ECE750T-28:
Computer-aided Reasoning for Software Engineering

Lecture 5: Conflict-driven Clause Learning SAT solving (Part 2)

Vijay Ganesh
(Original notes from Isil Dillig)

Announcements

- Posted two papers related to this lecture on the course webpage

Announcements

- Posted two papers related to this lecture on the course webpage
- One is about a SAT solver called GRASP and the other about the Chaff SAT solver

Announcements

- Posted two papers related to this lecture on the course webpage
- One is about a SAT solver called GRASP and the other about the Chaff SAT solver
- Posted papers on concolic testing and applications to automatic exploit construction

Overview

- Last lecture: Basic CDCL algorithm for deciding satisfiability in boolean logic
- Focus was on overall architecture, conflict analysis and conflict-clause learning

Overview

- Last lecture: Basic CDCL algorithm for deciding satisfiability in boolean logic
- Focus was on overall architecture, conflict analysis and conflict-clause learning
- Today: Focus on backjumping (non-chronological backtracking), VSIDS, Fast BCP through 2-watched literal scheme

Overview

- Last lecture: Basic CDCL algorithm for deciding satisfiability in boolean logic
- Focus was on overall architecture, conflict analysis and conflict-clause learning
- Today: Focus on backjumping (non-chronological backtracking), VSIDS, Fast BCP through 2-watched literal scheme
- Many competitive solvers based on DPLL, but extend it in three important ways:

Overview

- Last lecture: Basic CDCL algorithm for deciding satisfiability in boolean logic
- Focus was on overall architecture, conflict analysis and conflict-clause learning
- Today: Focus on backjumping (non-chronological backtracking), VSIDS, Fast BCP through 2-watched literal scheme
- Many competitive solvers based on DPLL, but extend it in three important ways:
  1. Non-chronological backtracking

Overview

- Last lecture: Basic CDCL algorithm for deciding satisfiability in boolean logic
- Focus was on overall architecture, conflict analysis and conflict-clause learning
- Today: Focus on backjumping (non-chronological backtracking), VSIDS, Fast BCP through 2-watched literal scheme
- Many competitive solvers based on DPLL, but extend it in three important ways:
  1. Non-chronological backtracking
  2. Learning from past "mistakes"

Overview

- Last lecture: Basic CDCL algorithm for deciding satisfiability in boolean logic
- Focus was on overall architecture, conflict analysis and conflict-clause learning
- Today: Focus on backjumping (non-chronological backtracking), VSIDS, Fast BCP through 2-watched literal scheme
- Many competitive solvers based on DPLL, but extend it in three important ways:
  1. Non-chronological backtracking
  2. Learning from past "mistakes"
  3. Heuristics for choosing variables and assignments

Overview

- Last lecture: Basic CDCL algorithm for deciding satisfiability in boolean logic
- Focus was on overall architecture, conflict analysis and conflict-clause learning
- Today: Focus on backjumping (non-chronological backtracking), VSIDS, Fast BCP through 2-watched literal scheme
- Many competitive solvers based on DPLL, but extend it in three important ways:
  1. Non-chronological backtracking
  2. Learning from past "mistakes"
  3. Heuristics for choosing variables and assignments
- In addition, some implementation tricks to perform BCP fast

Non-Chronological Backtracking

- Recall basic DPLL: First try assigning p to $\top$; if doesn't work, backtrack to most recent decision level and try $p = \bot$

Non-Chronological Backtracking

- Recall basic DPLL: First try assigning p to $\top$; if doesn't work, backtrack to most recent decision level and try $p = \bot$
- This is called chronological backtracking because we backtrack to the most recent branching point

Non-Chronological Backtracking

- Recall basic DPLL: First try assigning p to $\top$; if doesn't work, backtrack to most recent decision level and try $p = \bot$
- This is called chronological backtracking because we backtrack to the most recent branching point
- But in some cases this is sub-optimal!

Non-Chronological Backtracking

- Recall basic DPLL: First try assigning p to $\top$; if doesn't work, backtrack to most recent decision level and try $p = \bot$
- This is called chronological backtracking because we backtrack to the most recent branching point
- But in some cases this is sub-optimal!
- Suppose we assigned to variables $p_1, p_2, \ldots p_{100}$ and discovered that assignment to p_4 was a bad choice

Non-Chronological Backtracking

- Recall basic DPLL: First try assigning p to $\top$; if doesn't work, backtrack to most recent decision level and try $p = \bot$
- This is called chronological backtracking because we backtrack to the most recent branching point
- But in some cases this is sub-optimal!
- Suppose we assigned to variables $p_1, p_2, \ldots p_{100}$ and discovered that assignment to p_4 was a bad choice
- Backtracking to decision level associated with p_{100} is a bad idea because we were already doomed after assigning to p_4

Non-Chronological Backtracking

- Recall basic DPLL: First try assigning p to $\top$; if doesn't work, backtrack to most recent decision level and try $p = \bot$
- This is called chronological backtracking because we backtrack to the most recent branching point
- But in some cases this is sub-optimal!
- Suppose we assigned to variables $p_1, p_2, \ldots p_{100}$ and discovered that assignment to p_4 was a bad choice
- Backtracking to decision level associated with p_{100} is a bad idea because we were already doomed after assigning to p_4
- In non-chronological backtracking, we don't have to go back to most recent decision level

Learning

- Learning = acquisition of new clauses that prevent bad assignments similar to those already explored

Learning

- Learning = acquisition of new clauses that prevent bad assignments similar to those already explored
- For instance, suppose the SAT solver makes an assignment and discovers that $p_5 = \top, p_{32} = \bot, p_{100} = \top$ is inconsistent

Learning

- Learning = acquisition of new clauses that prevent bad assignments similar to those already explored
- For instance, suppose the SAT solver makes an assignment and discovers that $p_5 = \top, p_{32} = \bot, p_{100} = \top$ is inconsistent
- What can we learn from this?

Learning

- Learning = acquisition of new clauses that prevent bad assignments similar to those already explored
- For instance, suppose the SAT solver makes an assignment and discovers that $p_5 = \top, p_{32} = \bot, p_{100} = \top$ is inconsistent
- What can we learn from this?

$$\phi \Rightarrow \neg(p_5 \wedge \neg p_{32} \wedge p_{100})$$

Learning

- Learning = acquisition of new clauses that prevent bad assignments similar to those already explored
- For instance, suppose the SAT solver makes an assignment and discovers that $p_5 = \top, p_{32} = \bot, p_{100} = \top$ is inconsistent
- What can we learn from this?

$$\phi \Rightarrow \neg(p_5 \wedge \neg p_{32} \wedge p_{100})$$

- Thus, we can add this clause without changing ϕ's satisfiability (why?)

Learning

- Learning = acquisition of new clauses that prevent bad assignments similar to those already explored
- For instance, suppose the SAT solver makes an assignment and discovers that $p_5 = \top, p_{32} = \bot, p_{100} = \top$ is inconsistent
- What can we learn from this?

$$\phi \Rightarrow \neg(p_5 \wedge \neg p_{32} \wedge p_{100})$$

- Thus, we can add this clause without changing ϕ's satisfiability (why?)
- Such clauses "learned" by SAT solvers called conflict clauses

Learning

- Learning = acquisition of new clauses that prevent bad assignments similar to those already explored
- For instance, suppose the SAT solver makes an assignment and discovers that $p_5 = \top, p_{32} = \bot, p_{100} = \top$ is inconsistent
- What can we learn from this?

$$\phi \Rightarrow \neg(p_5 \wedge \neg p_{32} \wedge p_{100})$$

- Thus, we can add this clause without changing ϕ's satisfiability (why?)
- Such clauses "learned" by SAT solvers called conflict clauses
- SAT solvers maintain a database of conflict clauses to prevent bad future assignments

Decision Heuristics

- In the basic DPLL algorithm, we chose variables in a random order, and always tried $\top$ first before $\bot$

Decision Heuristics

- In the basic DPLL algorithm, we chose variables in a random order, and always tried $\top$ first before $\bot$
- But we can do better!

Decision Heuristics

- In the basic DPLL algorithm, we chose variables in a random order, and always tried $\top$ first before $\bot$
- But we can do better!
- Making assignment to certain variables can make formula much easier to solve!

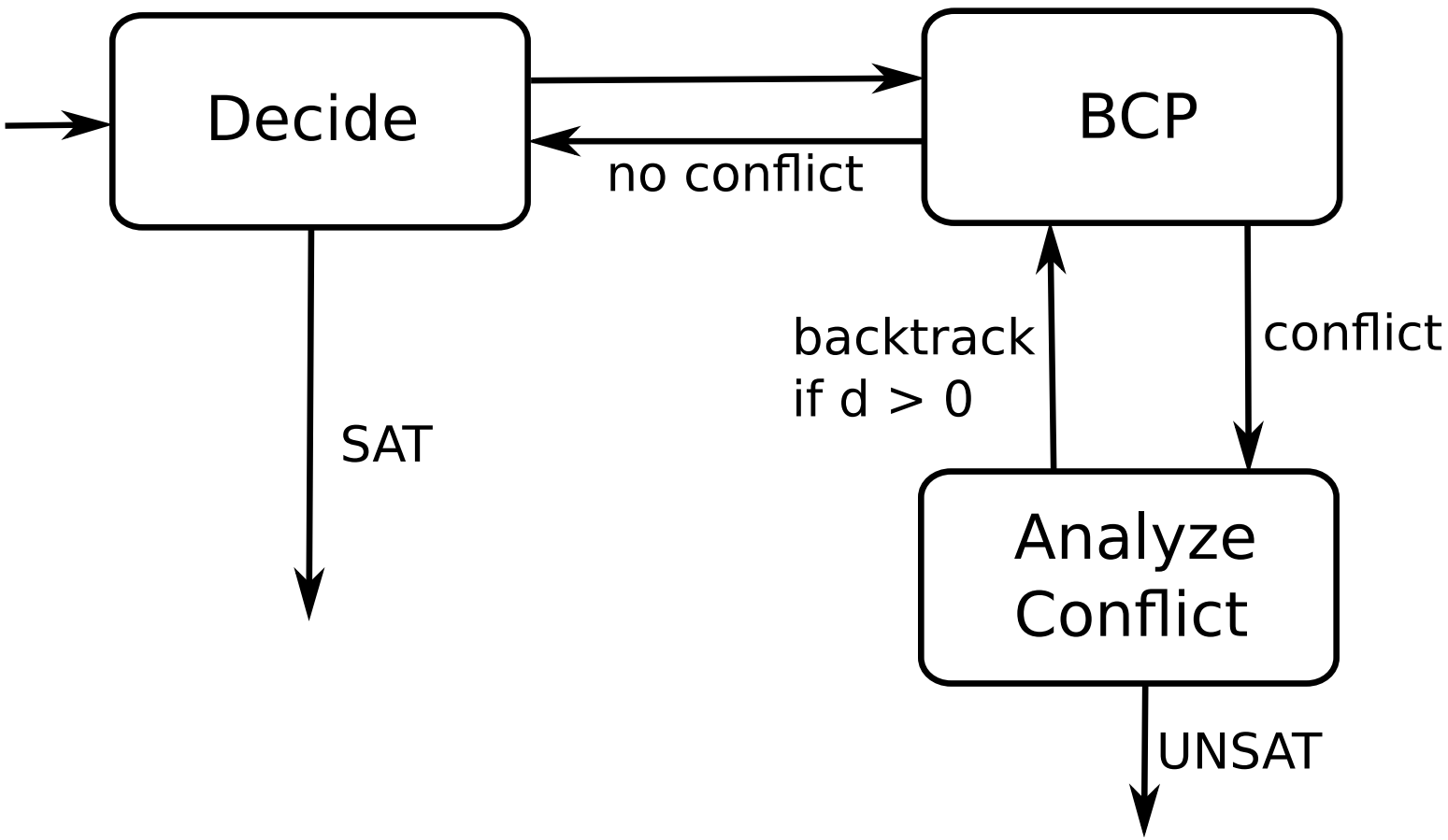
Decision Heuristics

- In the basic DPLL algorithm, we chose variables in a random order, and always tried $\top$ first before $\bot$
- But we can do better!
- Making assignment to certain variables can make formula much easier to solve!
- Practical DPLL-based solvers use more sophisticated heuristics to choose variable order and truth assignments

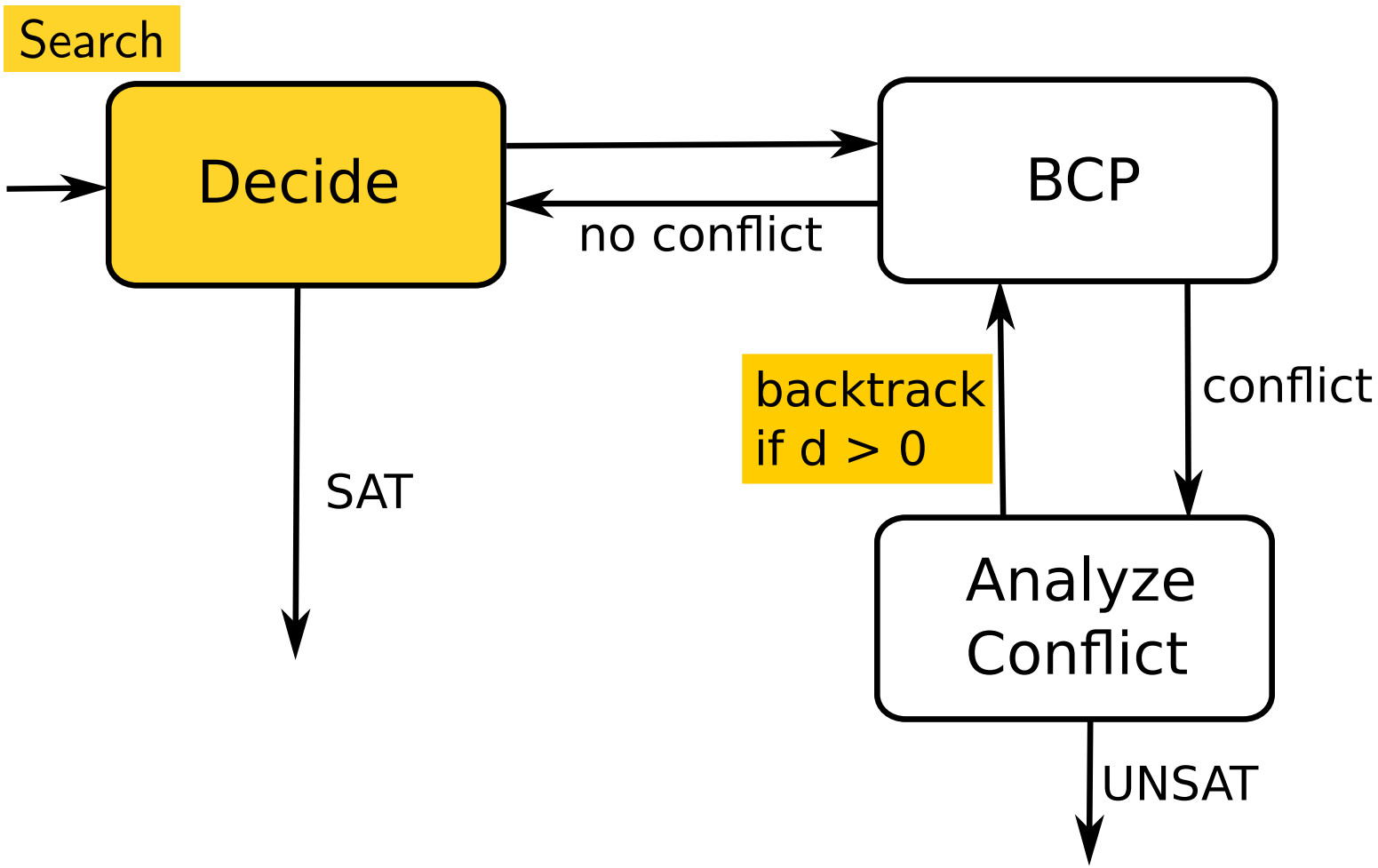
Decision Heuristics

- In the basic DPLL algorithm, we chose variables in a random order, and always tried $\top$ first before $\bot$
- But we can do better!
- Making assignment to certain variables can make formula much easier to solve!
- Practical DPLL-based solvers use more sophisticated heuristics to choose variable order and truth assignments
- This is something of a black art, but one of the most important elements in SAT solving ...

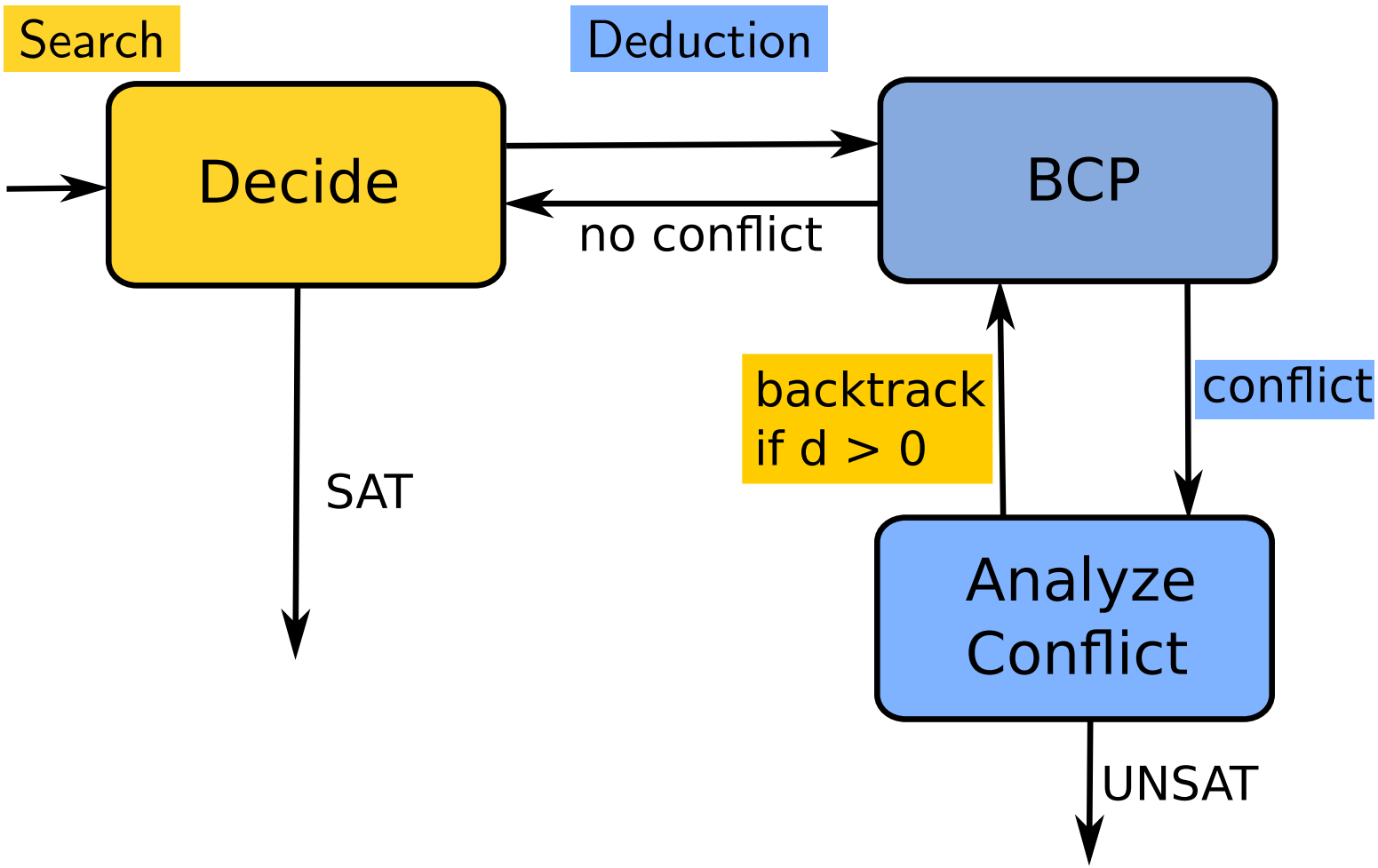
Architecture of DPLL-Based SAT Solvers



Architecture of DPLL-Based SAT Solvers



Architecture of DPLL-Based SAT Solvers



BCP in SAT Solvers

- Recall: BCP is all possible applications of unit resolution

BCP in SAT Solvers

- Recall: BCP is all possible applications of unit resolution
- In addition to performing BCP, SAT solvers also remember deductions performed in the BCP process

BCP in SAT Solvers

- Recall: BCP is all possible applications of unit resolution
- In addition to performing BCP, SAT solvers also remember deductions performed in the BCP process
- Necessary for analyzing conflicts, inferring conflict clauses

BCP in SAT Solvers

- Recall: BCP is all possible applications of unit resolution
- In addition to performing BCP, SAT solvers also remember deductions performed in the BCP process
- Necessary for analyzing conflicts, inferring conflict clauses
- Thus, BCP process recorded as implication graph

BCP in SAT Solvers

- Recall: BCP is all possible applications of unit resolution
- In addition to performing BCP, SAT solvers also remember deductions performed in the BCP process
- Necessary for analyzing conflicts, inferring conflict clauses
- Thus, BCP process recorded as implication graph
- First some terminology . . .

Some Terminology and Conventions

- Decision variable: variable assigned in the Decide step

Some Terminology and Conventions

- Decision variable: variable assigned in the Decide step
- Variables assigned due to BCP are not decision variables

Some Terminology and Conventions

- Decision variable: variable assigned in the Decide step
- Variables assigned due to BCP are not decision variables
- The decision level of a decision variable is the level (order) in which it was assigned

Some Terminology and Conventions

- Decision variable: variable assigned in the Decide step
- Variables assigned due to BCP are not decision variables
- The decision level of a decision variable is the level (order) in which it was assigned
- The decision level of a variable assigned due to BCP is the decision level of the last assigned decision variable

Some Terminology and Conventions

- Decision variable: variable assigned in the Decide step
- Variables assigned due to BCP are not decision variables
- The decision level of a decision variable is the level (order) in which it was assigned
- The decision level of a variable assigned due to BCP is the decision level of the last assigned decision variable
- Important note: Think of assignments as literals: Assignment $p = \top$ is literal p; assignment $p = \bot$ as literal $\neg p$

Some Terminology and Conventions

- Decision variable: variable assigned in the Decide step
- Variables assigned due to BCP are not decision variables
- The decision level of a decision variable is the level (order) in which it was assigned
- The decision level of a variable assigned due to BCP is the decision level of the last assigned decision variable
- Important note: Think of assignments as literals: Assignment $p = \top$ is literal p; assignment $p = \bot$ as literal $\neg p$
- Also: An assignment corresponds to a new unit clause added to our set of clauses

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top$

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields:

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$
- Is x_2 a decision variable?

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$
- Is x_2 a decision variable? No

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$
- Is x_2 a decision variable? No
- Decision level of x_2?

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$
- Is x_2 a decision variable? No
- Decision level of x_2? 1

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$
- Is x_2 a decision variable? No
- Decision level of x_2? 1
- Decide next assigns $x_4 = \top$. BCP deduces:

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$
- Is x_2 a decision variable? No
- Decision level of x_2? 1
- Decide next assigns $x_4 = \top$. BCP deduces: $x_3 = \bot$

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$
- Is x_2 a decision variable? No
- Decision level of x_2? 1
- Decide next assigns $x_4 = \top$. BCP deduces: $x_3 = \bot$
- x_4 decision variable with decision level:

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$
- Is x_2 a decision variable? No
- Decision level of x_2? 1
- Decide next assigns $x_4 = \top$. BCP deduces: $x_3 = \bot$
- x_4 decision variable with decision level: 2

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$
- Is x_2 a decision variable? No
- Decision level of x_2? 1
- Decide next assigns $x_4 = \top$. BCP deduces: $x_3 = \bot$
- x_4 decision variable with decision level: 2
- x_3's decision level:

Decision Level Example

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$$

- Decide assigns $x_1 = \top \Rightarrow x_1$ decision var at level 1
- BCP yields: $x_2 = \top$
- Is x_2 a decision variable? No
- Decision level of x_2? 1
- Decide next assigns $x_4 = \top$. BCP deduces: $x_3 = \bot$
- x_4 decision variable with decision level: 2
- x_3's decision level: 2

Implication Graph

- An implication graph is a labeled directed acyclic graph

Implication Graph

- An implication graph is a labeled directed acyclic graph
- Nodes: literals in the current partial assignment

Implication Graph

- An implication graph is a labeled directed acyclic graph
- Nodes: literals in the current partial assignment
- Node labels: Indicate assignment and decision level.

Implication Graph

- An implication graph is a labeled directed acyclic graph
- Nodes: literals in the current partial assignment
- Node labels: Indicate assignment and decision level.
- Example: Node labeled $\neg x : 3$ means variable x was assigned to $\perp$ at decision level 3

Implication Graph

- An implication graph is a labeled directed acyclic graph
- Nodes: literals in the current partial assignment
- Node labels: Indicate assignment and decision level.
- Example: Node labeled $\neg x : 3$ means variable x was assigned to $\perp$ at decision level 3
- Edges from $l_1, \ldots l_k$ to l labeled with c: Assignments $l_1, \ldots, l_k$ caused assignment l due to clause c during BCP

Implication Graph

- An implication graph is a labeled directed acyclic graph
- Nodes: literals in the current partial assignment
- Node labels: Indicate assignment and decision level.
- Example: Node labeled $\neg x : 3$ means variable x was assigned to $\perp$ at decision level 3
- Edges from $l_1, \ldots l_k$ to l labeled with c: Assignments $l_1, \ldots, l_k$ caused assignment l due to clause c during BCP
- A special node C is called the conflict node.

Implication Graph

- An implication graph is a labeled directed acyclic graph
- Nodes: literals in the current partial assignment
- Node labels: Indicate assignment and decision level.
- Example: Node labeled $\neg x : 3$ means variable x was assigned to $\perp$ at decision level 3
- Edges from $l_1, \ldots l_k$ to l labeled with c: Assignments $l_1, \ldots, l_k$ caused assignment l due to clause c during BCP
- A special node C is called the conflict node.
- Edge to conflict node labeled with c: current partial assignment contradicts clause c.

Implication Graph Example

- Consider the following set of clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg a \vee \neg b) \quad c_3 : (\neg c \vee b)$$

Implication Graph Example

- Consider the following set of clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg a \vee \neg b) \quad c_3 : (\neg c \vee b)$$

- Assume *Decide* assigned $a = \top$ at decision level 2

Implication Graph Example

- Consider the following set of clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg a \vee \neg b) \quad c_3 : (\neg c \vee b)$$

- Assume *Decide* assigned $a = \top$ at decision level 2
- BCP yields:

Implication Graph Example

- Consider the following set of clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg a \vee \neg b) \quad c_3 : (\neg c \vee b)$$

- Assume *Decide* assigned $a = \top$ at decision level 2
- BCP yields: $c = \top, b = \bot$

Implication Graph Example

- Consider the following set of clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg a \vee \neg b) \quad c_3 : (\neg c \vee b)$$

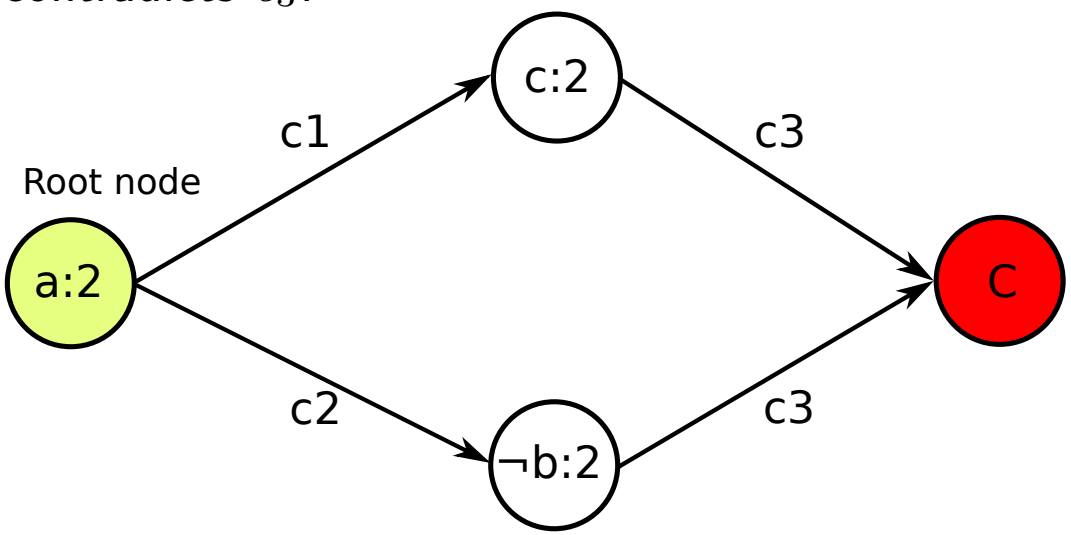
- Assume *Decide* assigned $a = \top$ at decision level 2
- BCP yields: $c = \top, b = \bot$
- Assignment contradicts c_3!

Implication Graph Example

- Consider the following set of clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg a \vee \neg b) \quad c_3 : (\neg c \vee b)$$

- Assume *Decide* assigned $a = \top$ at decision level 2
- BCP yields: $c = \top, b = \bot$
- Assignment contradicts c_3!



Another Example

- Consider the following clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg c \vee \neg a \vee b) \quad c_3 : (\neg c \vee d) \quad c_4 : (\neg d \vee \neg b)$$

Another Example

- Consider the following clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg c \vee \neg a \vee b) \quad c_3 : (\neg c \vee d) \quad c_4 : (\neg d \vee \neg b)$$

- Suppose *Decide* assigned $a = \top$ at decision level 1

Another Example

- Consider the following clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg c \vee \neg a \vee b) \quad c_3 : (\neg c \vee d) \quad c_4 : (\neg d \vee \neg b)$$

- Suppose *Decide* assigned $a = \top$ at decision level 1
- Using clause c_1, BCP yields:

Another Example

- Consider the following clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg c \vee \neg a \vee b) \quad c_3 : (\neg c \vee d) \quad c_4 : (\neg d \vee \neg b)$$

- Suppose *Decide* assigned $a = \top$ at decision level 1
- Using clause c_1, BCP yields: $c = \top$

Another Example

- Consider the following clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg c \vee \neg a \vee b) \quad c_3 : (\neg c \vee d) \quad c_4 : (\neg d \vee \neg b)$$

- Suppose *Decide* assigned $a = \top$ at decision level 1
- Using clause c_1, BCP yields: $c = \top$
- Using clause c_2, BCP yields:

Another Example

- Consider the following clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg c \vee \neg a \vee b) \quad c_3 : (\neg c \vee d) \quad c_4 : (\neg d \vee \neg b)$$

- Suppose *Decide* assigned $a = \top$ at decision level 1
- Using clause c_1, BCP yields: $c = \top$
- Using clause c_2, BCP yields: $b = \top$

Another Example

- Consider the following clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg c \vee \neg a \vee b) \quad c_3 : (\neg c \vee d) \quad c_4 : (\neg d \vee \neg b)$$

- Suppose *Decide* assigned $a = \top$ at decision level 1
- Using clause c_1, BCP yields: $c = \top$
- Using clause c_2, BCP yields: $b = \top$
- Using clause c_3, BCP yields:

Another Example

- Consider the following clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg c \vee \neg a \vee b) \quad c_3 : (\neg c \vee d) \quad c_4 : (\neg d \vee \neg b)$$

- Suppose *Decide* assigned $a = \top$ at decision level 1
- Using clause c_1, BCP yields: $c = \top$
- Using clause c_2, BCP yields: $b = \top$
- Using clause c_3, BCP yields: $d = \top$

Another Example

- Consider the following clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg c \vee \neg a \vee b) \quad c_3 : (\neg c \vee d) \quad c_4 : (\neg d \vee \neg b)$$

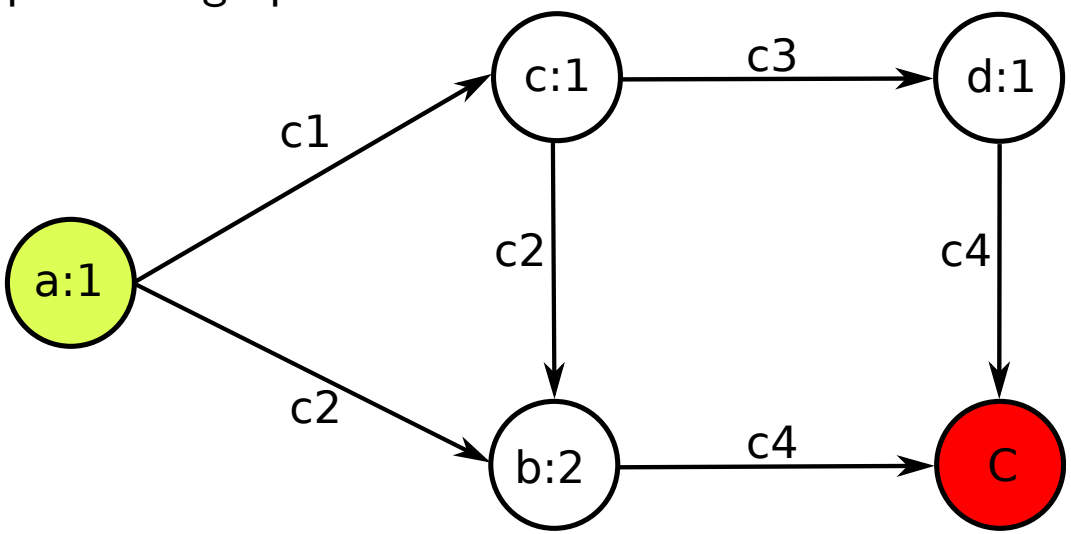
- Suppose *Decide* assigned $a = \top$ at decision level 1
- Using clause c_1, BCP yields: $c = \top$
- Using clause c_2, BCP yields: $b = \top$
- Using clause c_3, BCP yields: $d = \top$
- Assignment $b = \top, d = \top$ contradicts: $c_4 : (\neg d \vee \neg b)$

Example cont.

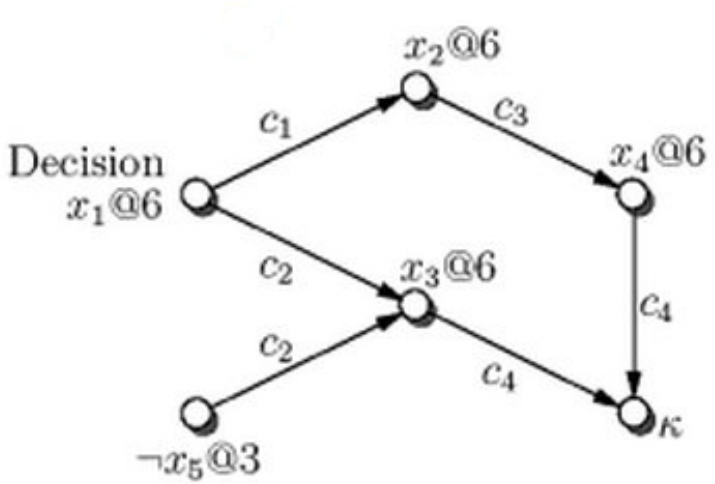
- Consider the following clauses:

$$c_1 : (\neg a \vee c) \quad c_2 : (\neg c \vee \neg a \vee b) \quad c_3 : (\neg c \vee d) \quad c_4 : (\neg d \vee \neg b)$$

- Suppose *Decide* assigned $a = \top$ at decision level 1
- Resulting implication graph:

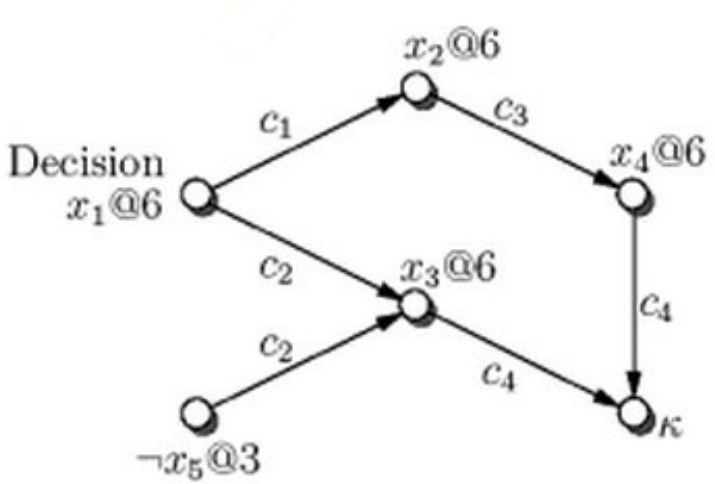


Example 3



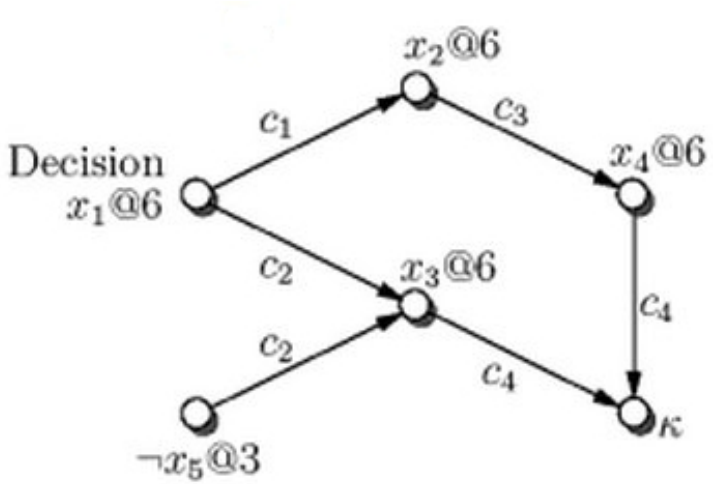
- Based on this implication graph, what is c_4?

Example 3



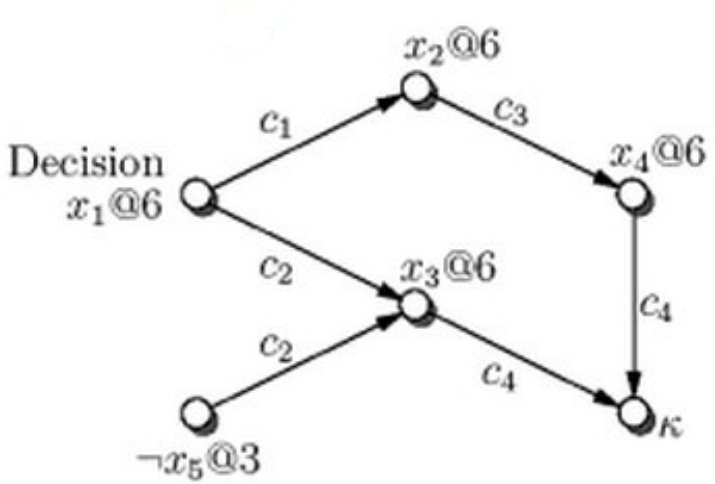
- Based on this implication graph, what is c_4? $\neg x_3 \vee \neg x_4$

Example 3



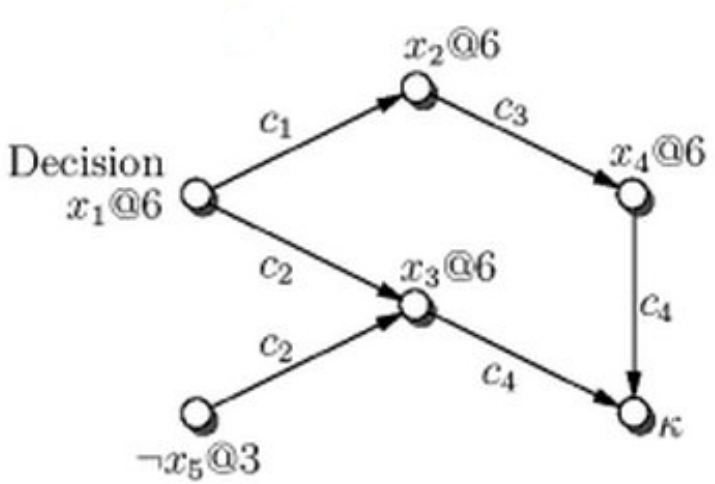
- Based on this implication graph, what is c_4? $\neg x_3 \vee \neg x_4$
- What is c_3?

Example 3



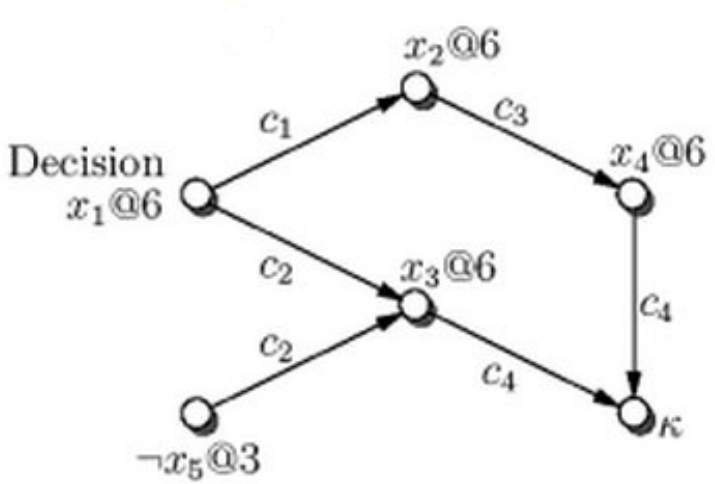
- Based on this implication graph, what is c_4? $\neg x_3 \vee \neg x_4$
- What is c_3? $\neg x_2 \vee x_4$

Example 3



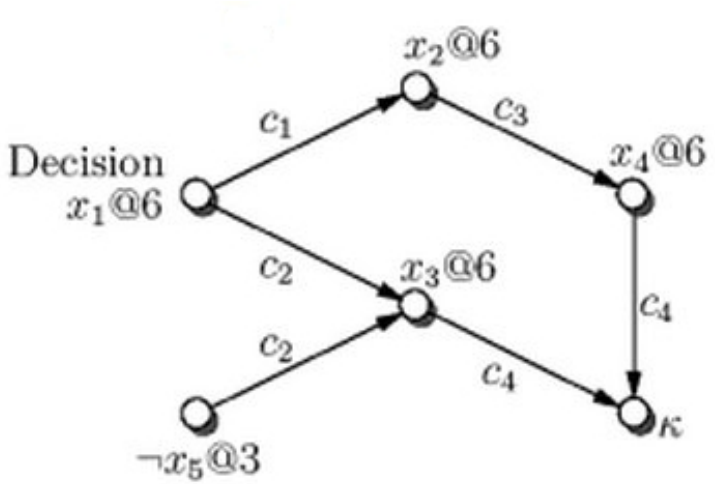
- Based on this implication graph, what is c_4? $\neg x_3 \vee \neg x_4$
- What is c_3? $\neg x_2 \vee x_4$
- What is c_1?

Example 3



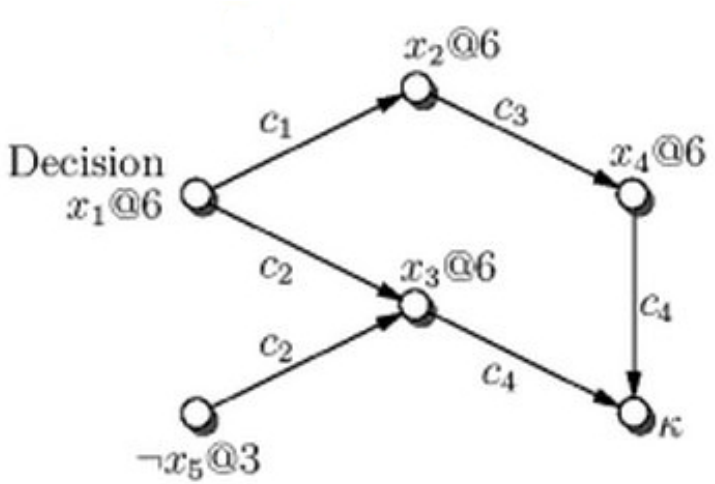
- Based on this implication graph, what is c_4? $\neg x_3 \vee \neg x_4$
- What is c_3? $\neg x_2 \vee x_4$
- What is c_1? $\neg x_1 \vee x_2$

Example 3



- Based on this implication graph, what is c_4? $\neg x_3 \vee \neg x_4$
- What is c_3? $\neg x_2 \vee x_4$
- What is c_1? $\neg x_1 \vee x_2$
- What is c_2?

Example 3



- Based on this implication graph, what is c_4? $\neg x_3 \vee \neg x_4$
- What is c_3? $\neg x_2 \vee x_4$
- What is c_1? $\neg x_1 \vee x_2$
- What is c_2? $\neg x_1 \vee x_5 \vee x_3$

Implication Graph Properties

- Root nodes in the implication graph correspond to what kind of variables?

Implication Graph Properties

- Root nodes in the implication graph correspond to what kind of variables?
  decision variables

Implication Graph Properties

- Root nodes in the implication graph correspond to what kind of variables?
  decision variables
- Edges and internal nodes arise due to BCP

Implication Graph Properties

- Root nodes in the implication graph correspond to what kind of variables?
  decision variables
- Edges and internal nodes arise due to BCP
- If literal l has incoming edge labeled c, what do we know about c?

Implication Graph Properties

- Root nodes in the implication graph correspond to what kind of variables? decision variables
- Edges and internal nodes arise due to BCP
- If literal l has incoming edge labeled c, what do we know about c? l must appear in c

Implication Graph Properties

- Root nodes in the implication graph correspond to what kind of variables? decision variables
- Edges and internal nodes arise due to BCP
- If literal l has incoming edge labeled c, what do we know about c? l must appear in c
- If literal l has outgoing edge labeled c, what do we know about c?

Implication Graph Properties

- Root nodes in the implication graph correspond to what kind of variables? decision variables
- Edges and internal nodes arise due to BCP
- If literal l has incoming edge labeled c, what do we know about c? l must appear in c
- If literal l has outgoing edge labeled c, what do we know about c? $\neg l$ must appear in c

Analyzing Conflicts

- So far: Implication graph used to record history of choices and subsequent BCP

Analyzing Conflicts

- So far: Implication graph used to record history of choices and subsequent BCP
- But whole point of recording this history is to analyze conflict

Analyzing Conflicts

- So far: Implication graph used to record history of choices and subsequent BCP
- But whole point of recording this history is to analyze conflict
- AnalyzeConflict has two goals:

Analyzing Conflicts

- So far: Implication graph used to record history of choices and subsequent BCP
- But whole point of recording this history is to analyze conflict
- AnalyzeConflict has two goals:
  1. Learn new conflict clauses

Analyzing Conflicts

- So far: Implication graph used to record history of choices and subsequent BCP
- But whole point of recording this history is to analyze conflict
- AnalyzeConflict has two goals:
  1. Learn new conflict clauses
  2. Figure out what level to backtrack to

Analyzing Conflicts

- So far: Implication graph used to record history of choices and subsequent BCP
- But whole point of recording this history is to analyze conflict
- AnalyzeConflict has two goals:
  1. Learn new conflict clauses
  2. Figure out what level to backtrack to
- Next: How to use the implication graph to derive conflict clauses and choose backtracking level

Conflict Clauses

- A conflict clause is a clause (disjunct) implied by the original formula

Conflict Clauses

- A conflict clause is a clause (disjunct) implied by the original formula
- Point of conflict clause: Prevent bad partial assignments by deriving contradiction as quickly as possible

Conflict Clauses

- A conflict clause is a clause (disjunct) implied by the original formula
- Point of conflict clause: Prevent bad partial assignments by deriving contradiction as quickly as possible
- Question: To achieve this goal, are small or large conflict clauses better?

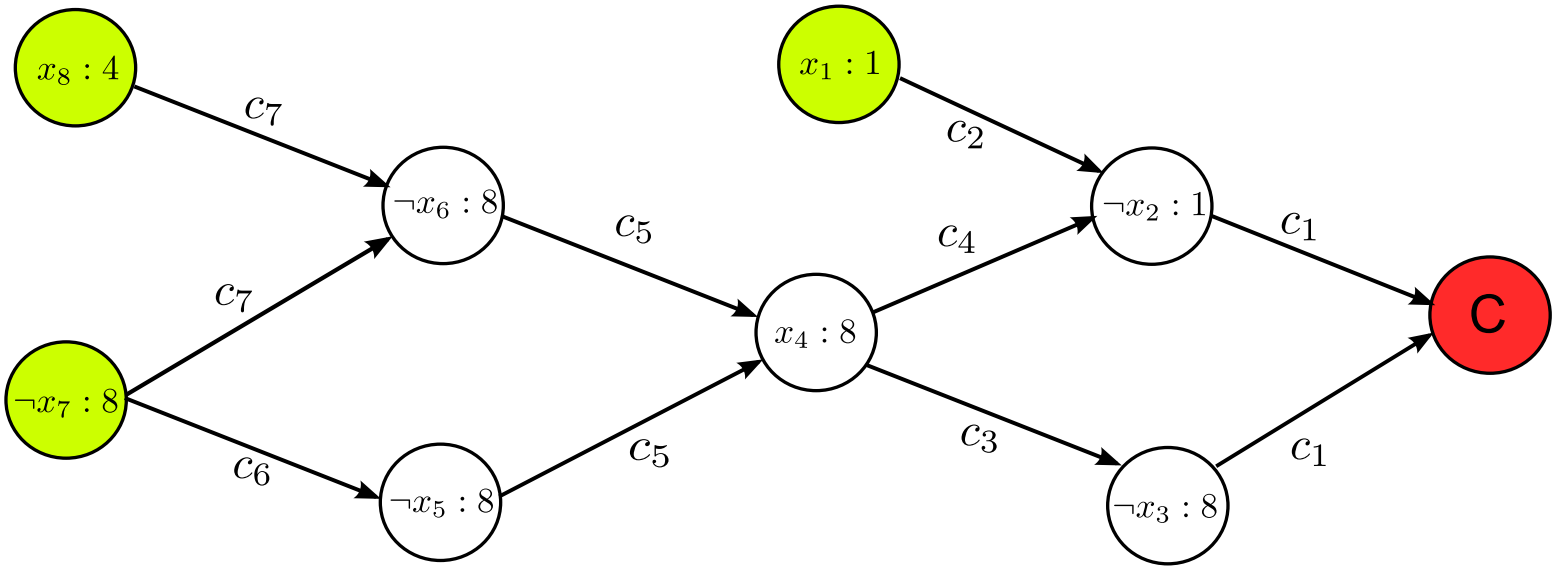
Conflict Clauses

- A conflict clause is a clause (disjunct) implied by the original formula
- Point of conflict clause: Prevent bad partial assignments by deriving contradiction as quickly as possible
- Question: To achieve this goal, are small or large conflict clauses better?
- Answer: Small ones because the smaller the clause, the quicker BCP forces variable assignments, and the quicker we derive contradictions!

Conflict Clauses

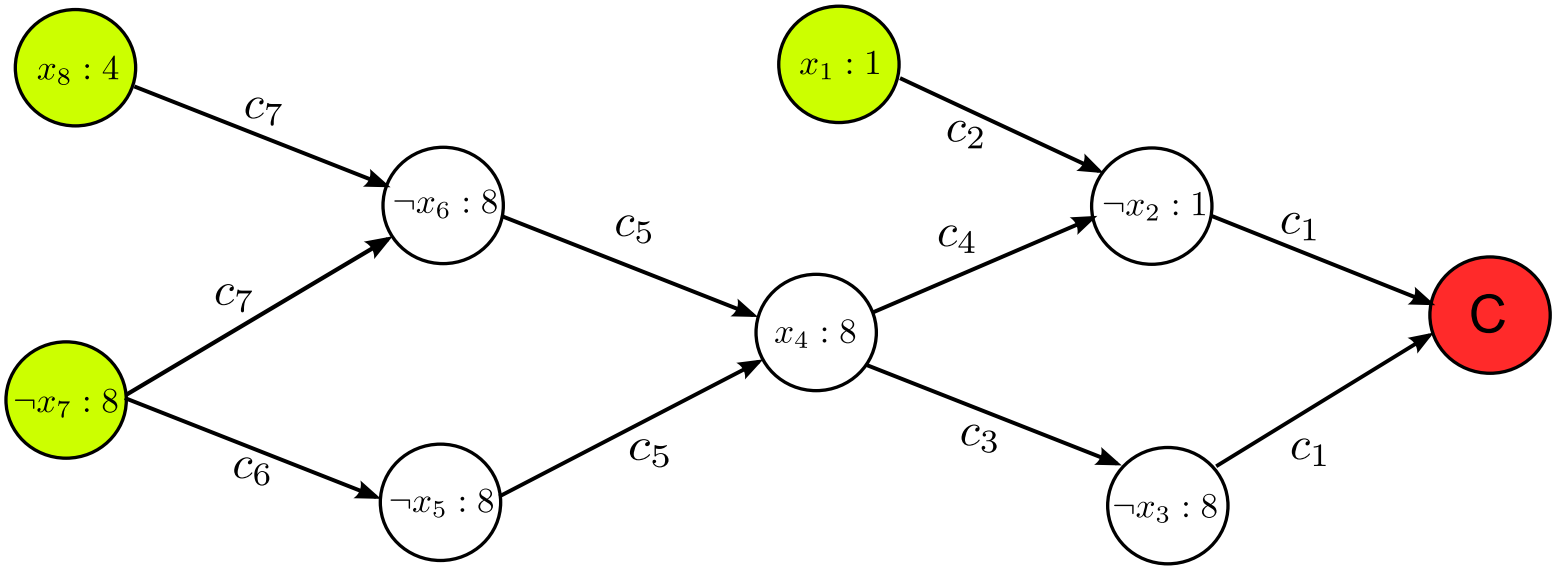
- A conflict clause is a clause (disjunct) implied by the original formula
- Point of conflict clause: Prevent bad partial assignments by deriving contradiction as quickly as possible
- Question: To achieve this goal, are small or large conflict clauses better?
- Answer: Small ones because the smaller the clause, the quicker BCP forces variable assignments, and the quicker we derive contradictions!
- The implication graph is very useful for deriving small clauses implied by the original formula!

Using Implication Graph to Analyze Conflicts



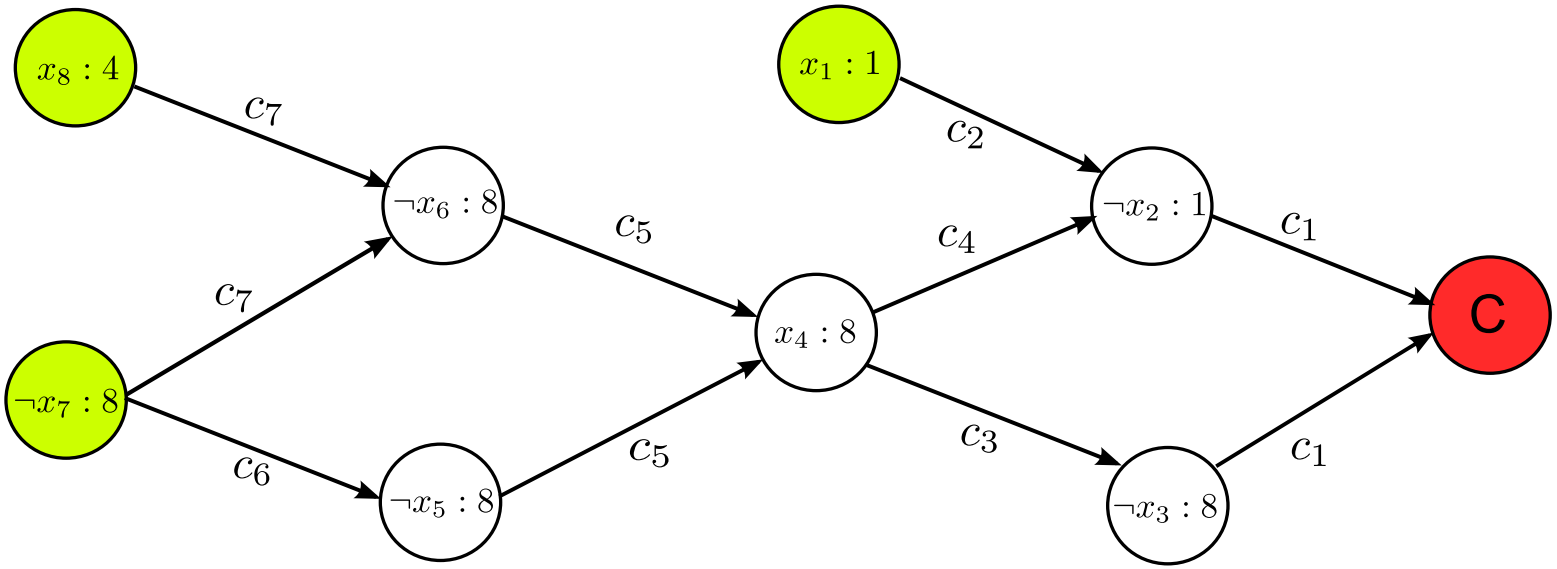
- What can we say about source of conflict based on this (partial) implication graph?

Using Implication Graph to Analyze Conflicts



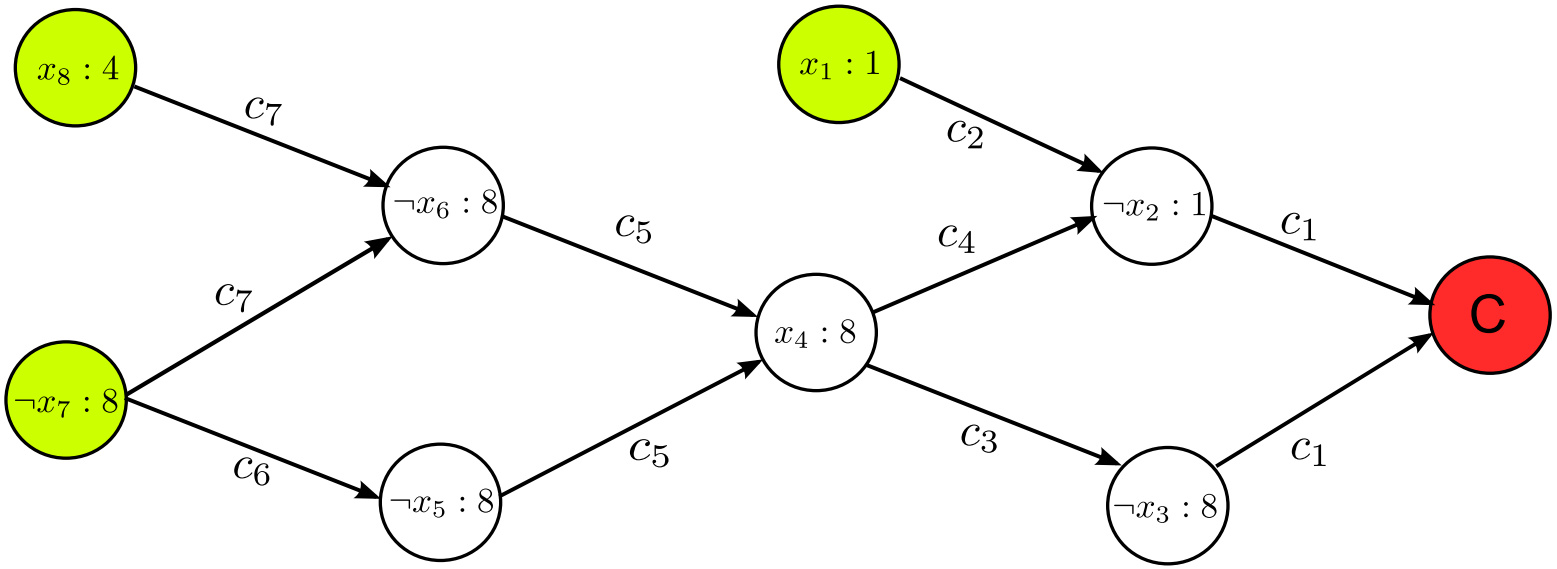
- What can we say about source of conflict based on this (partial) implication graph?
- Partial assignment $x_1, x_8, \neg x_7$ leads to conflict!

Using Implication Graph to Analyze Conflicts



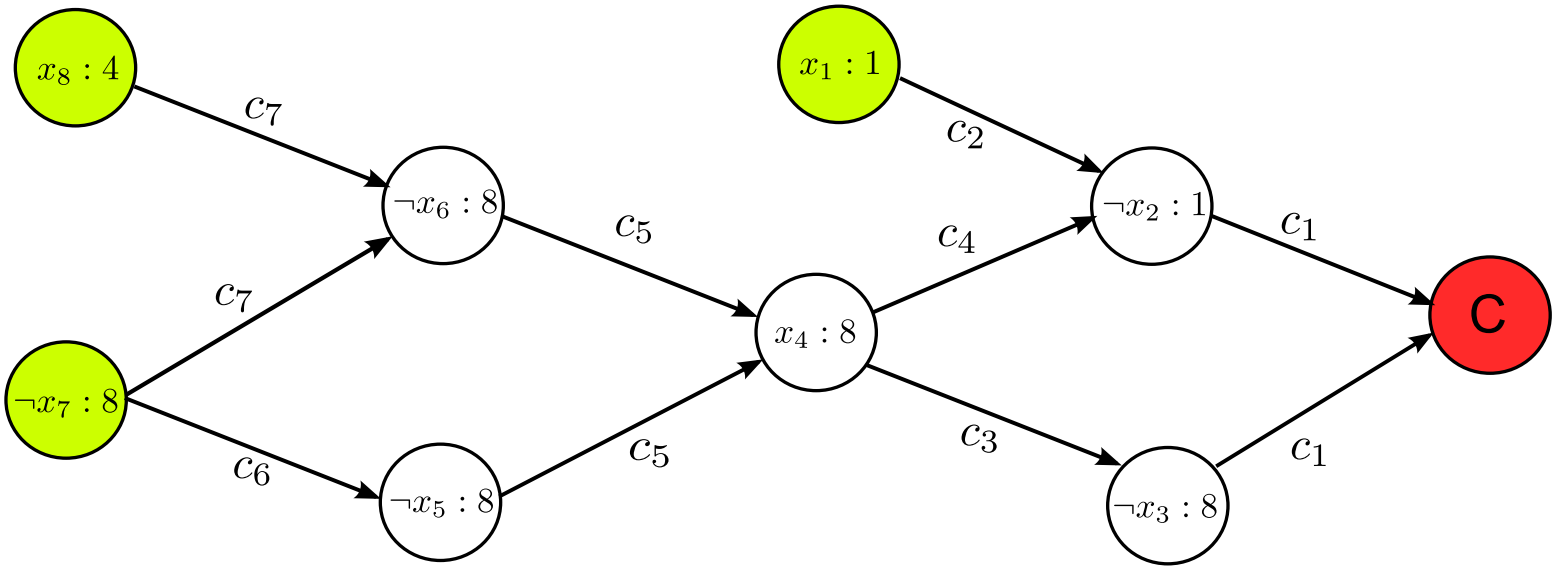
- What can we say about source of conflict based on this (partial) implication graph?
- Partial assignment $x_1, x_8, \neg x_7$ leads to conflict!
- Are other decision variables relevant to conflict?

Using Implication Graph to Analyze Conflicts



- What can we say about source of conflict based on this (partial) implication graph?
- Partial assignment $x_1, x_8, \neg x_7$ leads to conflict!
- Are other decision variables relevant to conflict? No!

Using Implication Graph to Analyze Conflicts



- What can we say about source of conflict based on this (partial) implication graph?
- Partial assignment $x_1, x_8, \neg x_7$ leads to conflict!
- Are other decision variables relevant to conflict? No!
- Implication graph allows us to identify a minimal set of "choices" (assignments) relevant to conflict!

One Strategy to Derive Conflict Clause

- One way to derive conflict clause: Conjoin all literals associated with root nodes reaching conflict node, use negation as conflict clause

One Strategy to Derive Conflict Clause

- One way to derive conflict clause: Conjoin all literals associated with root nodes reaching conflict node, use negation as conflict clause
- Why is this correct?

One Strategy to Derive Conflict Clause

- One way to derive conflict clause: Conjoin all literals associated with root nodes reaching conflict node, use negation as conflict clause
- Why is this correct?
- Literals associated with root nodes reaching conflict node form a partial assignment A sufficient to derive contradiction.

One Strategy to Derive Conflict Clause

- One way to derive conflict clause: Conjoin all literals associated with root nodes reaching conflict node, use negation as conflict clause
- Why is this correct?
- Literals associated with root nodes reaching conflict node form a partial assignment A sufficient to derive contradiction.
- Thus, $\phi \wedge A$ is unsat; hence $\neg A$ is implied by formula!

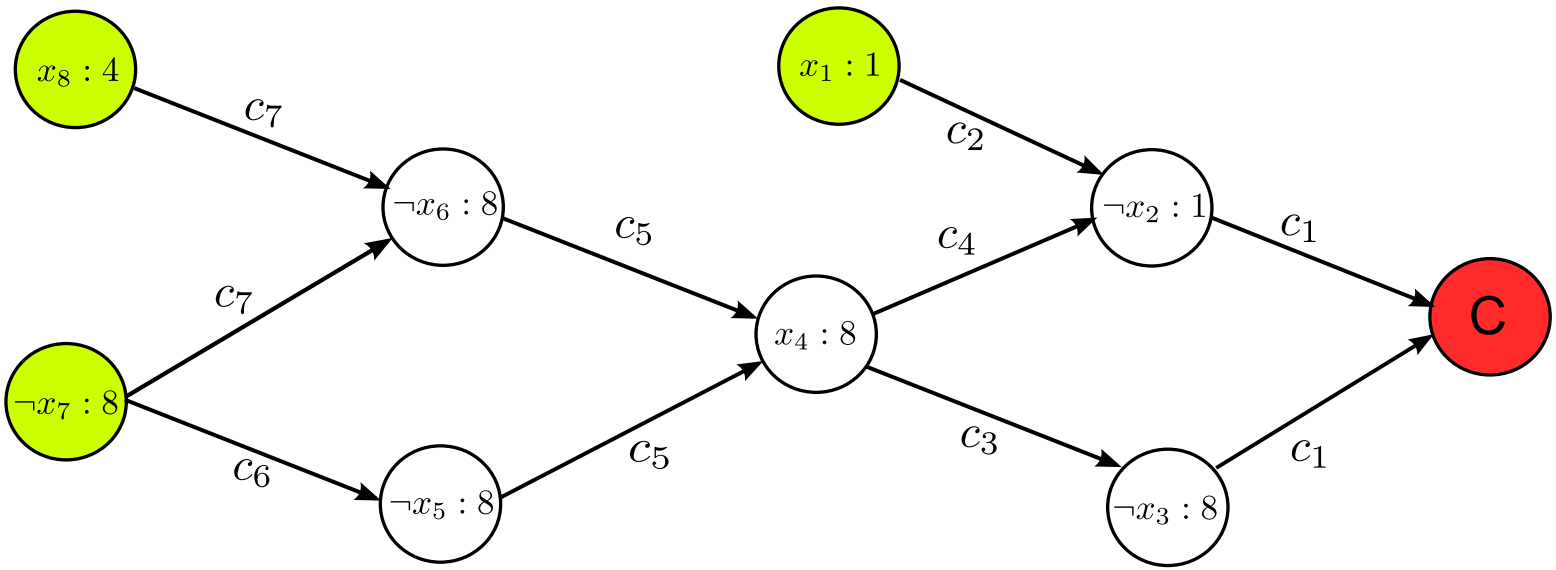
One Strategy to Derive Conflict Clause

- One way to derive conflict clause: Conjoin all literals associated with root nodes reaching conflict node, use negation as conflict clause
- Why is this correct?
- Literals associated with root nodes reaching conflict node form a partial assignment A sufficient to derive contradiction.
- Thus, $\phi \wedge A$ is unsat; hence $\neg A$ is implied by formula!
- Question: Ok, $\neg A$ is valid conflict clause, but why is it better than taking the negation of the whole partial assignment?

One Strategy to Derive Conflict Clause

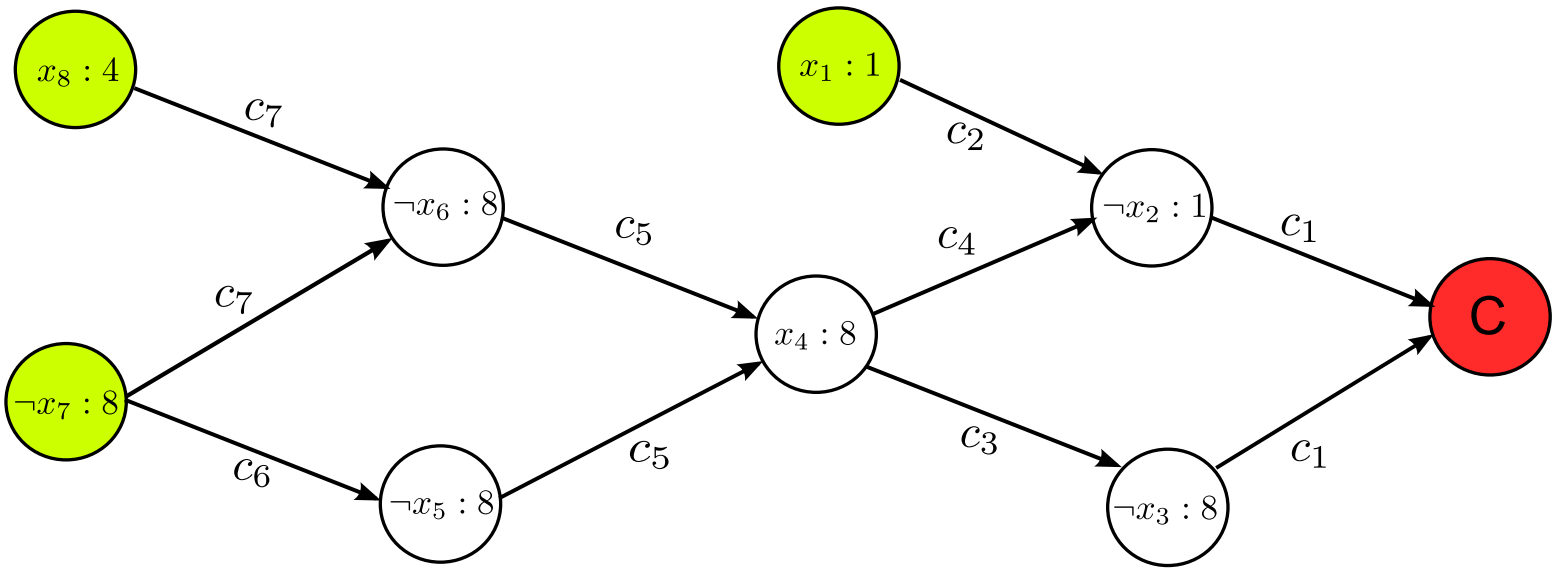
- One way to derive conflict clause: Conjoin all literals associated with root nodes reaching conflict node, use negation as conflict clause
- Why is this correct?
- Literals associated with root nodes reaching conflict node form a partial assignment A sufficient to derive contradiction.
- Thus, $\phi \wedge A$ is unsat; hence $\neg A$ is implied by formula!
- Question: Ok, $\neg A$ is valid conflict clause, but why is it better than taking the negation of the whole partial assignment?
- Answer: Because it only includes literals relevant to contradiction; thus resulting clause much smaller!

Using Implication Graph to Analyze Conflicts



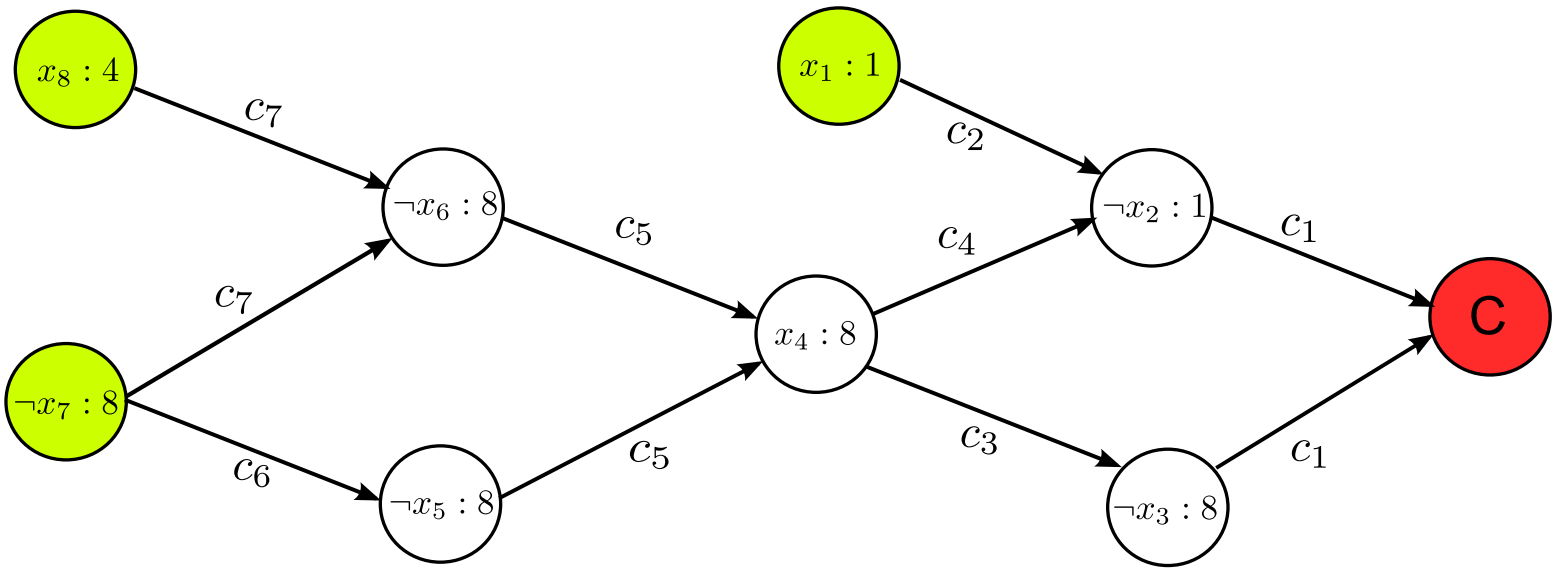
- In this example, this would yield:

Using Implication Graph to Analyze Conflicts



- In this example, this would yield: $c' = \neg x_8 \vee x_7 \vee \neg x_1$

Using Implication Graph to Analyze Conflicts



- In this example, this would yield: $c' = \neg x_8 \vee x_7 \vee \neg x_1$
- c' prevents the same partial assignment in the next step

Analyzing Conflicts

- This strategy is one of the earliest strategies proposed for inferring conflict clauses

Analyzing Conflicts

- This strategy is one of the earliest strategies proposed for inferring conflict clauses
- Original GRASP SAT solver derived conflict clauses this way

Analyzing Conflicts

- This strategy is one of the earliest strategies proposed for inferring conflict clauses
- Original GRASP SAT solver derived conflict clauses this way
- But people have improved upon this; possible to derive even better conflict clauses!

Analyzing Conflicts

- This strategy is one of the earliest strategies proposed for inferring conflict clauses
- Original GRASP SAT solver derived conflict clauses this way
- But people have improved upon this; possible to derive even better conflict clauses!
- A key concept is unique implication points

Unique Implication Point

- A node N in the implication graph is a unique implication point (UIP) if all paths from current decision node to the conflict node must go through N

Unique Implication Point

- A node N in the implication graph is a unique implication point (UIP) if all paths from current decision node to the conflict node must go through N
- Same concept as dominator

Unique Implication Point

- A node N in the implication graph is a unique implication point (UIP) if all paths from current decision node to the conflict node must go through N
- Same concept as dominator
- Is the current decision node a UIP?

Unique Implication Point

- A node N in the implication graph is a unique implication point (UIP) if all paths from current decision node to the conflict node must go through N
- Same concept as dominator
- Is the current decision node a UIP? Yes

Unique Implication Point

- A node N in the implication graph is a unique implication point (UIP) if all paths from current decision node to the conflict node must go through N
- Same concept as dominator
- Is the current decision node a UIP? Yes
- Can there be multiple unique implication points?

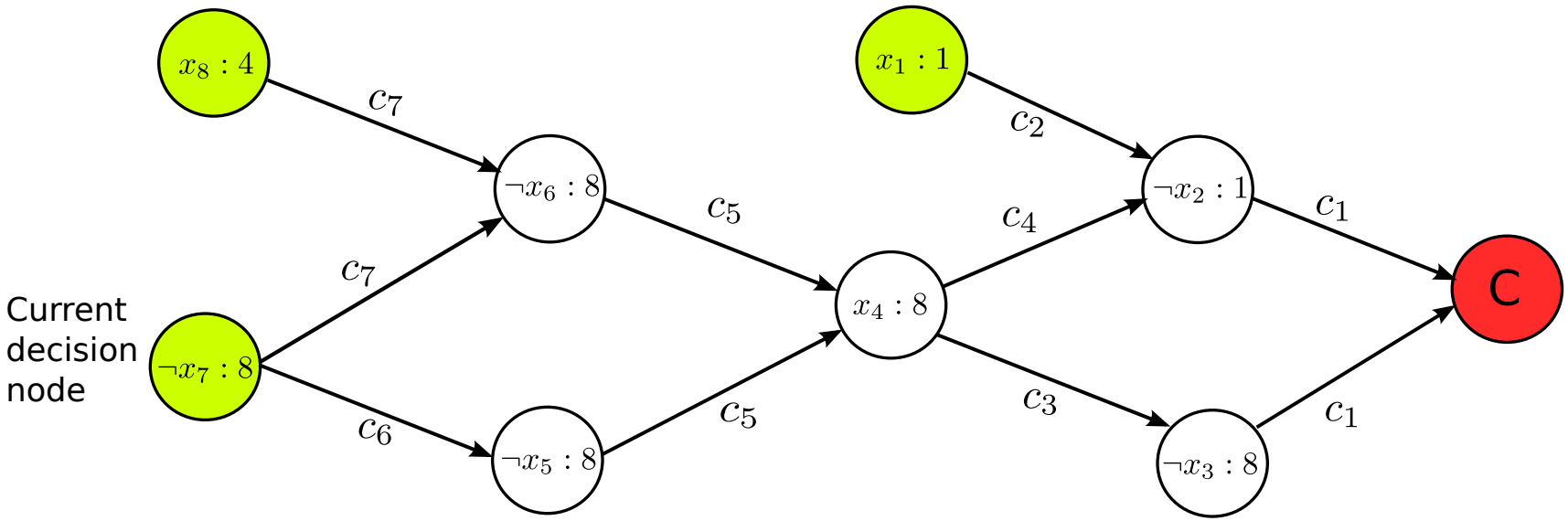
Unique Implication Point

- A node N in the implication graph is a unique implication point (UIP) if all paths from current decision node to the conflict node must go through N
- Same concept as dominator
- Is the current decision node a UIP? Yes
- Can there be multiple unique implication points? Yes

Unique Implication Point

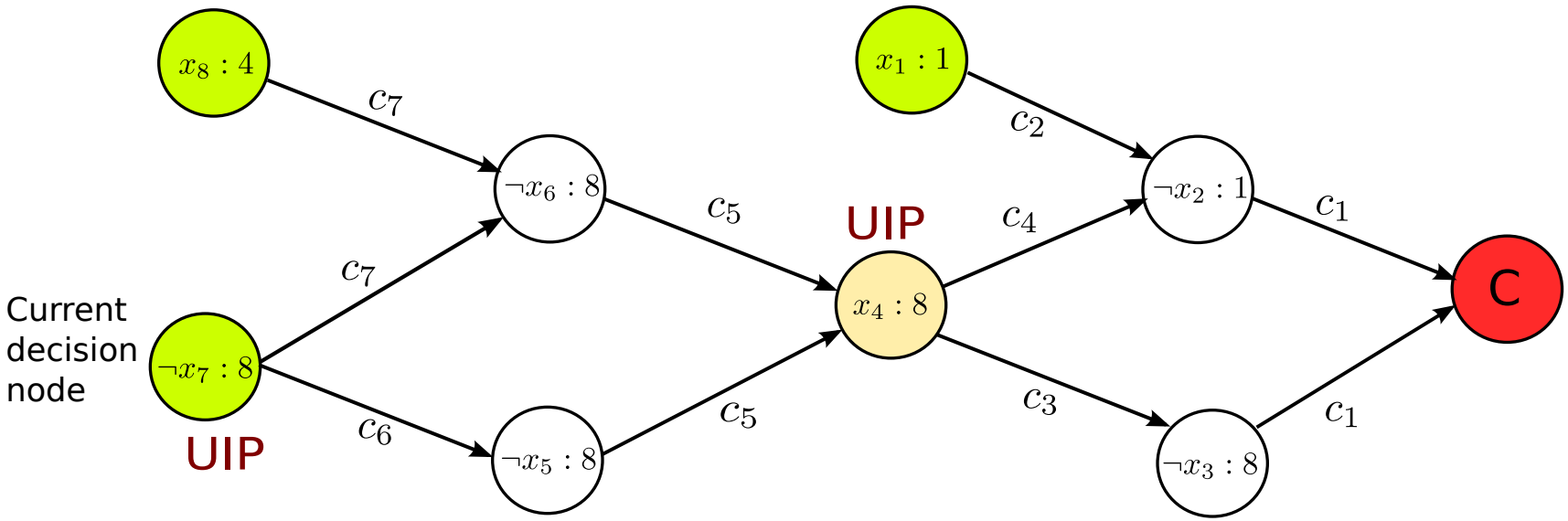
- A node N in the implication graph is a unique implication point (UIP) if all paths from current decision node to the conflict node must go through N
- Same concept as dominator
- Is the current decision node a UIP? Yes
- Can there be multiple unique implication points? Yes
- First unique implication point: UIP closest to conflict node

UIP Example



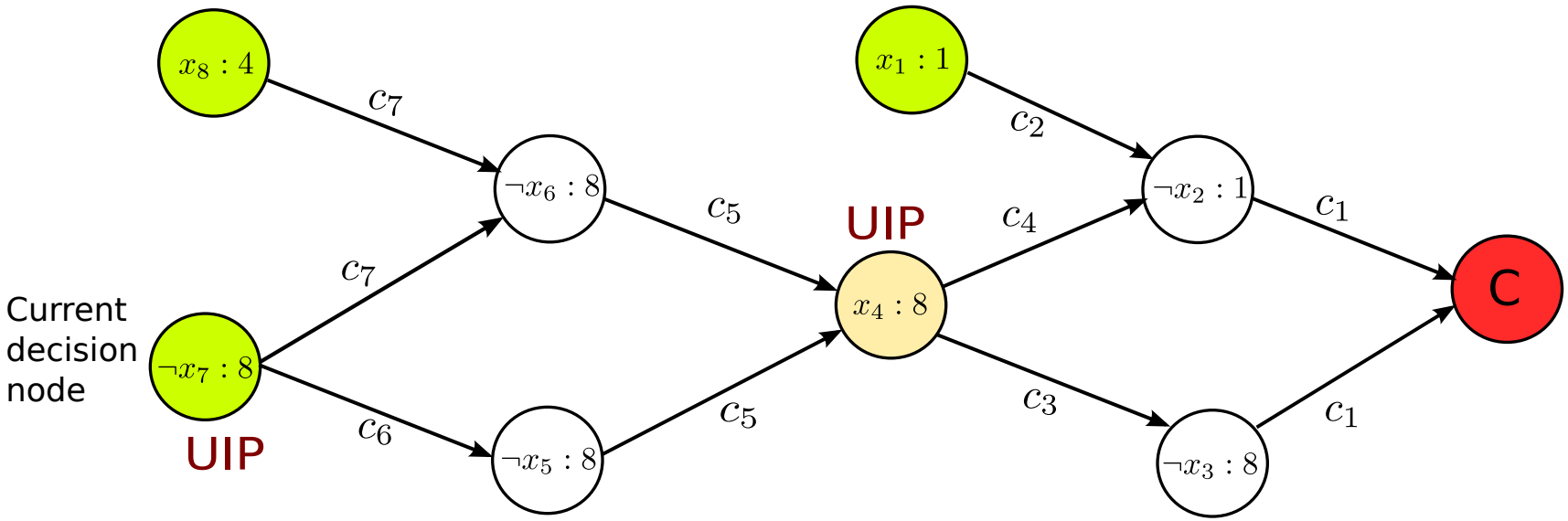
- Which nodes are UIP's?

UIP Example



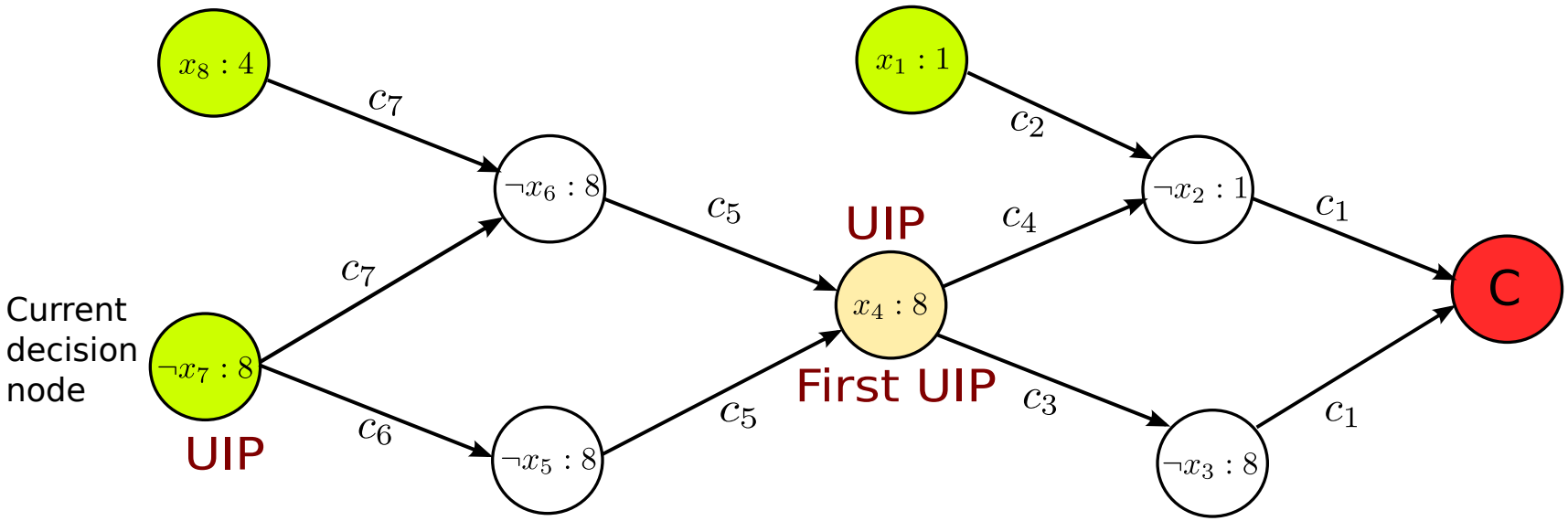
- Which nodes are UIP's? $\neg x_7 : 8, x_4 : 8$

UIP Example



- Which nodes are UIP's? $\neg x_7 : 8, x_4 : 8$
- Which node is first UIP?

UIP Example



- Which nodes are UIP's? $\neg x_7 : 8, x_4 : 8$
- Which node is first UIP? $x_4 : 8$

Using UIP and Resolution for Deriving Conflict Clause

- Inferring better conflict clauses: Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP

Using UIP and Resolution for Deriving Conflict Clause

- Inferring better conflict clauses: Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- Specifically: In current clause c, find last assigned literal l in c.

Using UIP and Resolution for Deriving Conflict Clause

- Inferring better conflict clauses: Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- Specifically: In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c'.

Using UIP and Resolution for Deriving Conflict Clause

- Inferring better conflict clauses: Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- Specifically: In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c'.
- Resolve c and c'.

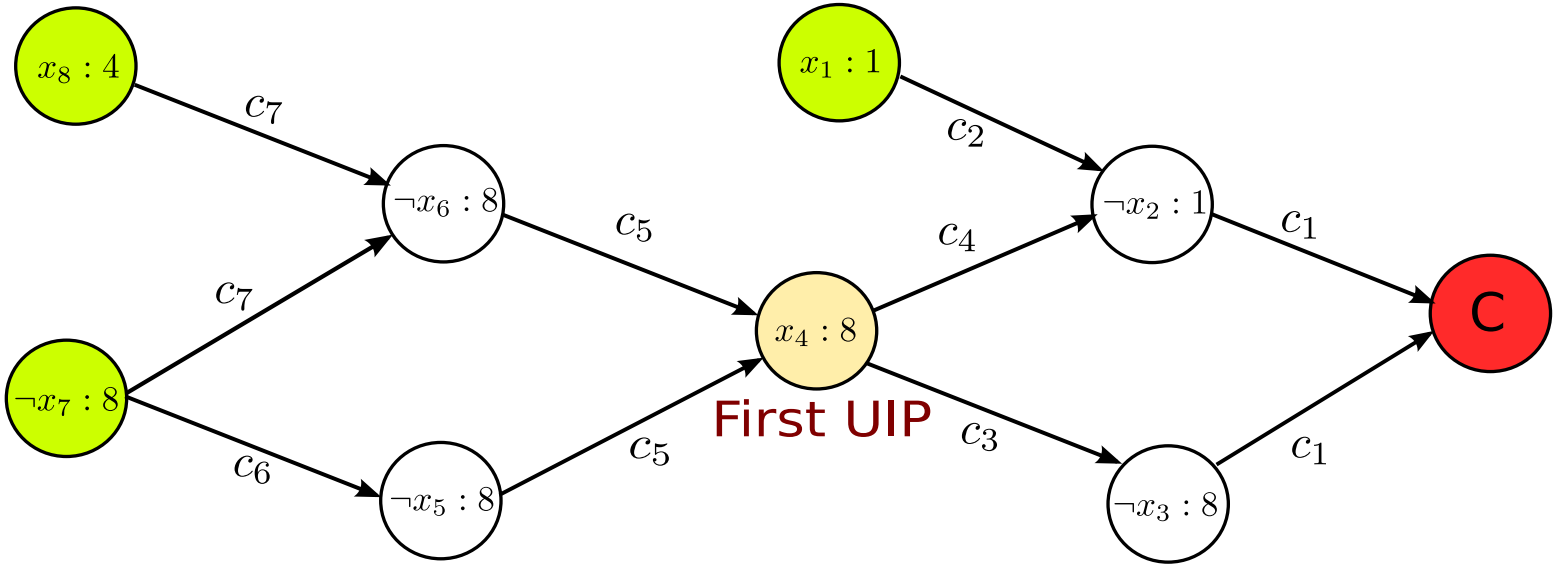
Using UIP and Resolution for Deriving Conflict Clause

- Inferring better conflict clauses: Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- Specifically: In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c'.
- Resolve c and c'.
- Set current clause be resolvent of c and c'.

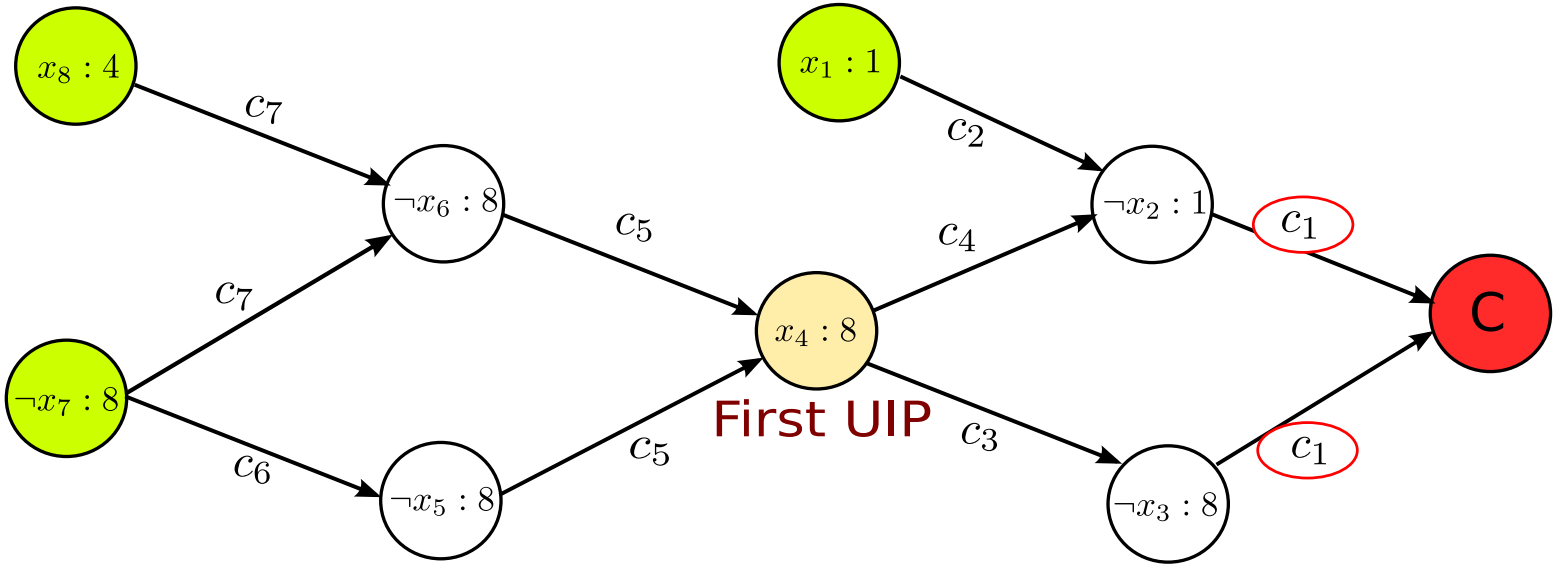
Using UIP and Resolution for Deriving Conflict Clause

- Inferring better conflict clauses: Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- Specifically: In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c'.
- Resolve c and c'.
- Set current clause be resolvent of c and c'.
- Repeat until current clause contains negation of the first UIP literal (as the single literal at current decision level)

Analyzing Conflict via Resolution Example

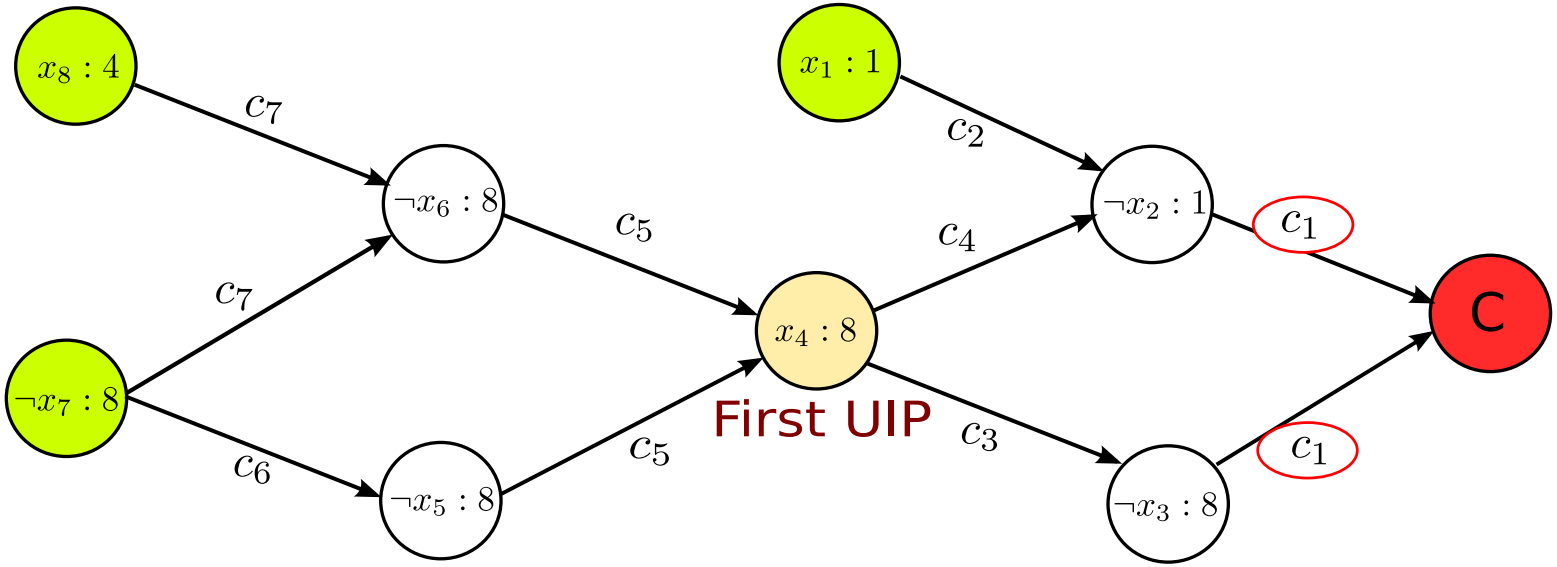


Analyzing Conflict via Resolution Example



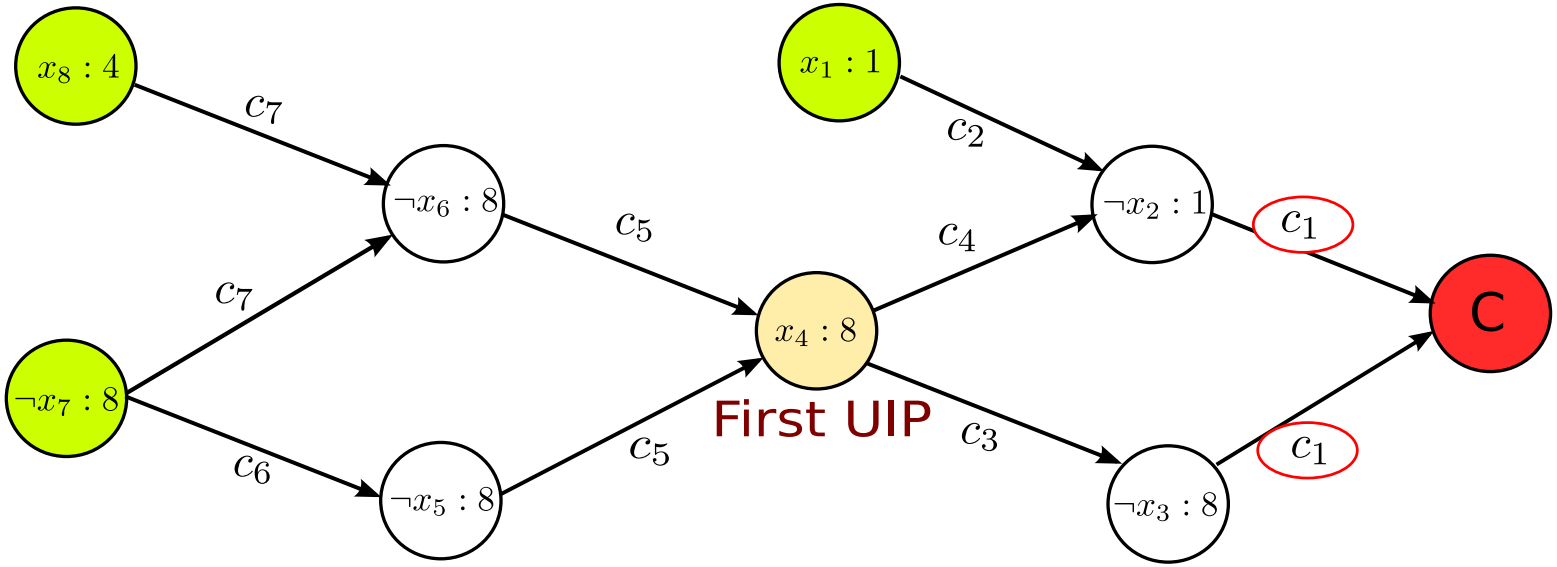
- What is c_1?

Analyzing Conflict via Resolution Example



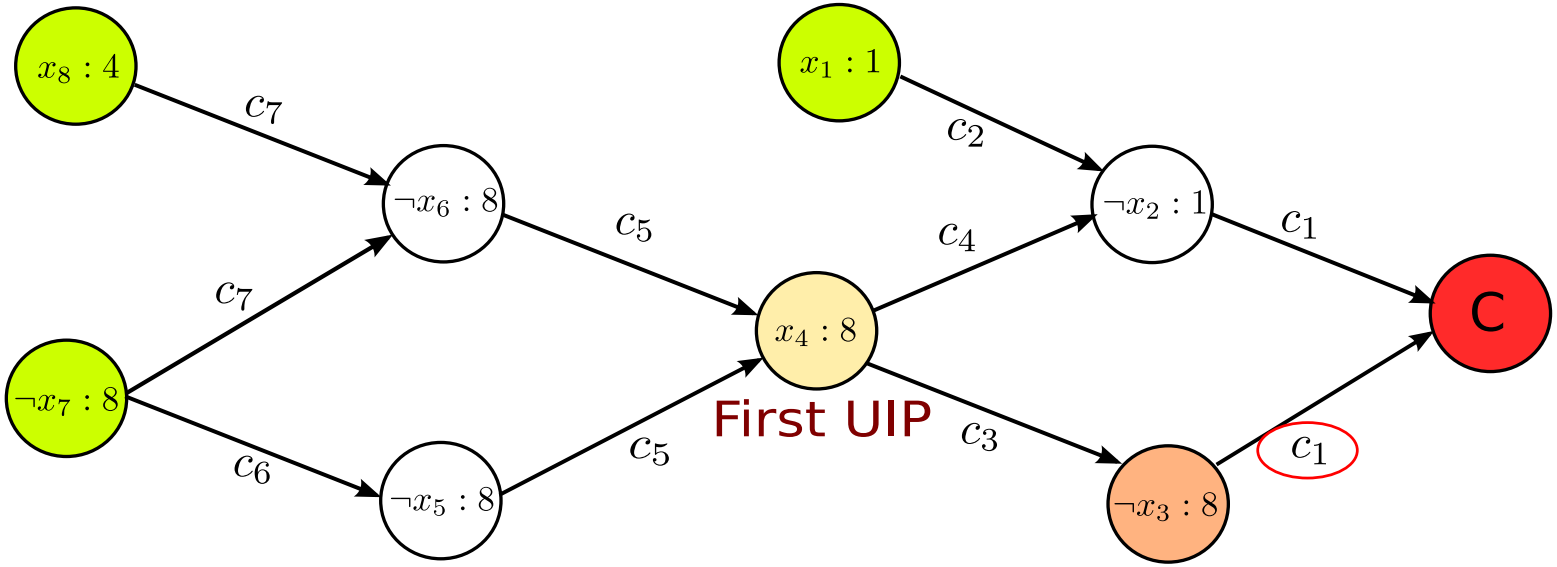
- What is c_1? $(x_2 \vee x_3)$

Analyzing Conflict via Resolution Example



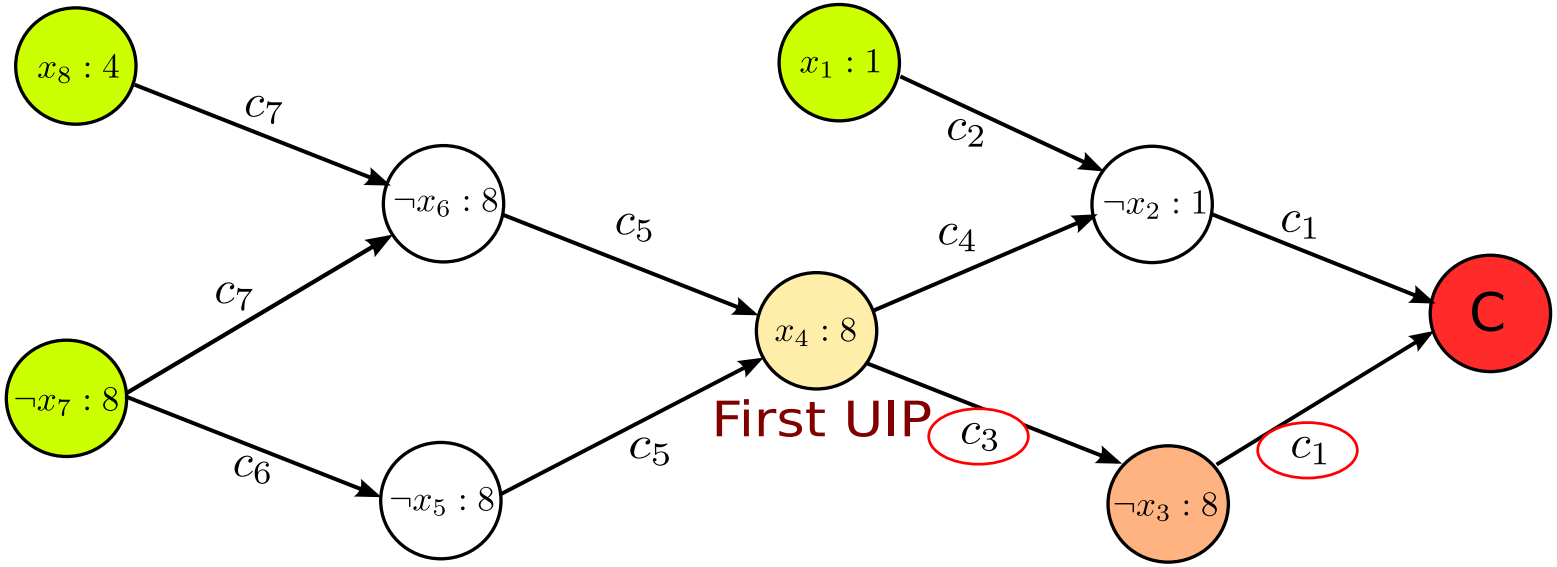
- What is c_1? $(x_2 \vee x_3)$
- Last assigned literal in c_1:

Analyzing Conflict via Resolution Example



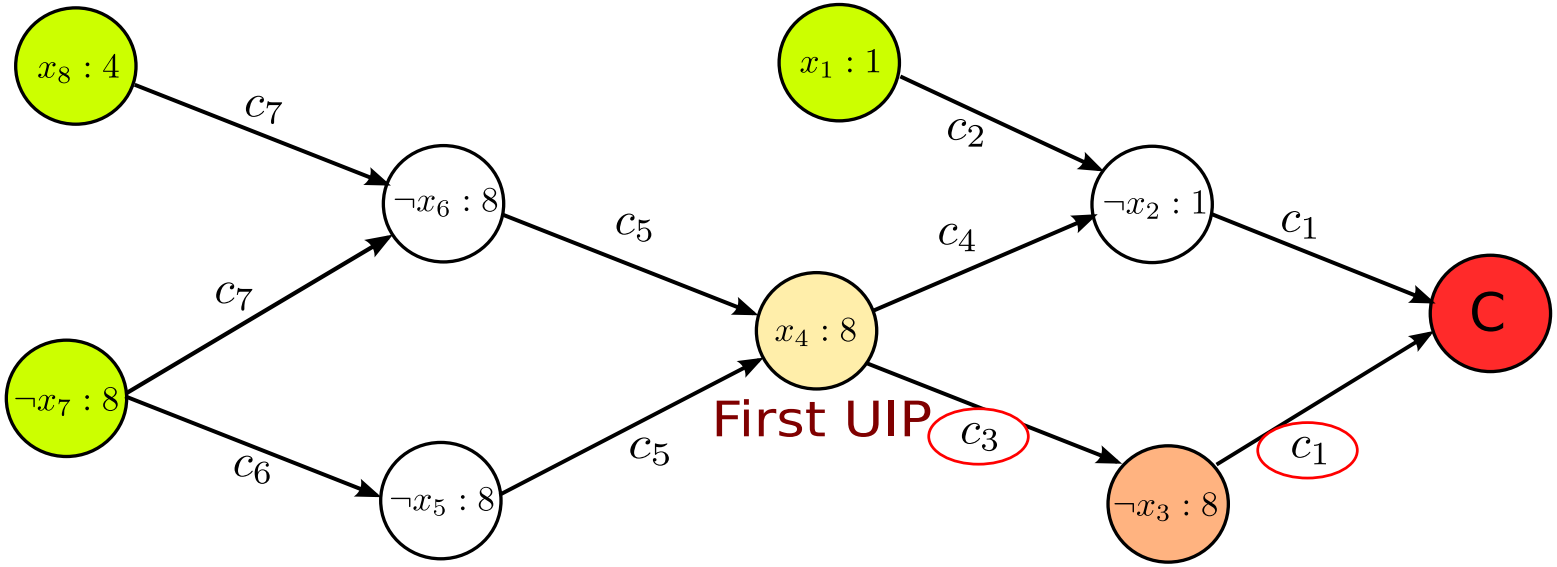
- What is c_1? $(x_2 \vee x_3)$
- Last assigned literal in c_1: $(\neg x_3)$

Analyzing Conflict via Resolution Example



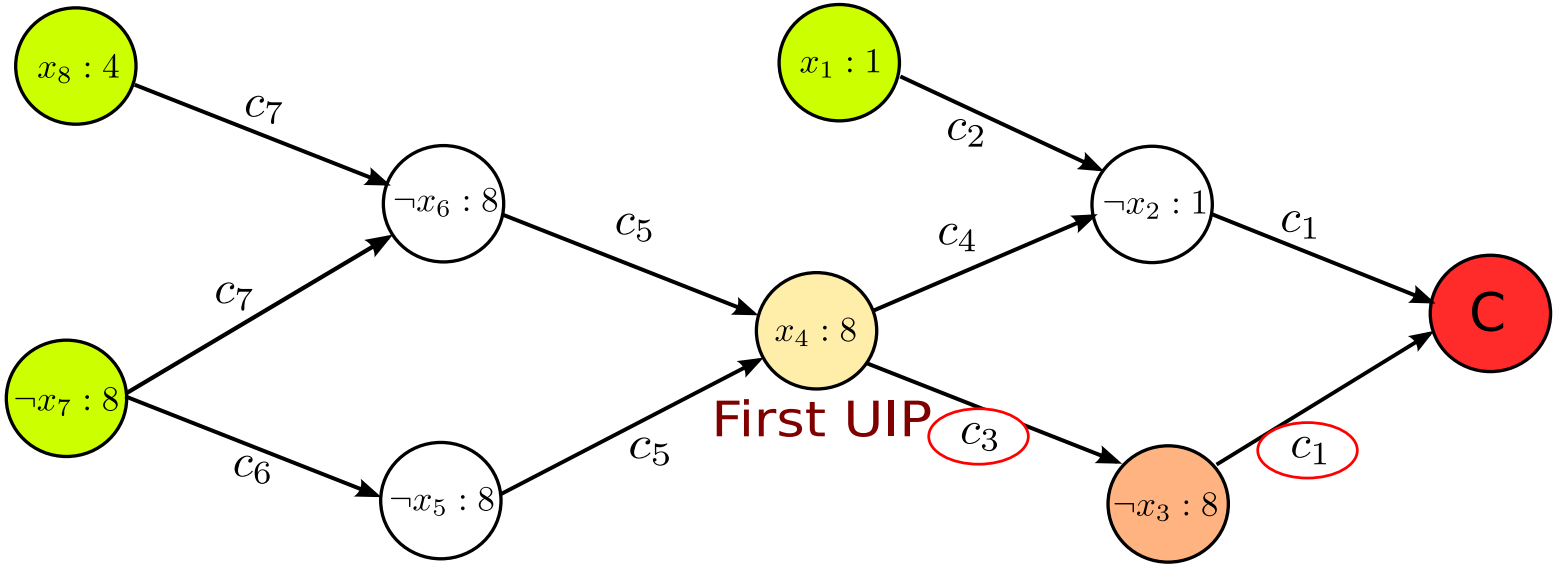
- What is c_1? $(x_2 \vee x_3)$
- Last assigned literal in c_1: $(\neg x_3)$
- Clause c_3 labeling incoming edge:

Analyzing Conflict via Resolution Example



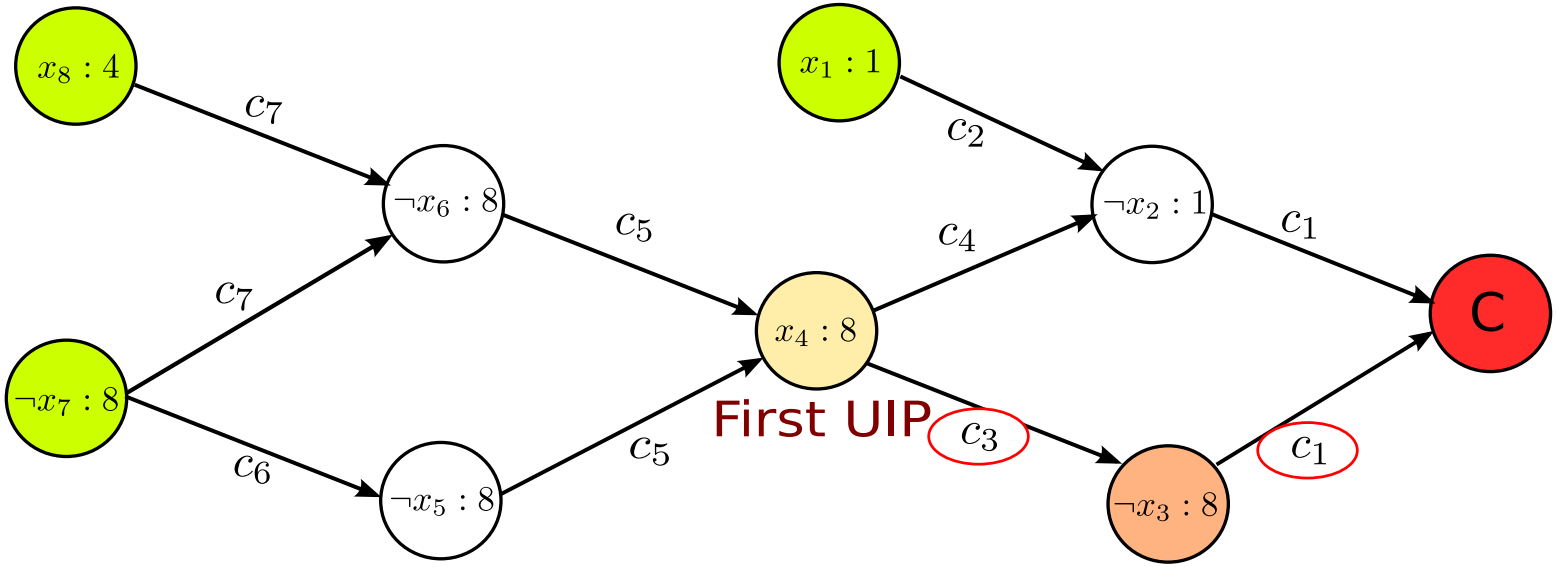
- What is c_1? $(x_2 \vee x_3)$
- Last assigned literal in c_1: $(\neg x_3)$
- Clause c_3 labeling incoming edge: $(\neg x_3 \vee \neg x_4)$

Analyzing Conflict via Resolution Example



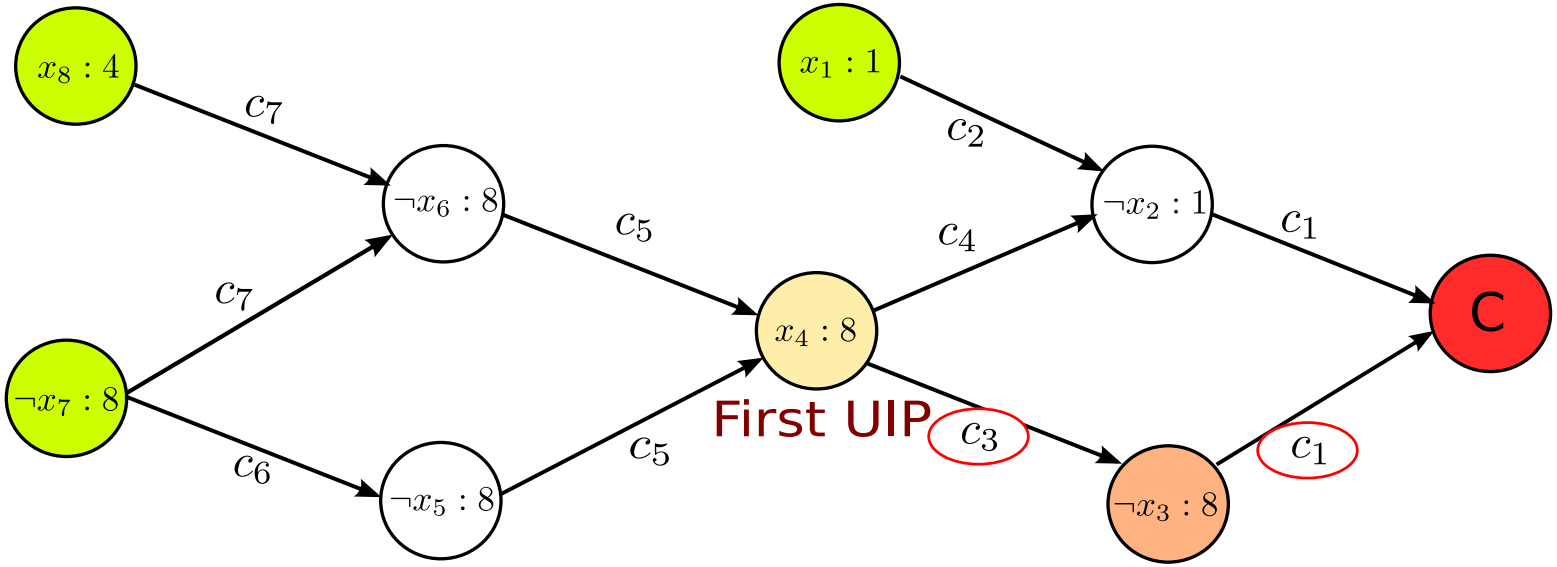
- What is c_1? $(x_2 \vee x_3)$
- Last assigned literal in c_1: $(\neg x_3)$
- Clause c_3 labeling incoming edge: $(\neg x_3 \vee \neg x_4)$
- Resolve c_1 and c_3:

Analyzing Conflict via Resolution Example



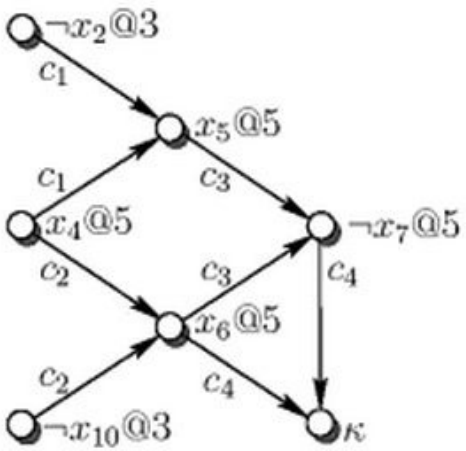
- What is c_1? $(x_2 \vee x_3)$
- Last assigned literal in c_1: $(\neg x_3)$
- Clause c_3 labeling incoming edge: $(\neg x_3 \vee \neg x_4)$
- Resolve c_1 and c_3: $x_2 \vee \neg x_4$

Analyzing Conflict via Resolution Example



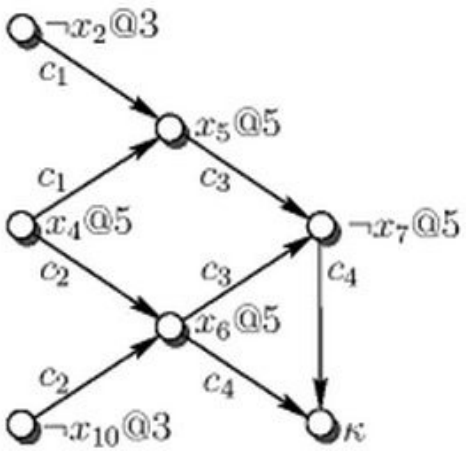
- What is c_1? $(x_2 \vee x_3)$
- Last assigned literal in c_1: $(\neg x_3)$
- Clause c_3 labeling incoming edge: $(\neg x_3 \vee \neg x_4)$
- Resolve c_1 and c_3: $x_2 \vee \neg x_4$
- $\neg x_4$ only literal from decision level 8 $\Rightarrow$ $x_2 \vee \neg x_4$ conflict clause

Another Example



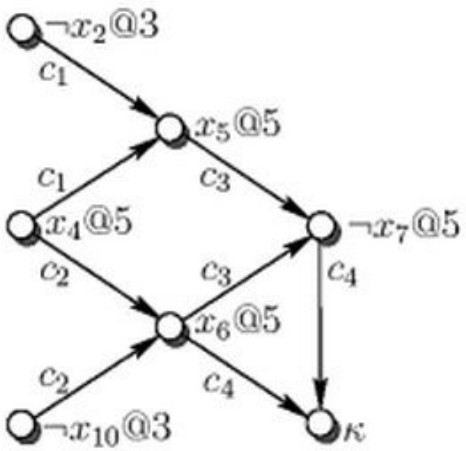
- What is the first UIP?

Another Example



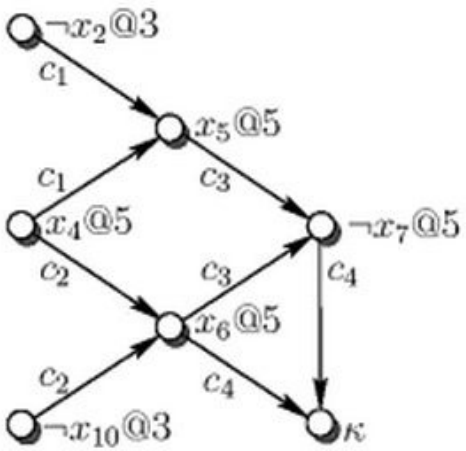
- What is the first UIP? $x_4@5$

Another Example



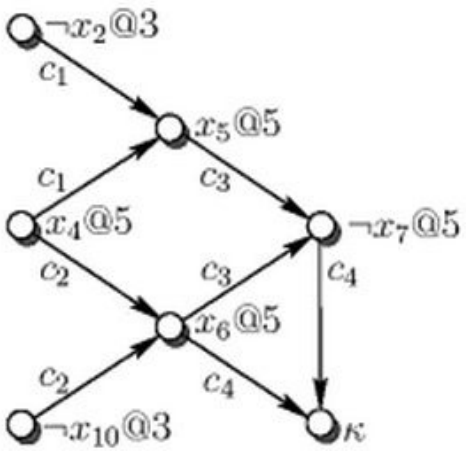
- What is the first UIP? $x_4@5$
- Start with clause c_4:

Another Example



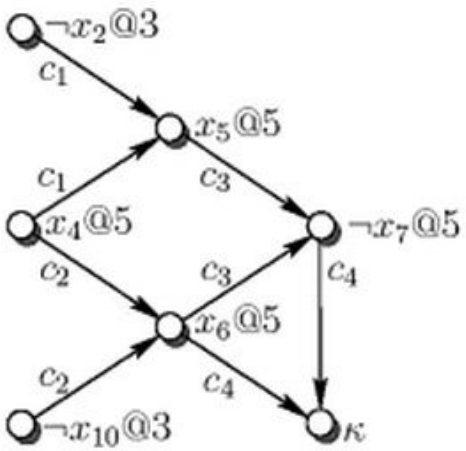
- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$

Another Example



- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$
- Suppose $\neg x_7$ assigned later than x_6, so pick $\neg x_7$

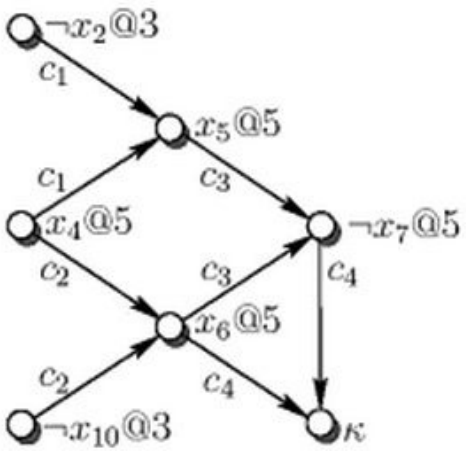
Another Example



- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$
- Suppose $\neg x_7$ assigned later than x_6, so pick $\neg x_7$

- Clause on incoming edge to $\neg x_7$:

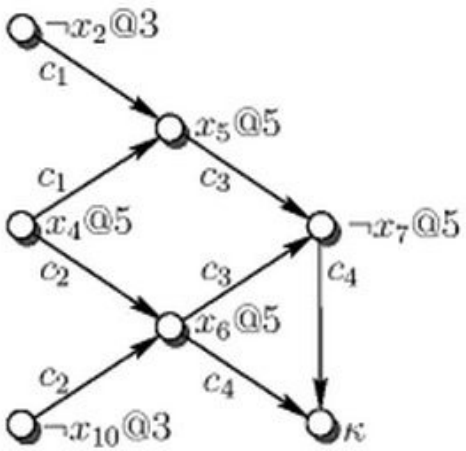
Another Example



- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$
- Suppose $\neg x_7$ assigned later than x_6, so pick $\neg x_7$

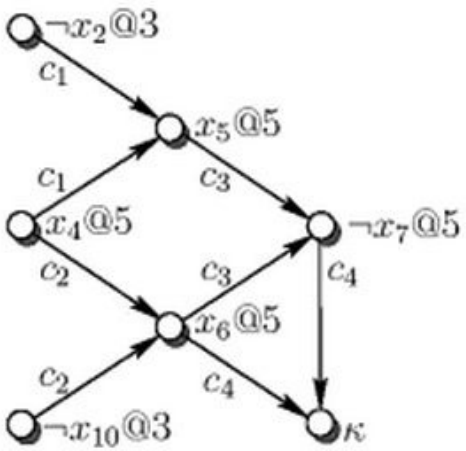
- Clause on incoming edge to $\neg x_7$: $c_3 : (\neg x_5 \vee \neg x_6 \vee \neg x_7)$

Another Example



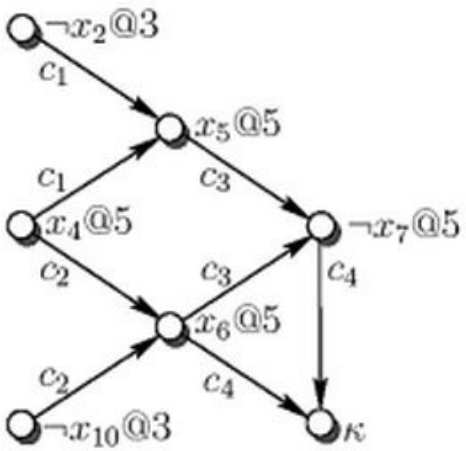
- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$
- Suppose $\neg x_7$ assigned later than x_6, so pick $\neg x_7$
- Clause on incoming edge to $\neg x_7$: $c_3 : (\neg x_5 \vee \neg x_6 \vee \neg x_7)$
- Resolve c_3, c_4:

Another Example



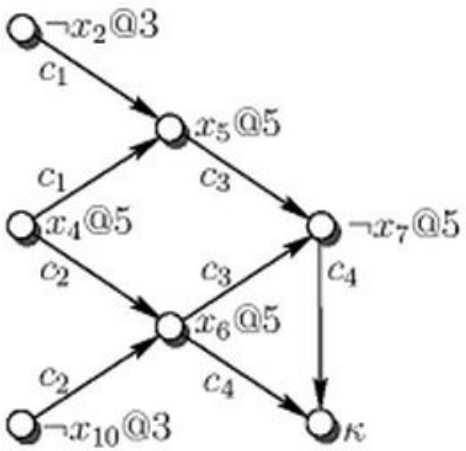
- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$
- Suppose $\neg x_7$ assigned later than x_6, so pick $\neg x_7$
- Clause on incoming edge to $\neg x_7$: $c_3 : (\neg x_5 \vee \neg x_6 \vee \neg x_7)$
- Resolve c_3, c_4: $\neg x_5 \vee \neg x_6$

Another Example



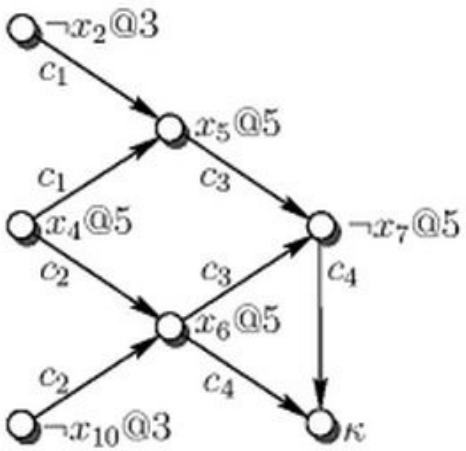
- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$
- Suppose $\neg x_7$ assigned later than x_6, so pick $\neg x_7$
- Clause on incoming edge to $\neg x_7$: $c_3 : (\neg x_5 \vee \neg x_6 \vee \neg x_7)$
- Resolve c_3, c_4: $\neg x_5 \vee \neg x_6$
- Suppose x_6 assigned later, pick x_6

Another Example



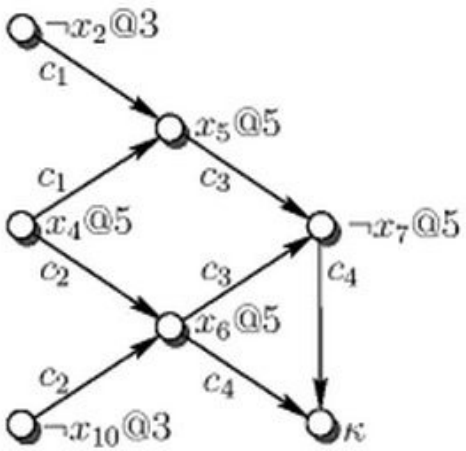
- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$
- Suppose $\neg x_7$ assigned later than x_6, so pick $\neg x_7$
- Clause on incoming edge to $\neg x_7$: $c_3 : (\neg x_5 \vee \neg x_6 \vee \neg x_7)$
- Resolve c_3, c_4: $\neg x_5 \vee \neg x_6$
- Suppose x_6 assigned later, pick x_6
- Clause on incoming edge:

Another Example



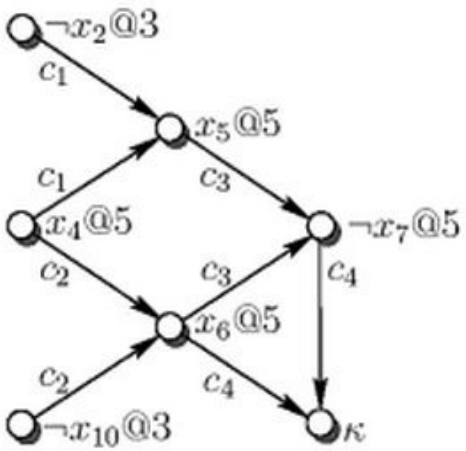
- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$
- Suppose $\neg x_7$ assigned later than x_6, so pick $\neg x_7$
- Clause on incoming edge to $\neg x_7$: $c_3 : (\neg x_5 \vee \neg x_6 \vee \neg x_7)$
- Resolve c_3, c_4: $\neg x_5 \vee \neg x_6$
- Suppose x_6 assigned later, pick x_6
- Clause on incoming edge: $c_2 : \neg x_4 \vee x_{10} \vee x_6$

Another Example



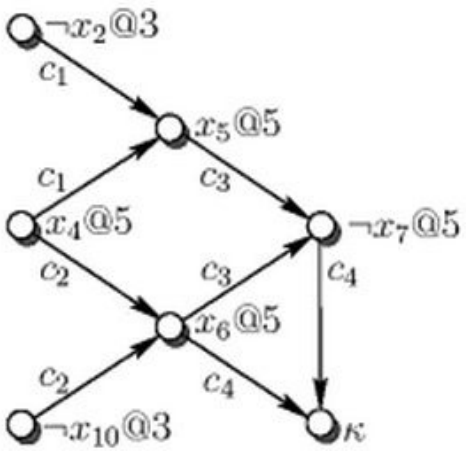
- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$
- Suppose $\neg x_7$ assigned later than x_6, so pick $\neg x_7$
- Clause on incoming edge to $\neg x_7$: $c_3 : (\neg x_5 \vee \neg x_6 \vee \neg x_7)$
- Resolve c_3, c_4: $\neg x_5 \vee \neg x_6$
- Suppose x_6 assigned later, pick x_6
- Clause on incoming edge: $c_2 : \neg x_4 \vee x_{10} \vee x_6$
- Resolve current clause with c_2:

Another Example



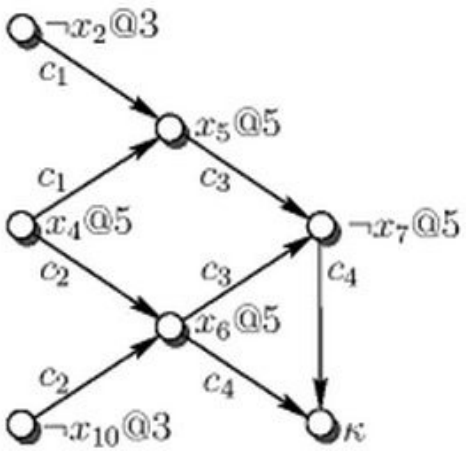
- What is the first UIP? $x_4@5$
- Start with clause c_4: $\neg x_6 \vee x_7$
- Suppose $\neg x_7$ assigned later than x_6, so pick $\neg x_7$
- Clause on incoming edge to $\neg x_7$: $c_3 : (\neg x_5 \vee \neg x_6 \vee \neg x_7)$
- Resolve c_3, c_4: $\neg x_5 \vee \neg x_6$
- Suppose x_6 assigned later, pick x_6
- Clause on incoming edge: $c_2 : \neg x_4 \vee x_{10} \vee x_6$
- Resolve current clause with c_2: $\neg x_4 \vee x_{10} \vee \neg x_5$

Another Example, cont.



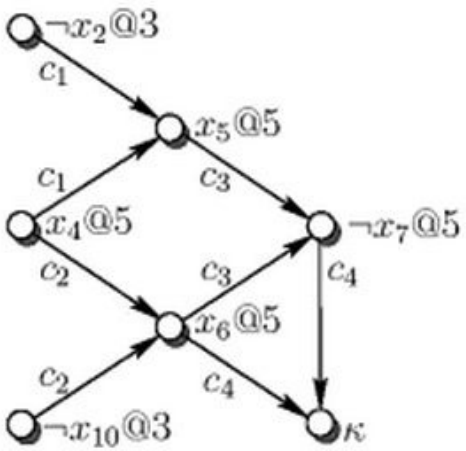
- Current clause:

Another Example, cont.



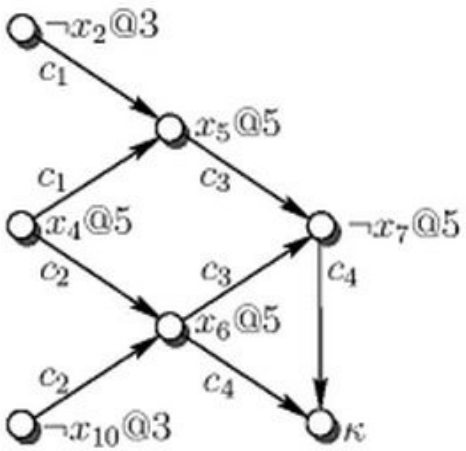
- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$

Another Example, cont.



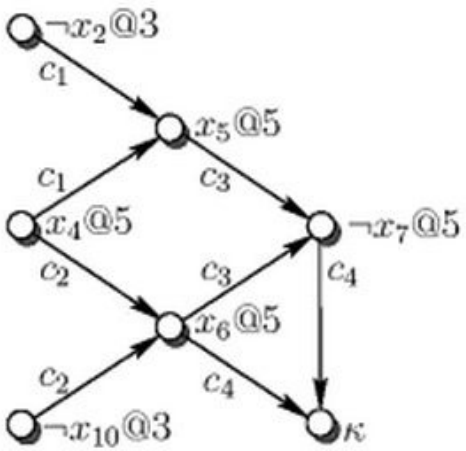
- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$
- Are we done?

Another Example, cont.



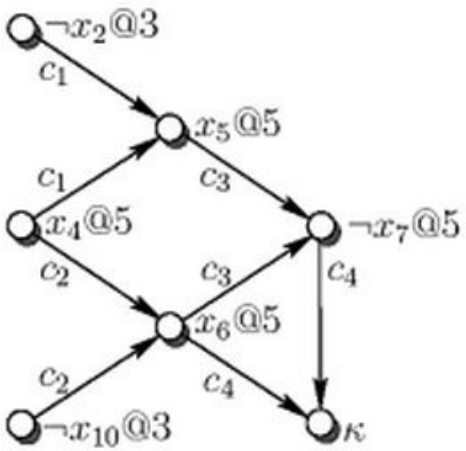
- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$
- Are we done? No (because x_5 is also from current decision level)

Another Example, cont.



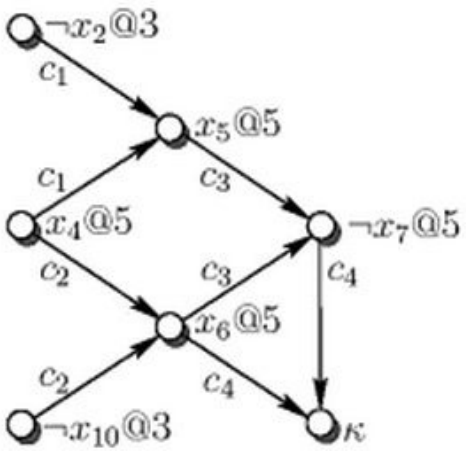
- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$
- Are we done? No (because x_5 is also from current decision level)
- Pick last assigned literal: x_5

Another Example, cont.



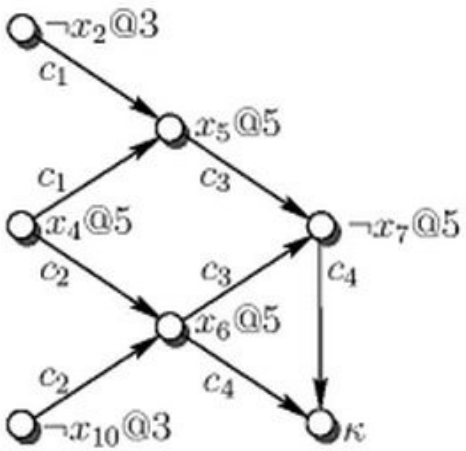
- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$
- Are we done? No (because x_5 is also from current decision level)
- Pick last assigned literal: x_5
- Incoming edge to x_5:

Another Example, cont.



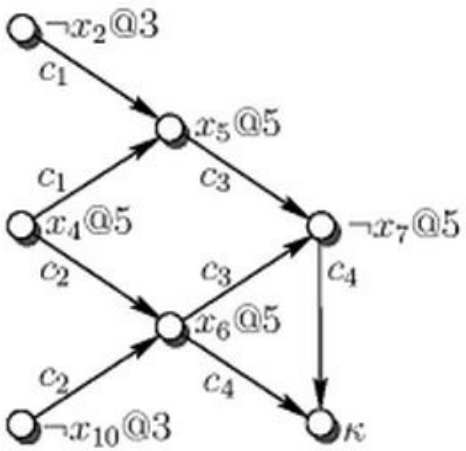
- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$
- Are we done? No (because x_5 is also from current decision level)
- Pick last assigned literal: x_5
- Incoming edge to x_5: $x_2 \vee \neg x_4 \vee x_5$

Another Example, cont.



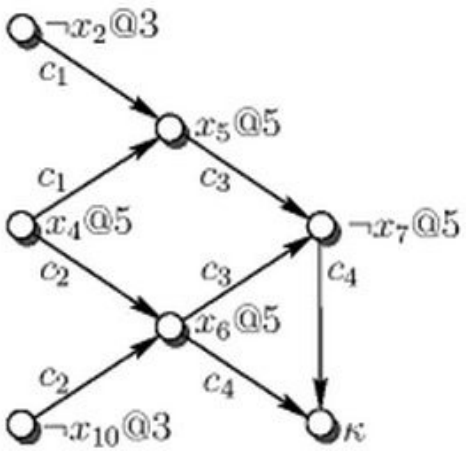
- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$
- Are we done? No (because x_5 is also from current decision level)
- Pick last assigned literal: x_5
- Incoming edge to x_5: $x_2 \vee \neg x_4 \vee x_5$
- Resolve with current clause:

Another Example, cont.



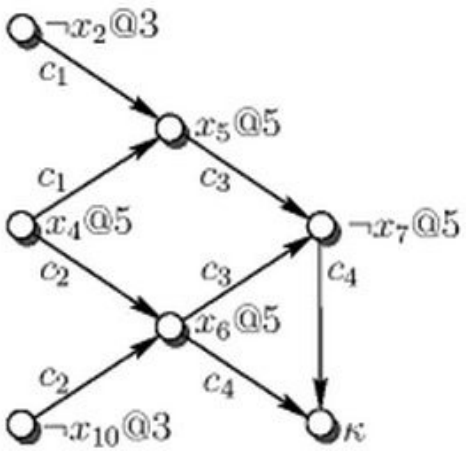
- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$
- Are we done? No (because x_5 is also from current decision level)
- Pick last assigned literal: x_5
- Incoming edge to x_5: $x_2 \vee \neg x_4 \vee x_5$
- Resolve with current clause: $x_2 \vee \neg x_4 \vee x_{10}$

Another Example, cont.



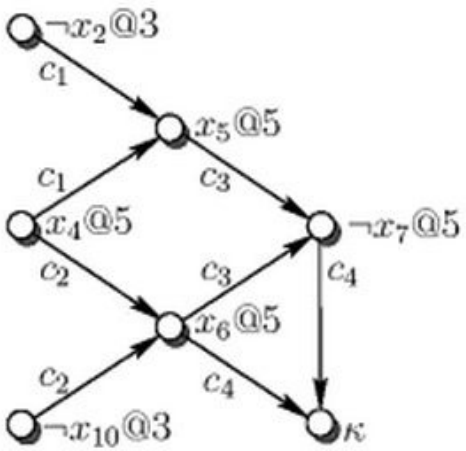
- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$
- Are we done? No (because x_5 is also from current decision level)
- Pick last assigned literal: x_5
- Incoming edge to x_5: $x_2 \vee \neg x_4 \vee x_5$
- Resolve with current clause: $x_2 \vee \neg x_4 \vee x_{10}$
- Are we done?

Another Example, cont.



- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$
- Are we done? No (because x_5 is also from current decision level)
- Pick last assigned literal: x_5
- Incoming edge to x_5: $x_2 \vee \neg x_4 \vee x_5$
- Resolve with current clause: $x_2 \vee \neg x_4 \vee x_{10}$
- Are we done? Yes!

Another Example, cont.



- Current clause: $\neg x_4 \vee x_{10} \vee \neg x_5$
- Are we done? No (because x_5 is also from current decision level)
- Pick last assigned literal: x_5
- Incoming edge to x_5: $x_2 \vee \neg x_4 \vee x_5$
- Resolve with current clause: $x_2 \vee \neg x_4 \vee x_{10}$
- Are we done? Yes!
- New conflict clause: $x_2 \vee \neg x_4 \vee x_{10}$

Why is this correct?

- Observe: At each step, we perform resolution between a clause c on incoming edge of node l and a clause c' on outgoing edge of l

Why is this correct?

- Observe: At each step, we perform resolution between a clause c on incoming edge of node l and a clause c' on outgoing edge of l
- Why can we always resolve c and c'?

Why is this correct?

- Observe: At each step, we perform resolution between a clause c on incoming edge of node l and a clause c' on outgoing edge of l
- Why can we always resolve c and c'? By construction, c must contain l, and c' must contain $\neg l$

Why is this correct?

- Observe: At each step, we perform resolution between a clause c on incoming edge of node l and a clause c' on outgoing edge of l
- Why can we always resolve c and c'? By construction, c must contain l, and c' must contain $\neg l$
- Furthermore, since c and c' are clauses from original formula, any clause we derive is implied by the original formula

Why is this correct?

- Observe: At each step, we perform resolution between a clause c on incoming edge of node l and a clause c' on outgoing edge of l
- Why can we always resolve c and c'? By construction, c must contain l, and c' must contain $\neg l$
- Furthermore, since c and c' are clauses from original formula, any clause we derive is implied by the original formula
- Thus, final conflict clause is implied by the original formula!

Why is this correct?

- Observe: At each step, we perform resolution between a clause c on incoming edge of node l and a clause c' on outgoing edge of l
- Why can we always resolve c and c'? By construction, c must contain l, and c' must contain $\neg l$
- Furthermore, since c and c' are clauses from original formula, any clause we derive is implied by the original formula
- Thus, final conflict clause is implied by the original formula!
- It's unclear whether there is a deep reason this works well

Why is this correct?

- Observe: At each step, we perform resolution between a clause c on incoming edge of node l and a clause c' on outgoing edge of l
- Why can we always resolve c and c'? By construction, c must contain l, and c' must contain $\neg l$
- Furthermore, since c and c' are clauses from original formula, any clause we derive is implied by the original formula
- Thus, final conflict clause is implied by the original formula!
- It's unclear whether there is a deep reason this works well
- Empirical results show this strategy is effective ...

Backtracking

- Recall: AnalyzeConflict has two goals.

Backtracking

- Recall: AnalyzeConflict has two goals.
- First goal: Deriving conflict clauses ✓

Backtracking

- Recall: AnalyzeConflict has two goals.
- First goal: Deriving conflict clauses ✓
- Second goal: Figure out what level to backtrack to

Backtracking

- Recall: AnalyzeConflict has two goals.
- First goal: Deriving conflict clauses ✓
- Second goal: Figure out what level to backtrack to
- Backtrack to level d means delete all variable assignments made after level d (but assignments at level d not deleted)

Backtracking

- Recall: AnalyzeConflict has two goals.
- First goal: Deriving conflict clauses ✓
- Second goal: Figure out what level to backtrack to
- Backtrack to level d means delete all variable assignments made after level d (but assignments at level d not deleted)
- Next: Talk about how to infer a good level to backtrack to

Backtracking and Asserting Clauses

- A good strategy: We want to backtrack to a level that makes conflict clause c an asserting clause in the next step

Backtracking and Asserting Clauses

- A good strategy: We want to backtrack to a level that makes conflict clause c an asserting clause in the next step
- Asserting clause is a clause with exactly one unassigned literal

Backtracking and Asserting Clauses

- A good strategy: We want to backtrack to a level that makes conflict clause c an asserting clause in the next step
- Asserting clause is a clause with exactly one unassigned literal
- Why do we want to make c an asserting clause?

Backtracking and Asserting Clauses

- A good strategy: We want to backtrack to a level that makes conflict clause c an asserting clause in the next step
- Asserting clause is a clause with exactly one unassigned literal
- Why do we want to make c an asserting clause?
- BCP will force an assignment to unassigned literal l in c

Choosing Backtracking Level

- Question: If we want to make conflict clause c an asserting clause in the next step, what level do we need to backtrack to?

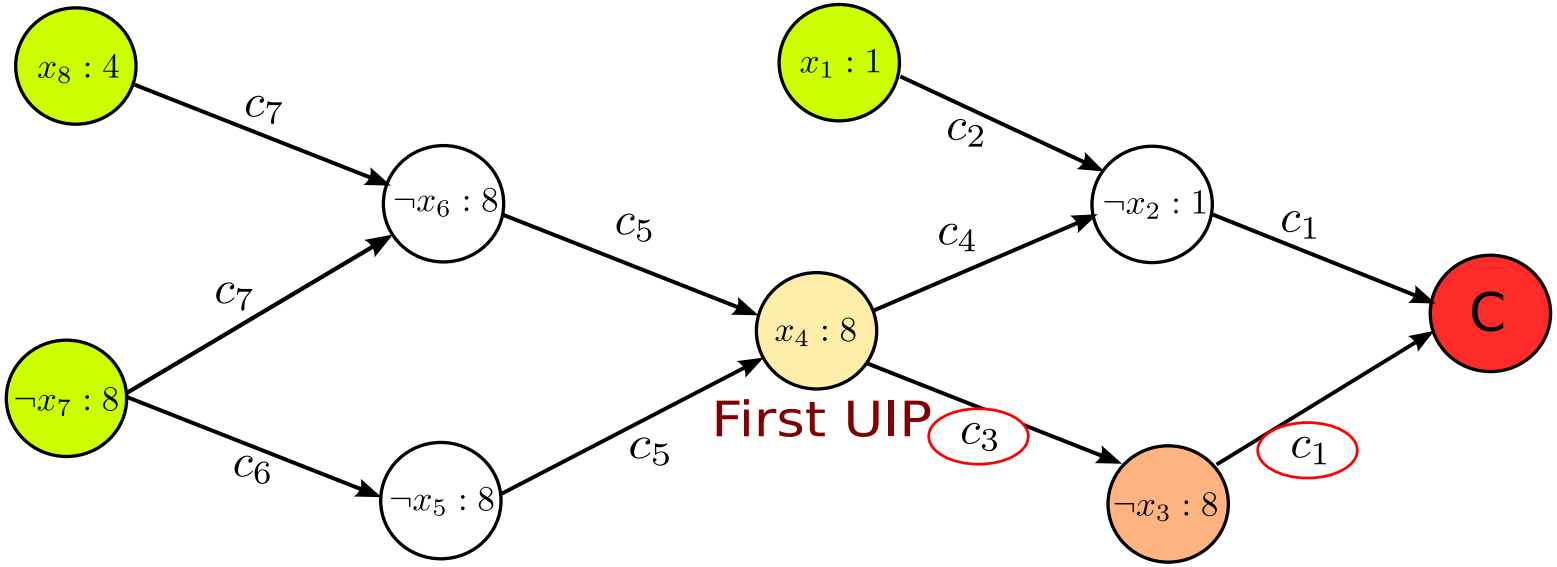
Choosing Backtracking Level

- Question: If we want to make conflict clause c an asserting clause in the next step, what level do we need to backtrack to?
- Answer: If l is the literal in c with second highest decision level d, backtrack to the level d

Choosing Backtracking Level

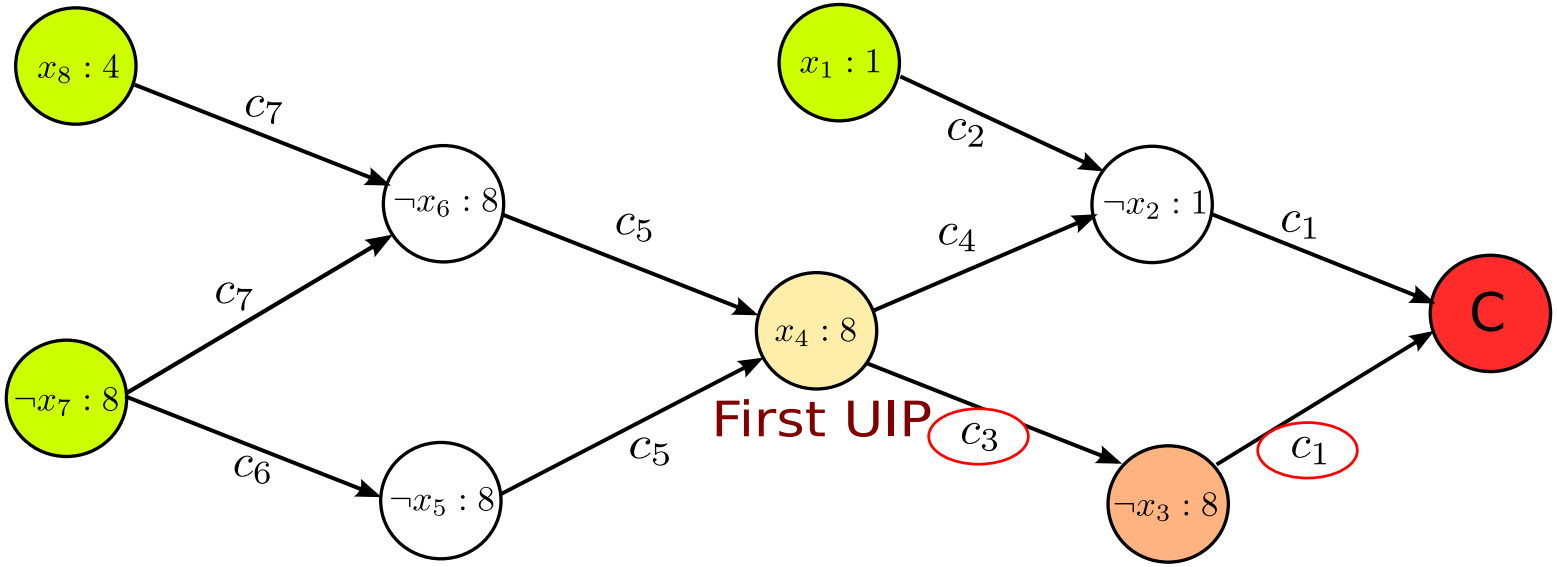
- Question: If we want to make conflict clause c an asserting clause in the next step, what level do we need to backtrack to?
- Answer: If l is the literal in c with second highest decision level d, backtrack to the level d
- Why? Since conflict clause contains only one literal, say l', from the first highest decision level, backtracking to d will assert l'!

Going Back to Example



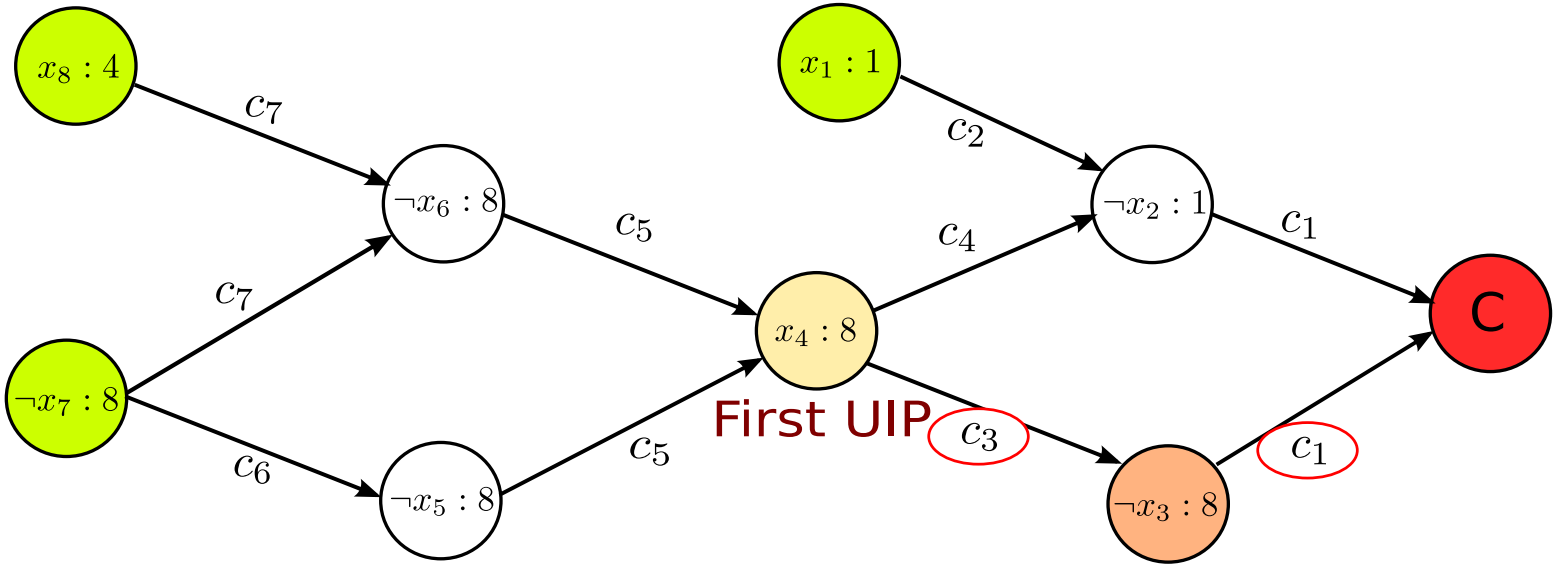
- Recall: We obtained the conflict clause $x_2 \vee \neg x_4$

Going Back to Example



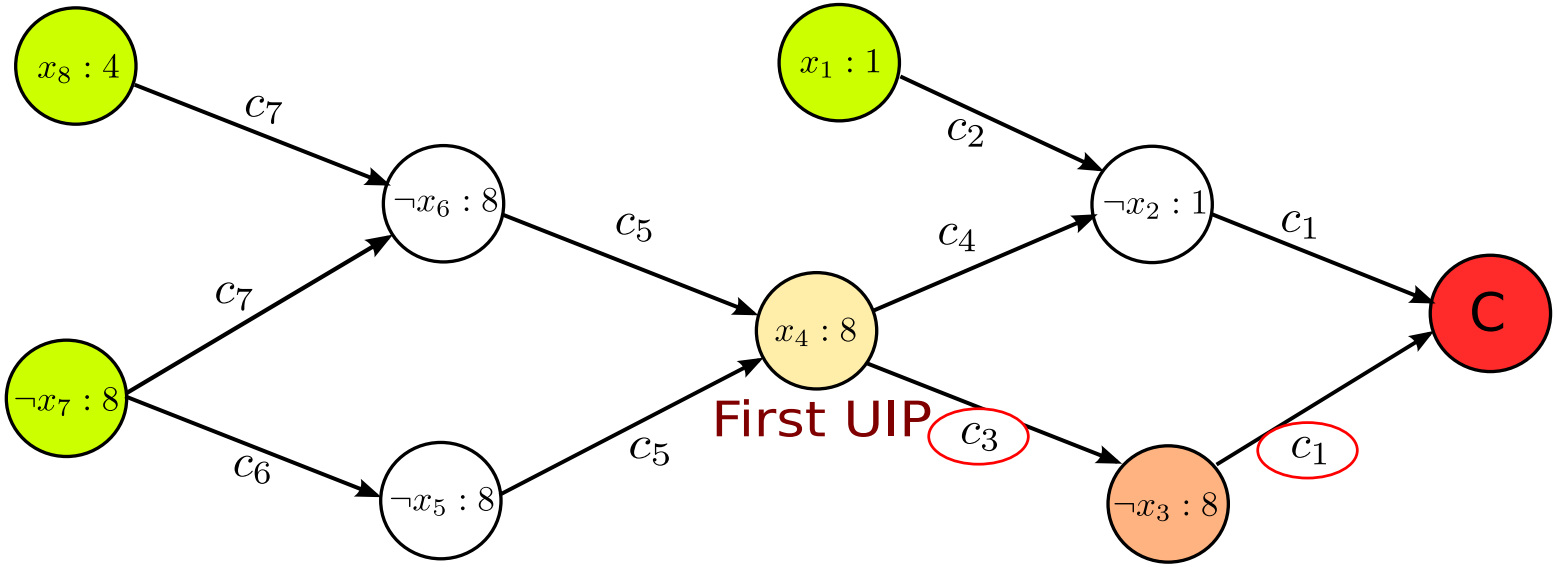
- Recall: We obtained the conflict clause $x_2 \vee \neg x_4$
- What level do we backtrack to?

Going Back to Example



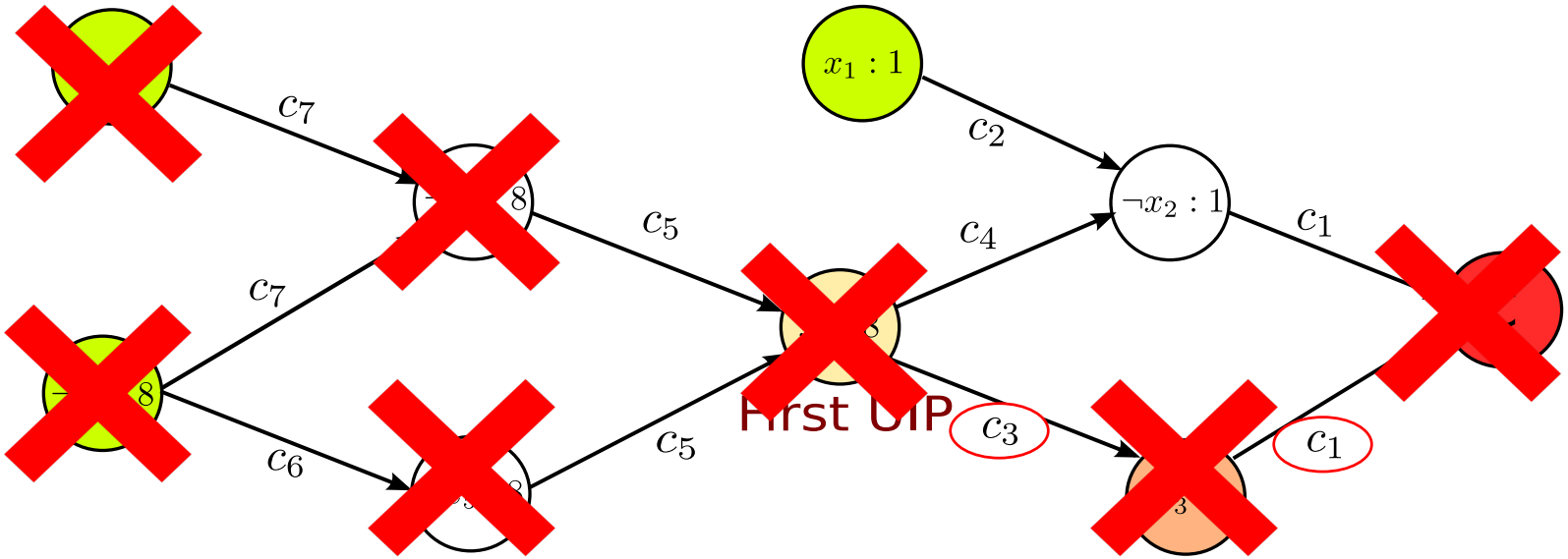
- Recall: We obtained the conflict clause $x_2 \vee \neg x_4$
- What level do we backtrack to? decision level 1

Going Back to Example



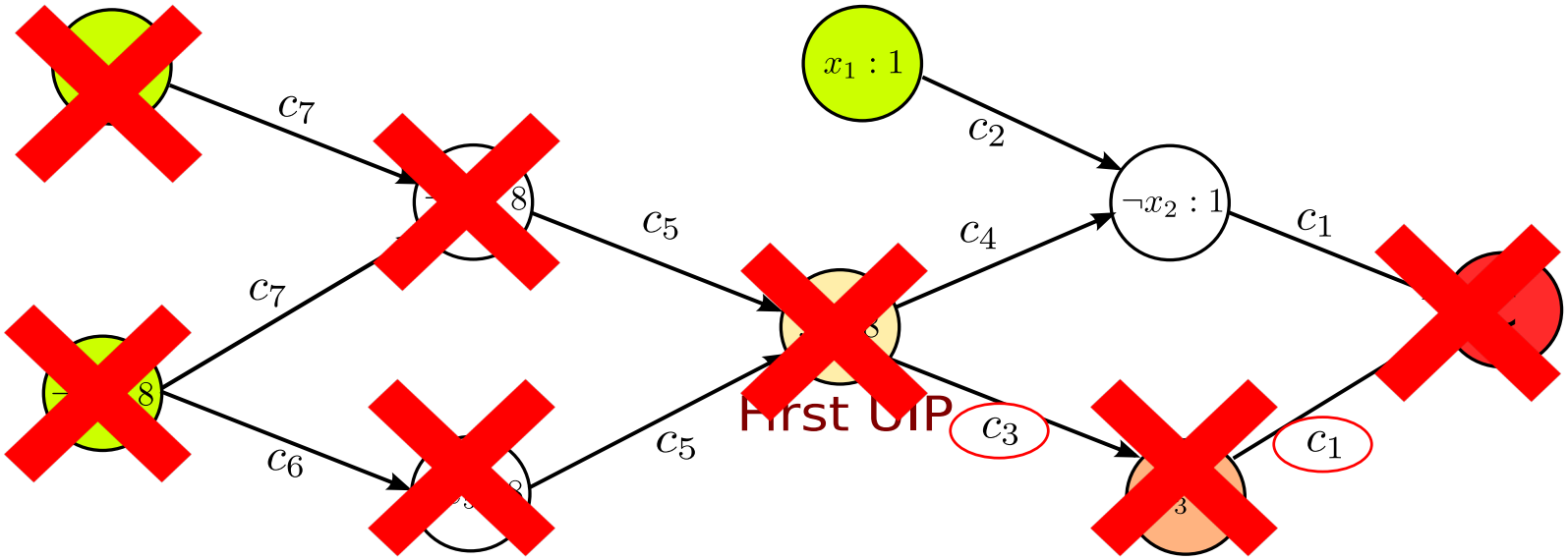
- Recall: We obtained the conflict clause $x_2 \vee \neg x_4$
- What level do we backtrack to? decision level 1
- What do we delete in the graph?

Going Back to Example



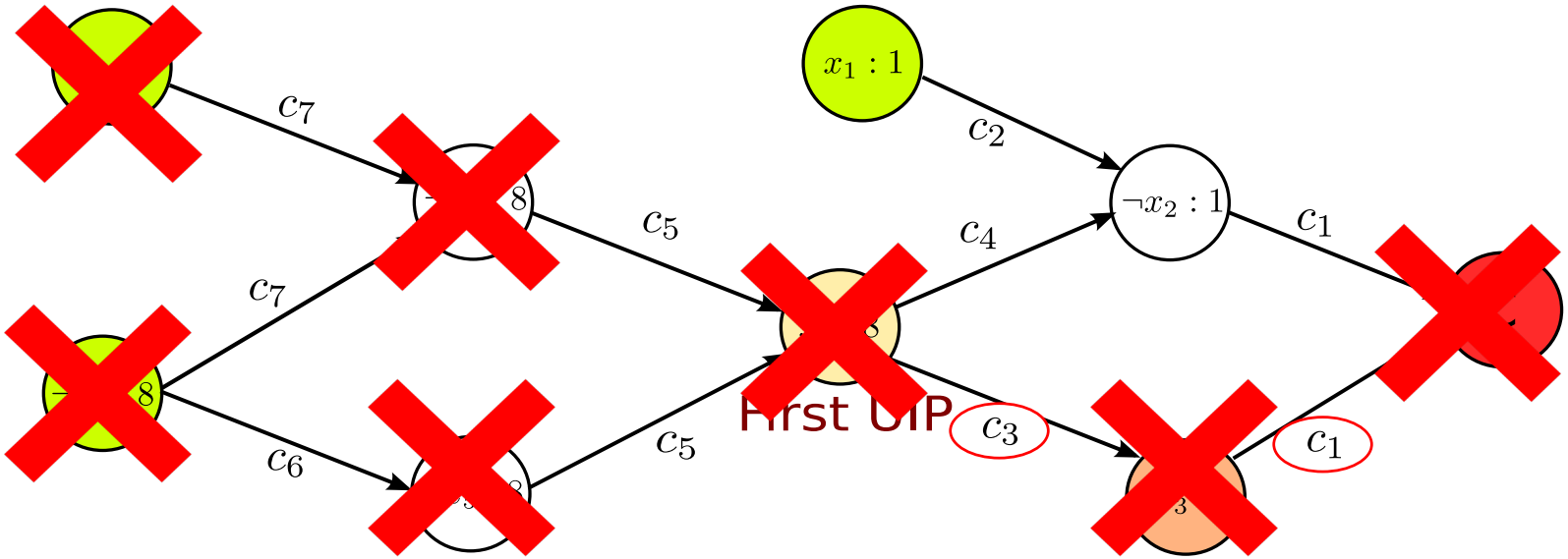
- Recall: We obtained the conflict clause $x_2 \vee \neg x_4$
- What level do we backtrack to? decision level 1
- What do we delete in the graph? everything except x_1 and $\neg x_2$

Going Back to Example



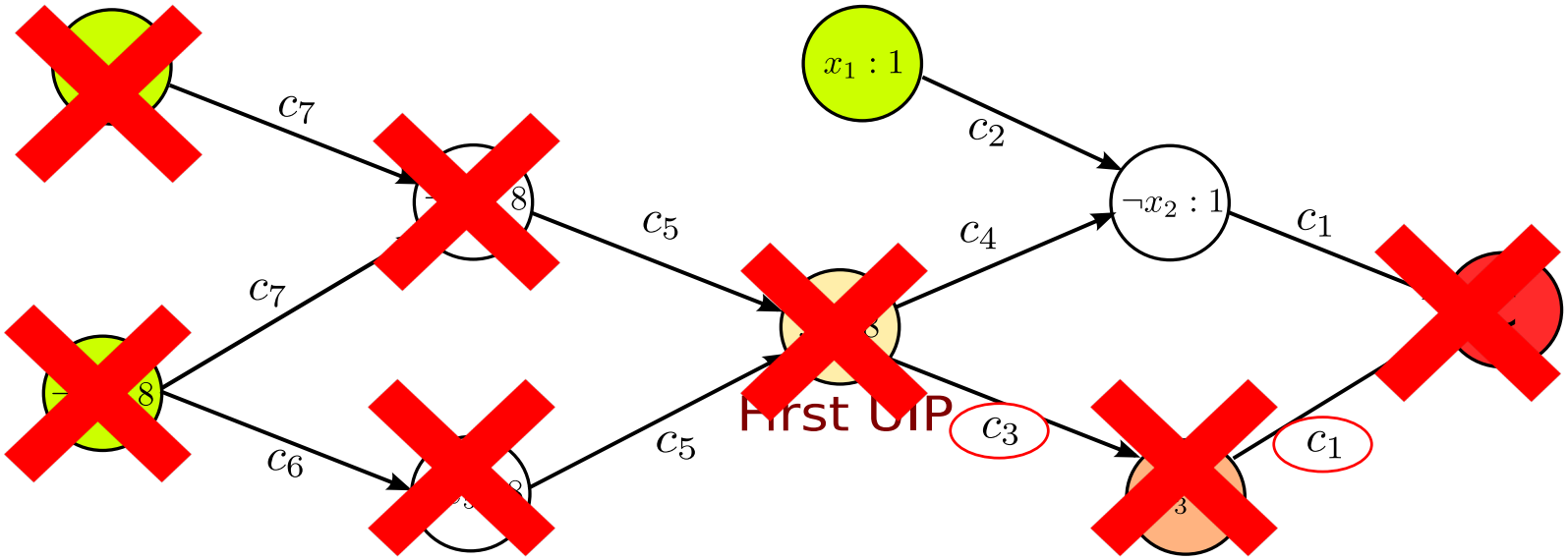
- Recall: We obtained the conflict clause $x_2 \vee \neg x_4$
- What level do we backtrack to? decision level 1
- What do we delete in the graph? everything except x_1 and $\neg x_2$
- After we add $x_2 \vee \neg x_4$ to clause database, BCP implies:

Going Back to Example



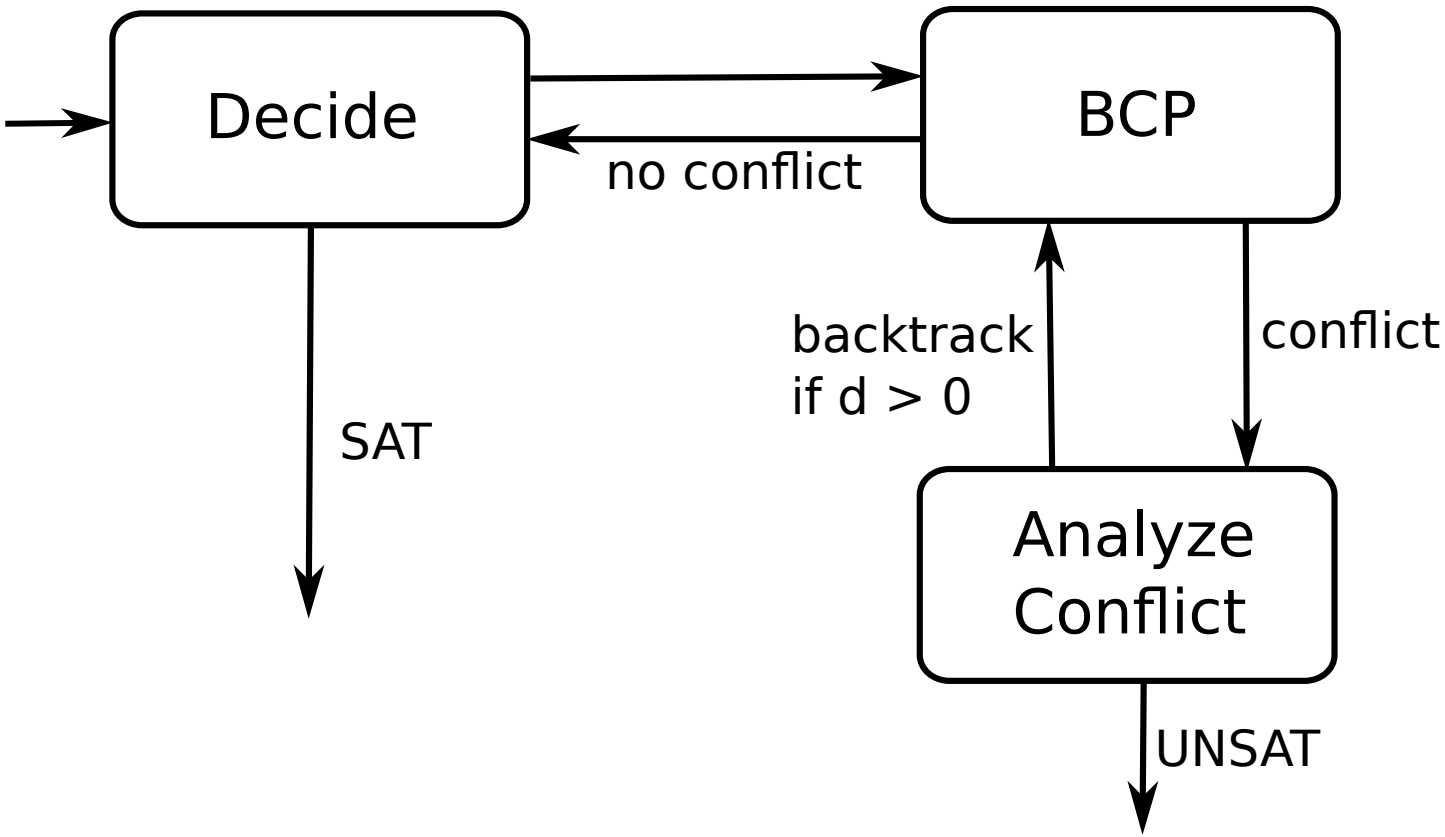
- Recall: We obtained the conflict clause $x_2 \vee \neg x_4$
- What level do we backtrack to? decision level 1
- What do we delete in the graph? everything except x_1 and $\neg x_2$
- After we add $x_2 \vee \neg x_4$ to clause database, BCP implies: $\neg x_4$

Going Back to Example

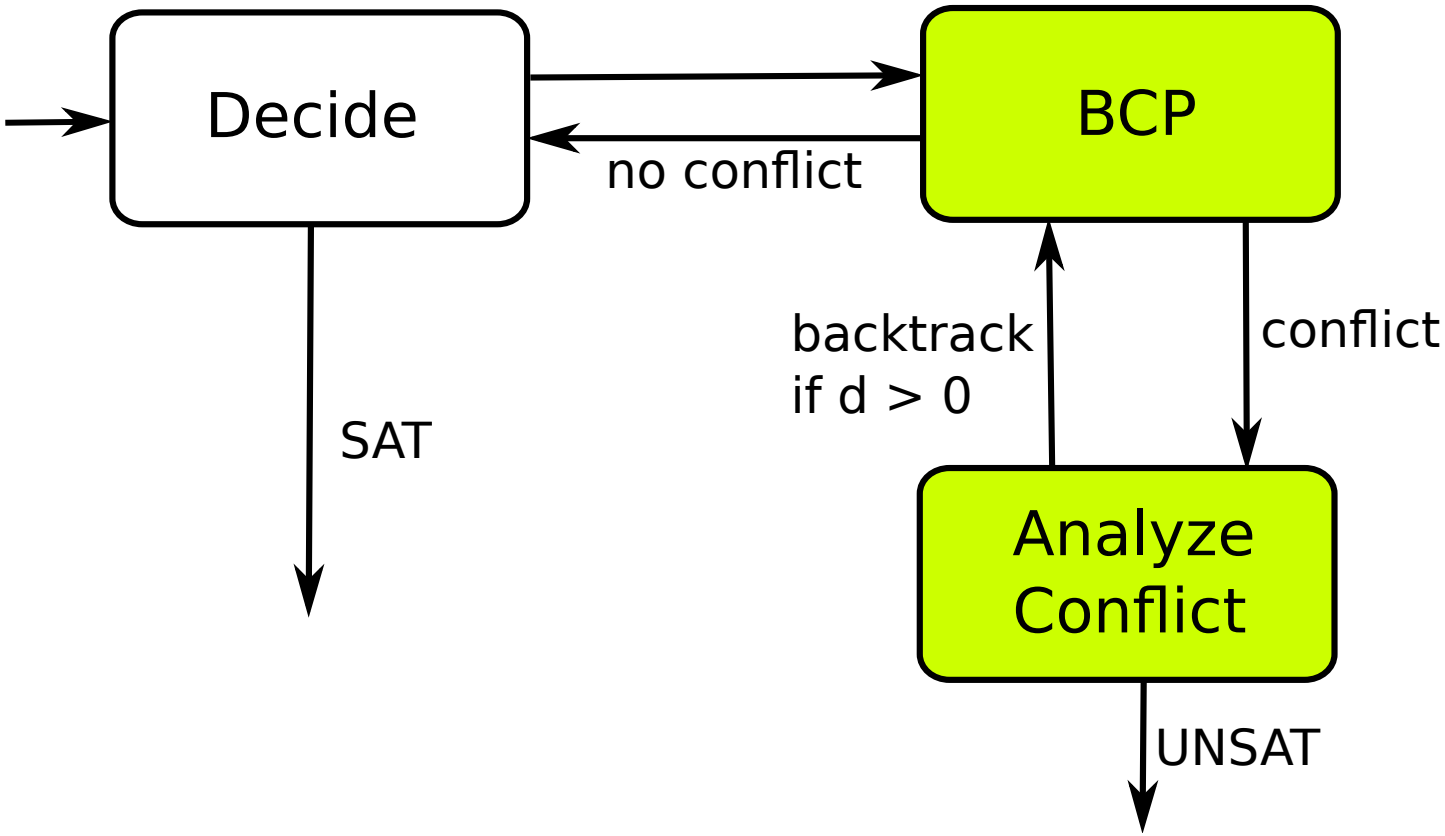


- Recall: We obtained the conflict clause $x_2 \vee \neg x_4$
- What level do we backtrack to? decision level 1
- What do we delete in the graph? everything except x_1 and $\neg x_2$
- After we add $x_2 \vee \neg x_4$ to clause database, BCP implies: $\neg x_4$
- Different assignment than before!

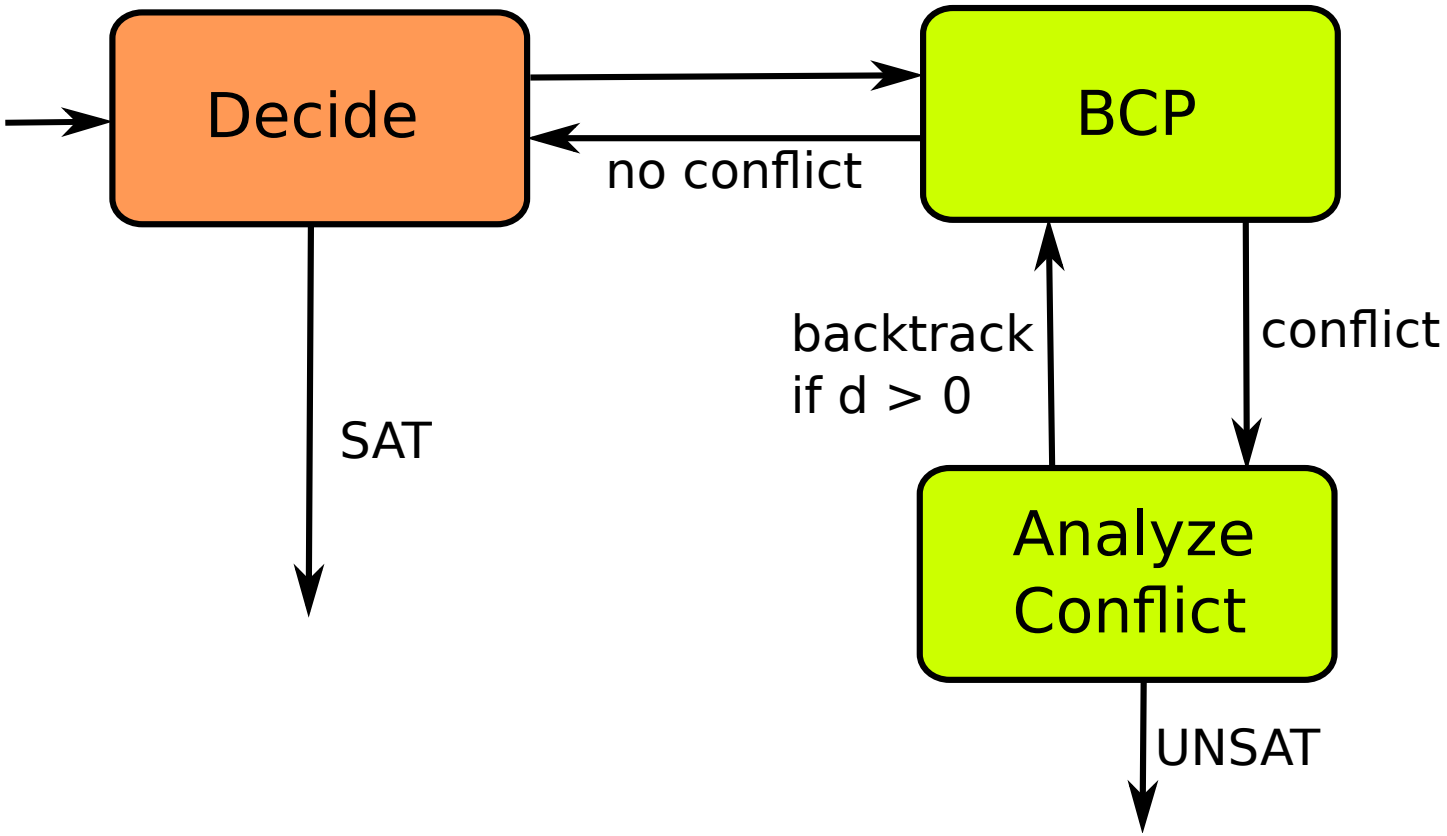
Recall: SAT Solver Architecture



Recall: SAT Solver Architecture



Recall: SAT Solver Architecture



- Decision heuristics for choosing variable order and truth assignment

Decision Heuristics

- Important part of SAT solvers, but something of a black art

Decision Heuristics

- Important part of SAT solvers, but something of a black art
- Can come up with hundreds of heuristics with varying tradeoffs

Decision Heuristics

- Important part of SAT solvers, but something of a black art
- Can come up with hundreds of heuristics with varying tradeoffs
- We'll only talk about two:

Decision Heuristics

- Important part of SAT solvers, but something of a black art
- Can come up with hundreds of heuristics with varying tradeoffs
- We'll only talk about two:
    1. dynamic largest individual sum (DLIS)

Decision Heuristics

- Important part of SAT solvers, but something of a black art
- Can come up with hundreds of heuristics with varying tradeoffs
- We'll only talk about two:
  1. dynamic largest individual sum (DLIS)
  2. variable state independent decaying sum (VSIDS)

Dynamic Largest Individual Sum (DLIS)

- This heuristic chooses the literal that satisfies the largest number of currently unsatisfied clauses.

Dynamic Largest Individual Sum (DLIS)

- This heuristic chooses the literal that satisfies the largest number of currently unsatisfied clauses.
- A clause is unsatisfied if the clause does not evaluate to true under the current partial assignment.

Dynamic Largest Individual Sum (DLIS)

- This heuristic chooses the literal that satisfies the largest number of currently unsatisfied clauses.
- A clause is unsatisfied if the clause does not evaluate to true under the current partial assignment.
- Example: $(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$

Dynamic Largest Individual Sum (DLIS)

- This heuristic chooses the literal that satisfies the largest number of currently unsatisfied clauses.
- A clause is unsatisfied if the clause does not evaluate to true under the current partial assignment.
- Example: $(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$
- What assignment would DLIS pick for this formula? (assuming no assignments so far)

Dynamic Largest Individual Sum (DLIS)

- This heuristic chooses the literal that satisfies the largest number of currently unsatisfied clauses.
- A clause is unsatisfied if the clause does not evaluate to true under the current partial assignment.
- Example: $(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$
- What assignment would DLIS pick for this formula? (assuming no assignments so far) $\neg x_2$

Dynamic Largest Individual Sum (DLIS)

- This heuristic chooses the literal that satisfies the largest number of currently unsatisfied clauses.
- A clause is unsatisfied if the clause does not evaluate to true under the current partial assignment.
- Example: $(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$
- What assignment would DLIS pick for this formula? (assuming no assignments so far) $\neg x_2$
- How is this heuristic is dynamic?

Dynamic Largest Individual Sum (DLIS)

- This heuristic chooses the literal that satisfies the largest number of currently unsatisfied clauses.
- A clause is unsatisfied if the clause does not evaluate to true under the current partial assignment.
- Example: $(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$
- What assignment would DLIS pick for this formula? (assuming no assignments so far) $\neg x_2$
- How is this heuristic is dynamic? It must be recomputed at each decision point (because unsatisfied clauses change)

Dynamic Largest Individual Sum (DLIS)

- This heuristic chooses the literal that satisfies the largest number of currently unsatisfied clauses.
- A clause is unsatisfied if the clause does not evaluate to true under the current partial assignment.
- Example: $(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$
- What assignment would DLIS pick for this formula? (assuming no assignments so far) $\neg x_2$
- How is this heuristic is dynamic? It must be recomputed at each decision point (because unsatisfied clauses change)
- Thus, overhead can be high and must be implemented carefully to minimize bookkeeping

Variable State Independent Decaying Sum (VSIDS)

- Similar to DLIS, but the goal is to reduce high overhead and favor literals that occur in a lot of conflicts (i.e. conflict-driven)

Variable State Independent Decaying Sum (VSIDS)

- Similar to DLIS, but the goal is to reduce high overhead and favor literals that occur in a lot of conflicts (i.e. conflict-driven)
- To reduce overhead, we count the total number of clauses in which the literal appears, but disregard if the clause it appears in is satisfied or not

Variable State Independent Decaying Sum (VSIDS)

- Similar to DLIS, but the goal is to reduce high overhead and favor literals that occur in a lot of conflicts (i.e. conflict-driven)
- To reduce overhead, we count the total number of clauses in which the literal appears, but disregard if the clause it appears in is satisfied or not
- Specifically, initialize the score of each literal to the number of clauses in which literal appears

Variable State Independent Decaying Sum (VSIDS)

- Similar to DLIS, but the goal is to reduce high overhead and favor literals that occur in a lot of conflicts (i.e. conflict-driven)
- To reduce overhead, we count the total number of clauses in which the literal appears, but disregard if the clause it appears in is satisfied or not
- Specifically, initialize the score of each literal to the number of clauses in which literal appears
- Every time we add a conflict clause involving literal l, increase the score of that literal by 1

Variable State Independent Decaying Sum (VSIDS)

- Similar to DLIS, but the goal is to reduce high overhead and favor literals that occur in a lot of conflicts (i.e. conflict-driven)
- To reduce overhead, we count the total number of clauses in which the literal appears, but disregard if the clause it appears in is satisfied or not
- Specifically, initialize the score of each literal to the number of clauses in which literal appears
- Every time we add a conflict clause involving literal l, increase the score of that literal by 1
- Much cheaper compared to DLIS because we don't need to scan all clauses to figure out which ones are satisfied

Variable State Independent Decaying Sum (VSIDS), cont.

- Second aspect of VSIDS: To favor literals that appear in recent conflicts, periodically divide scores of all literals by a constant
  $\Rightarrow$ decaying sum

Variable State Independent Decaying Sum (VSIDS), cont.

- Second aspect of VSIDS: To favor literals that appear in recent conflicts, periodically divide scores of all literals by a constant
  $\Rightarrow$ decaying sum

- If a literal doesn't appear in recent conflict, its score will decay over time

Variable State Independent Decaying Sum (VSIDS), cont.

- Second aspect of VSIDS: To favor literals that appear in recent conflicts, periodically divide scores of all literals by a constant
  $\Rightarrow$ decaying sum
- If a literal doesn't appear in recent conflict, its score will decay over time
- On the other hand, if literal appears in recent conflict, its score will be increased, so its score won't decay as much

Variable State Independent Decaying Sum (VSIDS), cont.

- Second aspect of VSIDS: To favor literals that appear in recent conflicts, periodically divide scores of all literals by a constant
  $\Rightarrow$ decaying sum
- If a literal doesn't appear in recent conflict, its score will decay over time
- On the other hand, if literal appears in recent conflict, its score will be increased, so its score won't decay as much
- Thus, the VSIDS heuristic favors literals that appear in recent conflicts

Variable State Independent Decaying Sum (VSIDS), cont.

- Second aspect of VSIDS: To favor literals that appear in recent conflicts, periodically divide scores of all literals by a constant
  $\Rightarrow$ decaying sum
- If a literal doesn't appear in recent conflict, its score will decay over time
- On the other hand, if literal appears in recent conflict, its score will be increased, so its score won't decay as much
- Thus, the VSIDS heuristic favors literals that appear in recent conflicts
- Introduced in the CHAFF SAT solver from Princeton, written by undergrads!

Implementation Tricks

- To build competitive SAT solvers, it is important to minimize overhead of implementing Decide, BCP, and Analyze Conflict

Implementation Tricks

- To build competitive SAT solvers, it is important to minimize overhead of implementing Decide, BCP, and Analyze Conflict
- Very important because SAT solver might be searching through hundreds of thousands of assignments!

Implementation Tricks

- To build competitive SAT solvers, it is important to minimize overhead of implementing Decide, BCP, and Analyze Conflict
- Very important because SAT solver might be searching through hundreds of thousands of assignments!
- We'll talk about two issues:
  1. number of conflict clauses
  2. trick to perform BCP fast: watch literals

Conflict Clauses

- Recall: After analyzing conflict, we add new conflict clause to our clause database

Conflict Clauses

- Recall: After analyzing conflict, we add new conflict clause to our clause database
- Pro: Conflict clauses quickly block bad assignments and prevent future mistakes

Conflict Clauses

- Recall: After analyzing conflict, we add new conflict clause to our clause database
- Pro: Conflict clauses quickly block bad assignments and prevent future mistakes
- Con: More clauses = more overhead

Conflict Clauses

- Recall: After analyzing conflict, we add new conflict clause to our clause database
- Pro: Conflict clauses quickly block bad assignments and prevent future mistakes
- Con: More clauses = more overhead

$\Rightarrow$ Tradeoff between conflict prevention and minimizing overhead

Conflict Clauses, cont.

- For this reason, many SAT solvers do not keep all the conflict clauses they derive

Conflict Clauses, cont.

- For this reason, many SAT solvers do not keep all the conflict clauses they derive
- For example, they put a limit on the number of conflict clauses they derive

Conflict Clauses, cont.

- For this reason, many SAT solvers do not keep all the conflict clauses they derive
- For example, they put a limit on the number of conflict clauses they derive
- Typically, keep most recent conflict clauses since they are most relevant to current part of search space

Implementing BCP

- Implementing BCP efficiently is very important because SAT solvers spend a lot of time doing BCP

Implementing BCP

- Implementing BCP efficiently is very important because SAT solvers spend a lot of time doing BCP
- Naive implementation of BCP: Requires scanning all currently unsatisfied clauses

Implementing BCP

- Implementing BCP efficiently is very important because SAT solvers spend a lot of time doing BCP
- Naive implementation of BCP: Requires scanning all currently unsatisfied clauses
- But industrial instances of boolean SAT problems contain hundreds of thousands of clauses

Implementing BCP

- Implementing BCP efficiently is very important because SAT solvers spend a lot of time doing BCP
- Naive implementation of BCP: Requires scanning all currently unsatisfied clauses
- But industrial instances of boolean SAT problems contain hundreds of thousands of clauses
- Thus, scanning all unsatisfied clauses too expensive!

Implementing BCP

- Implementing BCP efficiently is very important because SAT solvers spend a lot of time doing BCP
- Naive implementation of BCP: Requires scanning all currently unsatisfied clauses
- But industrial instances of boolean SAT problems contain hundreds of thousands of clauses
- Thus, scanning all unsatisfied clauses too expensive!
- A more intelligent implementation: Keep mapping from each literal to all clauses in which each literal appears (because we perform unit resolution after each variable assignment)

Implementing BCP

- Implementing BCP efficiently is very important because SAT solvers spend a lot of time doing BCP
- Naive implementation of BCP: Requires scanning all currently unsatisfied clauses
- But industrial instances of boolean SAT problems contain hundreds of thousands of clauses
- Thus, scanning all unsatisfied clauses too expensive!
- A more intelligent implementation: Keep mapping from each literal to all clauses in which each literal appears (because we perform unit resolution after each variable assignment)
- But this is still very expensive because typically each literals appears in many clauses

The Trick: Watch Literals

- Modern SAT solvers use a much more clever trick to perform BCP fast: watch literals

The Trick: Watch Literals

- Modern SAT solvers use a much more clever trick to perform BCP fast: watch literals
- Observe: Ultimate purpose of BCP is to figure out which variable assignments imply which others

The Trick: Watch Literals

- Modern SAT solvers use a much more clever trick to perform BCP fast: watch literals
- Observe: Ultimate purpose of BCP is to figure out which variable assignments imply which others
- Question: If we are performing unit resolution between l and clause $c = (\neg l \vee l_1, \ldots \vee l_k)$, under what condition will a new assignment be implied?

The Trick: Watch Literals

- Modern SAT solvers use a much more clever trick to perform BCP fast: watch literals
- Observe: Ultimate purpose of BCP is to figure out which variable assignments imply which others
- Question: If we are performing unit resolution between l and clause $c = (\neg l \vee l_1, \ldots \vee l_k)$, under what condition will a new assignment be implied?
- Answer: If clause c has only two literals left!

The Trick: Watch Literals

- Modern SAT solvers use a much more clever trick to perform BCP fast: watch literals
- Observe: Ultimate purpose of BCP is to figure out which variable assignments imply which others
- Question: If we are performing unit resolution between l and clause $c = (\neg l \vee l_1, \ldots \vee l_k)$, under what condition will a new assignment be implied?
- Answer: If clause c has only two literals left!
- Idea: Since a clause will not imply new variable assignment unless it has only two literals left, we only need to look at clauses that have at most two unassigned literals!

Watch Literals

- To efficiently detect clauses with at most two unassigned literals, select two unassigned literals in each unsatisfied clause as watch literals

Watch Literals

- To efficiently detect clauses with at most two unassigned literals, select two unassigned literals in each unsatisfied clause as watch literals
- Invariant: Either a clause has two watched unassigned literals or it is unit

Watch Literals

- To efficiently detect clauses with at most two unassigned literals, select two unassigned literals in each unsatisfied clause as watch literals
- Invariant: Either a clause has two watched unassigned literals or it is unit
- To maintain invariant: If a watch literal is assigned a truth value and clause has other unassigned literals, choose any unassigned literal in clause to be new watch literal

Watch Literals

- To efficiently detect clauses with at most two unassigned literals, select two unassigned literals in each unsatisfied clause as watch literals
- Invariant: Either a clause has two watched unassigned literals or it is unit
- To maintain invariant: If a watch literal is assigned a truth value and clause has other unassigned literals, choose any unassigned literal in clause to be new watch literal
- If a watch literal is assigned a truth value and there are no other unassigned non-watch literals left, BCP implies an assignment to the only remaining watch literal!

Watch Literals, cont.

- Question: Given this invariant, if we make assignment l, which clauses can imply new variable assignments?

Watch Literals, cont.

- Question: Given this invariant, if we make assignment l, which clauses can imply new variable assignments?
- Answer: Only those clauses in which $\neg l$ appears as watch literal

Watch Literals, cont.

- Question: Given this invariant, if we make assignment l, which clauses can imply new variable assignments?
- Answer: Only those clauses in which $\neg l$ appears as watch literal
- If $\neg l$ does not appear, we can't perform unit resolution

Watch Literals, cont.

- Question: Given this invariant, if we make assignment l, which clauses can imply new variable assignments?
- Answer: Only those clauses in which $\neg l$ appears as watch literal
- If $\neg l$ does not appear, we can't perform unit resolution
- If $\neg l$ appears but is not a watch literal, then clause has more than two unassigned literals $\Rightarrow$ won't imply new assignment!

Watch Literals, cont.

- Question: Given this invariant, if we make assignment l, which clauses can imply new variable assignments?
- Answer: Only those clauses in which $\neg l$ appears as watch literal
- If $\neg l$ does not appear, we can't perform unit resolution
- If $\neg l$ appears but is not a watch literal, then clause has more than two unassigned literals $\Rightarrow$ won't imply new assignment!
- Watch literal trick makes BCP much faster because much fewer clauses contain negation of current literal as a watch literal!

Watch Literals, cont.

- Question: Given this invariant, if we make assignment l, which clauses can imply new variable assignments?
- Answer: Only those clauses in which $\neg l$ appears as watch literal
- If $\neg l$ does not appear, we can't perform unit resolution
- If $\neg l$ appears but is not a watch literal, then clause has more than two unassigned literals $\Rightarrow$ won't imply new assignment!
- Watch literal trick makes BCP much faster because much fewer clauses contain negation of current literal as a watch literal!
- Yielded huge improvement in SAT solver performance!

Practical SAT Solving Summary

- Most competitive solvers today are based on DPLL

Practical SAT Solving Summary

- Most competitive solvers today are based on DPLL
- But they extend DPLL in three ways: non-chronological backtracking, conflict clause learning, and decision heuristics

Practical SAT Solving Summary

- Most competitive solvers today are based on DPLL
- But they extend DPLL in three ways: non-chronological backtracking, conflict clause learning, and decision heuristics
- In addition, clever implementation tricks like watch literals

Practical SAT Solving Summary

- Most competitive solvers today are based on DPLL
- But they extend DPLL in three ways: non-chronological backtracking, conflict clause learning, and decision heuristics
- In addition, clever implementation tricks like watch literals
- Some competitive DPLL-based SAT solvers: ZChaff, MiniSAT, PicoSAT ...

Practical SAT Solving Summary

- Most competitive solvers today are based on DPLL
- But they extend DPLL in three ways: non-chronological backtracking, conflict clause learning, and decision heuristics
- In addition, clever implementation tricks like watch literals
- Some competitive DPLL-based SAT solvers: ZChaff, MiniSAT, PicoSAT ...
- There are also other kinds of SAT solvers not based on DPLL, for instance, perform stochastic search (e.g., WalkSAT)

Practical SAT Solving Summary

- Most competitive solvers today are based on DPLL
- But they extend DPLL in three ways: non-chronological backtracking, conflict clause learning, and decision heuristics
- In addition, clever implementation tricks like watch literals
- Some competitive DPLL-based SAT solvers: ZChaff, MiniSAT, PicoSAT ...
- There are also other kinds of SAT solvers not based on DPLL, for instance, perform stochastic search (e.g., WalkSAT)
- Stochastic SAT solvers perform well on randomly-generated SAT instances, but not so well on industrial ones

Practical SAT Solving Summary

- Most competitive solvers today are based on DPLL
- But they extend DPLL in three ways: non-chronological backtracking, conflict clause learning, and decision heuristics
- In addition, clever implementation tricks like watch literals
- Some competitive DPLL-based SAT solvers: ZChaff, MiniSAT, PicoSAT ...
- There are also other kinds of SAT solvers not based on DPLL, for instance, perform stochastic search (e.g., WalkSAT)
- Stochastic SAT solvers perform well on randomly-generated SAT instances, but not so well on industrial ones
- DPLL-based ones are currently more popular