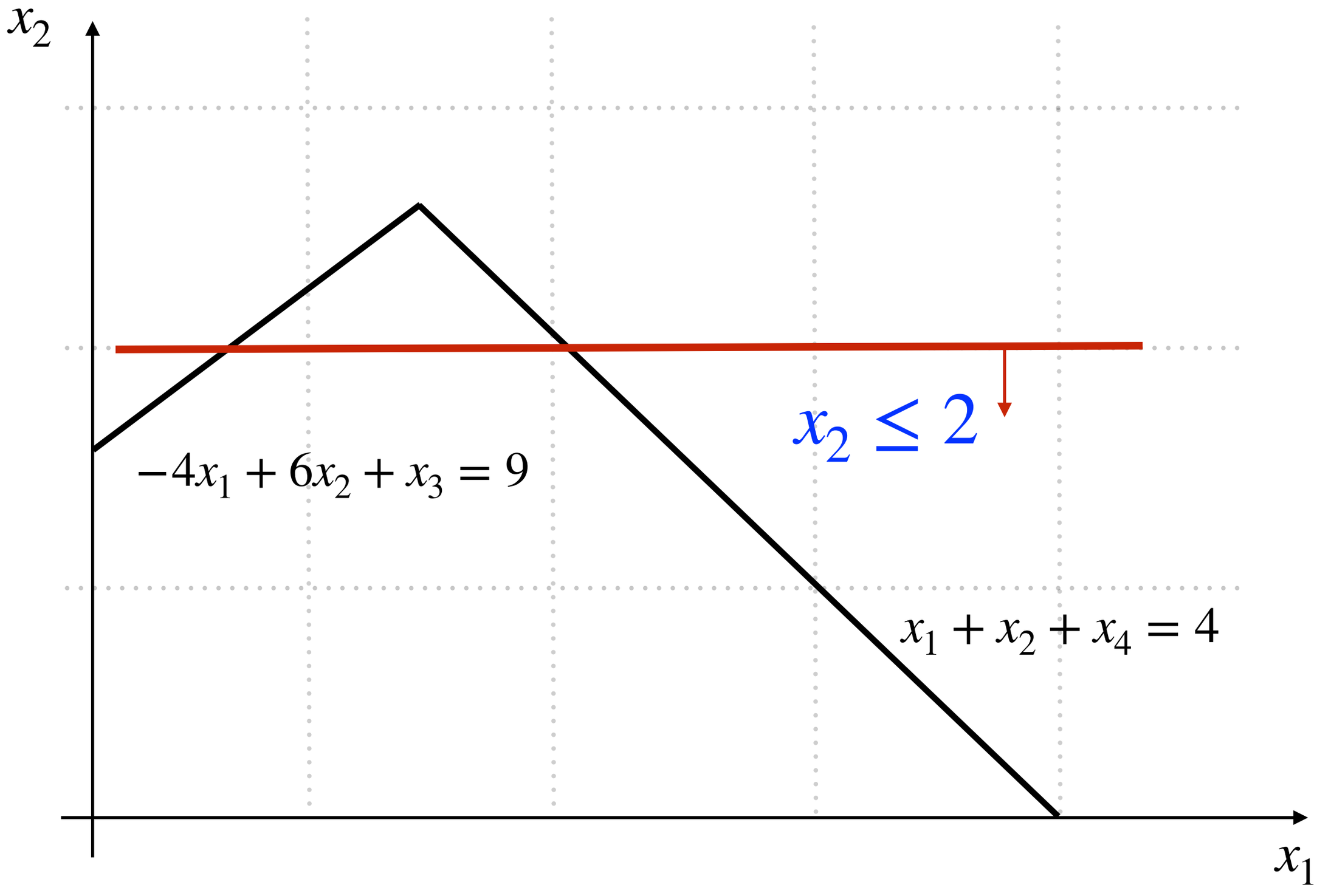
Cutting Planes



Cutting Planes



Cutting Planes



Example

$$\text{minimize } x_1 - 2x_2$$

s.t.

$$-4x_1 + 6x_2 + x_3 = 9$$

$$x_1 + x_2 + x_4 = 4$$

$$x_2 \leq 2$$

$$x_i \in \mathbb{N}$$

Example

$$\text{minimize } x_1 - 2x_2$$

s.t.

$$-4x_1 + 6x_2 + x_3 = 9$$

$$x_1 + x_2 + x_4 = 4$$

$$x_2 + x_5 = 2$$

$x_i \in \mathbb{N}$

Example

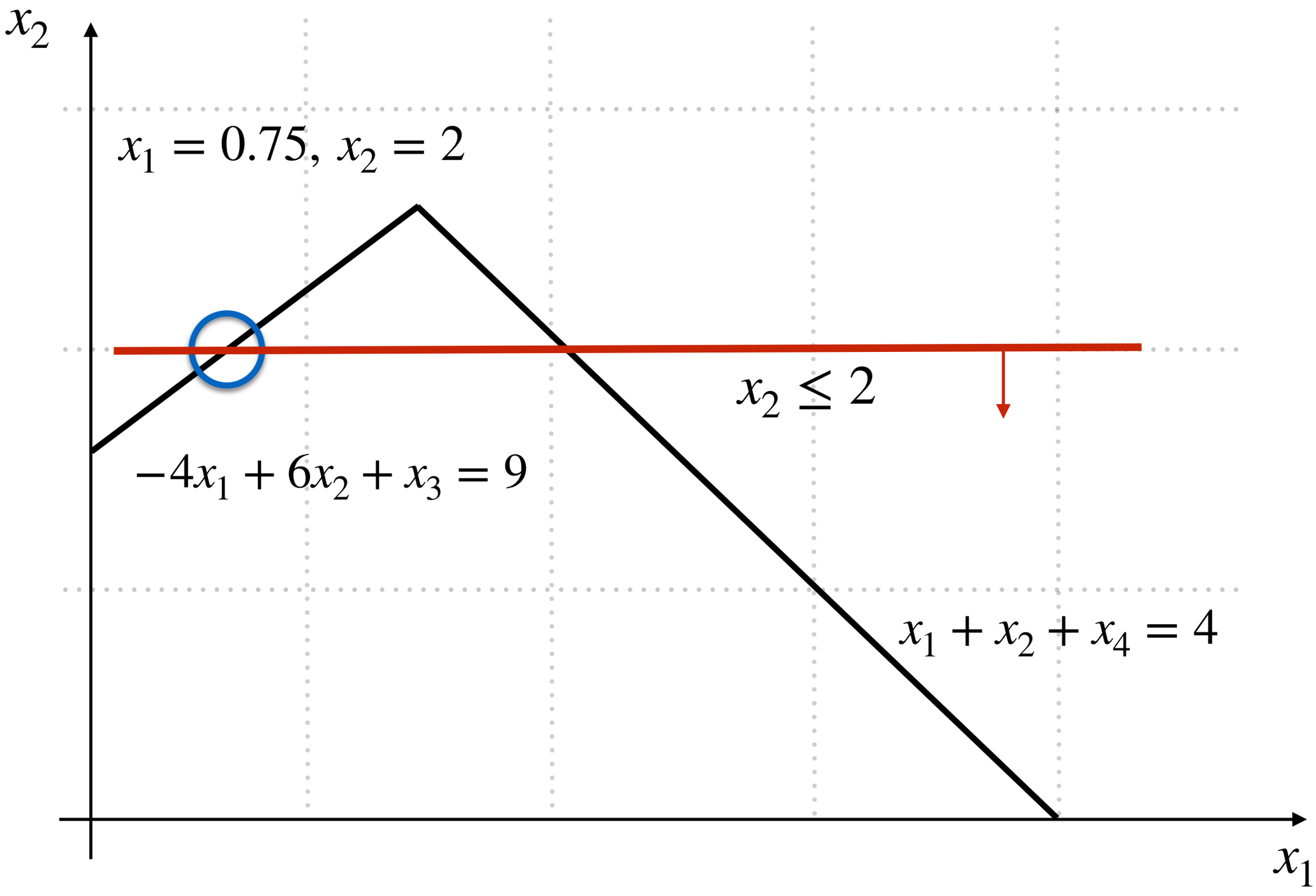
Solving with simplex, we get the optimal tableau

$$x_1 - \frac{1}{4}x_3 + \frac{3}{2}x_5 = \frac{3}{4},$$

$$x_2 + x_5 = 2.$$

Optimal solution: $x_1 = 0.75,\ x_2 = 2$

Cutting Planes



Example

Solving with simplex, we get the optimal tableau

$$x_1 - \frac{1}{4}x_3 + \frac{3}{2}x_5 = \frac{3}{4},$$

$$x_2 + x_5 = 2.$$

Optimal solution: $x_1 = 0.75,\ x_2 = 2$

Example

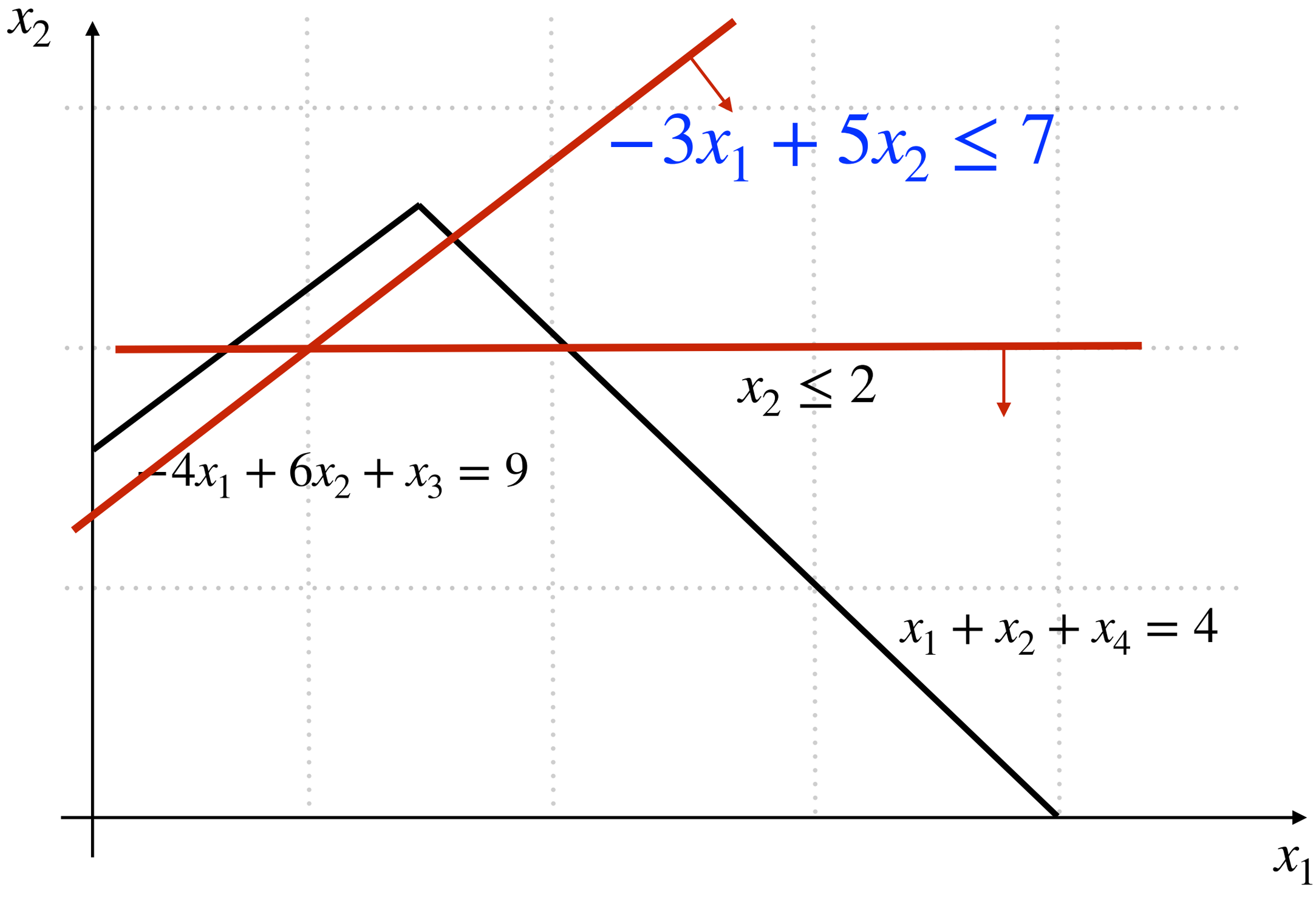
$$x_1 - \frac{1}{4}x_3 + \frac{3}{2}x_5 = \frac{3}{4}$$

$$x_1 - x_3 + x_5 \leq 0$$

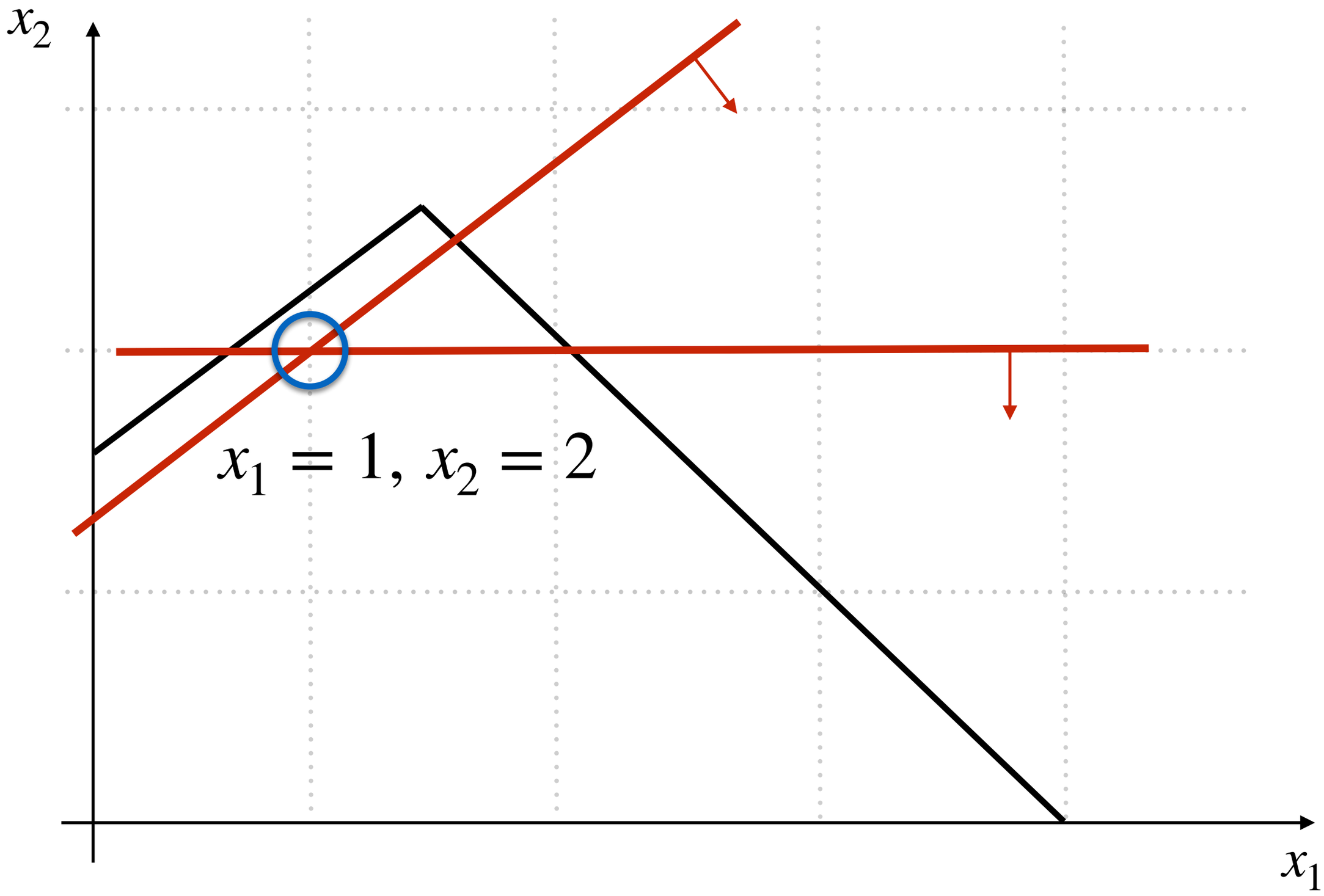
To visualize in the original variables, we have

$$-3x_1 - 5x_2 \leq 7$$

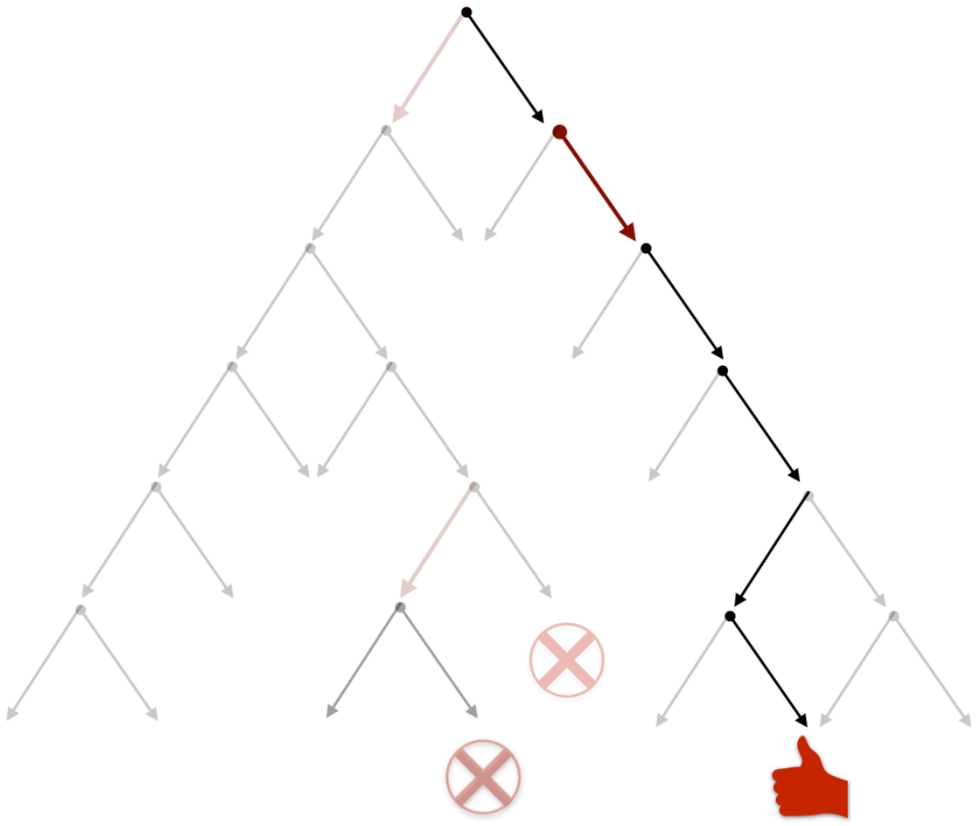
Cutting Planes

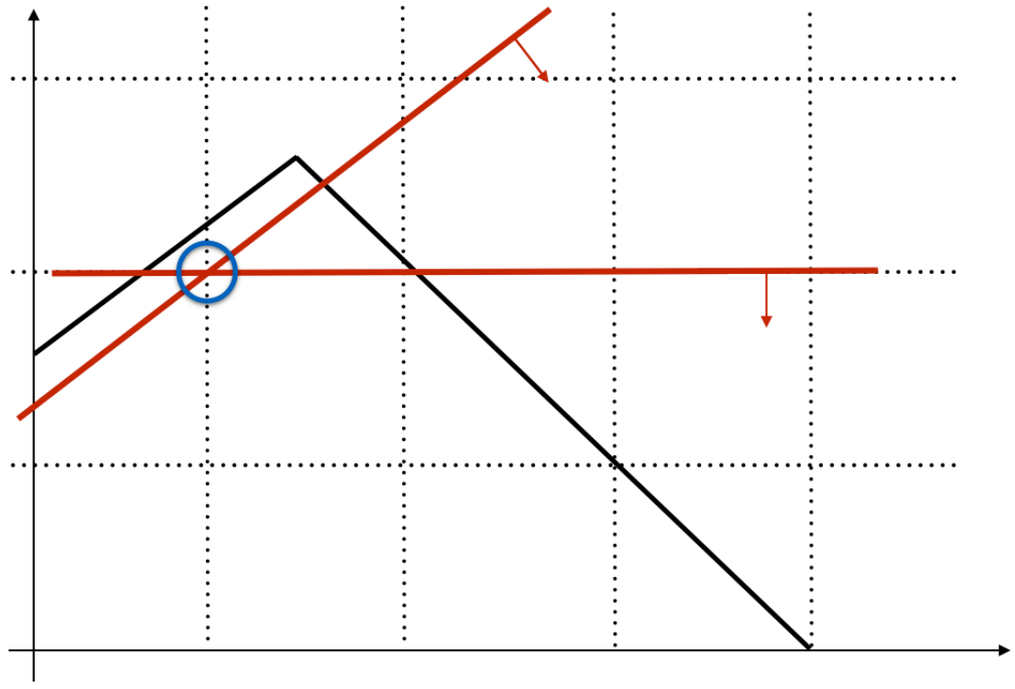


Cutting Planes



Branch and Cut





Recursively split the domain on some integer variables and create a branching tree structure. In each branch, do simplex and add cutting planes to the problem. Terminate when the cost function can't be improved.

Summary