COL:750/7250

Foundations of Automatic Verification

Instructor: Priyanka Golia

Course Webpage

https://priyanka-golia.github.io/teaching/COL-750-COL7250/index.html

DPLL algorithm (Davis -Putnam-Logemann-Loveland 1960)

```
$DPLL(F, m = \emptyset)\{$
    1. If F is True under m then Return SAT
    2. If F is False under m then Return UNSAT
    3. If there is a unit literal $l$ under m then Return $DPLL(F, m[l \mapsto 1])$
    4. If there is a unit literal $\neg l$ under m then Return $DPLL(F, m[l \mapsto 0])$
       Choose an unassigned variable $p$, and random bit $b \in \{0,1\}$
    5. If $DPLL(F, m[p \mapsto b]) ==$ SAT then Return SAT
       Else Return $DPLL(F, m[p \mapsto 1 - b])$
$\}$
```

Backtracking at conflict

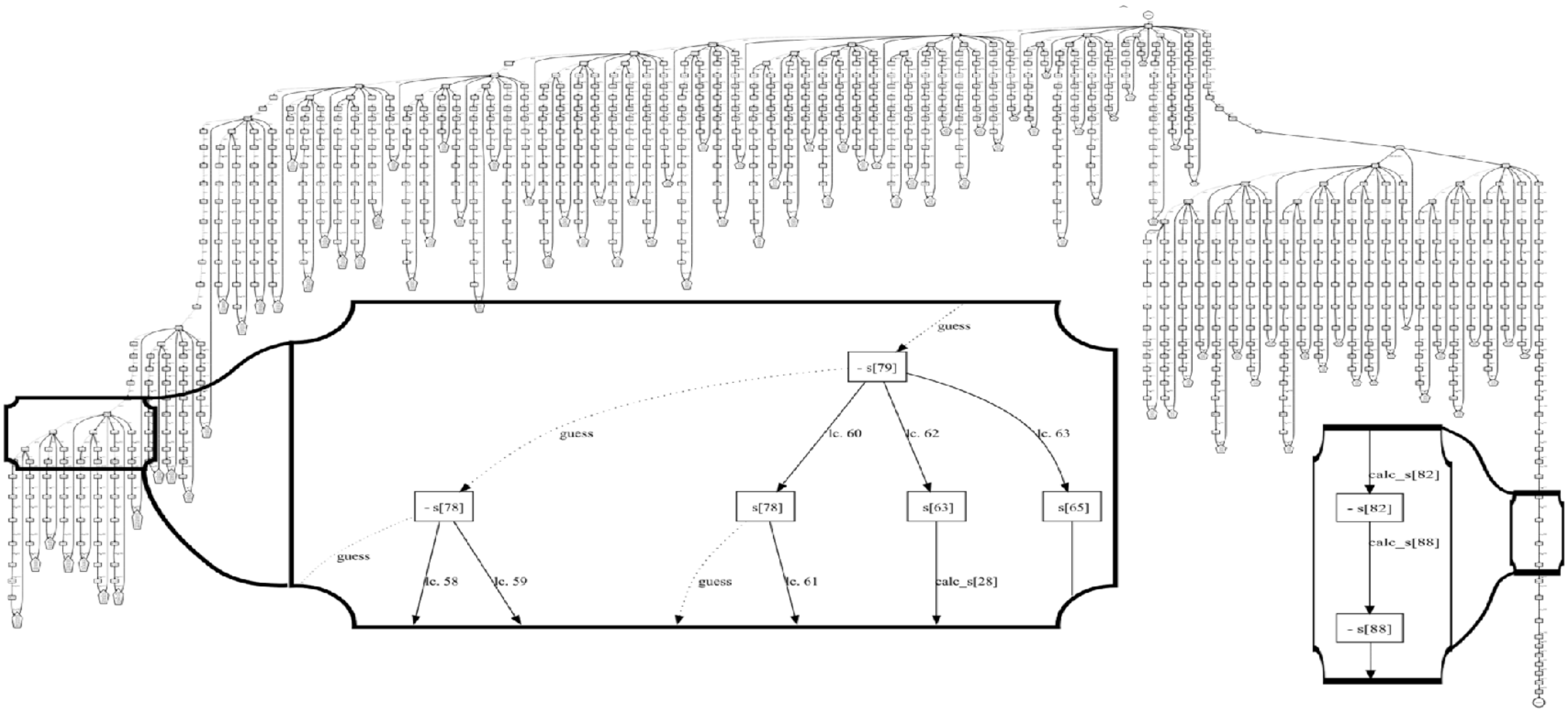
Unit Propagation

CDCL: Conflict Driven Clause Learning

1. UnitPropagation(m, F): applies unit propagation and extends m.

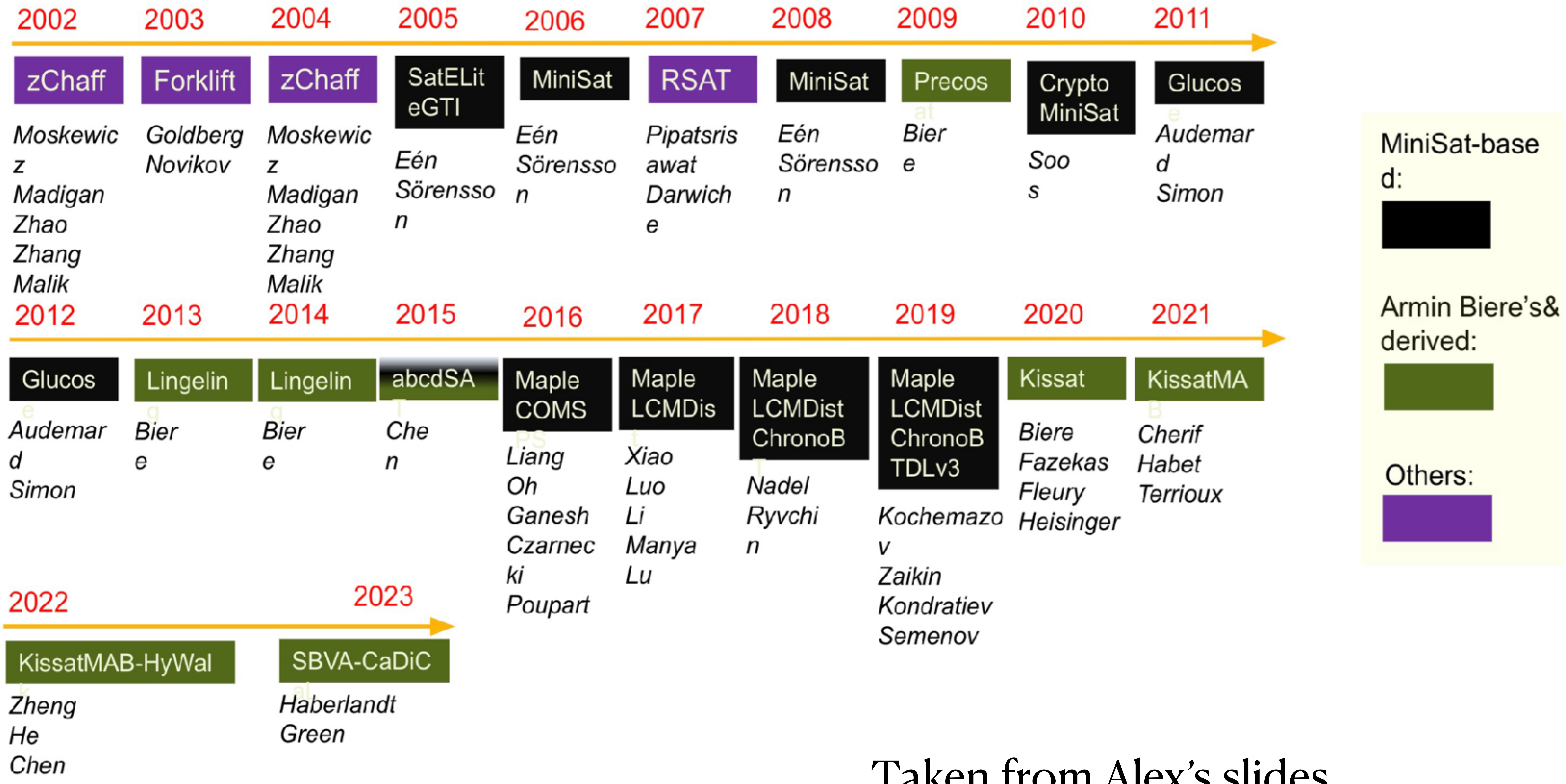
2. Decide(m, F): choose an unassigned variable in m and assign it a Boolean value.

3. AnalyzeConflict(m, F): returns a conflict clause learned using implication graph, and a decision level upto which the solver needs to backtrack.



Taken from Mate Soos 's slides.

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)



2002
zChaff
Moskewicz
Madigan
Zhao
Zhang
Malik

2003
Forklift
Goldberg
Novikov

2004
zChaff
Moskewicz
Madigan
Zhao
Zhang
Malik

2005
SatELiteGTI
Eén
Sörensson

2006
MiniSat
Eén
Sörensson

2007
RSAT
Pipatsrisawat
Darwiche

2008
MiniSat
Eén
Sörensson

2009
Precosat
Biere

2010
CryptoMiniSat
Soos

2011
Glucose
Audemard
Simon

2012
Glucose
Audemard
Simon

2013
Lingeling
Biere

2014
Lingeling
Biere

2015
abcdSAT
Chen

2016
MapleCOMSPS
Liang
Oh
Ganesh
Czarnecki
Poupart

2017
MapleLCMDist
Xiao
Luo
Li
Manya
Lu

2018
MapleLCMDistChronoBT
Nadel
Ryvchin

2019
MapleLCMDistChronoBTDLv3
Kochemazov
Zaikin
Kondratiev
Semenov

2020
Kissat
Biere
Fazekas
Fleury
Heisinger

2021
KissatMAB
Cherif
Habet
Terrioux

2022
KissatMAB-HyWalk
Zheng
He
Chen

2023
SBVA-CaDiCaL
Haberlandt
Green

MiniSat-based: ⬛

Armin Biere's& derived: 🟩

Others: 🟪

Taken from Alex's slides.

CDCL: Conflict Driven Clause Learning

1. UnitPropagation(m, F): applies unit propagation and extends m.

2. Decide(m, F): choose an unassigned variable in m and assign it a Boolean value.

Heuristics: which variables to pick, what value to assign?

3. ClauseLearning(m, F): returns a conflict clause learned using implication graph, and a decision level upto which the solver needs to backtrack.

Heuristics: how to learn a small conflict clause and unto which level to backtrack?

Heuristics: how to learn a small conflict clause and upto which level to backtrack?

AnalyzeConflict(m,F): some choices of clauses are found to be better than others.

Notations:

UIP (Unique Implication Point)

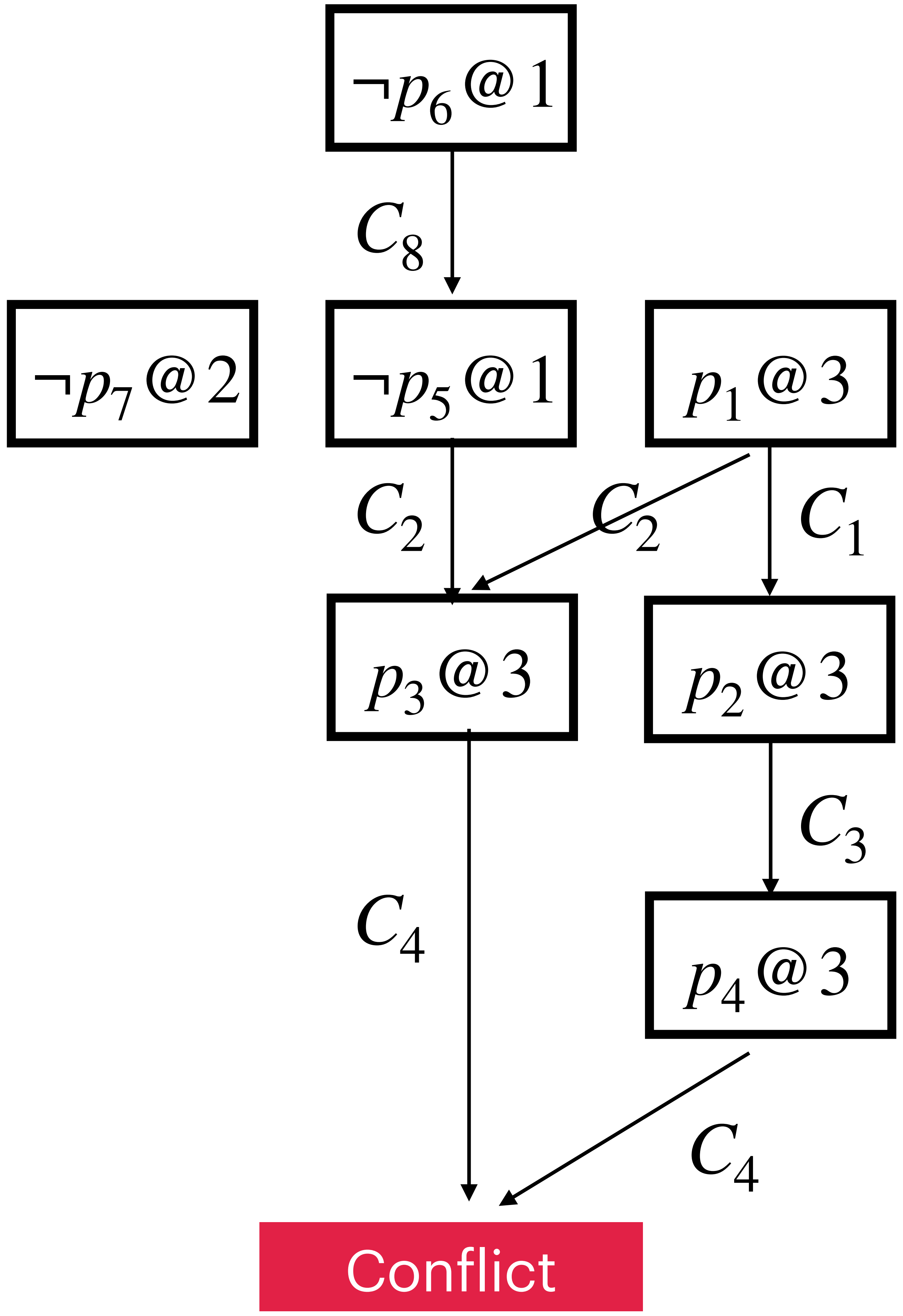
In an implication graph, node "$l@d$" is a UIP at decision level d if "$l@d$" occurs in each path from d^{th} decision literals to the conflict.

UIP points: In an implication graph, node "$l@d$" is a UIP at decision level d if "$l@d$" occurs in each path from d^{th} decision literals to the conflict.

UIP @ level 1:

UIP @ level 2:

UIP @ level 3:



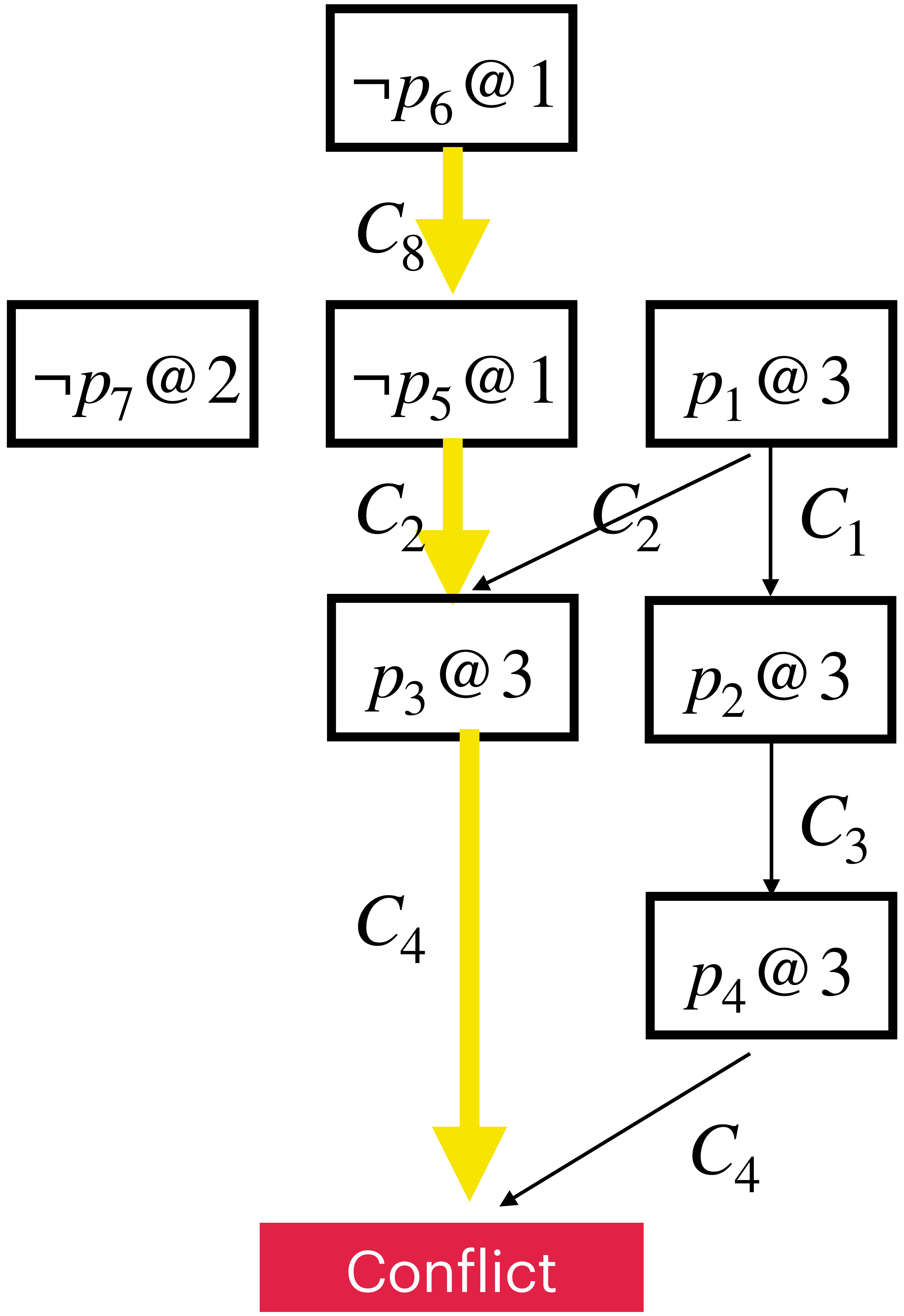
Implication Graph.

UIP points: In an implication graph, node “$l@d$” is a UIP at decision level d if “$l@d$” occurs in each path from d^{th} decision literals to the conflict.

UIP @ level 1: $\neg p_6@1, \neg p_5@1$

UIP @ level 2:

UIP @ level 3:



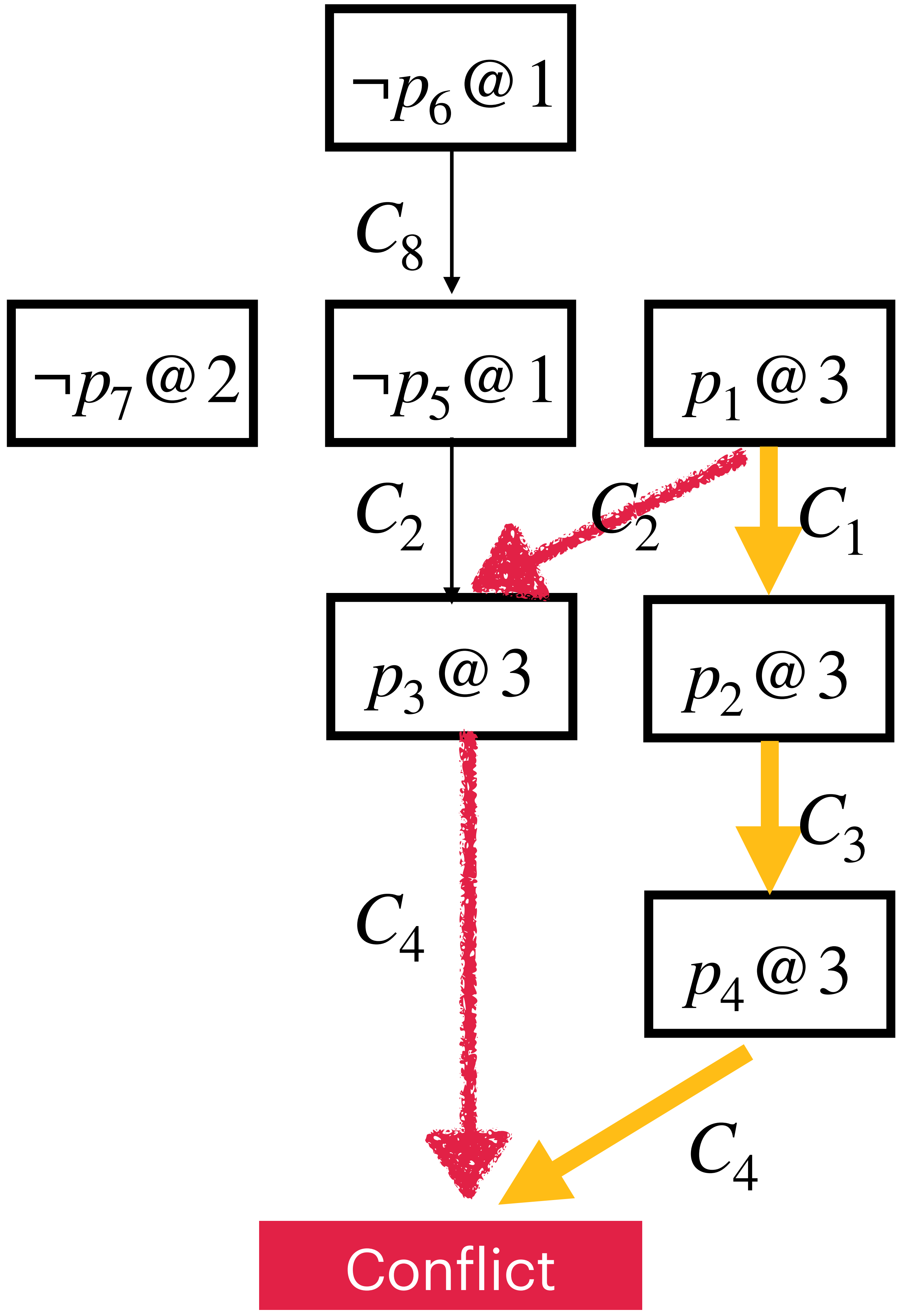
Implication Graph.

UIP points: In an implication graph, node “$l@d$” is a UIP at decision level d if “$l@d$” occurs in each path from d^{th} decision literals to the conflict.

UIP @ level 1: $\neg p_6 @ 1, \neg p_5 @ 1$

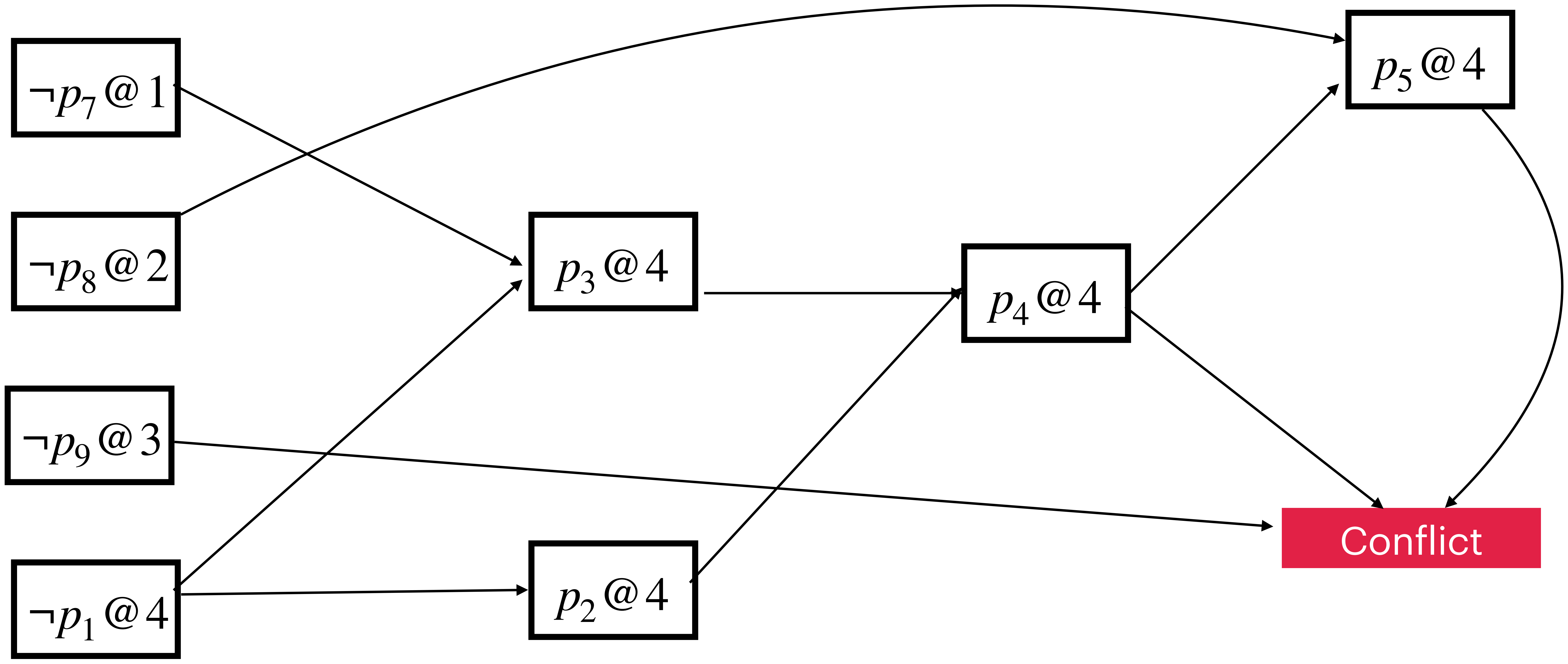
UIP @ level 2:

UIP @ level 3: $p_1 @ 3$



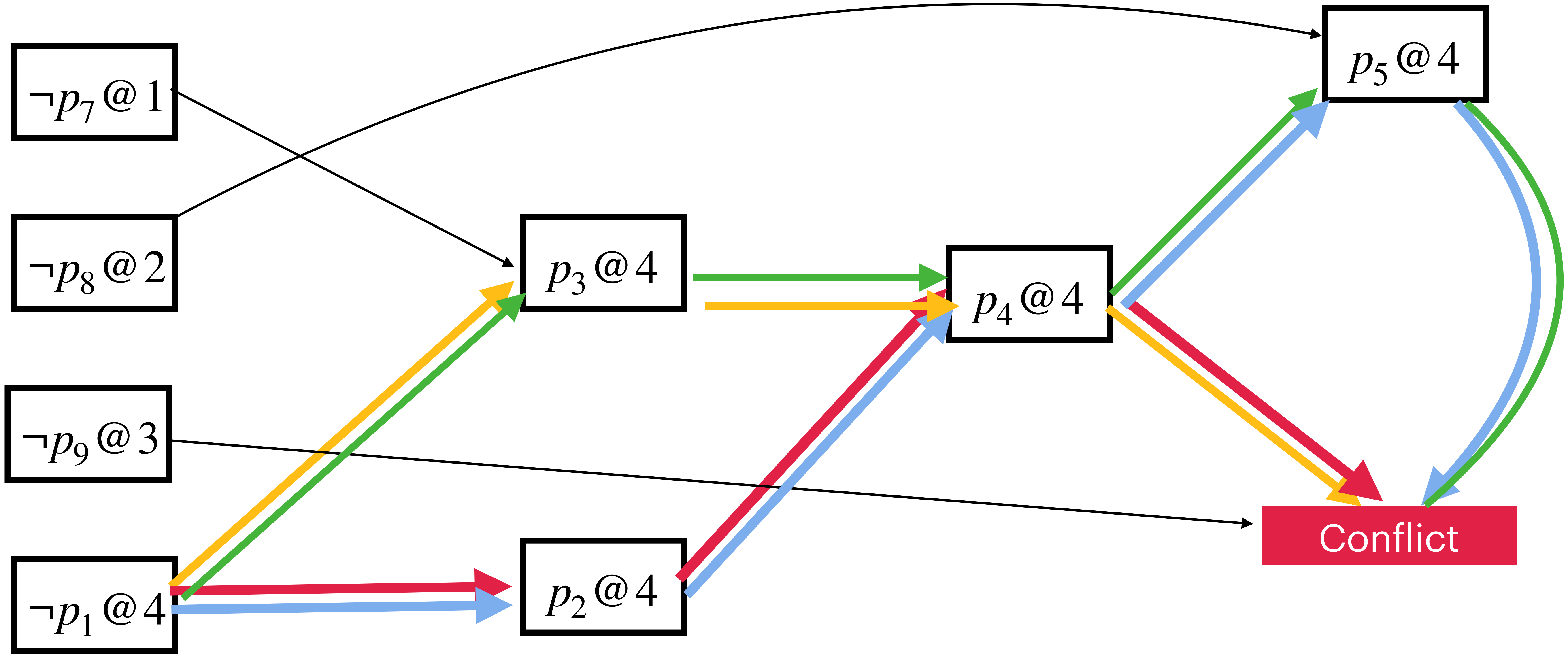
Implication Graph.

UIP points: In an implication graph, node "$l@d$" is a UIP at decision level d if "$l@d$" occurs in each path from d^{th} decision literals to the conflict.



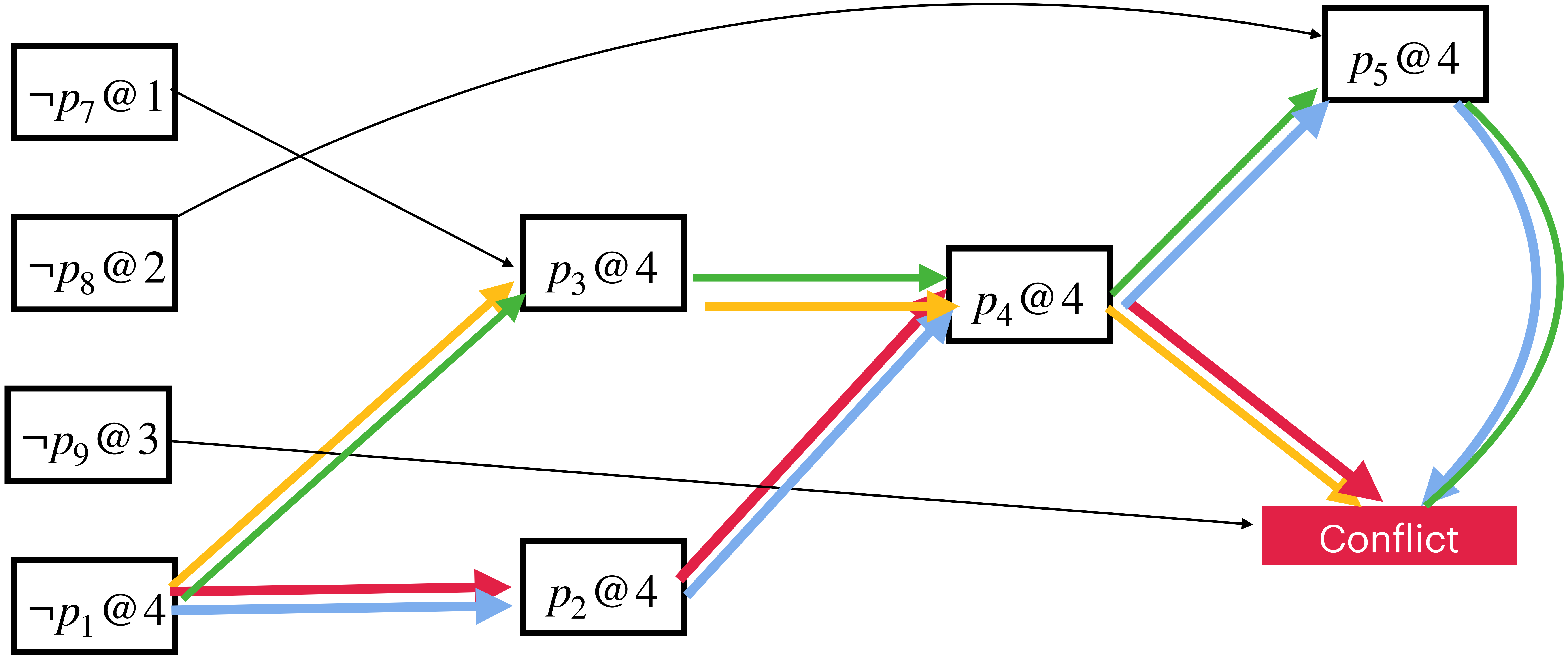
UIP $@4 = ???$

UIP points: In an implication graph, node "$l@d$" is a UIP at decision level d if "$l@d$" occurs in each path from d^{th} decision literals to the conflict.



UIP $@4 = ???$

UIP points: In an implication graph, node "$l@d$" is a UIP at decision level d if "$l@d$" occurs in each path from d^{th} decision literals to the conflict.



$$\text{UIP } @4 = \neg p_1@4, p_4@4$$

First UIP Point:
$p_4@4$

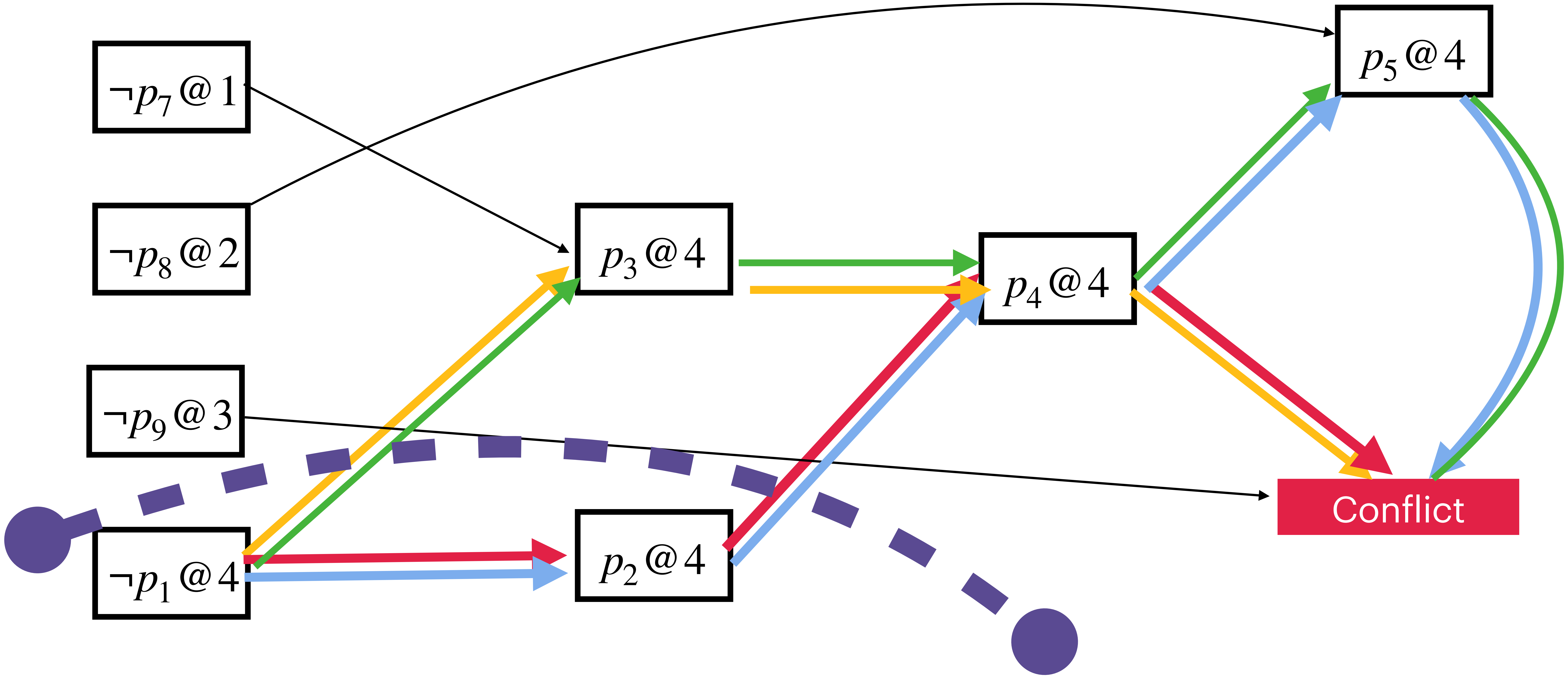
Last UIP Point:
$\neg p_1@4$

UIP cuts to analyze conflicts:

If l is UIP, then corresponding UIP cut is (A,B) of the implication graph.
Where,

B contains all the successors of l from which there is a path to conflict.
A contains the rest.

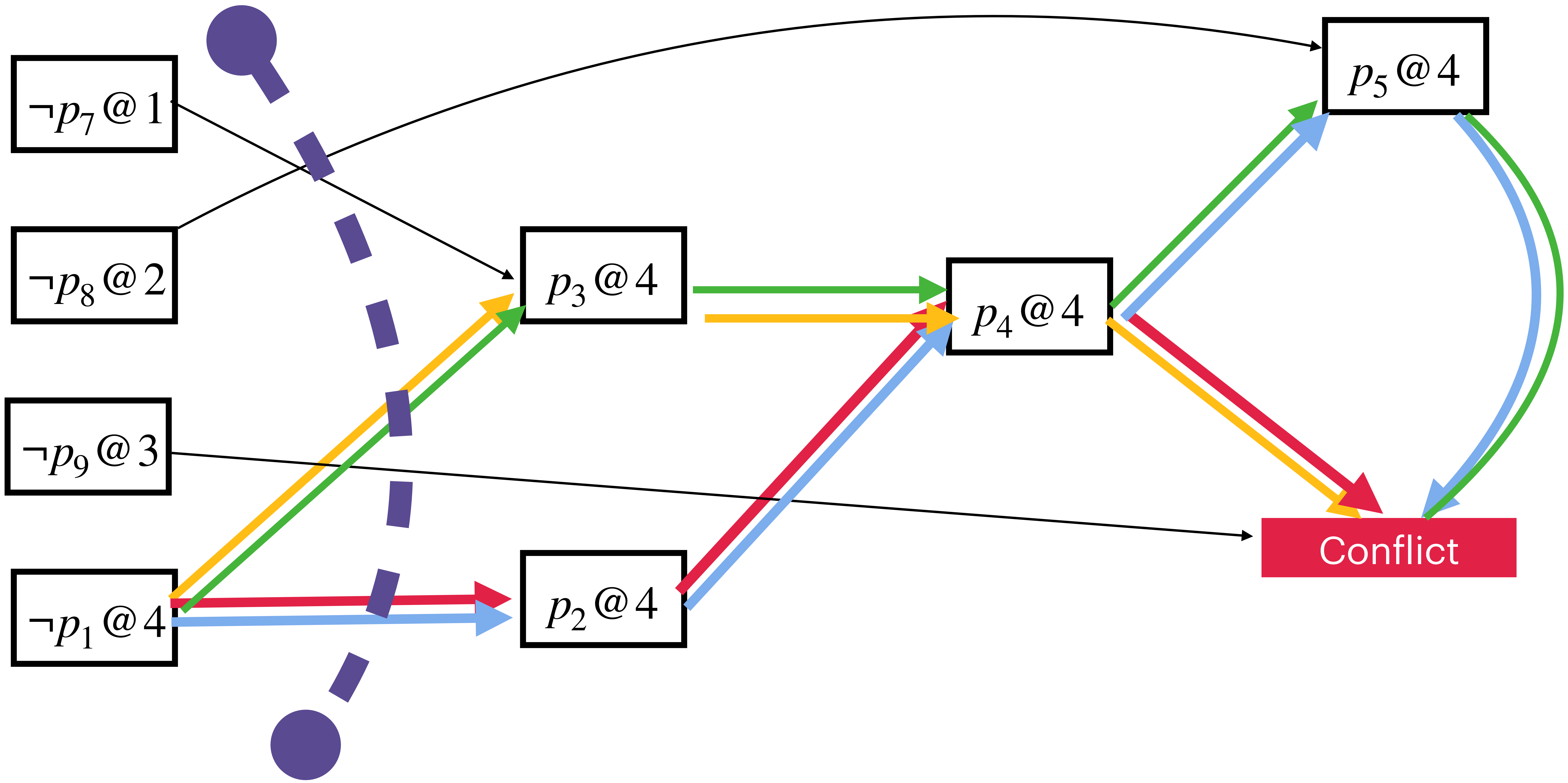
UIP cuts to analyze conflicts: If *l* is UIP, then corresponding UIP cut is (A,B) of the implication graph, where B contains all the successors of *l* from which there is a path to conflict, and A contains the rest.



UIP $@4 = \neg p_1@4, p_4@4$

Is it a UIP cut?

UIP cuts to analyze conflicts: If l is UIP, then corresponding UIP cut is (A,B) of the implication graph, where B contains all the successors of l from which there is a path to conflict, and A contains the rest.

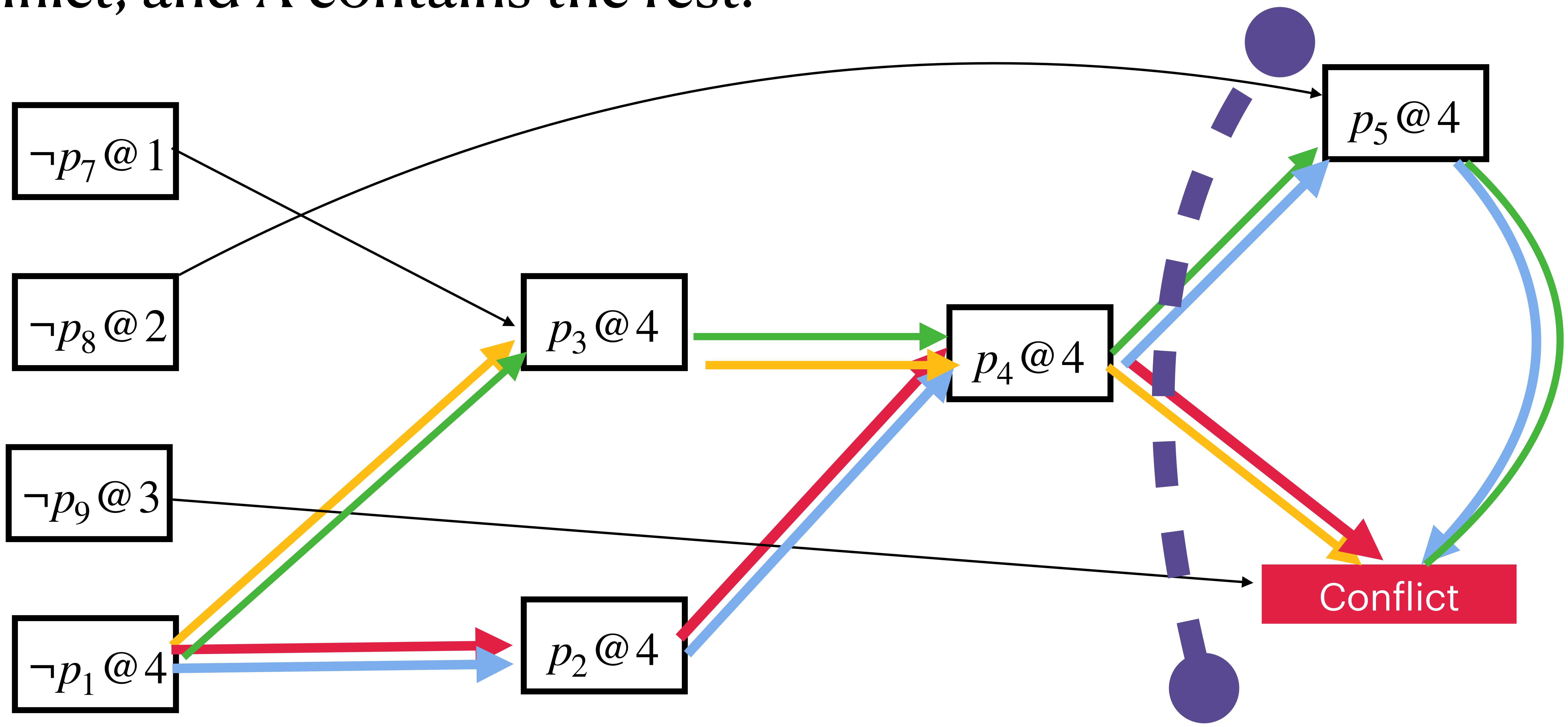


UIP $@4 = \neg p_1 @ 4, p_4 @ 4$

Is it a UIP cut?

Yes, with respect to $\neg p_1 @ 4$

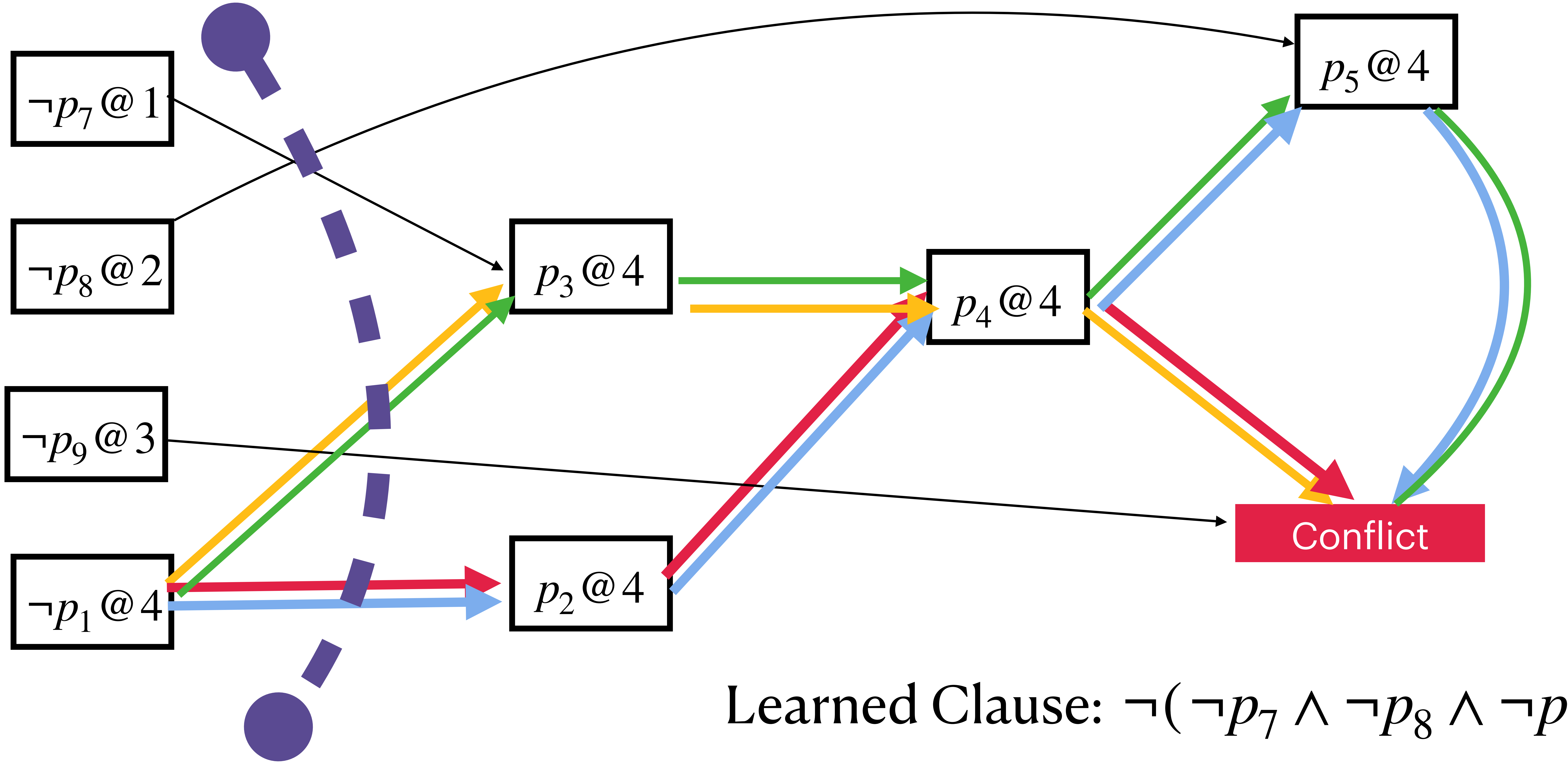
UIP cuts to analyze conflicts: If l is UIP, then corresponding UIP cut is (A,B) of the implication graph, where B contains all the successors of l from which there is a path to conflict, and A contains the rest.



UIP $@4 = \neg p_1 @ 4, p_4 @ 4$ Is it a UIP cut? Yes, with respect to $p_4 @ 4$

Learned Conflict Clause from UIP cut

The literals on the A side of the cut, which have an edge directed from A to B, form a clause. These literals are then negated and combined into a disjunction.

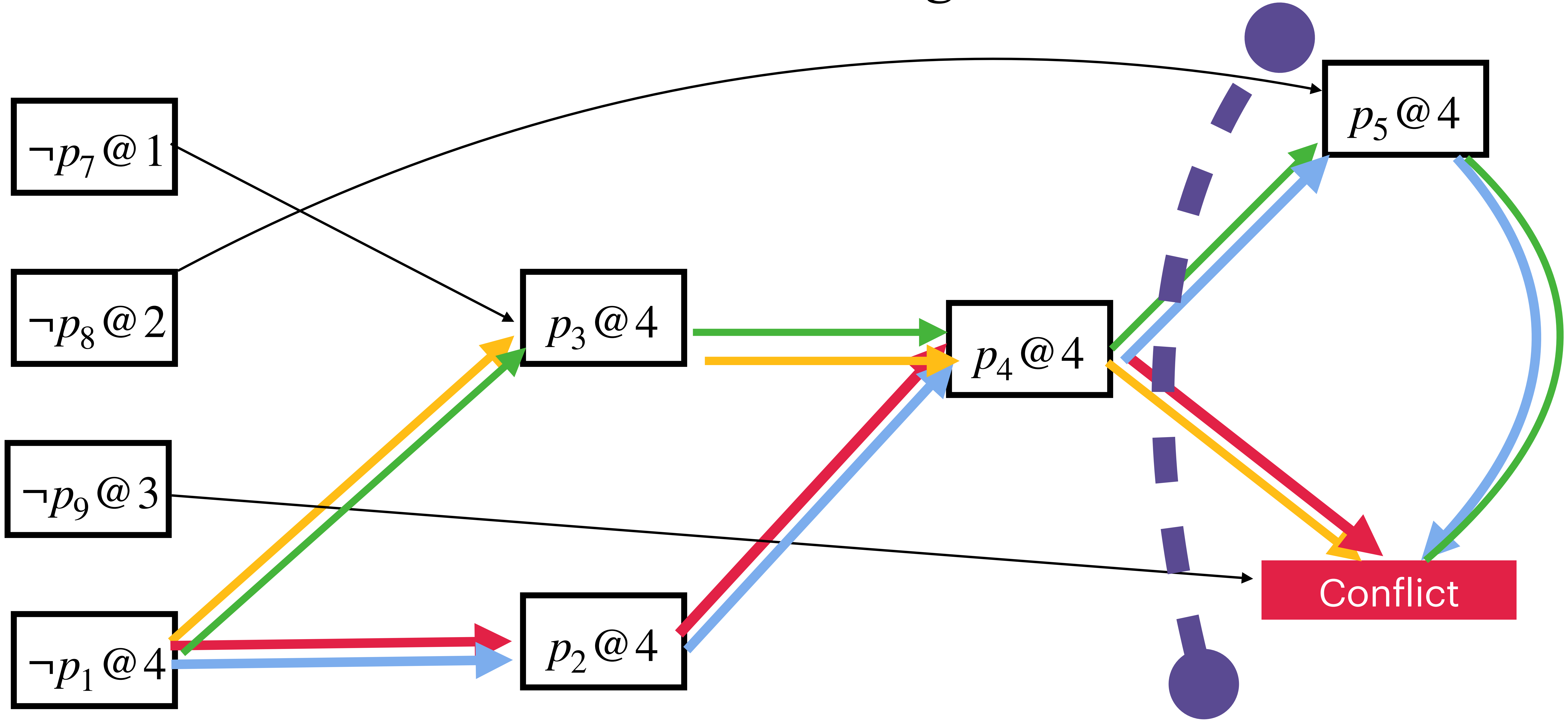


Learned Clause: $\neg(\neg p_7 \land \neg p_8 \land \neg p_9 \land \neg p_1)$

UIP $@4 = \neg p_1 @ 4, p_4 @ 4$

Learned Conflict Clause from UIP cut

The literals on the A side of the cut, which have an edge directed from A to B, form a clause. These literals are then negated and combined into a disjunction.



UIP $@4 = \neg p_1 @4, p_4 @4$

Learned Clause: $\neg(\neg p_8 \wedge p_4 \wedge \neg p_9)$

Heuristics: which variables to pick, what value to assign?

Variable ordering, Decision heuristics, Branching heuristics.

- # of variables occurrence in remaining unsatisfied clauses (different variants were studied in 90s).
- Dynamic heuristics:
    - Focus on variables which were useful recently in deriving learned clauses.
    - Can be interpreted as reinforcement learning.
    - VSIDS: Variable State Independent Decaying Sum.
- Look-ahead
    - Spent more time in selecting good variables.

VSIDS: Variable State Independent Decaying Sum

- Each literal l has a counter $S(l)$, initialized to zero.
- For every new clause $C = \{l_1, l_2, \ldots, l_n\}$, $S(l_i)$ is incremented if $l_i \in C$.
- The unassigned variable and polarity with highest counter is chosen.
- Ties are broken randomly.
- Periodically (once in 256 conflict), all counters are halved.

VSIDS: Variable State Independent Decaying Sum

Literals	Score
a	4
$\neg a$	5
b	3
$\neg b$	3
c	2
$\neg c$	3
d	2
$\neg d$	4
e	2
$\neg e$	6

……

Initial value occurrences of “a” in formula F

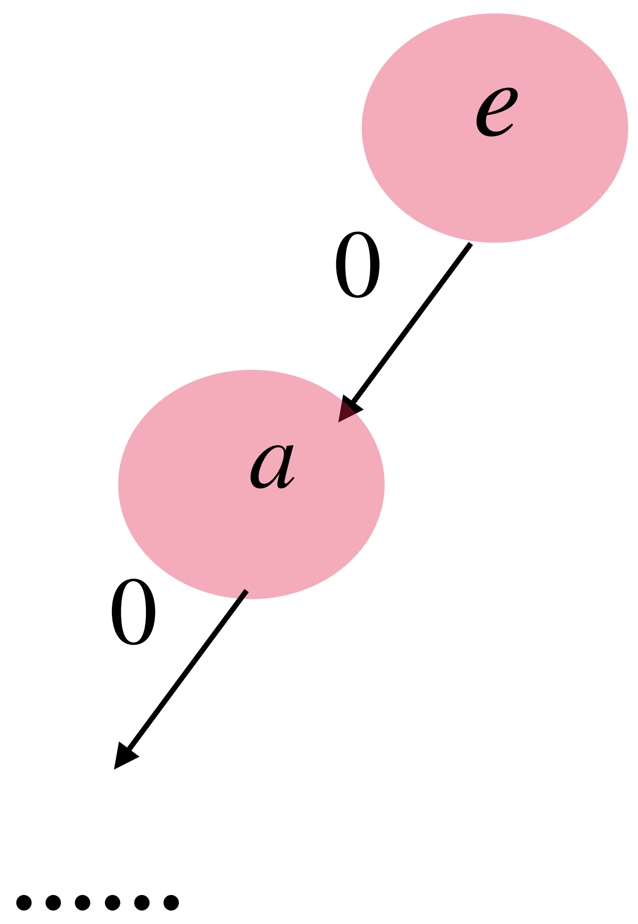
Count literal appearances in formula F

VSIDS: Variable State Independent Decaying Sum

Literals	Score
a	4
$\neg a$	5
b	3
$\neg b$	3
c	2
$\neg c$	3
d	2
$\neg d$	4
e	2
$\neg e$	6

……

Initial value occurrences of "a" in formula F



Conflict Learned clause $(\neg h \vee a \vee c \vee \neg b \vee k)$

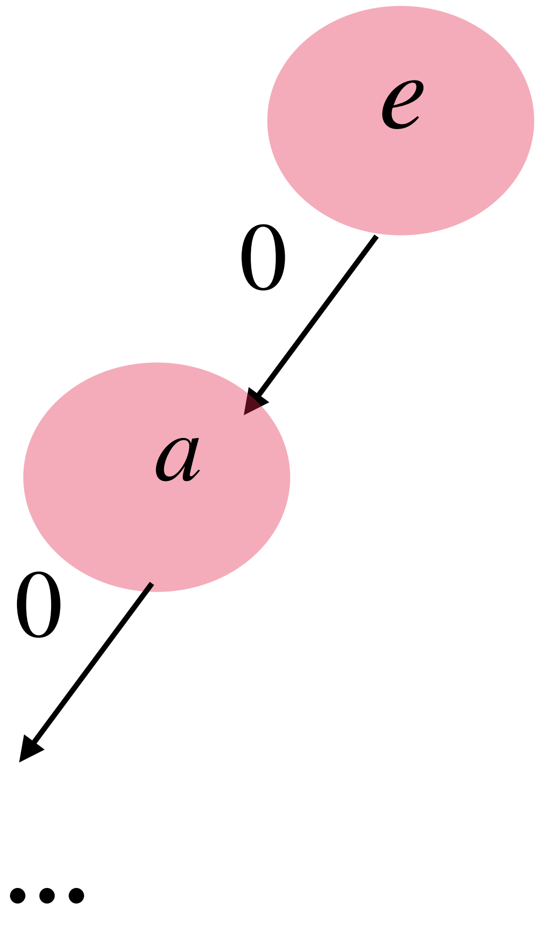
Count literal appearances in formula F

VSIDS: Variable State Independent Decaying Sum

Literals	Score
a	4 +1
$\neg a$	5
b	3+1
$\neg b$	3
c	2+1
$\neg c$	3
d	2
$\neg d$	4
e	2
$\neg e$	6

……

Initial value occurrences of “a” in formula F



Conflict

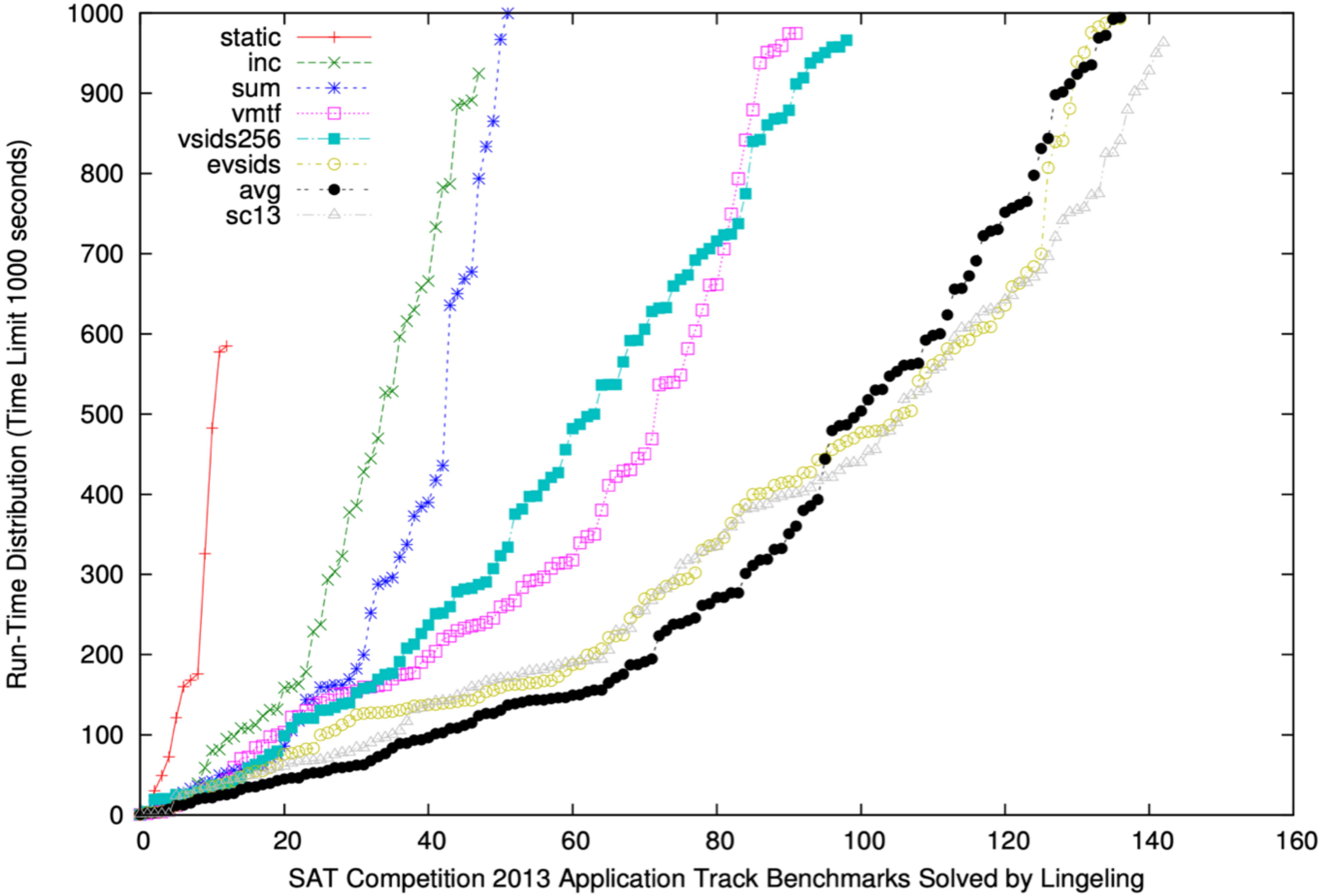
Learned clause $(\neg h \vee a \vee c \vee \neg b \vee k)$

Count literal appearances in formula F

VSIDS: Variable State Independent Decaying Sum

Why it was a breakthrough?

- Pre-chaff static heuristics — go over all clauses that are not satisfied and compute some function $f(a)$ for each literal “a”.
- VSLDS
    - Extremely low overhead.
    - Dynamic & local (conflict driven).
    - Focuses the search to learn from the local context.



static
inc
sum
vmtf
vsids256
evsids
avg
sc13
Run-Time Distribution (Time Limit 1000 seconds)
SAT Competition 2013 Application Track Benchmarks Solved by Lingeling
0
100
200
300
400
500
600
700
800
900
1000
0
20
40
60
80
100
120
140
160

Imagine a smart home with multiple devices (lights, fans, thermostats) spread across different rooms (kitchen, bedroom, living room). A control system needs to ensure certain rules are satisfied, such as:

1. All lights should be off when no one is in the room.
2. If the temperature is above 30°C, the fan should turn on.

Assume: m many person, n many lights.

Imagine a smart home with multiple devices (lights, fans, thermostats) spread across different rooms (kitchen, bedroom, living room). A control system needs to ensure certain rules are satisfied, such as:

1. All lights should be off when no one is in the room.
2. If the temperature is above 30°C, the fan should turn on.

$P = \{p_1, \ldots, p_m\}, L = \{L_1, \ldots, L_n\}$

Assume: m many person, n many lights.

Let p_i represents that i^{th} person is in the room, and L_j represents that j^{th} light is on.

Clauses n many, each clause has m+1 variables.

$$\neg(p_1 \vee p_2 \vee \ldots \vee p_m) \rightarrow (\neg L_1 \wedge \neg L_2 \wedge \ldots \wedge \neg L_n)$$

$$\equiv ((p_1 \vee p_2 \vee \ldots \vee p_m) \vee \neg L_1) \wedge ((p_1 \vee p_2 \vee \ldots \vee p_m) \vee \neg L_2) \wedge \ldots \wedge ((p_1 \vee p_2 \vee \ldots \vee p_m) \vee \neg L_n)$$

$P = \{p_1, \ldots, p_m\}, L = \{L_1, \ldots, L_n\}$

Assume: m many person, n many lights.

Let p_i represents that i^{th} person is in the room, and L_j represents that j^{th} light is on.

$$\neg(p_1 \vee p_2 \vee \ldots \vee p_m) \rightarrow (\neg L_1 \wedge \neg L_2 \wedge \ldots \wedge \neg L_n)$$

Clauses n many, each clause has m+1 variables.

$$\equiv ((p_1 \vee p_2 \vee \ldots \vee p_m) \vee \neg L_1) \wedge ((p_1 \vee p_2 \vee \ldots \vee p_m) \vee \neg L_2) \wedge \ldots \wedge ((p_1 \vee p_2 \vee \ldots \vee p_m) \vee \neg L_n)$$

Repetition: writing separate formulas for each room.
As the number of rooms increases, the formula grows linearly.

No generalization: We cannot express the general rule "For any room, if no one is present, the light should be off" without enumerating each case.

First Order Logic (FOL)

FOL is a logical system for reasoning about properties of objects.

Predicates — describes properties of objects.

Functions — maps objects to one another.

Quantifiers — to reason about multiple objects

First Order Logic (FOL): Objects

"John is happy" as P
“Mary is happy” as Q

Propositional variables don’t provide any structure about what the proposition refers to or relationships between entities — how P and Q are related ?

Objects: It represent entities in a domain of discourse (things we want to reason about), such as people, numbers, or physical objects.

Objects are: John, and Marry.
Happy(John) — property “happy” is applied to John.
Happy(Mary) — property “happy” is applied to Mary.
Likes(Mary,John): "Mary likes John."

Objects allow FOL to express relationships, properties, and reasoning about entities.

First Order Logic (FOL): Predicates

$Likes(You, Yogurt) \wedge Likes(You, Mango) \rightarrow Likes(You, MangoLassi)$.

Objects: { You, Yogurt, Mango, MangoLassi}.

Predicates: $Likes(Obj_1, Obj_2) \mapsto \{0,1\}$

Predicates takes objects as an arguments and evaluate to True or False.

Predicates — describes properties of objects.
Happy(John)
Cute(John)

First Order Logic (FOL): Functions

$\text{FavoriteMovieOf}(\text{You}) \neq \text{FavoriteMovieOf}(\text{Date}) \wedge$
$\text{StarOf}(\text{FavoriteMovieOf}(\text{You})) = \text{StarOf}(\text{FavoriteMovieOf}(\text{Date}))$

Functions take objects as an argument and return objects associated with it.

Medianof(x,y,z), +(x,y), Wife(John).

As with predicates, functions can take in any number of arguments, but always return a single object.

	Operate On	**And Produce**
Connectives $(\leftrightarrow , \rightarrow , \wedge , ..)$	Propositions	A Proposition
Predicates	Objects	A Proposition
Functions	Objects	An Object

First Order Logic (FOL): Quantifiers

There is a number which is both prime and even.

Variables: x.
Predicates: Even(x), Prime(x)

$$\exists x(Even(x) \wedge Prime(x))$$

There is someone who is taller than I am and weighs more than I do.

Objects: me, Variable: x
Predicates: Taller(x,me), WeighsMore(x,me)

$$\exists x Taller(x, me) \wedge WeighMore(x, me)$$

Existential Quantifier (∃): Expresses the existence of at least one element for which a statement is true.

First Order Logic (FOL): Quantifiers

For every number x, adding 0 to results in x itself.

Variable: x

Function: $+(x,0)$

Predicate: $=(x, +(x,0))$

$\forall x = (x, +(x,0))$

For all even numbers x, x is divisible by 2.

Variable: x

Function: $mod(x,2)$

Predicate: $even(x), = (mod(x,2),0)$

$\forall x \ \ (even(x) \rightarrow = (mod(x,2),0))$

Universal Quantifier (∀): Expresses generalization across all elements.

First Order Logic (FOL): Quantifiers

Scope of Quantifiers: refers to the part of the formula where the quantifier applies to the variable it introduces.

Bound Variable: A variable is bound if it lies within the scope of a quantifier.

Free Variable: A variable is free if it is not within the scope of any quantifier.

$\forall x\ P(x) \rightarrow Q(y)$. x is bounded and y is free

Nested Quantifiers: When quantifiers are nested, the scope of the inner quantifier is restricted by the outer quantifier.

$\forall x((\exists y P(x, y)) \rightarrow Q(x))$

Scope of $\forall x$ is entire formula.

Scope of $\exists y$ is limited to $P(x, y)$

First Order Logic (FOL): Quantifiers

When multiple quantifiers share overlapping scopes, their interactions can lead to significant differences in meaning.

$\forall x \exists y P(x, y)$	$\exists y \forall x P(x, y)$
For every x, there exists a y such that P(x, y).	There exists a y, for all x such that P(x, y).
Each person can know a different language, as long as they know at least one language.	There is a single language that everyone knows.

First Order Logic (FOL): Syntax

Well-Formed Formula (wff) of FOL are composed of six types of symbols (not including Parenthesis).

1. Constant symbols — representing objects.
2. Functions symbols — functions from pre-specified number of objects to an object.
3. Predicate symbols — more like specify properties to objects. Have specified arity.
Zero arity predicate symbols are treated as propositional symbols.
4. Variable symbols — will be used to quantify over objects.
5. Universal and existential quantifiers — will be used to indicate the type of quantification.
6. Logical connectives and negation.

First Order Logic (FOL): Syntax

Formula -> Atomic Formula
| Formula Connective Formula
| Quantifier Variable Formula
| ¬ Formula
| (Formula)

Connective -> $\leftrightarrow \mid \wedge \mid \vee \mid \rightarrow$
Quantifier -> $\forall \mid \exists$

Atomic Formula -> $P(T_1, \ldots, T_n)$ where
$P \in Predicates$, T_i are Terms, n is arity.

Term -> c, where $c \in$ CONST.
| v, where $v \in VAR$
| $F(T_1, \ldots, T_n)$, where $F \in$ Functions, T_i are Terms,
n is arity of F.

First Order Logic (FOL): Syntax

Is it a WFF?

$TallerThan(John, Fatherof(John)) \wedge TallerThan(Fatherof(Fatherof(John)), John)$.

Yes, notice, Term is recursive.

Term -> c, where $c \in$ CONST.

| v, where $v \in VAR$

| $F(T_1, \ldots, T_n)$, where $F \in$ Functions, T_i are Terms,

n is arity of F.