# Satisfiability as Abstract Interpretation

Leopold Haller
*(joint work with* Vijay D'Silva*)*

UNIVERSITY OF
OXFORD

# A Tale of Two Communities

AI
(1977)

*over-*
*approximate,*
*sound*

GAP

*under-*
*approximate,*
*precise*

DP
(1960)

DPLL
(1962)

CDCL
(1996)

Learning.
Efficiency!

# A Tale of Two Communities

AI
(1977)

*over-approximate, sound*

GAP

*under-approximate, precise*

DP
(1960)

DPLL
(1962)

CDCL
(1996)

BMC
(2003)

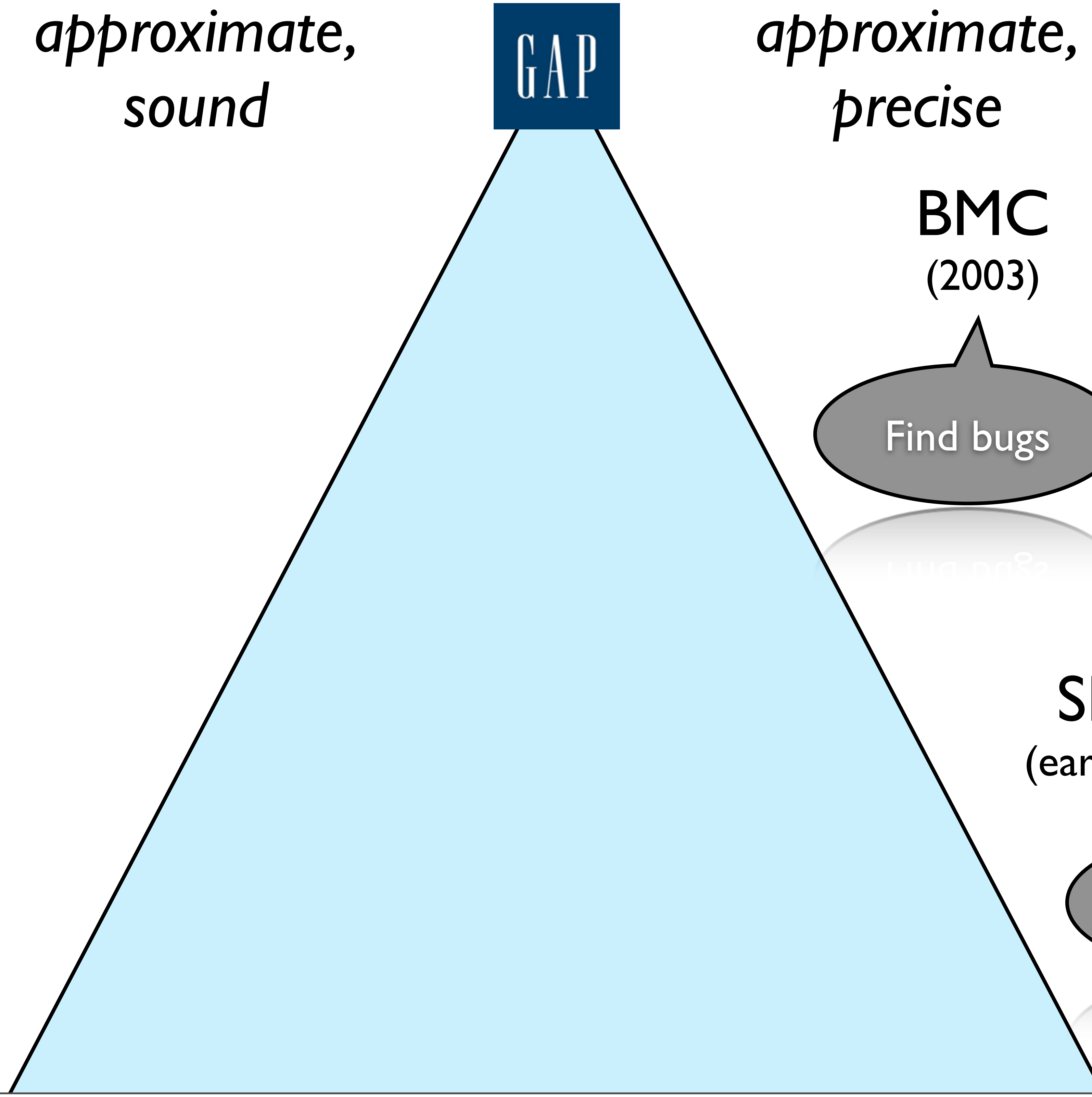
Find bugs

Learning. Efficiency!

# A Tale of Two Communities

AI
(1977)

*over-*
*approximate,*
*sound*




*under-*
*approximate,*
*precise*

DP
(1960)

DPLL
(1962)

CDCL
(1996)

Learning.
Efficiency!

BMC
(2003)

Find bugs

SMT
(early 00s)

Richer
logics

# A Tale of Two Communities

AI
(1977)

*over-approximate, sound*

GAP

*under-approximate, precise*

DP
(1960)

DPLL
(1962)

CDCL
(1996)

BMC
(2003)

Flexible Disjunction

Trace Partitioning
(2005)

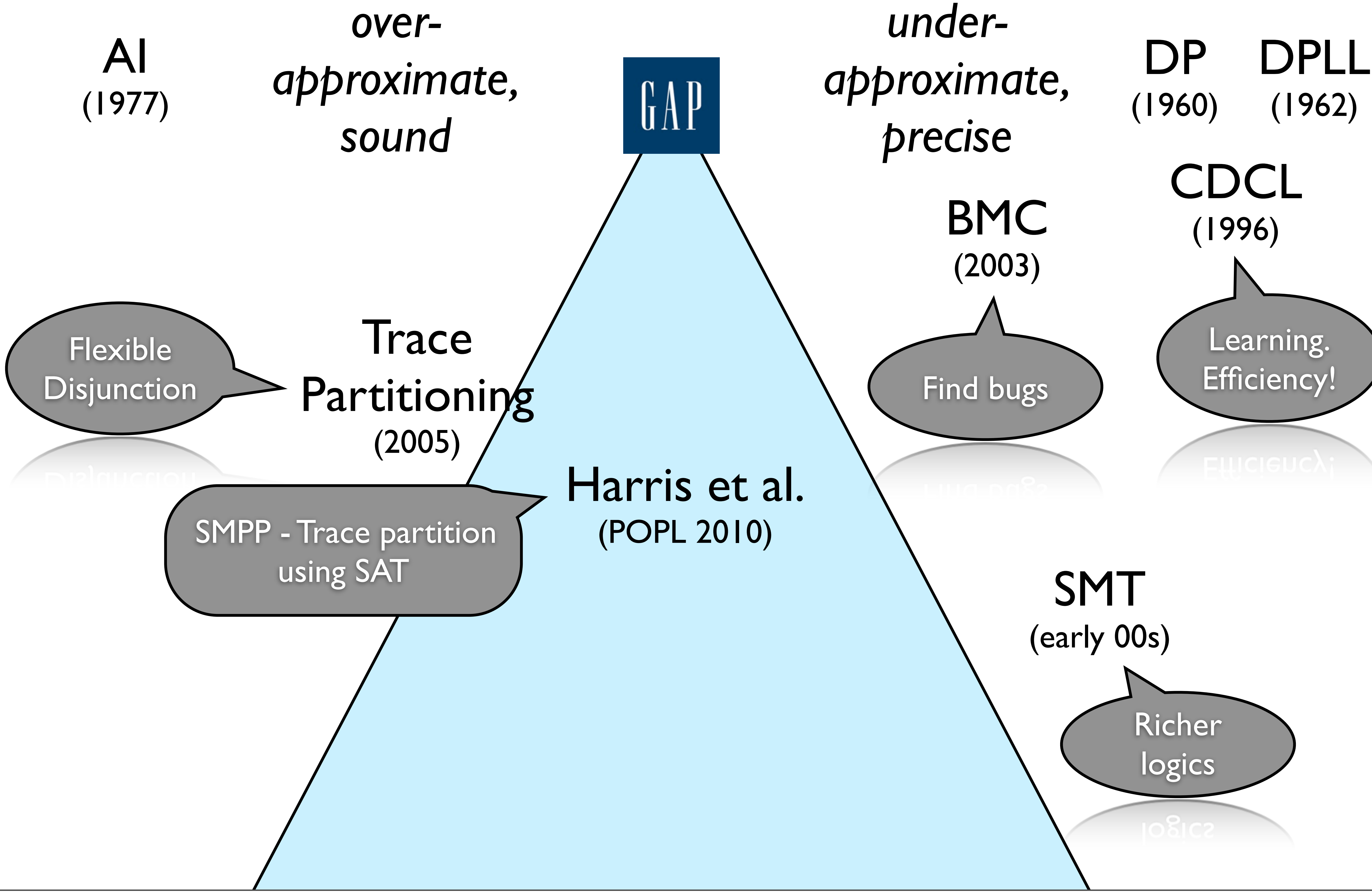
Find bugs

Learning. Efficiency!

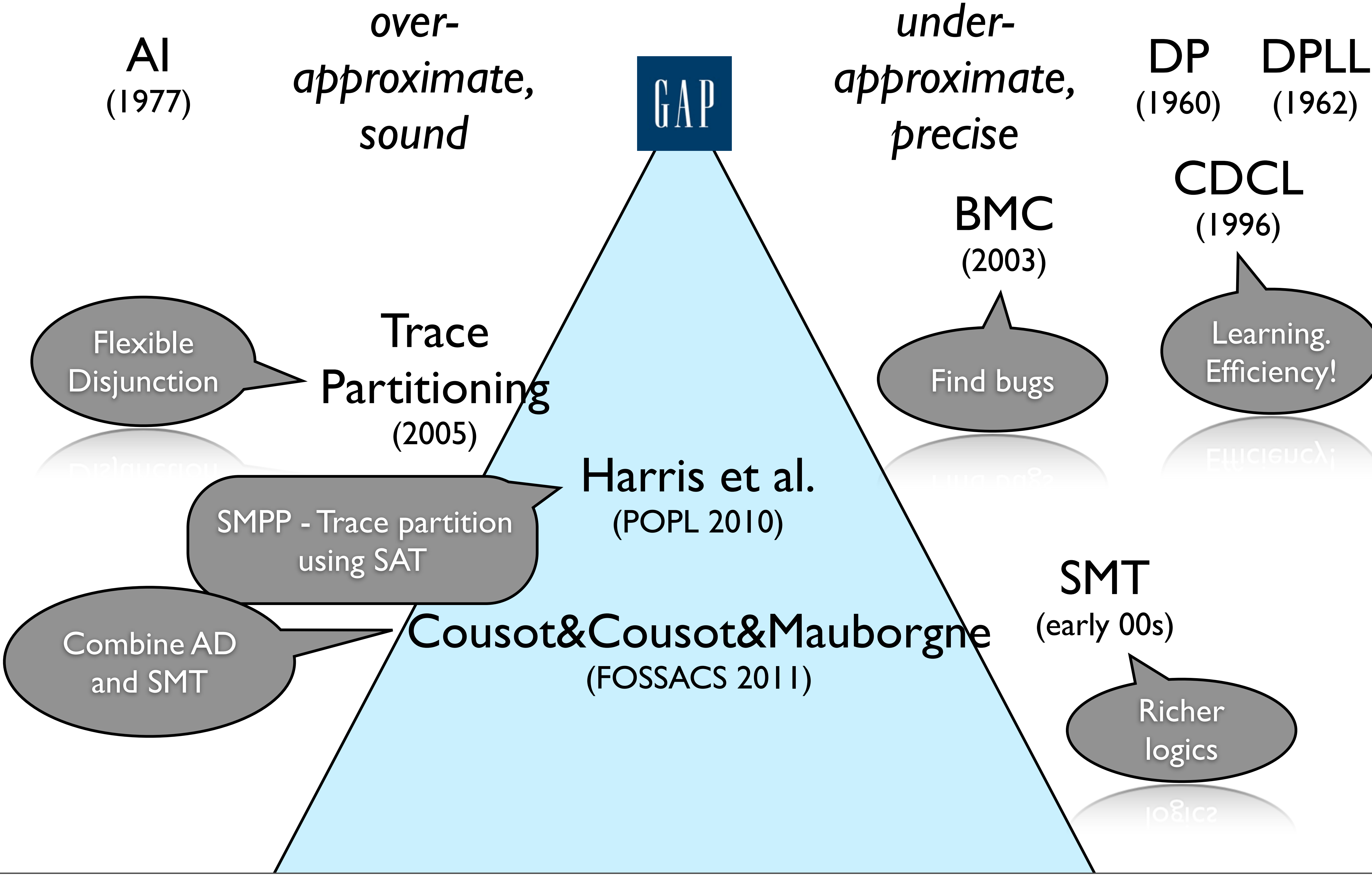
SMT
(early 00s)

Richer logics

# A Tale of Two Communities



# A Tale of Two Communities



# A Tale of Two Communities

AI
(1977)

*over-*
*approximate,*
*sound*

GAP

*under-*
*approximate,*
*precise*

DP
(1960)

DPLL
(1962)

CDCL
(1996)

Learning.
Efficiency!

BMC
(2003)

Find bugs

Flexible
Disjunction

Trace
Partitioning
(2005)

Harris et al.
(POPL 2010)

SMPP - Trace partition
using SAT

SMT
(early 00s)

Richer
logics

Combine AD
and SMT

Cousot&Cousot&Mauborgne
(FOSSACS 2011)

Monnieaux & Gonnord
(SAS 2011)

Focus on path
using SMT

# A Tale of Two Communities

DPLL (1962) CDCL (1996) solvers

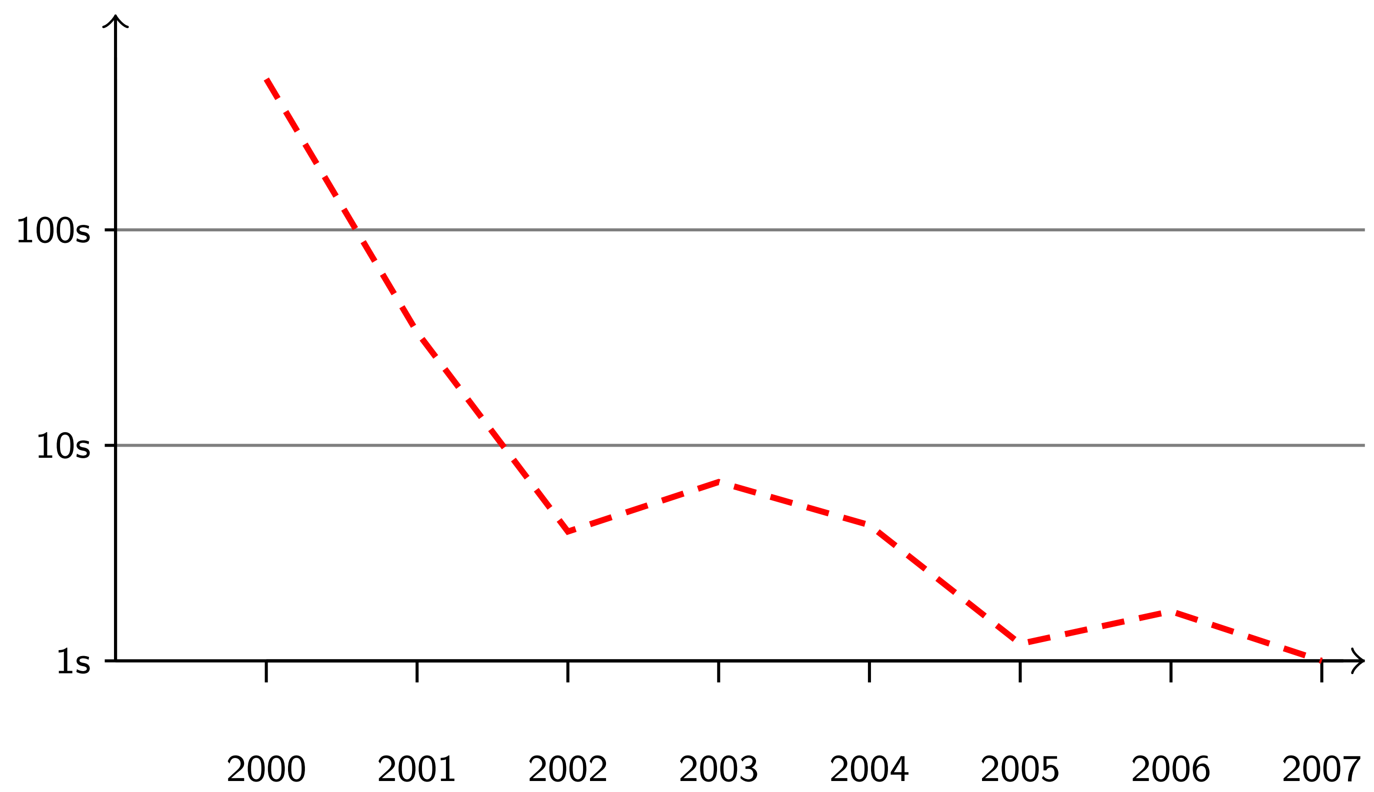
are

(proper) abstract interpreters

AI (1977)

Why does this matter?

# Why does this matter?

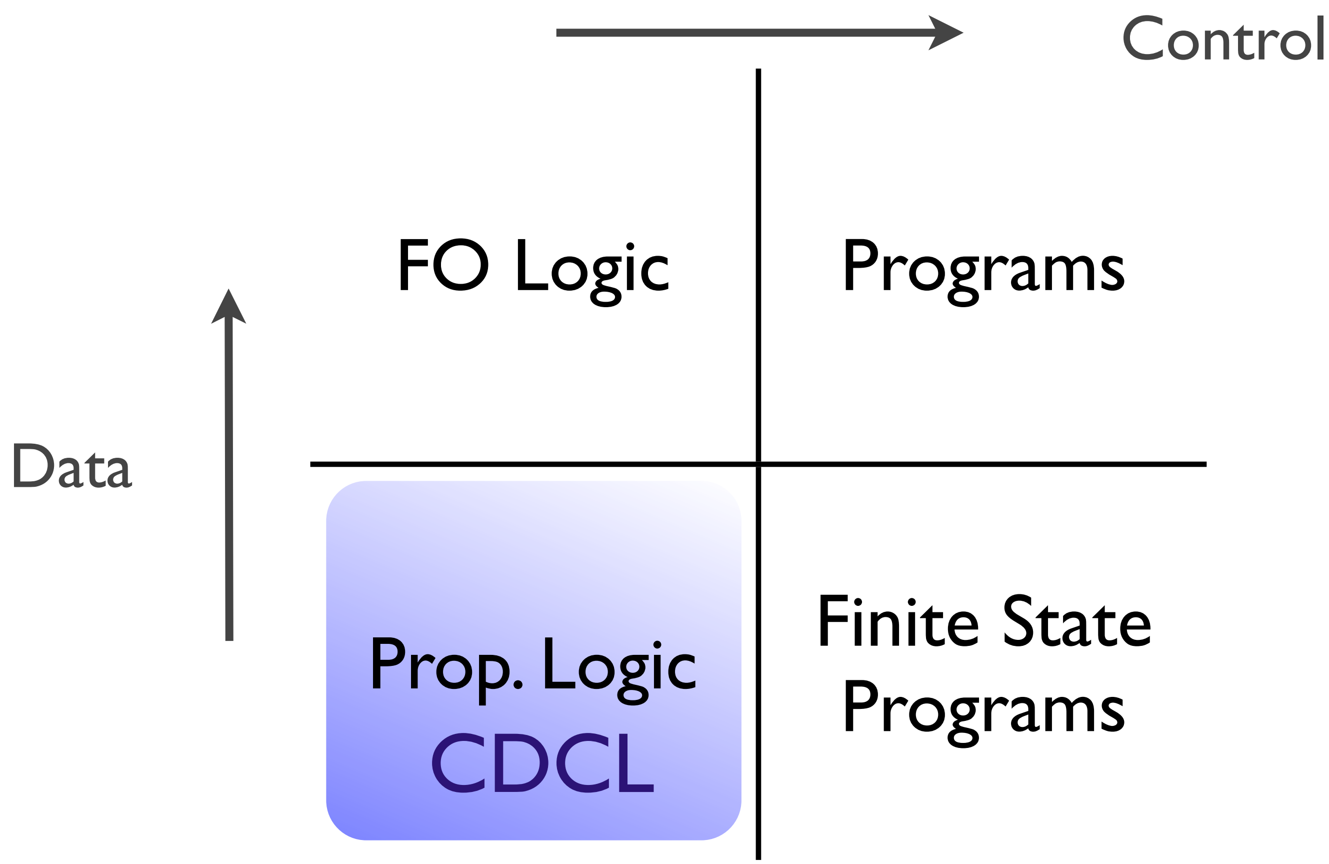



" the practical success of SAT has come as a surprise to many in the computer science community. The combination of strong practical drivers and open competition in this experimental research effort created enough momentum to overcome the pessimism based on theory. Can we take these lessons to other problems and domains?"

– Malik & Zhang, 2009

# Why does this matter?

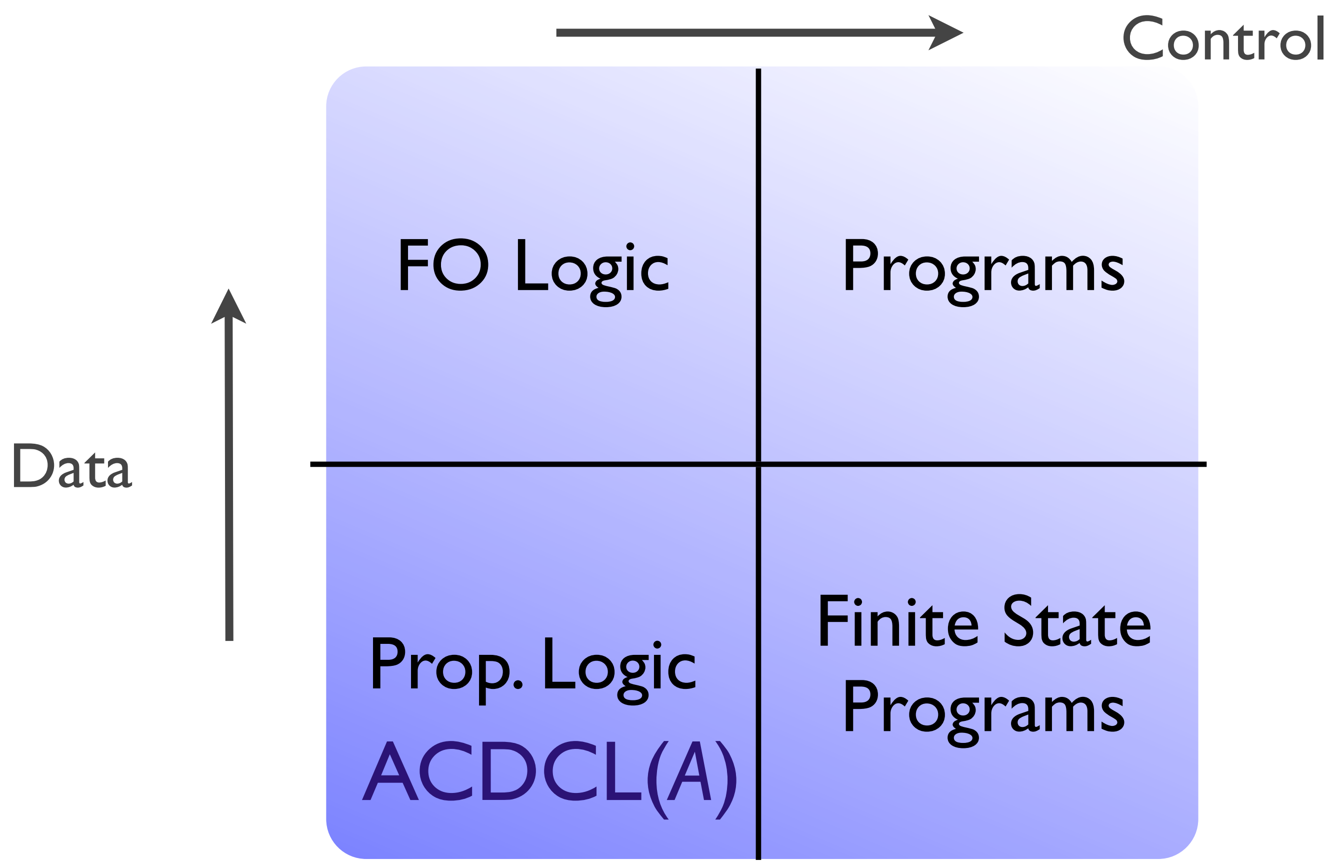
# Why does this matter?



# Why does this matter?

ACDCL(*A*) $\longleftrightarrow$ Abstract domain *A*

# Why does this matter?




ACDCL(*A*) ⟷ Abstract domain *A*

# Conflict Driven Clause Learning

Interpreting Logic

CDCL is Abstract Interpretation

ACDCL(*A*)

# The CDCL Algorithm
## Jargon Slide

| | |
|---|---|
| Propositions | finite set $V$ |
| Literal | $p, \neg p \quad p \in V$ |
| Clause | disjunction of literals |
| CNF formula | conjunction of clauses |

literals

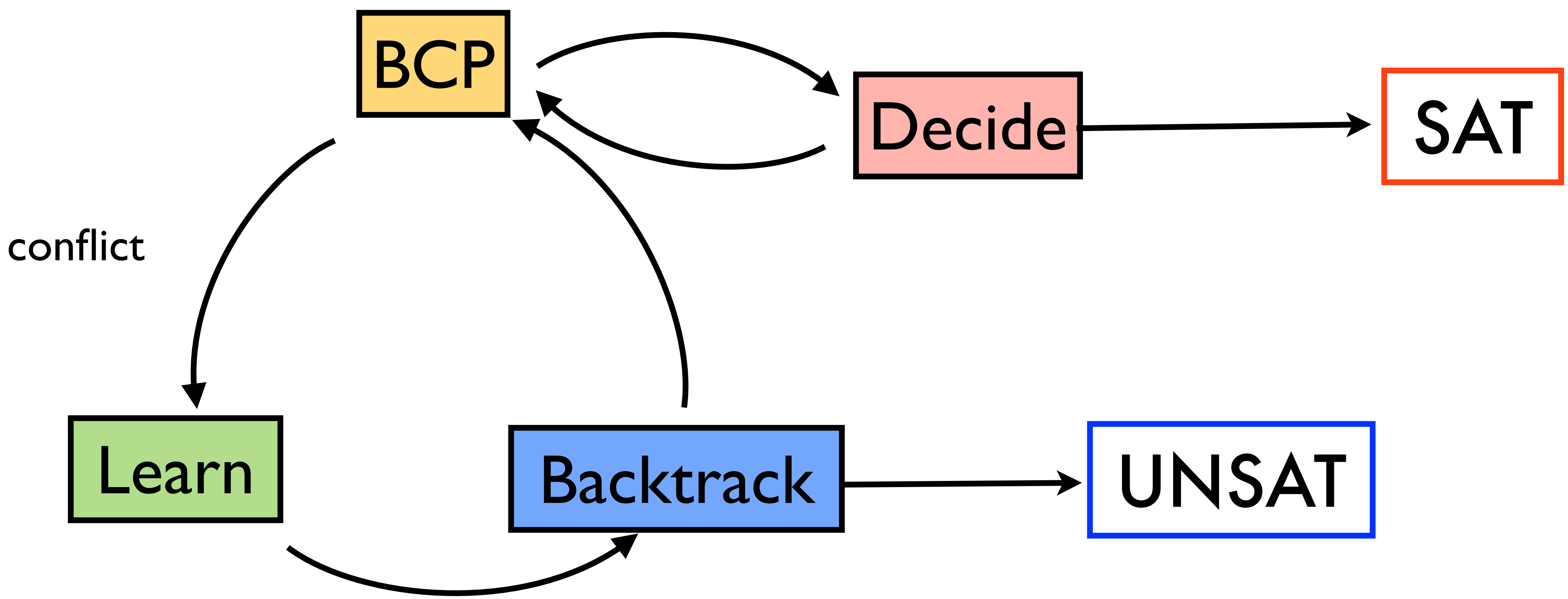
$$\varphi = (\underbrace{p \vee \neg q}) \wedge \ldots \wedge (\underbrace{\neg r \vee w \vee q})$$

clauses

| | |
|---|---|
| Assignment | partial function $V \to \{\mathsf{t}, \mathsf{f}\}$ |
| Satisfiability | Does there exists an assignment $V \to \{\mathsf{t}, \mathsf{f}\}$ such that $\varphi$ is true? |

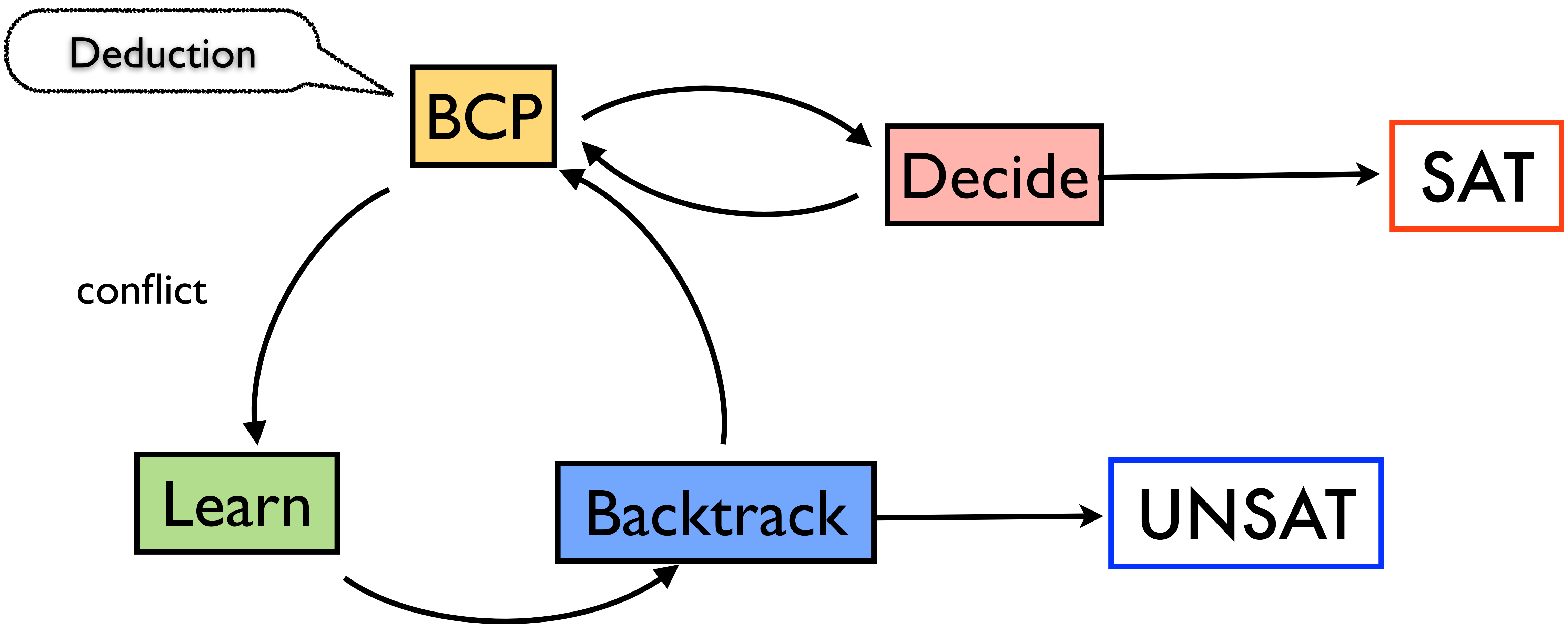
# The CDCL Algorithm

Propositional CNF formula $\varphi = (p \vee \neg q) \wedge \ldots \wedge (\neg r \wedge w \wedge q)$



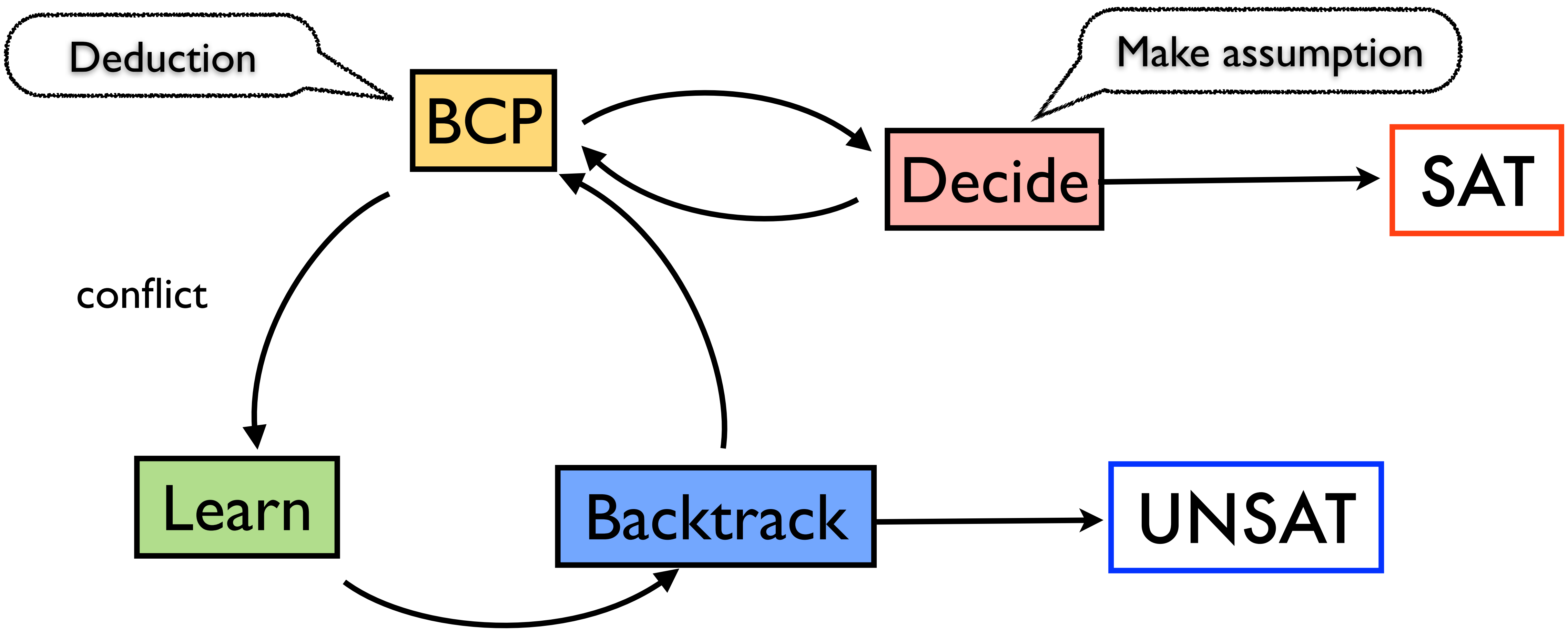
# The CDCL Algorithm

Propositional CNF formula $\varphi = (p \vee \neg q) \wedge \ldots \wedge (\neg r \wedge w \wedge q)$



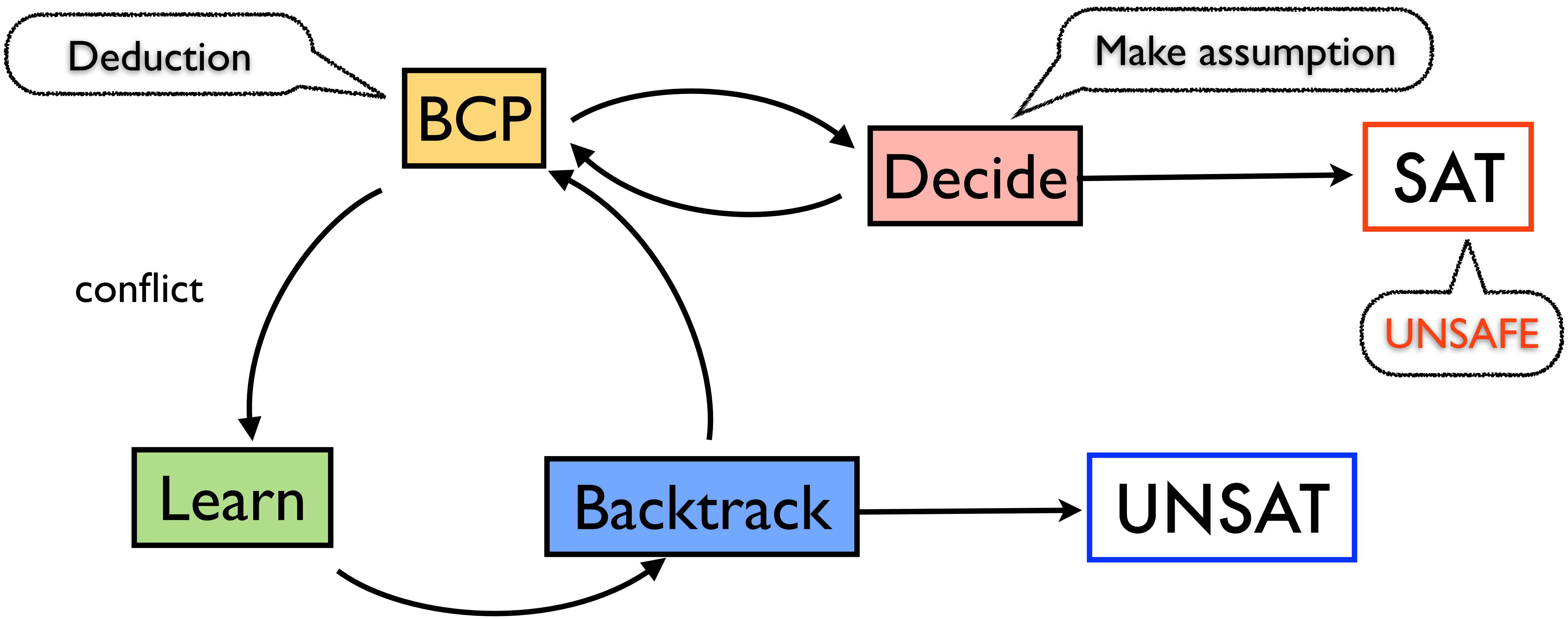
# The CDCL Algorithm

Propositional CNF formula $\varphi = (p \vee \neg q) \wedge \ldots \wedge (\neg r \wedge w \wedge q)$



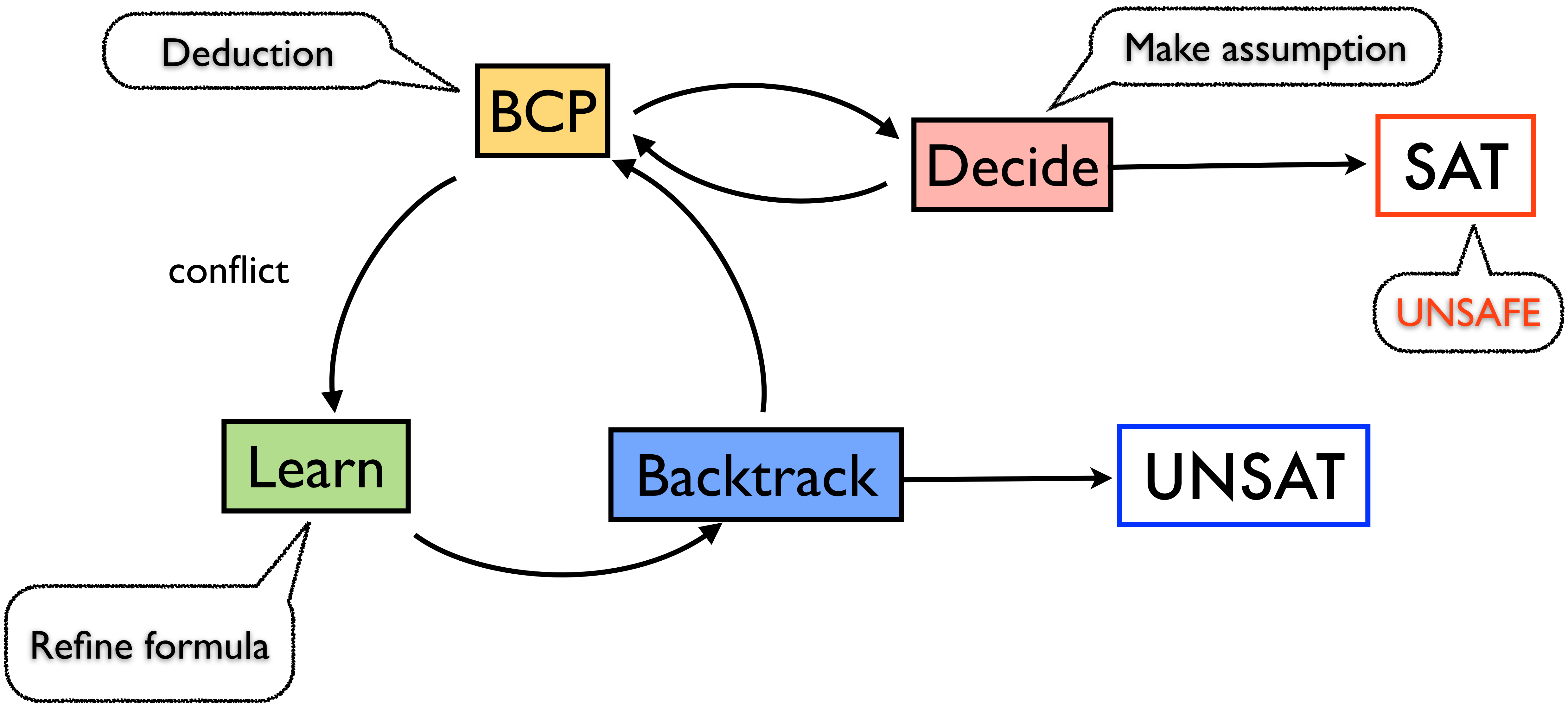
# The CDCL Algorithm

Propositional CNF formula $\varphi = (p \vee \neg q) \wedge \ldots \wedge (\neg r \wedge w \wedge q)$



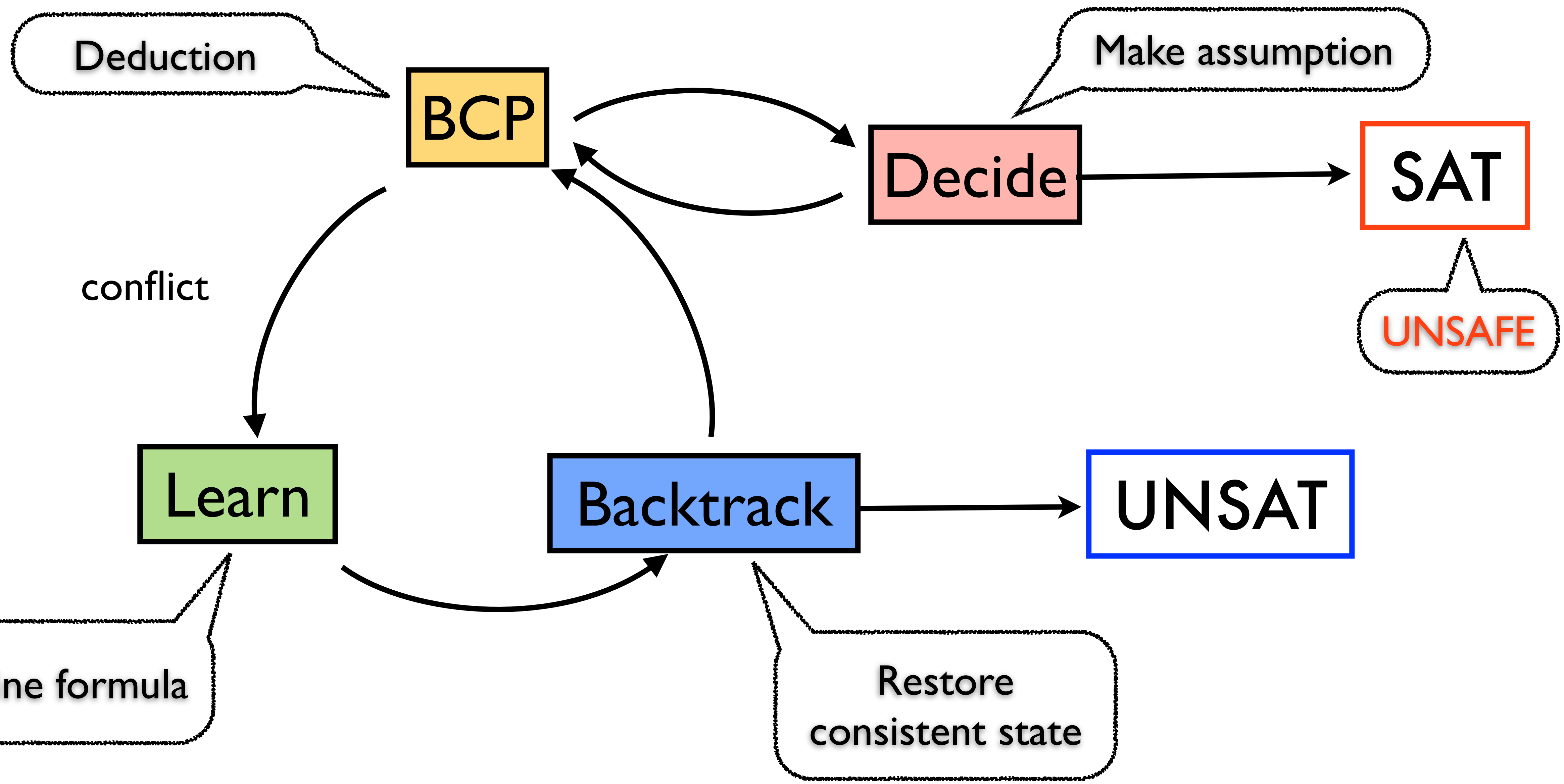
# The CDCL Algorithm

Propositional CNF formula $\varphi = (p \vee \neg q) \wedge \ldots \wedge (\neg r \wedge w \wedge q)$



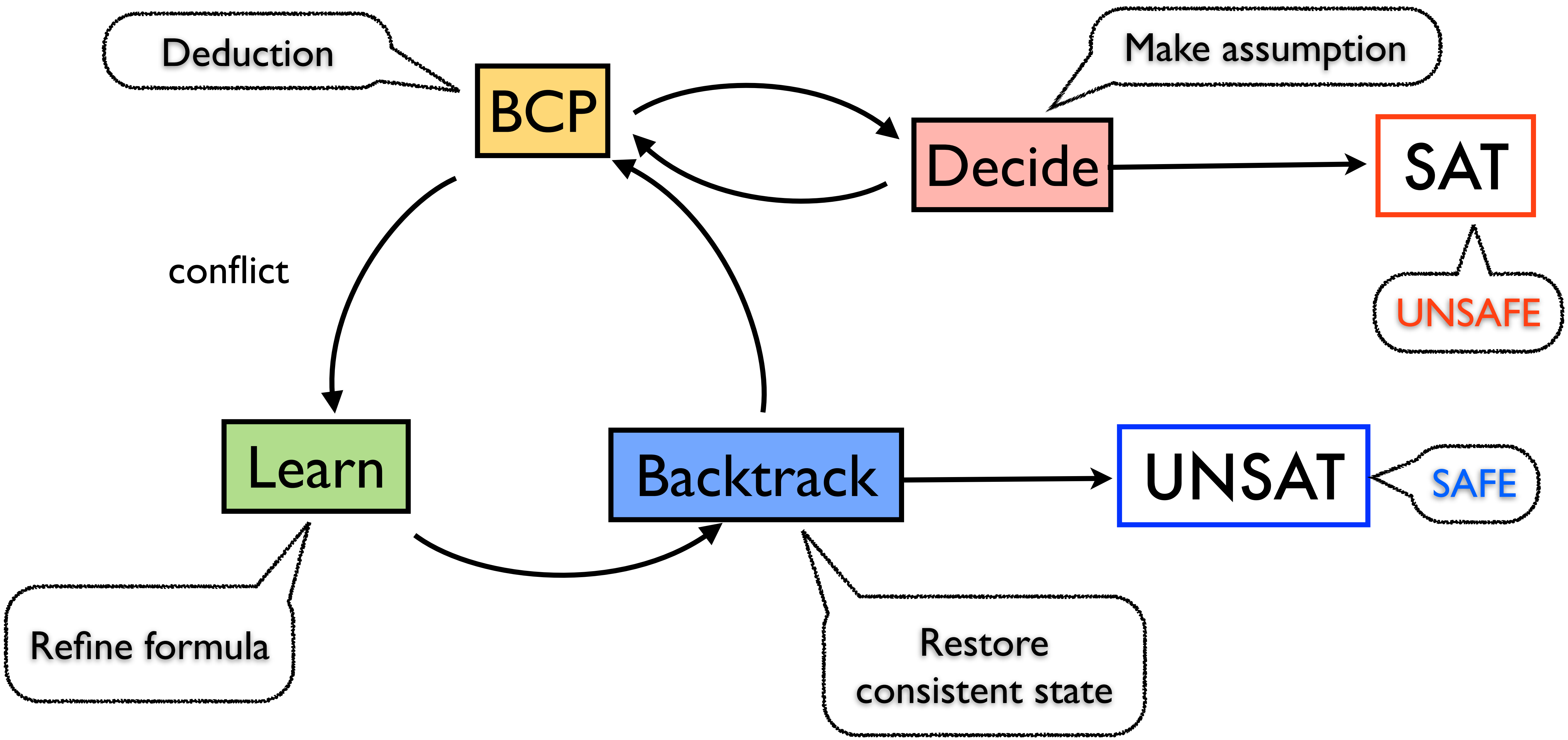
# The CDCL Algorithm

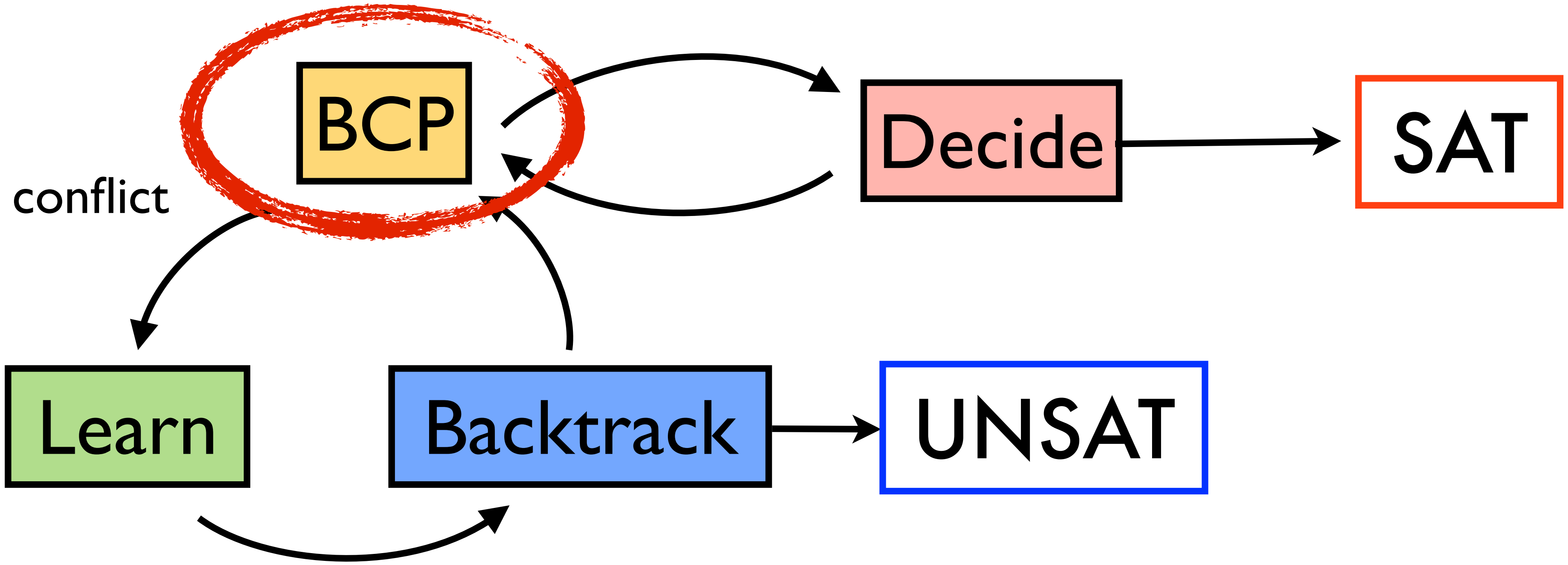
Propositional CNF formula $\varphi = (p \vee \neg q) \wedge \ldots \wedge (\neg r \wedge w \wedge q)$



# The CDCL Algorithm

Propositional CNF formula $\varphi = (p \vee \neg q) \wedge \ldots \wedge (\neg r \wedge w \wedge q)$

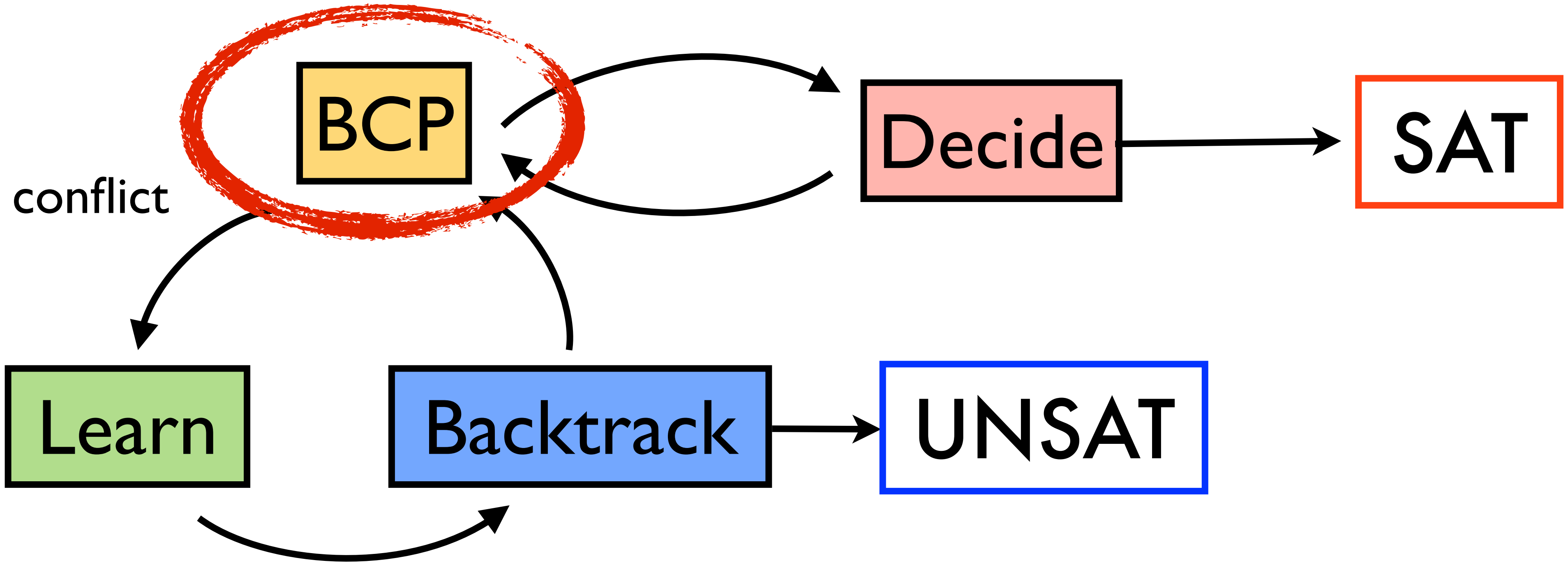





Boolean Constraint Propagation (BCP)

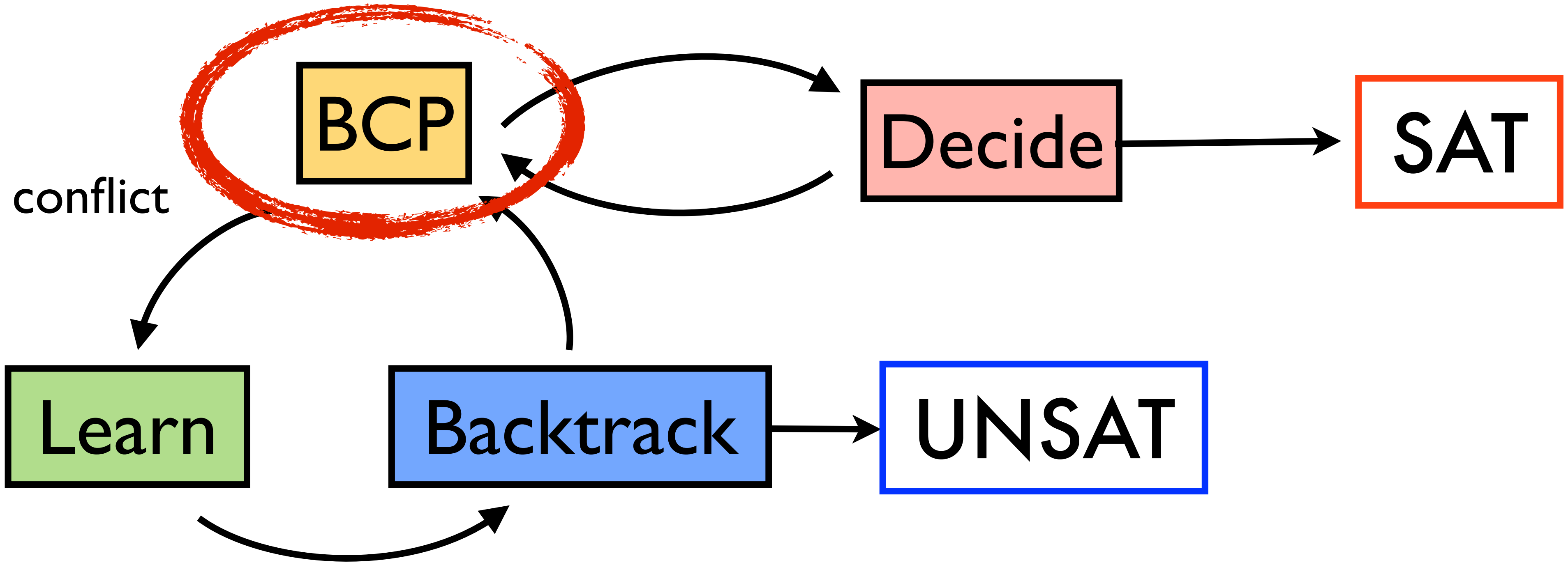
Operates over a partial function
(variable assignment)

$$V \to \{\mathrm{t}, \mathrm{f}\}$$




Boolean Constraint Propagation (BCP)

Unit Rule




# Boolean Constraint Propagation (BCP)

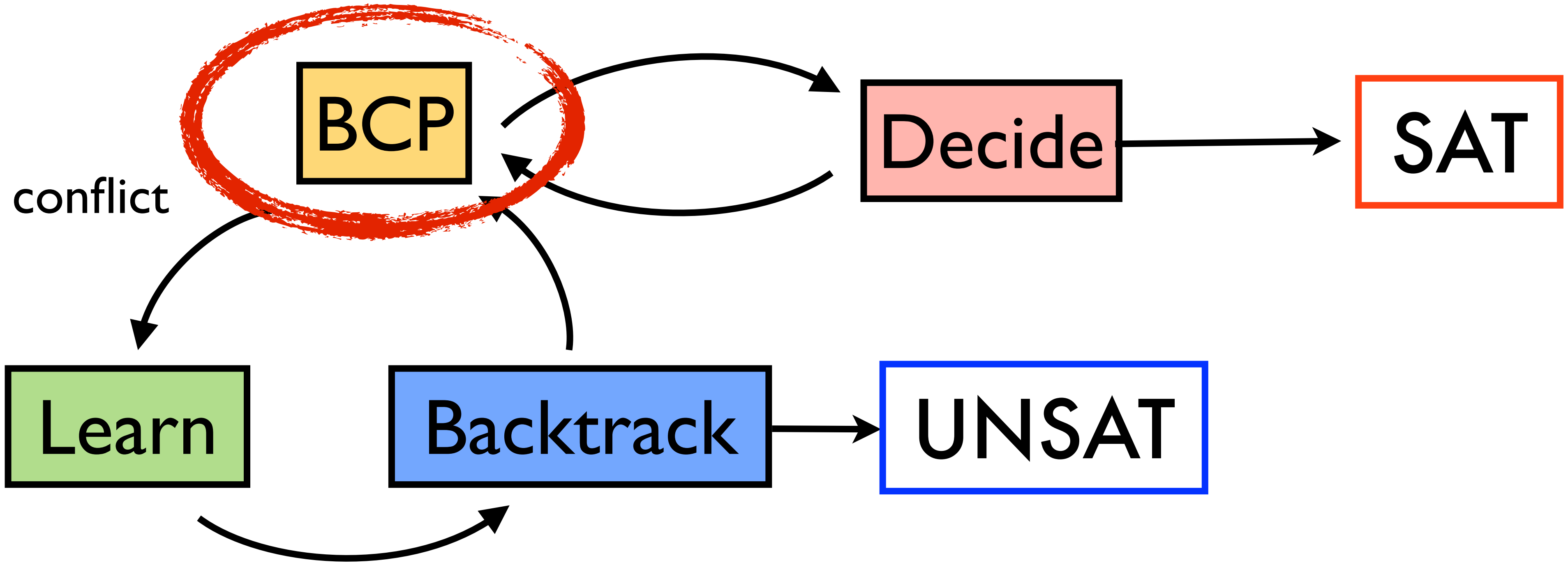
## Unit Rule

$p \mapsto \mathsf{t}$

$q \mapsto \mathsf{f}$

$r \mapsto \mathsf{f}$

$$\ldots \wedge (\neg p \vee q \vee r \vee \neg w) \wedge \ldots$$




# Boolean Constraint Propagation (BCP)

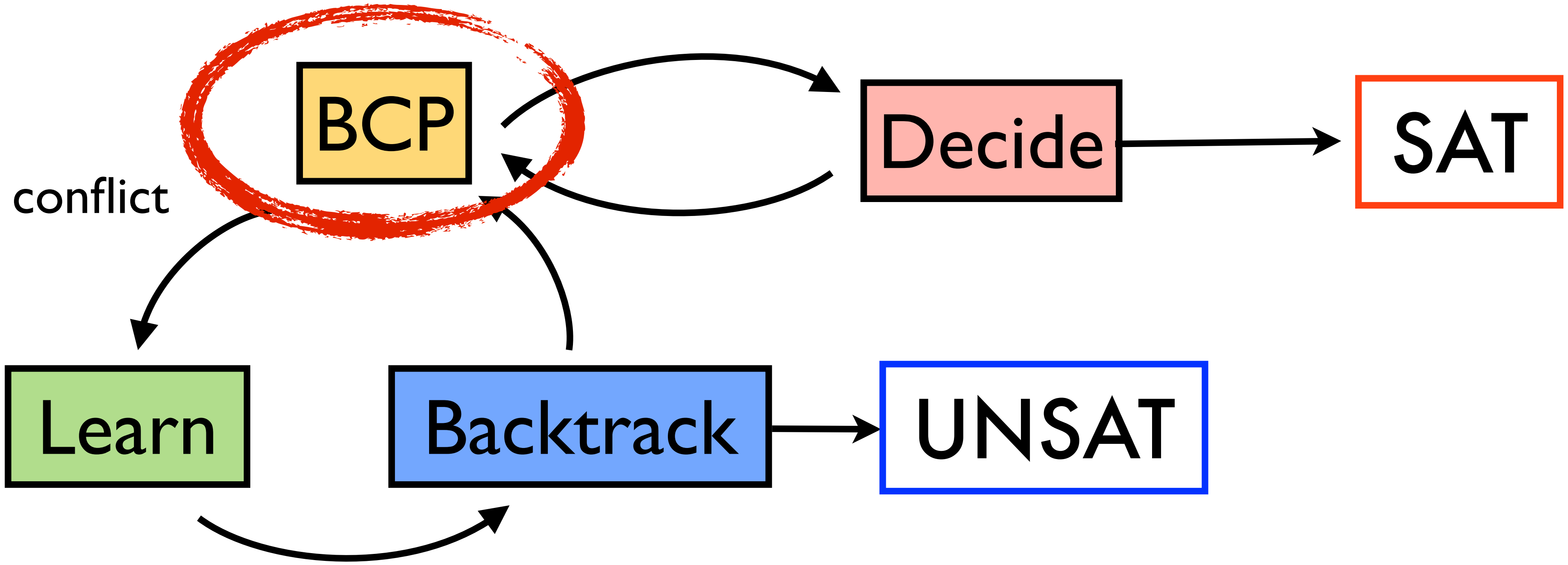
## Unit Rule

$p \mapsto \mathsf{t}$

$q \mapsto \mathsf{f}$

$r \mapsto \mathsf{f}$

$$\ldots \wedge (\neg p \vee q \vee r \vee \neg w) \wedge \ldots$$




# Boolean Constraint Propagation (BCP)

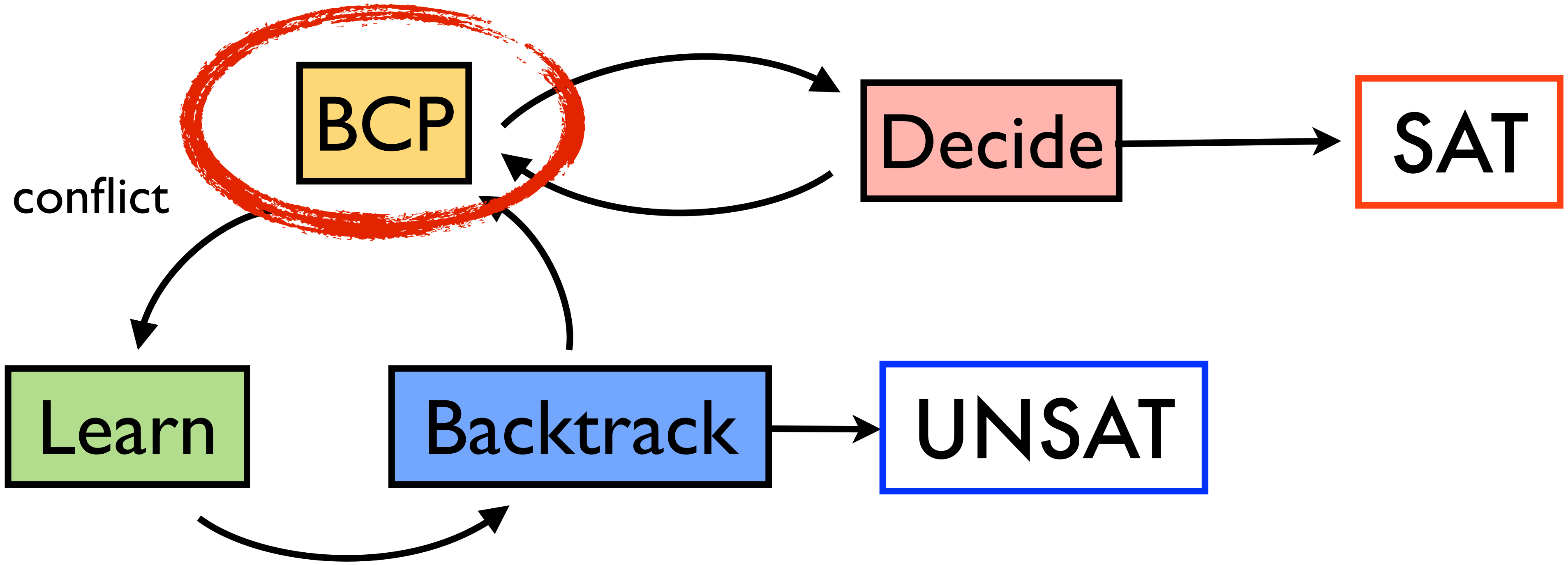
## Unit Rule

$p \mapsto \mathsf{t}$

$q \mapsto \mathsf{f}$

$r \mapsto \mathsf{f}$

$$\ldots \wedge (\neg p \vee q \vee r \vee \neg w) \wedge \ldots$$




# Boolean Constraint Propagation (BCP)

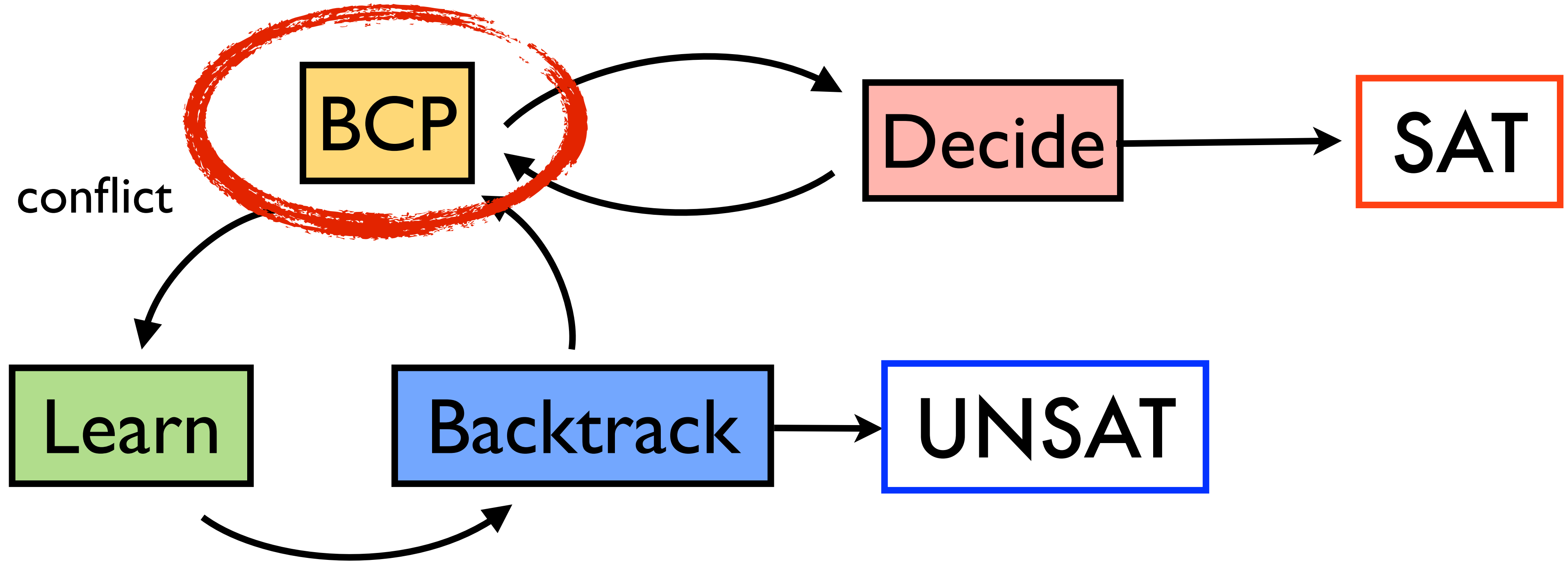
## Unit Rule

$p \mapsto \mathsf{t}$

$q \mapsto \mathsf{f}$

$r \mapsto \mathsf{f}$

$$\ldots \wedge (\neg p \vee q \vee r \vee \neg w) \wedge \ldots$$




# Boolean Constraint Propagation (BCP)

$p \mapsto \mathsf{t}$

$q \mapsto \mathsf{f}$

$r \mapsto \mathsf{f}$

**Unit Rule** $\Rightarrow$

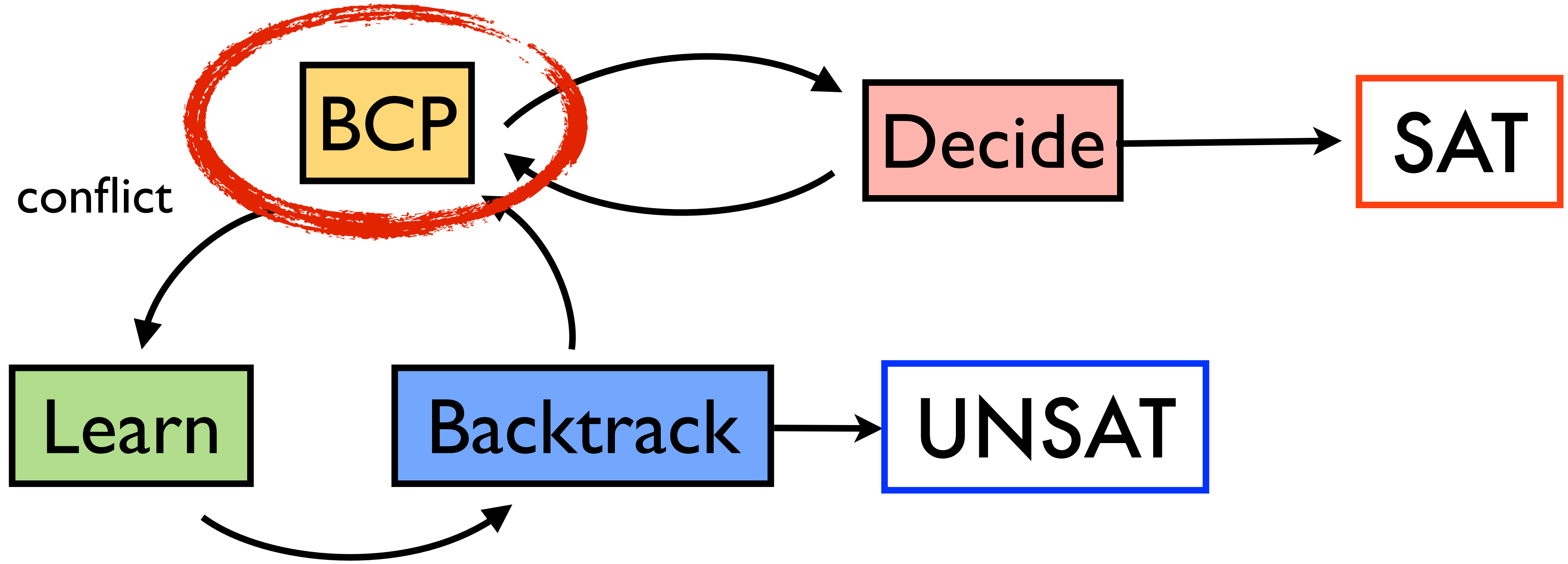
$$\ldots \wedge (\neg p \vee q \vee r \vee \boxed{\neg w}) \wedge \ldots$$

$p \mapsto \mathsf{t}$

$q \mapsto \mathsf{f}$

$r \mapsto \mathsf{f}$

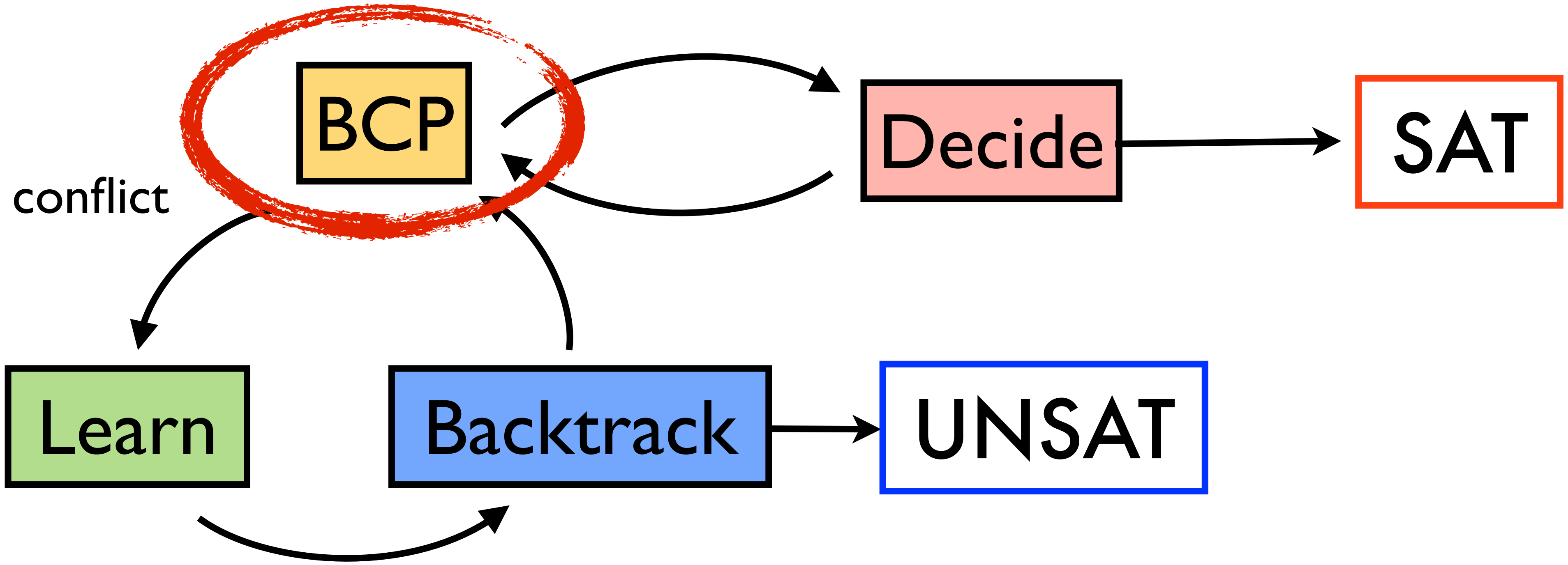
$w \mapsto \mathsf{f}$




$$\varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w)$$

Boolean Constraint Propagation (BCP)

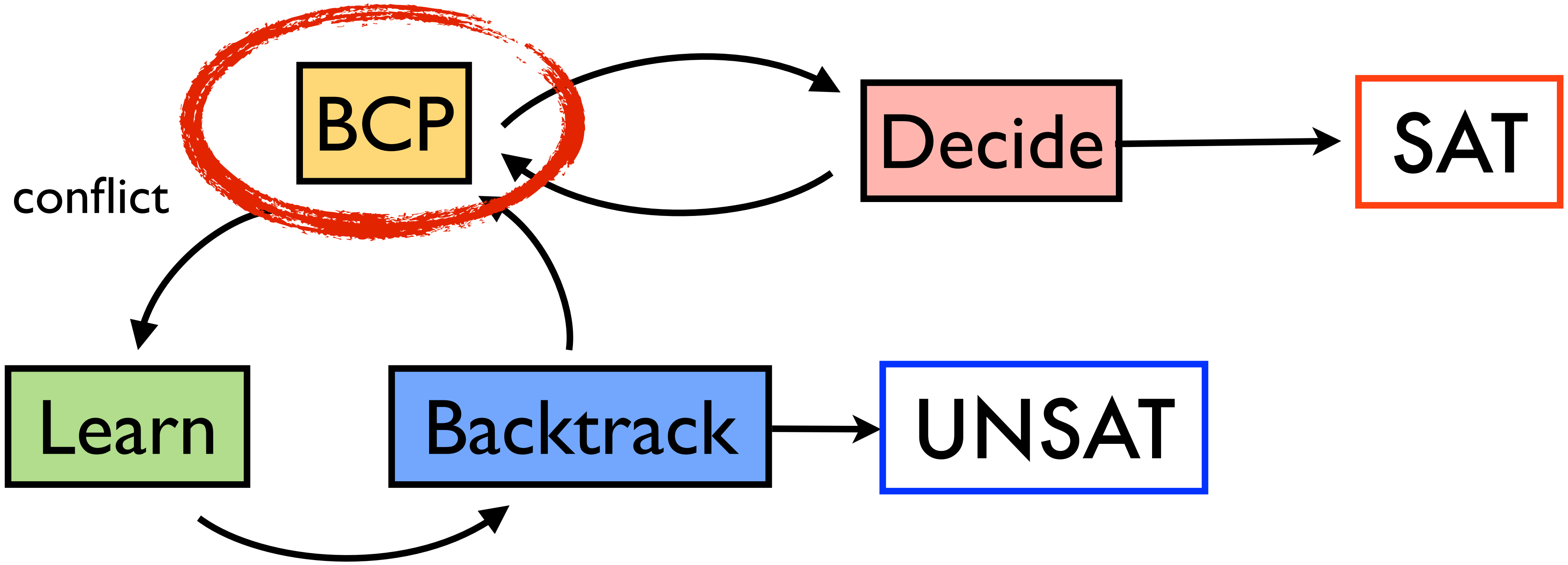
BCP = Exhaustive application of unit rule




$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

Boolean Constraint Propagation (BCP)

BCP = Exhaustive application of unit rule

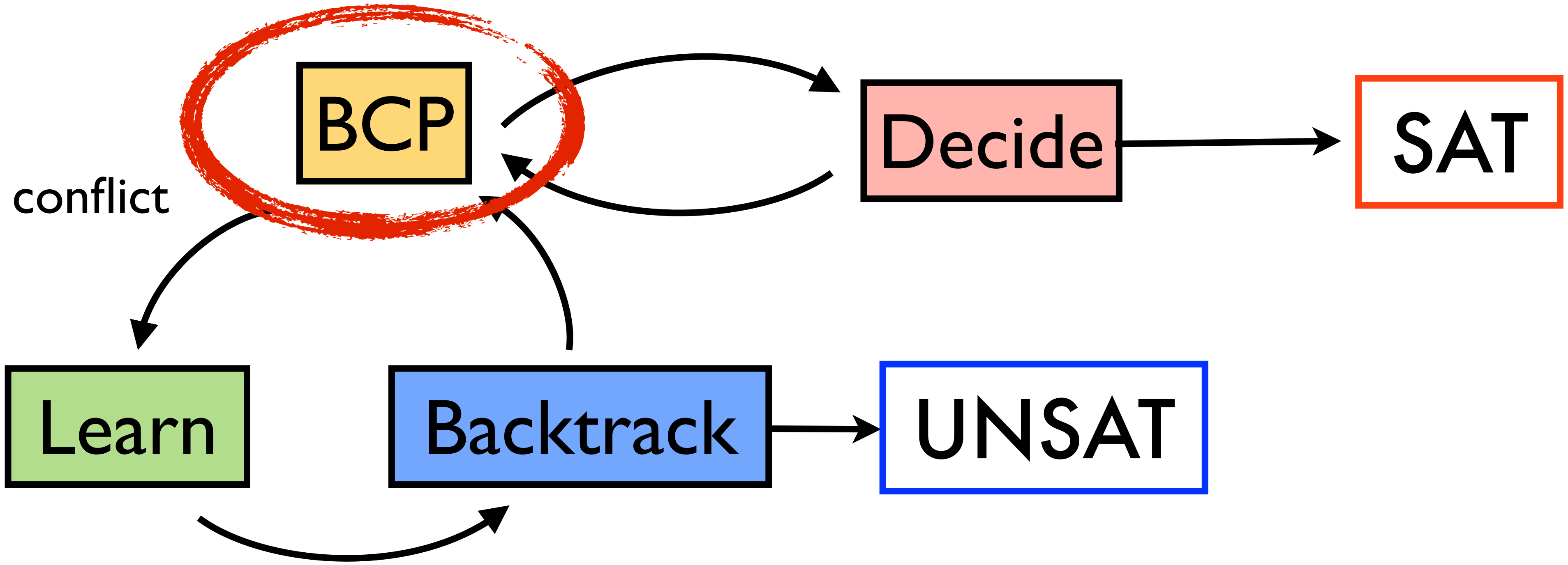



$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

Boolean Constraint Propagation (BCP)

BCP = Exhaustive application of unit rule

$\Longrightarrow$ $p \mapsto \mathsf{t}$

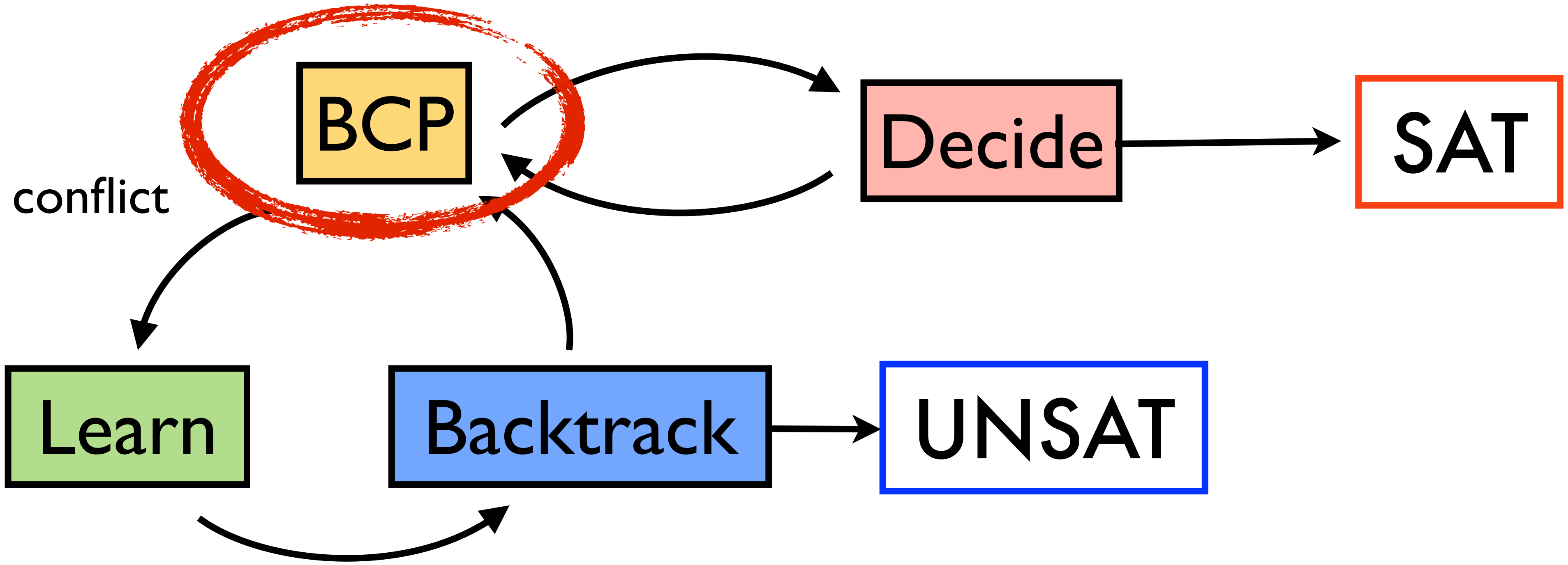



$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

**Boolean Constraint Propagation (BCP)**

BCP = Exhaustive application of unit rule

$\Longrightarrow$ $p \mapsto \mathsf{t}$



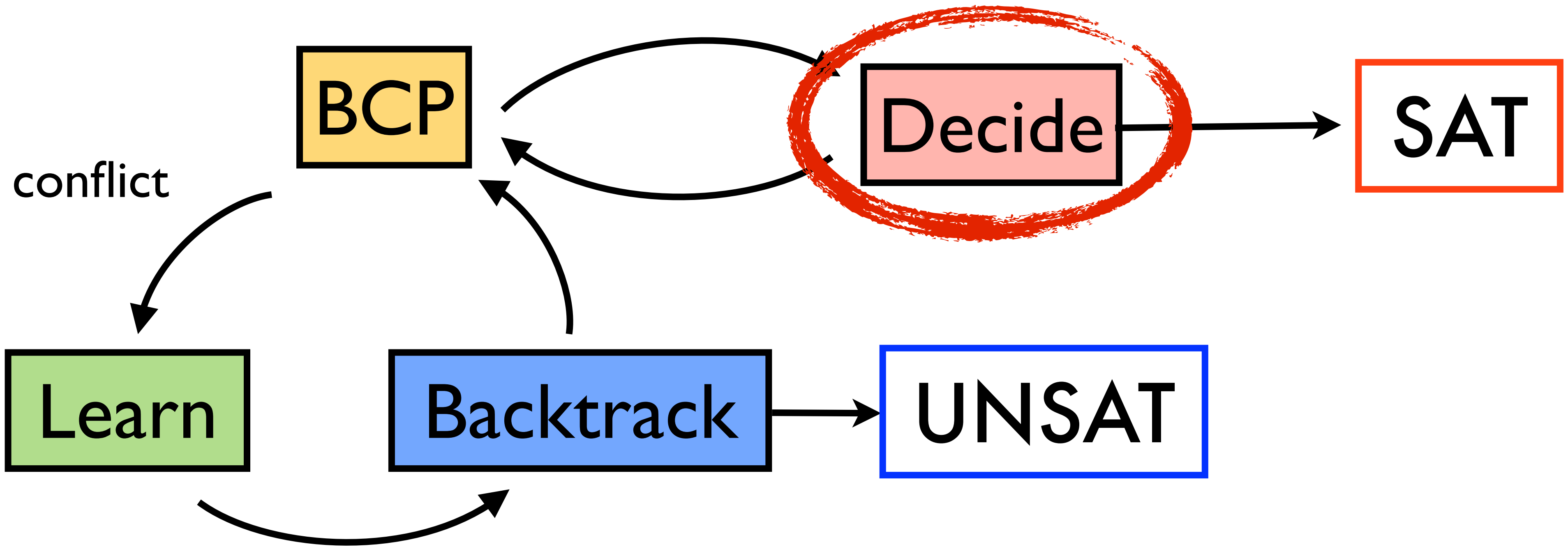

$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

**Boolean Constraint Propagation (BCP)**

BCP = Exhaustive application of unit rule

$\Rightarrow$ $p \mapsto \mathsf{t}$ $\Rightarrow$ $p \mapsto \mathsf{t}$

$q \mapsto \mathsf{f}$

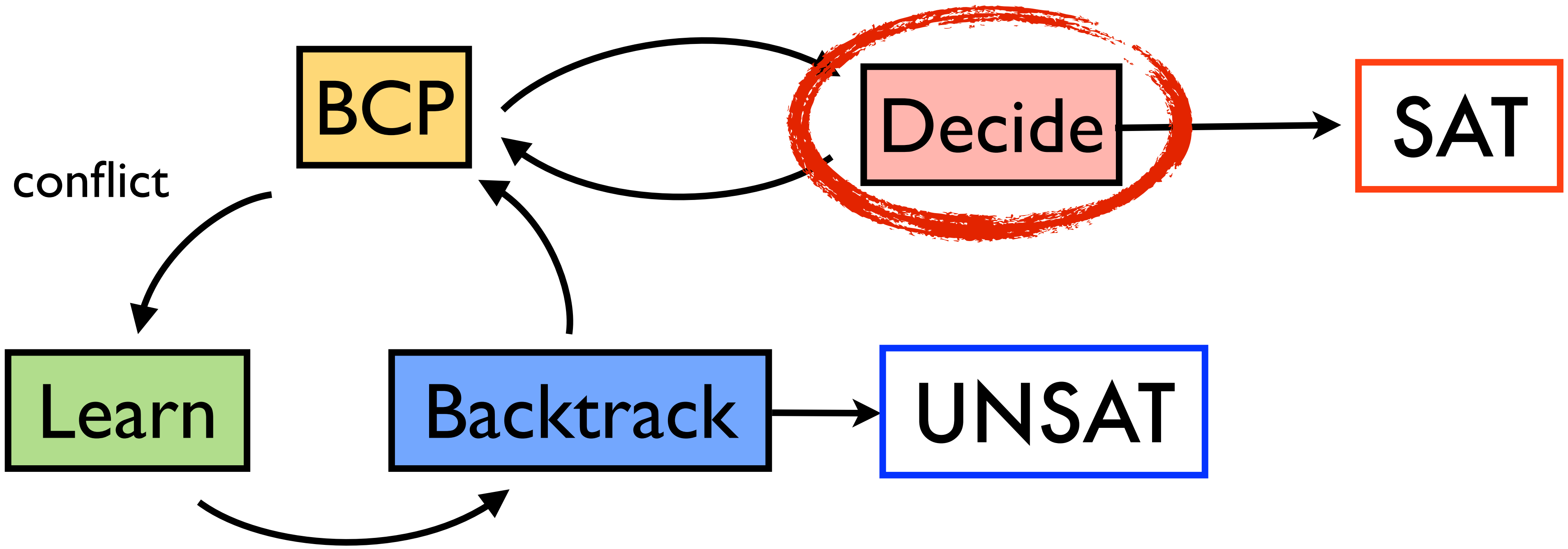



$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

Decisions

$p \mapsto \mathsf{t}$

$q \mapsto \mathsf{f}$



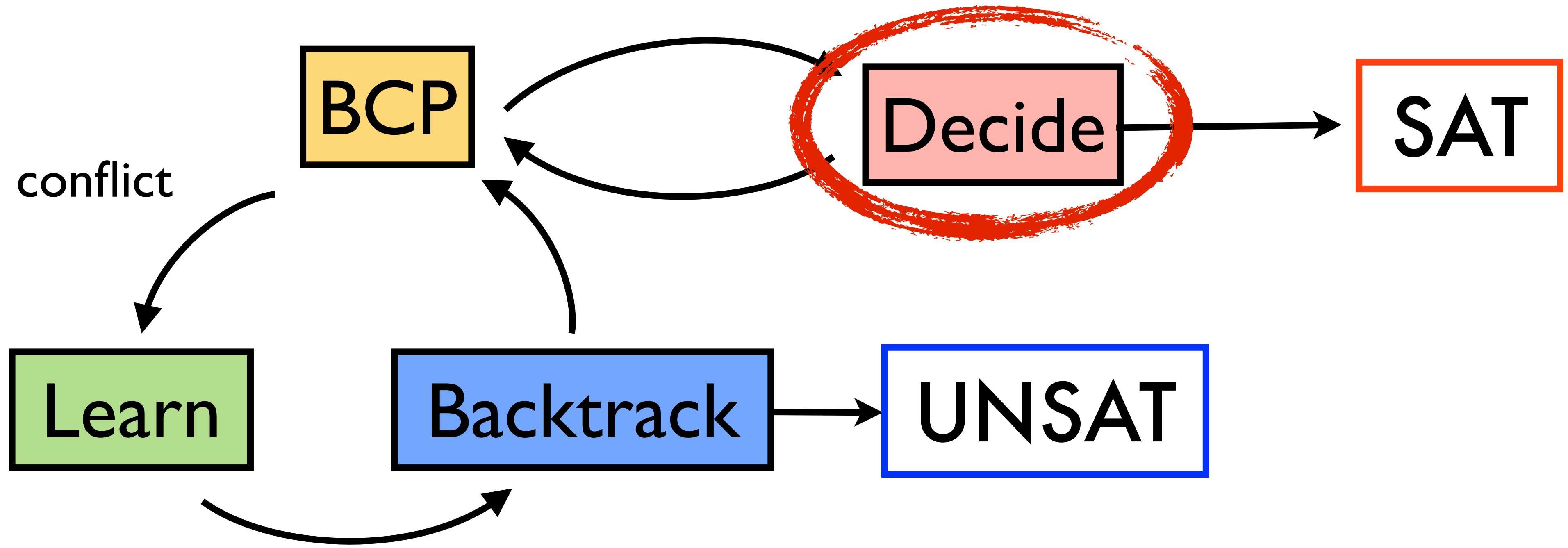

$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

$\boxed{\text{Decisions}}$

Pick an unassigned variable and assign a truth value

$$p \mapsto \mathsf{t}$$

$$q \mapsto \mathsf{f}$$

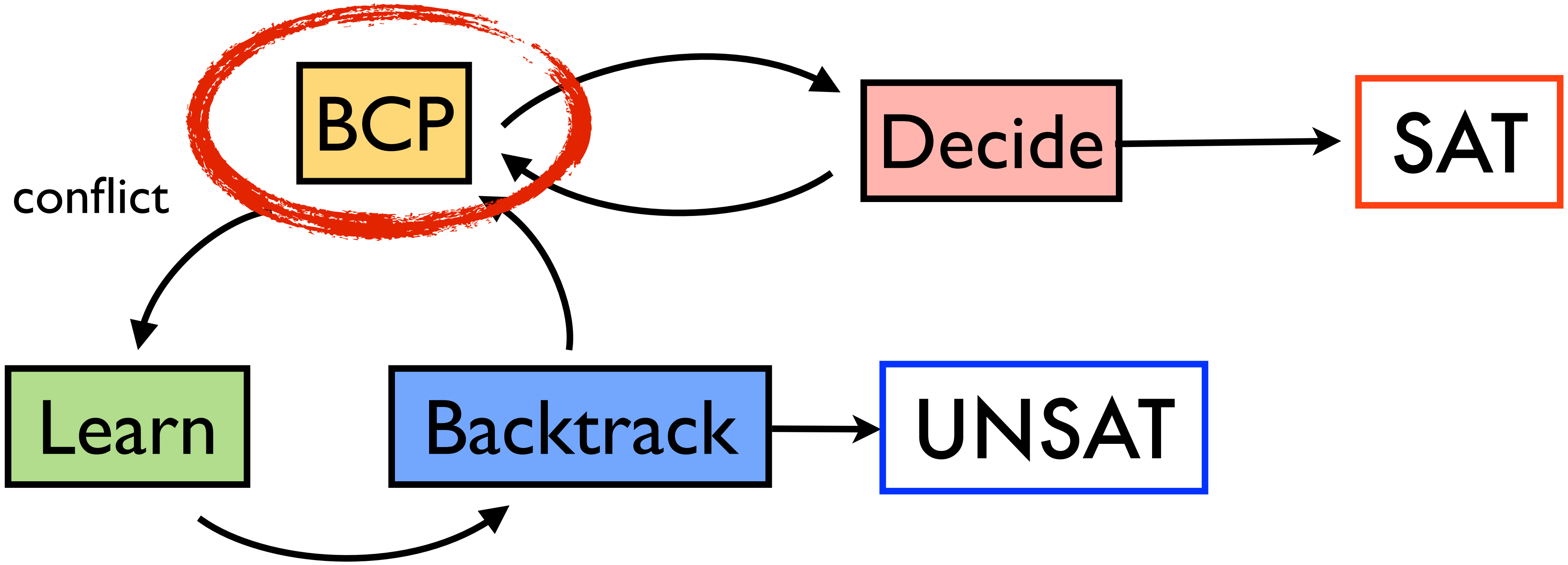



$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

## Decisions

Pick an unassigned variable and assign a truth value

$$\begin{array}{l} p \mapsto \mathsf{t} \\ q \mapsto \mathsf{f} \end{array} \quad \Longrightarrow \quad \begin{array}{l} p \mapsto \mathsf{t} \\ q \mapsto \mathsf{f} \\ r \mapsto \mathsf{f} \end{array}$$

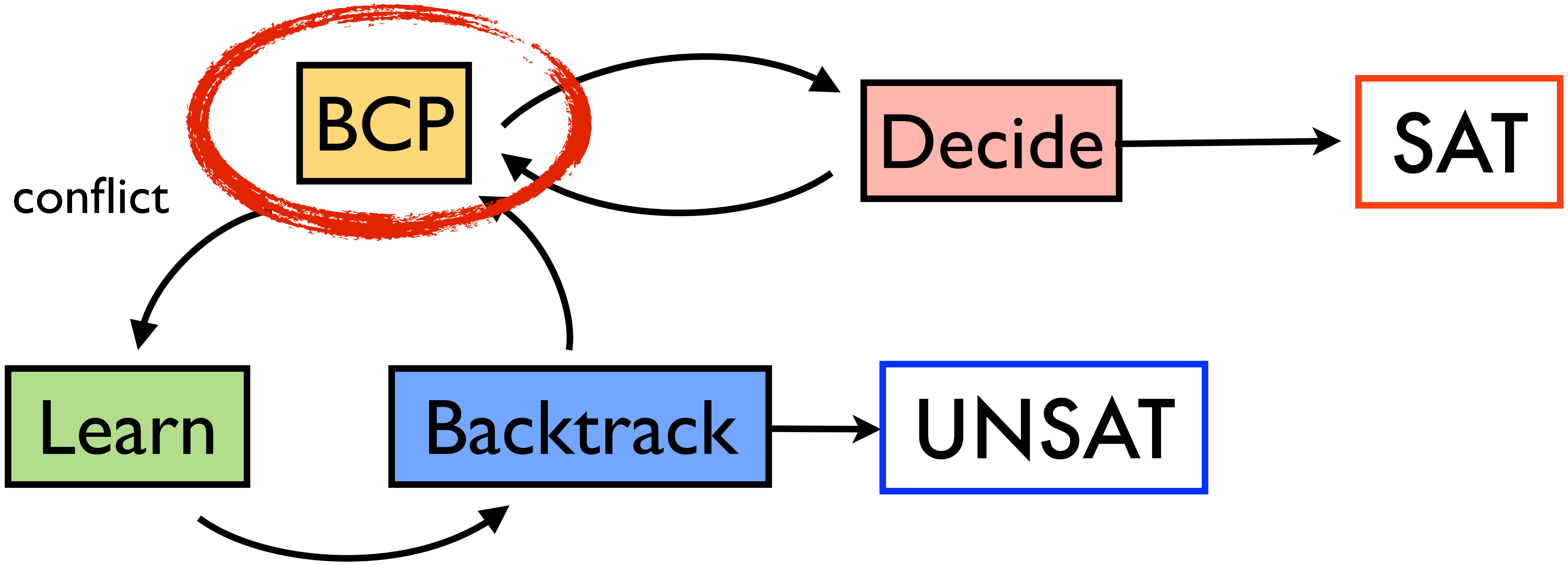



$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

$$p \mapsto \mathsf{t}$$

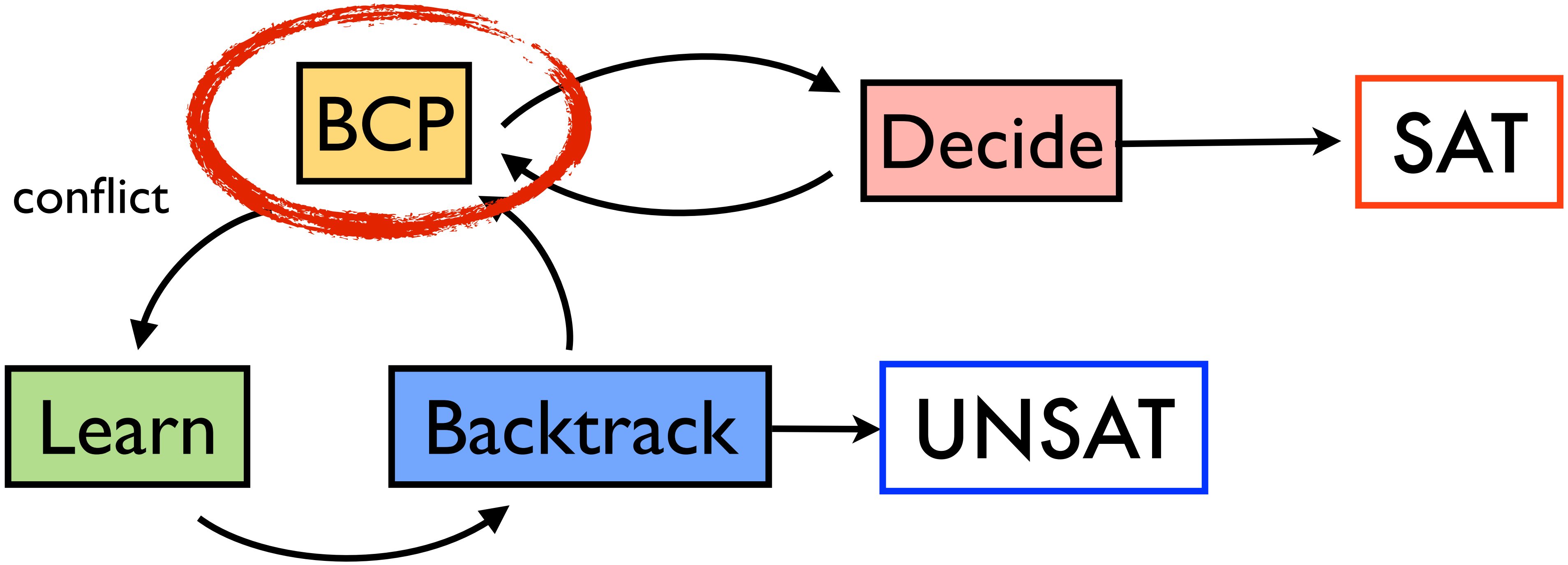
$$q \mapsto \mathsf{f}$$

$$r \mapsto \mathsf{f}$$




$$\varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w)$$

$$\begin{array}{l} p \mapsto \mathsf{t} \\ q \mapsto \mathsf{f} \\ r \mapsto \mathsf{f} \end{array} \quad \Longrightarrow \quad \begin{array}{l} p \mapsto \mathsf{t} \\ q \mapsto \mathsf{f} \\ r \mapsto \mathsf{f} \\ w \mapsto \mathsf{f} \end{array}$$



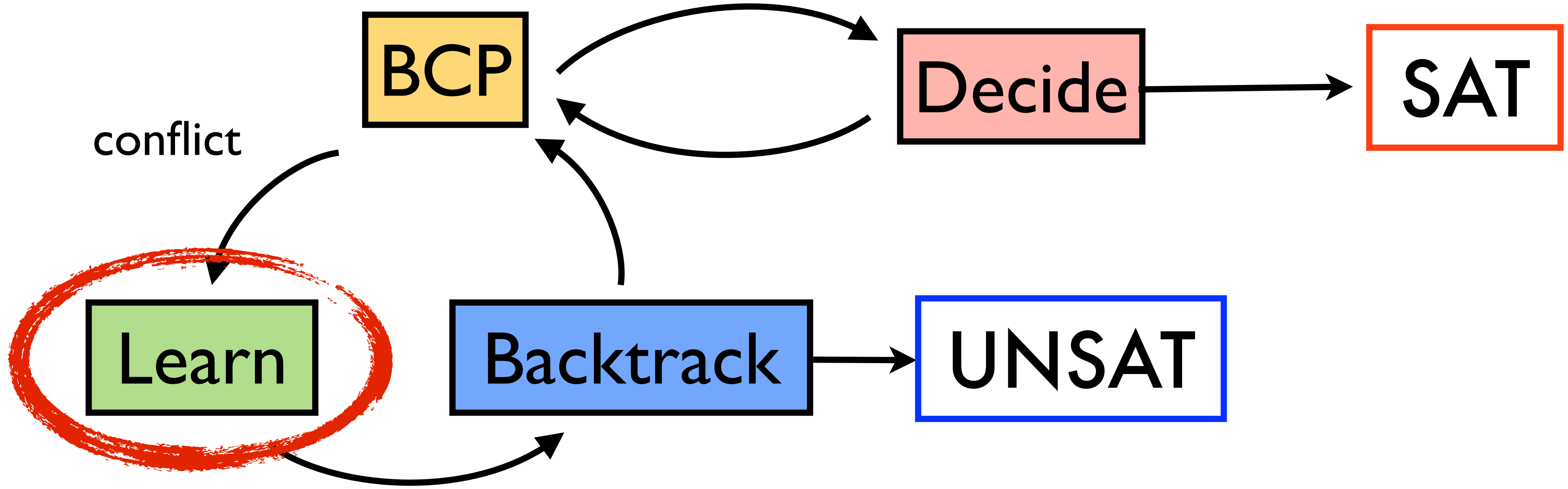

$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

$p \mapsto \mathsf{t}$
$q \mapsto \mathsf{f}$
$r \mapsto \mathsf{f}$

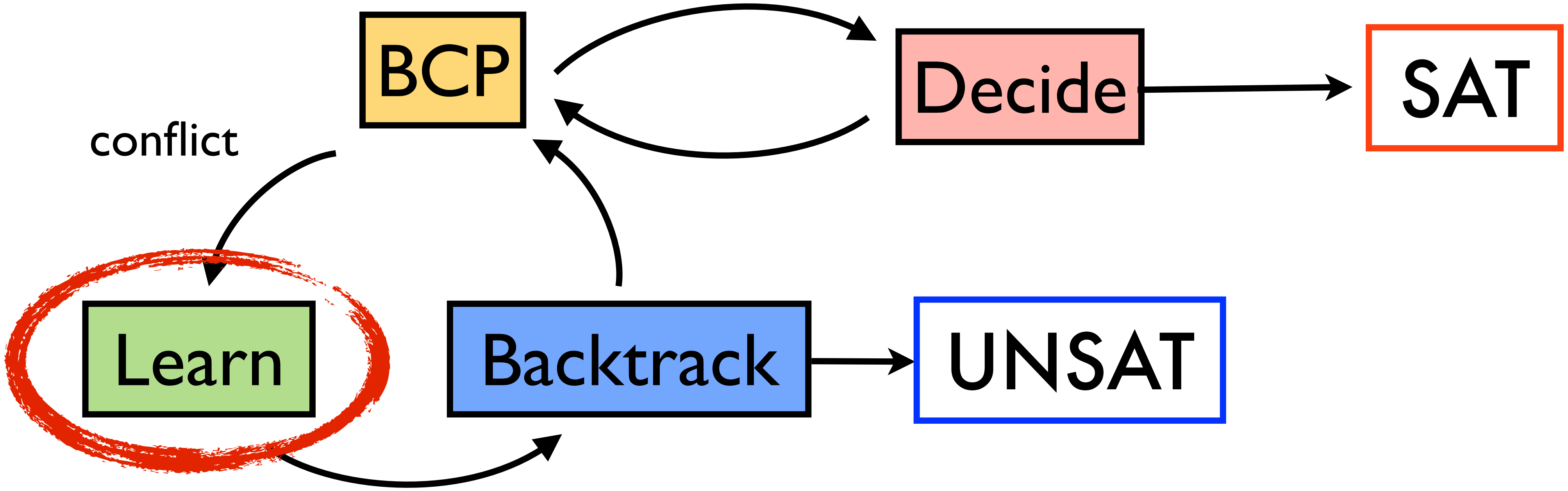
$\Longrightarrow$

$p \mapsto \mathsf{t}$
$q \mapsto \mathsf{f}$
$r \mapsto \mathsf{f}$
$w \mapsto \mathsf{f}$

$\Longrightarrow$ conflict

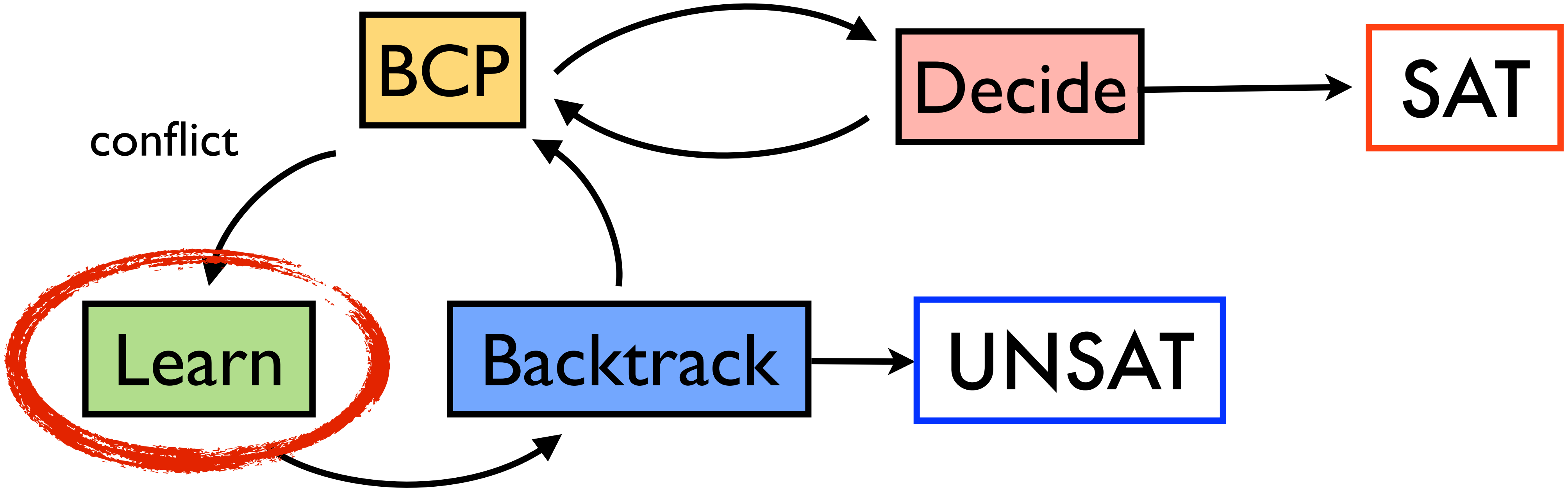



$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$




$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

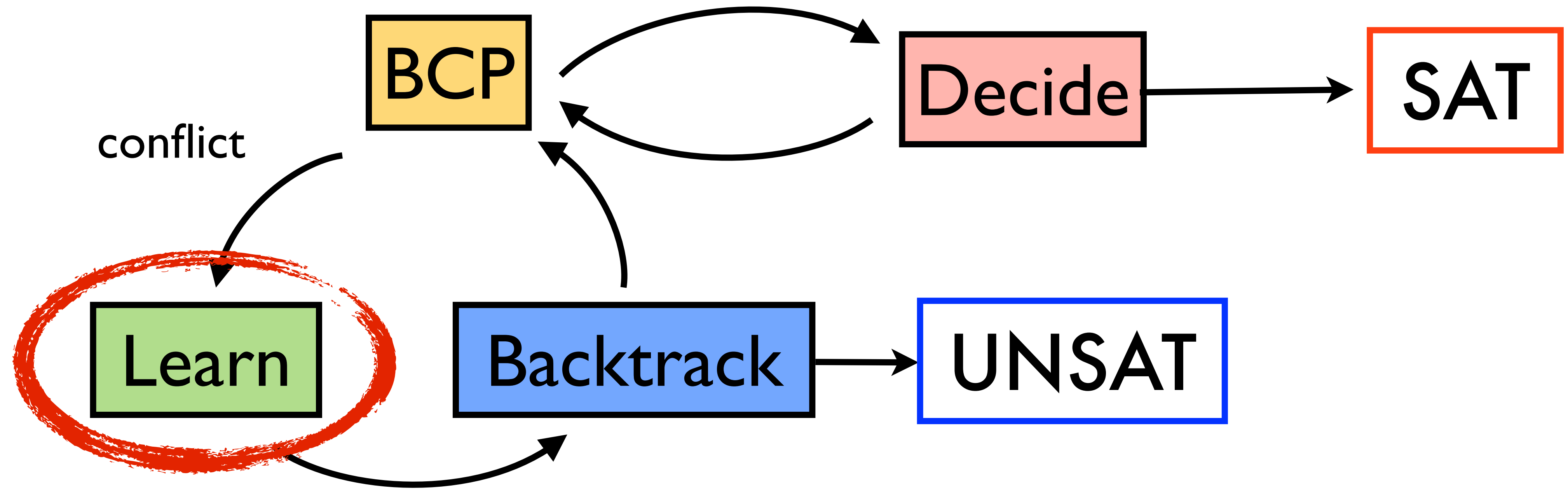
$p \mapsto \mathsf{t}$




$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

$p \mapsto \mathsf{t}$

$q \mapsto \mathsf{f}$

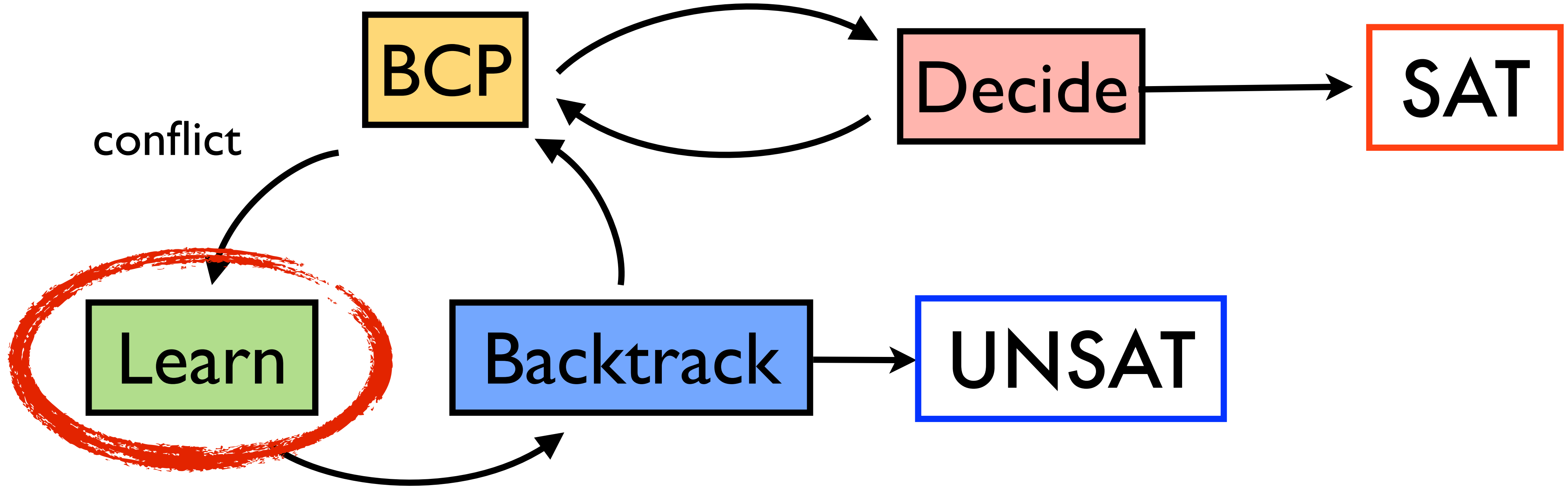



$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

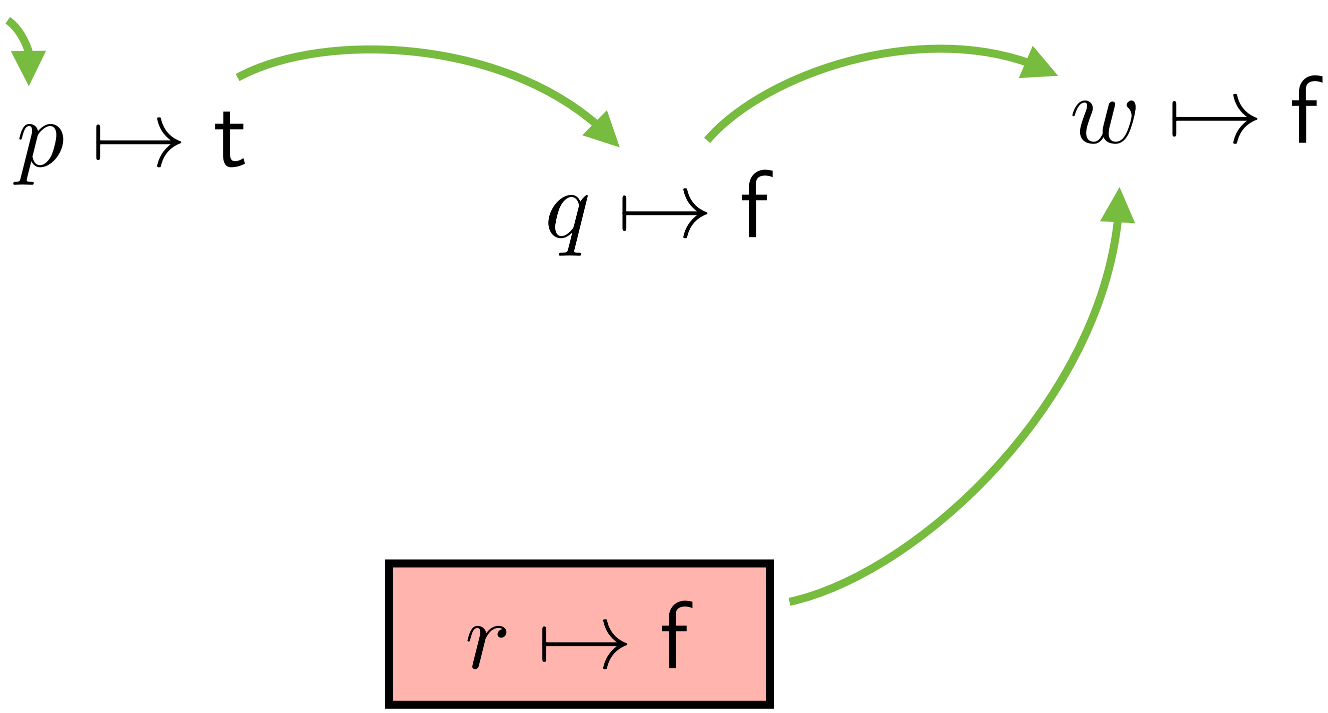
$p \mapsto \mathsf{t}$

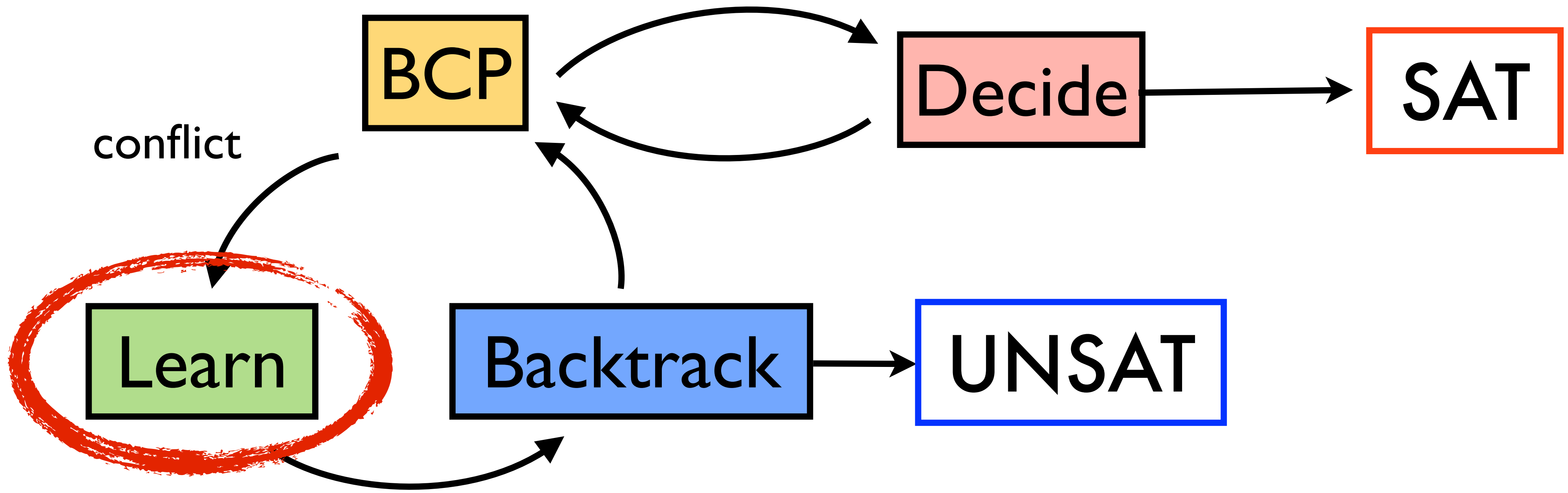
$q \mapsto \mathsf{f}$

$\boxed{r \mapsto \mathsf{f}}$

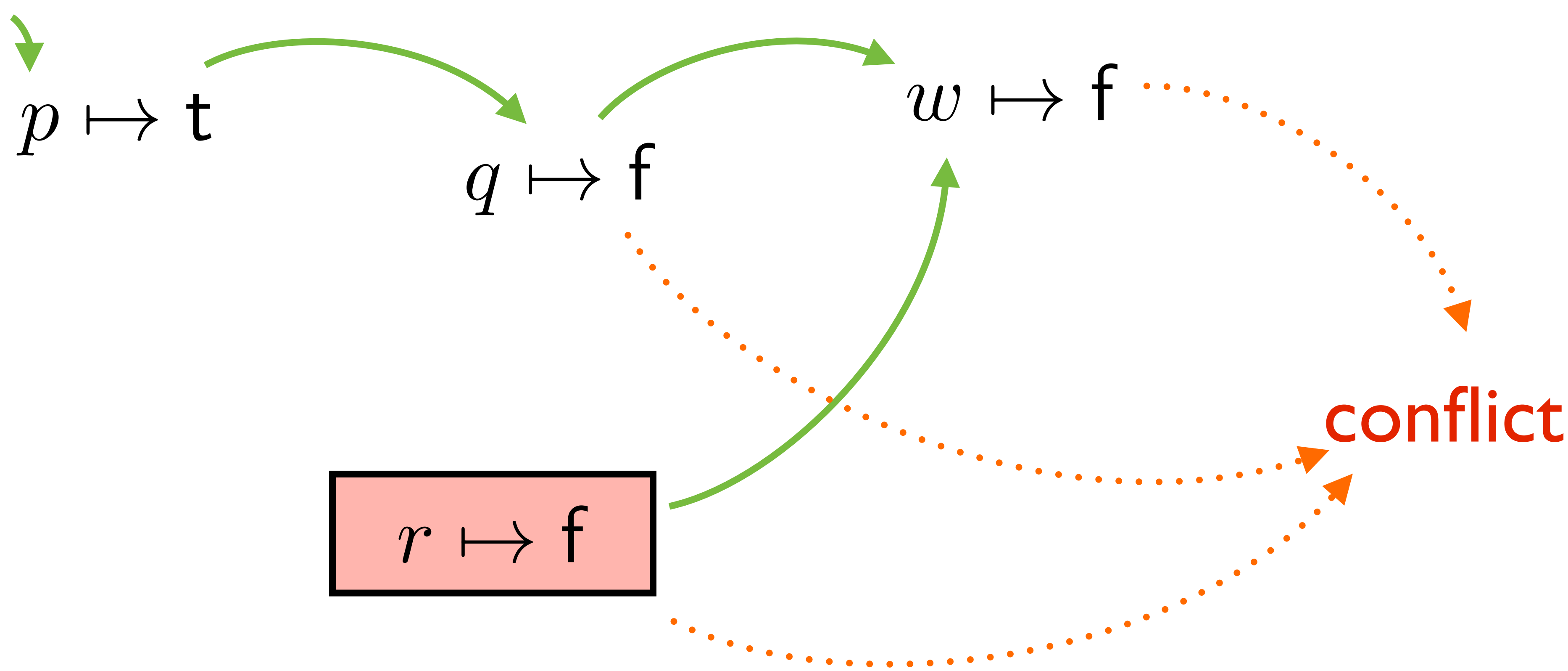


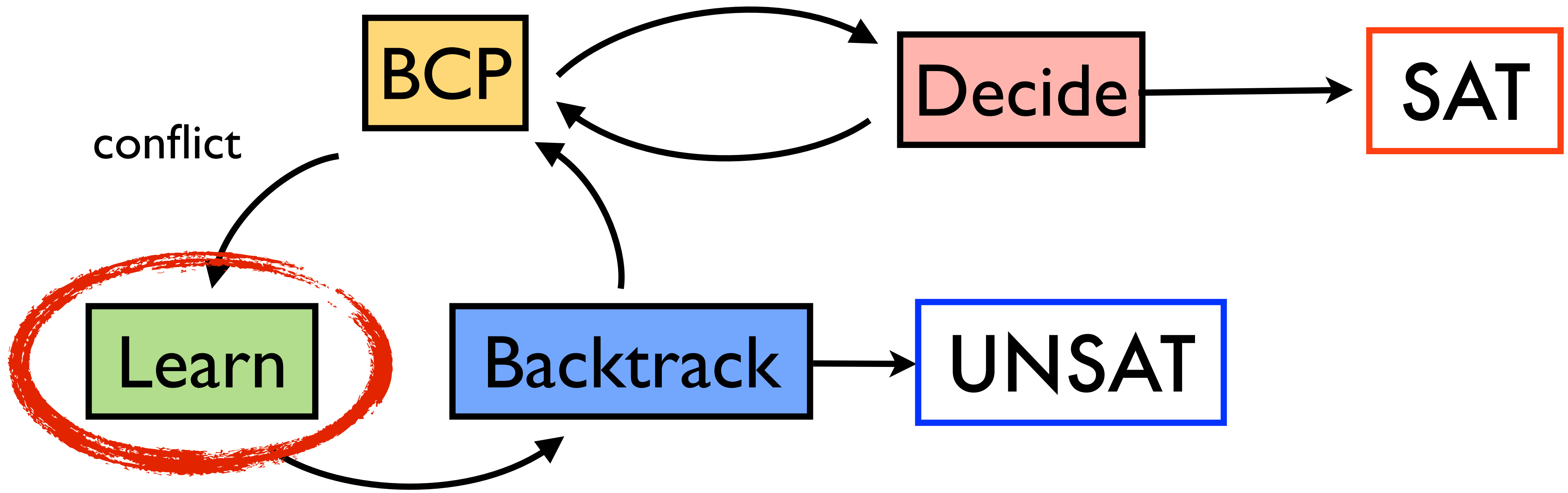

$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$



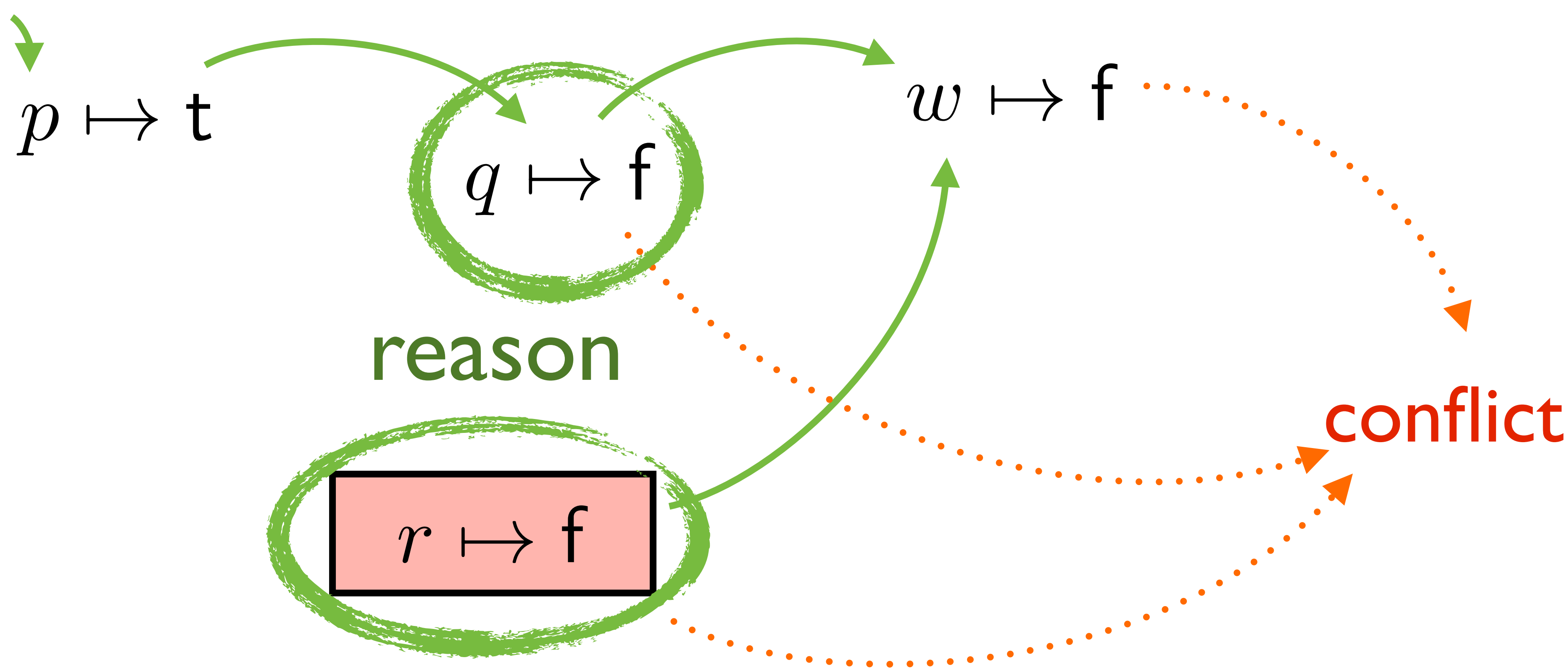


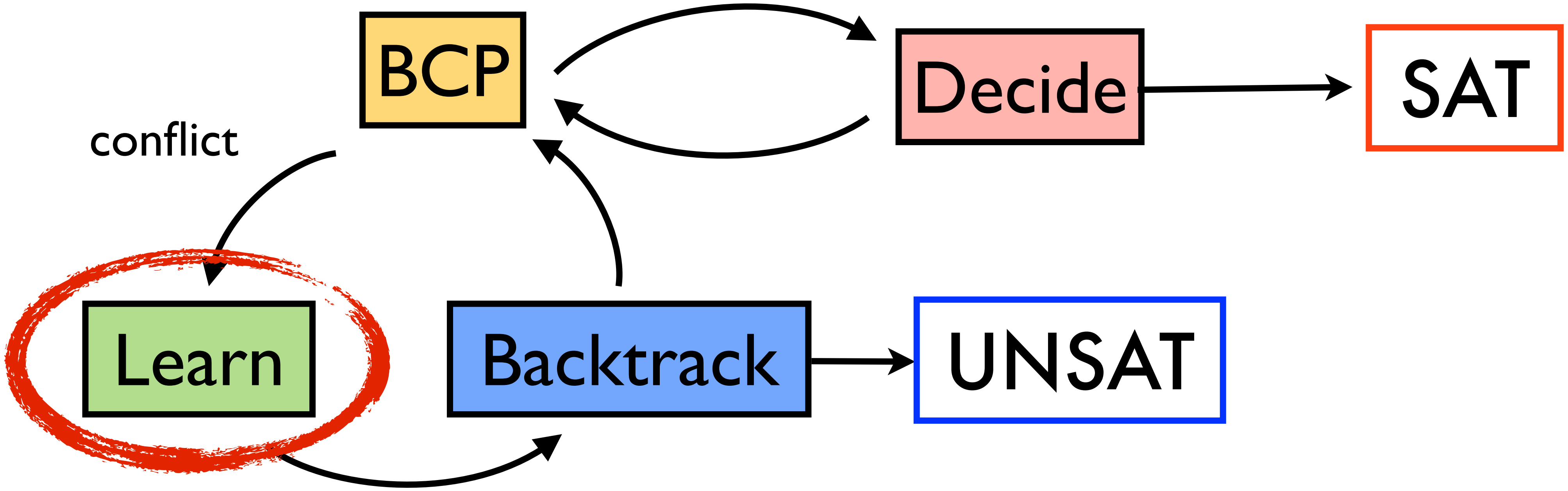

$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$



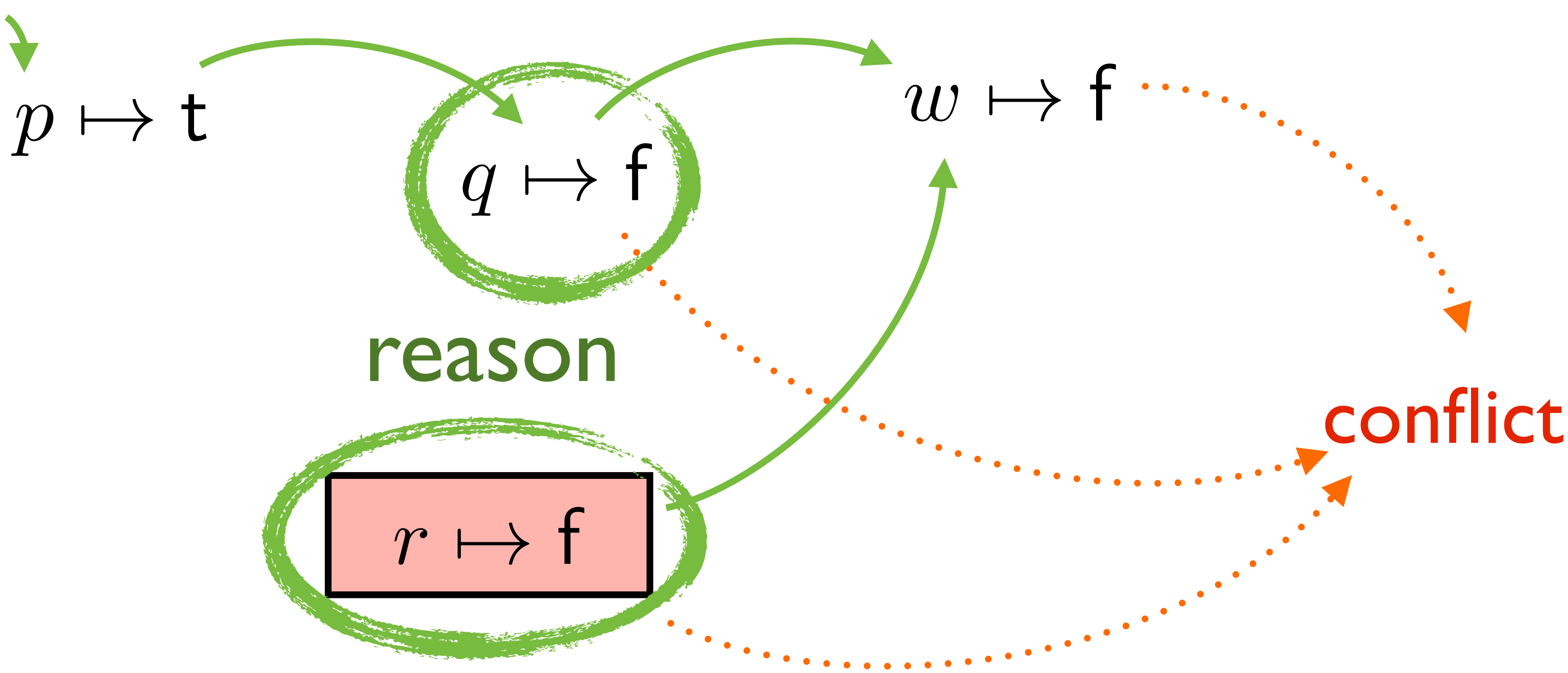



$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$

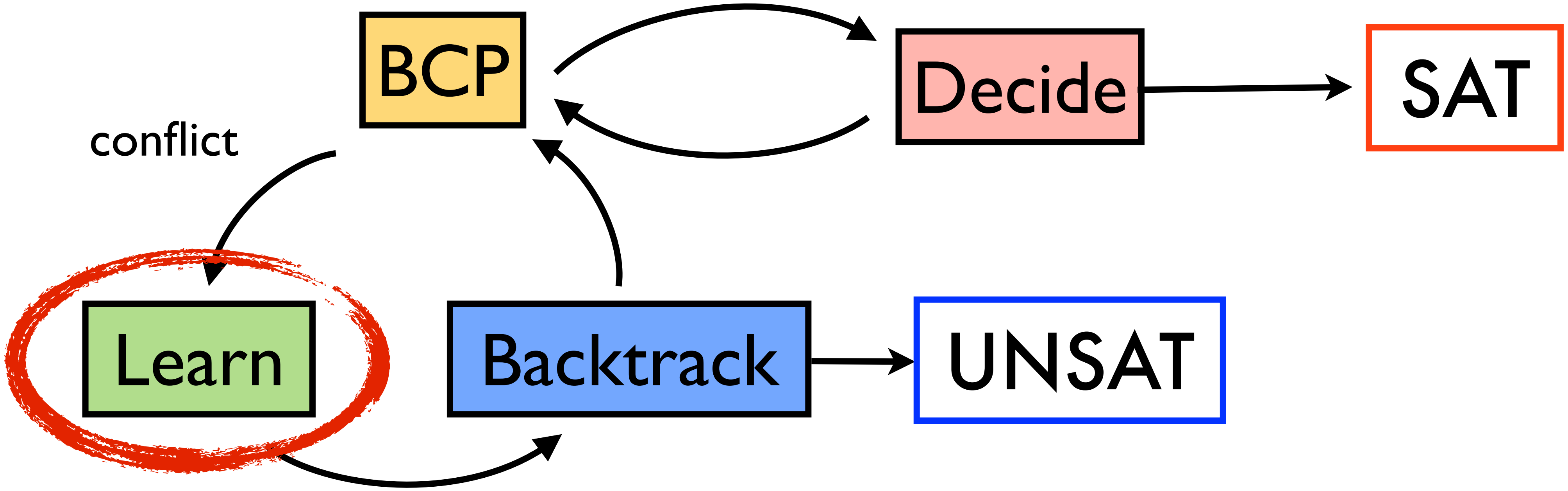




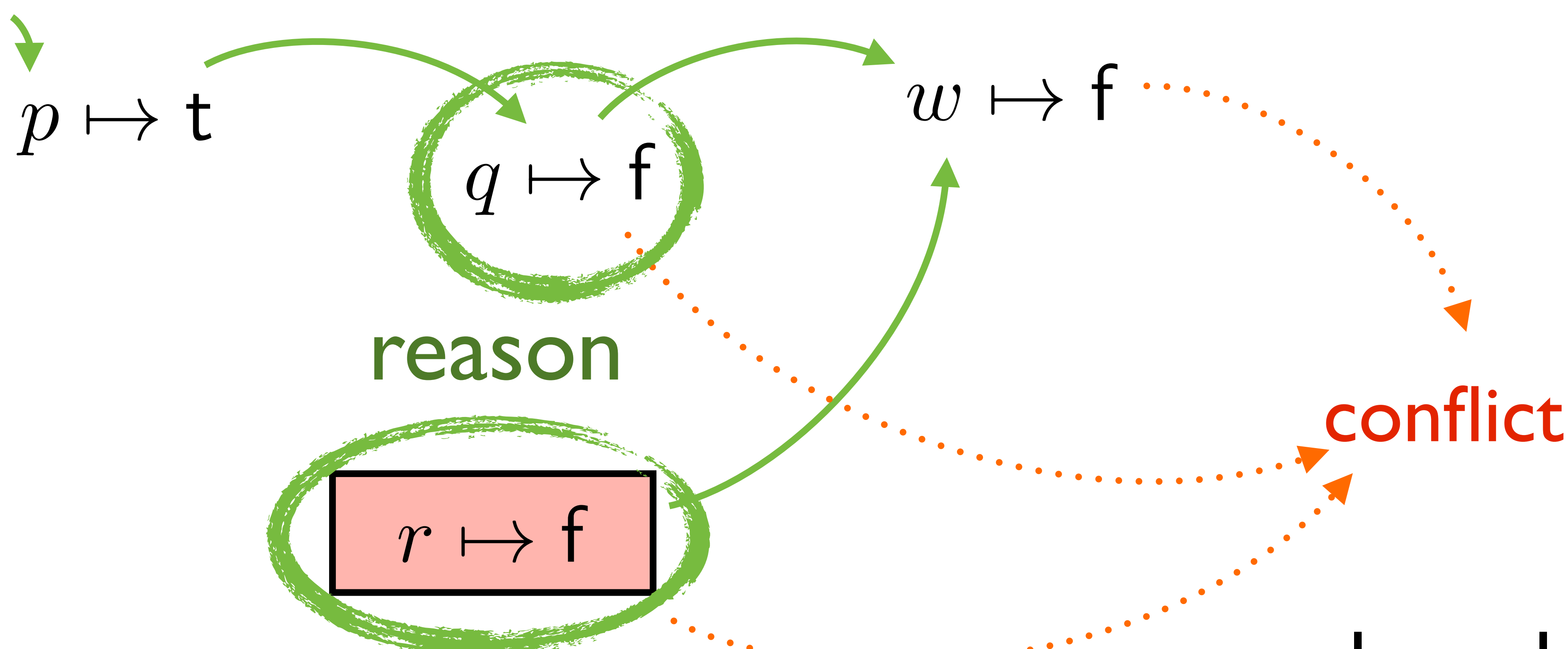

$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$



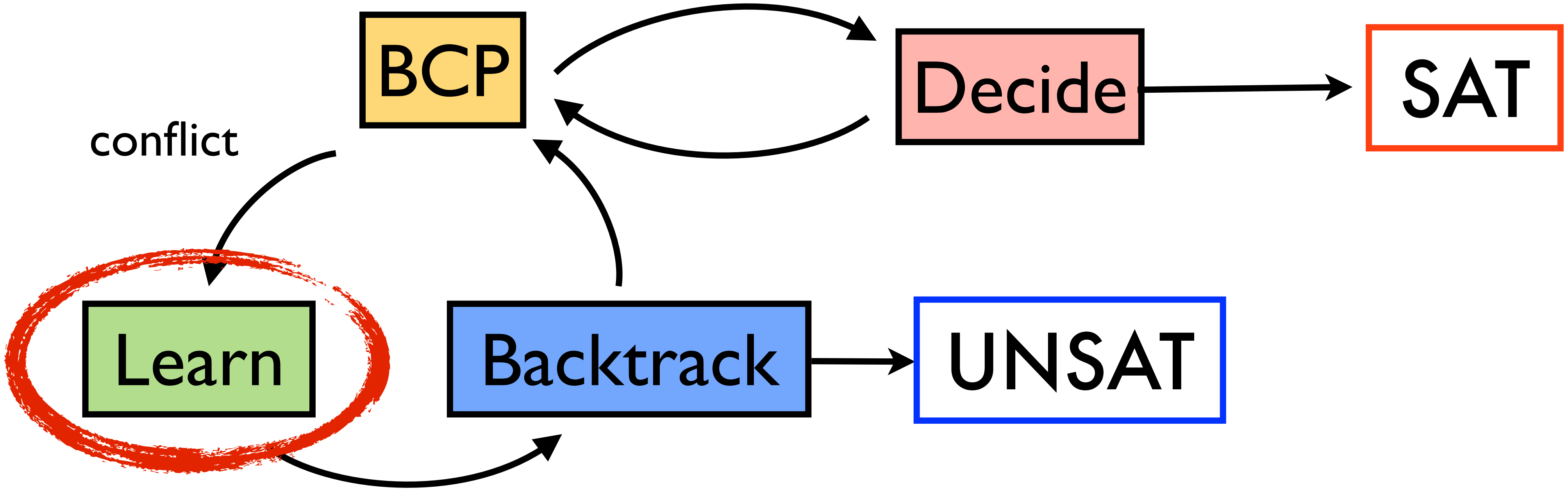

$q \mapsto \mathsf{f}$ and $r \mapsto \mathsf{f}$ is not possible



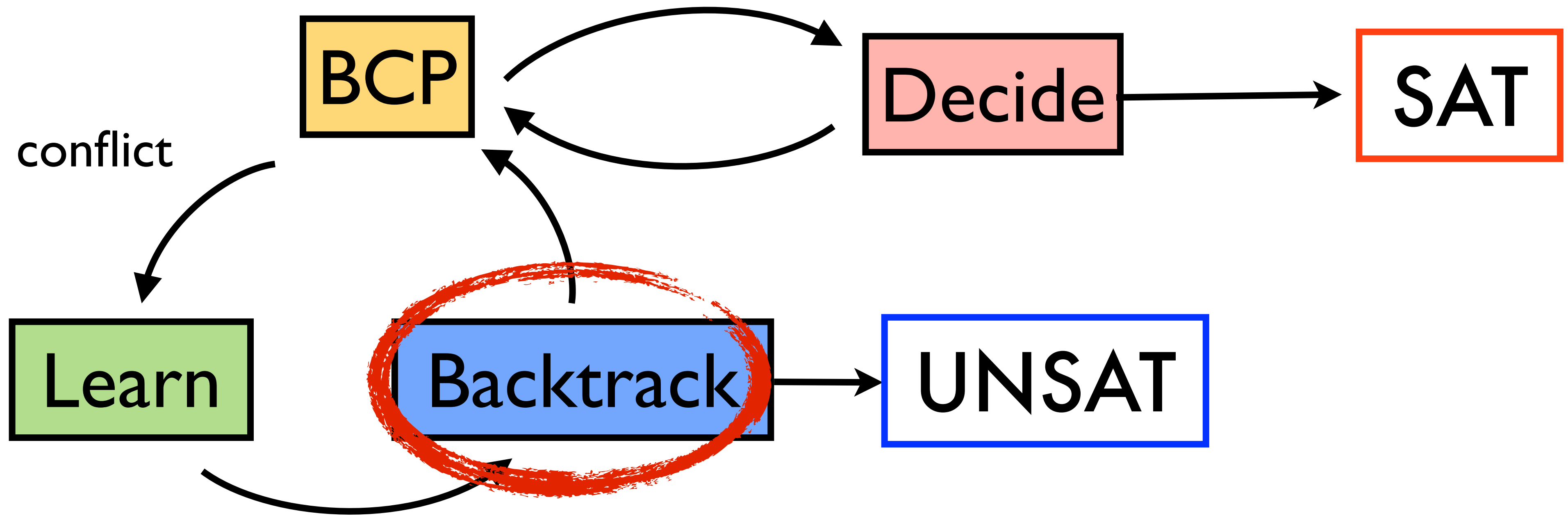

$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w)$$



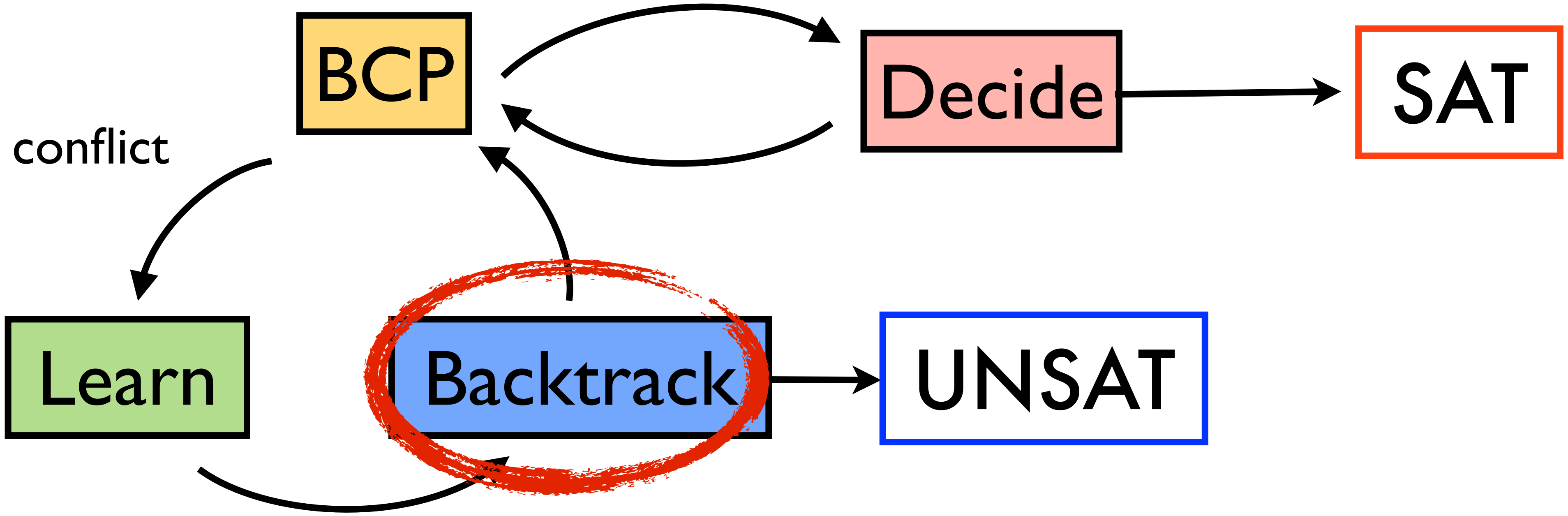

$q \mapsto$ f and $r \mapsto$ f is not possible ⟹ learn lemma $q \vee r$




$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w) \qquad q \vee r$$

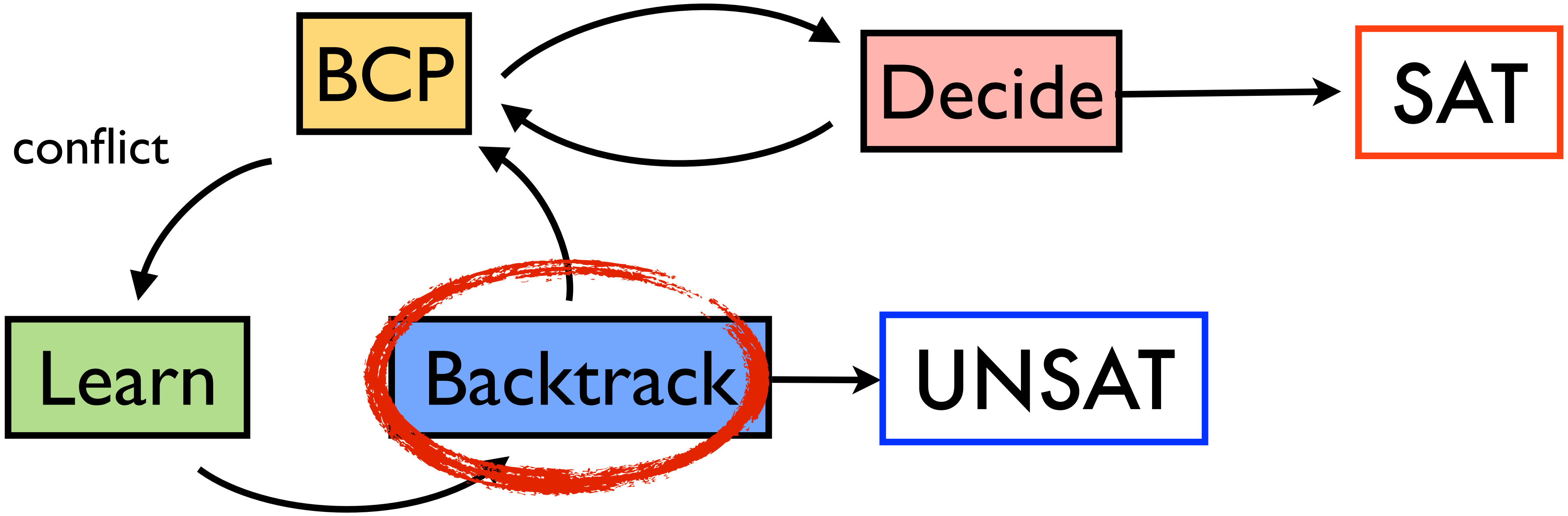



$$\varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w) \land (q \lor r)$$



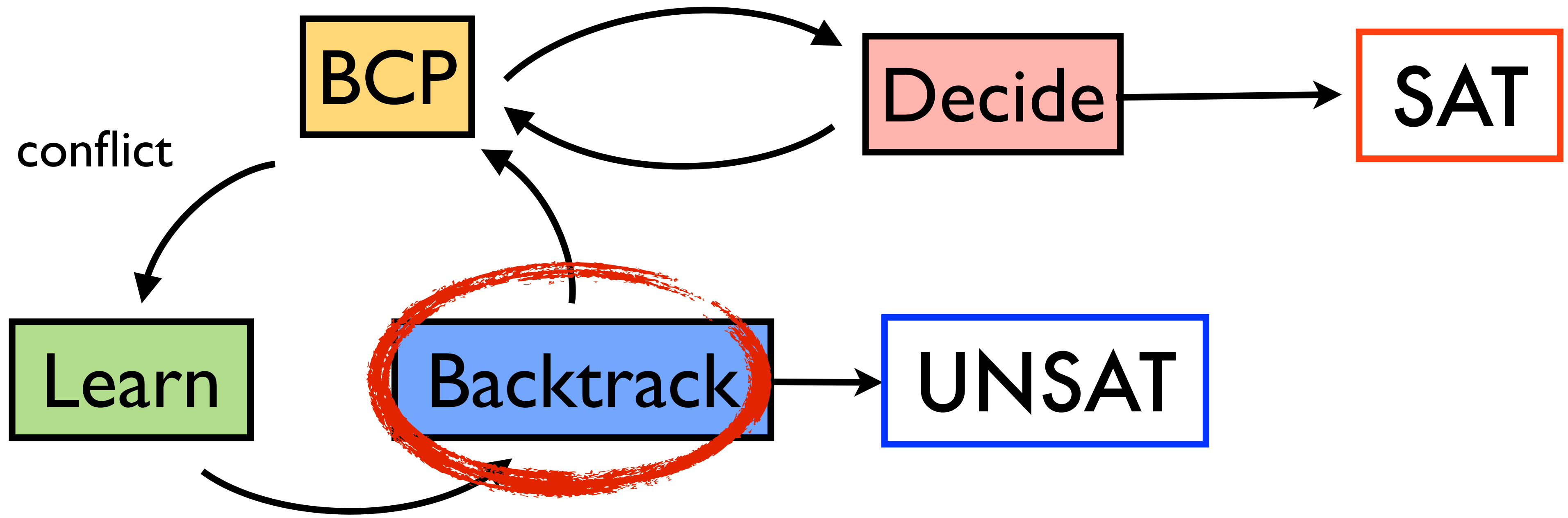

$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w) \wedge (q \vee r)$$

$$p \mapsto \mathsf{t}$$
$$q \mapsto \mathsf{f}$$
$$r \mapsto \mathsf{f}$$
$$w \mapsto \mathsf{f}$$




$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w) \wedge (q \vee r)$$

$$\begin{array}{l} p \mapsto \mathsf{t} \\ q \mapsto \mathsf{f} \\ r \mapsto \mathsf{f} \\ w \mapsto \mathsf{f} \end{array} \quad \Longrightarrow \quad \begin{array}{l} p \mapsto \mathsf{t} \\ q \mapsto \mathsf{f} \end{array}$$




$$\varphi = p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg w) \wedge (q \vee r \vee w) \wedge (q \vee r)$$

| | | |
|---|---|---|
| $p \mapsto$ t<br>$q \mapsto$ f<br>$r \mapsto$ f<br>$w \mapsto$ f | $p \mapsto$ t<br>$q \mapsto$ f | $p \mapsto$ t<br>$q \mapsto$ f<br>$r \mapsto$ t |

# The CDCL Algorithm
## One Line Summaries

BCP and decisions construct an assignment

Learning infers new clauses

Model theoretic search guides proof theoretic search

Important: CDCL is more than case splitting

Conflict Driven Clause Learning

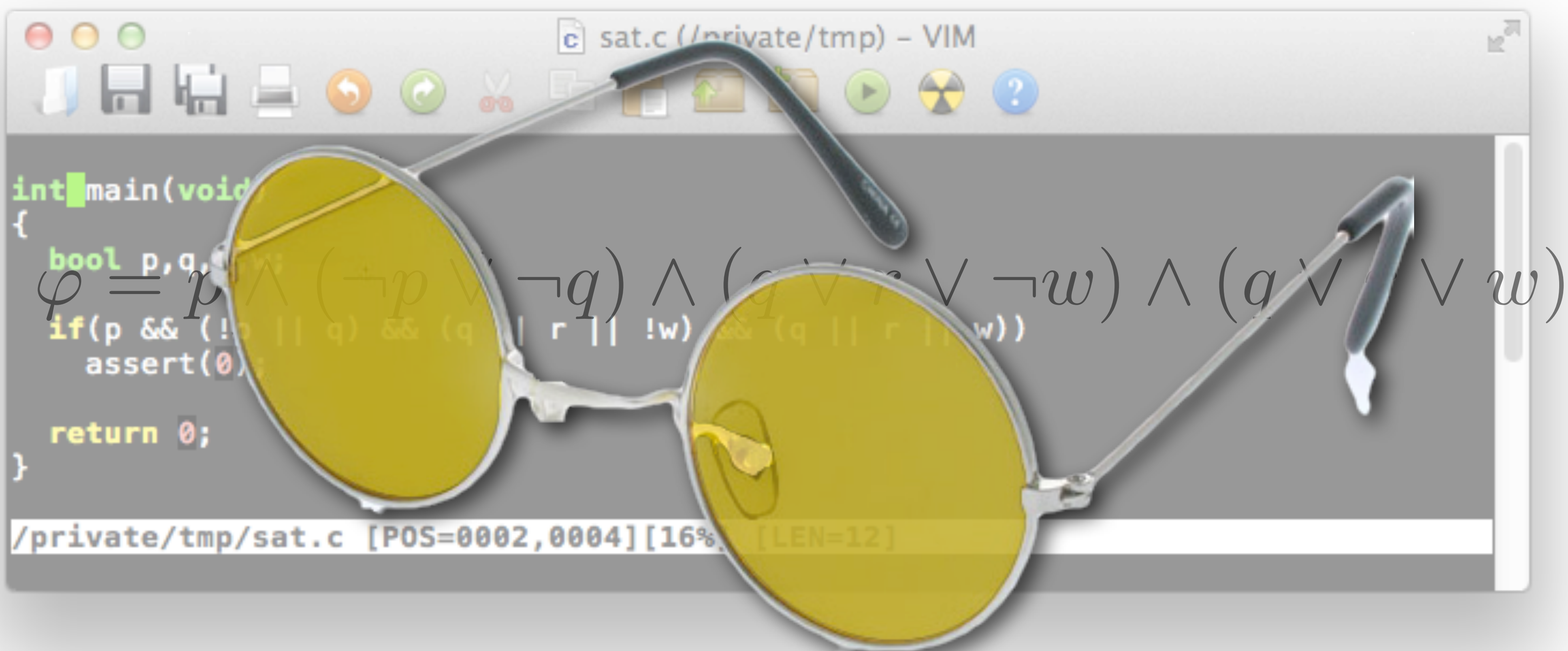
Interpreting Logic

CDCL is Abstract Interpretation

ACDCL(*A*)

$$\varphi = p \land (\neg p \lor \neg q) \land (q \lor r \lor \neg w) \land (q \lor r \lor w)$$

Imagine no assignments,
it’s easy if you try




Imagine only Booleans,
I wonder if you can

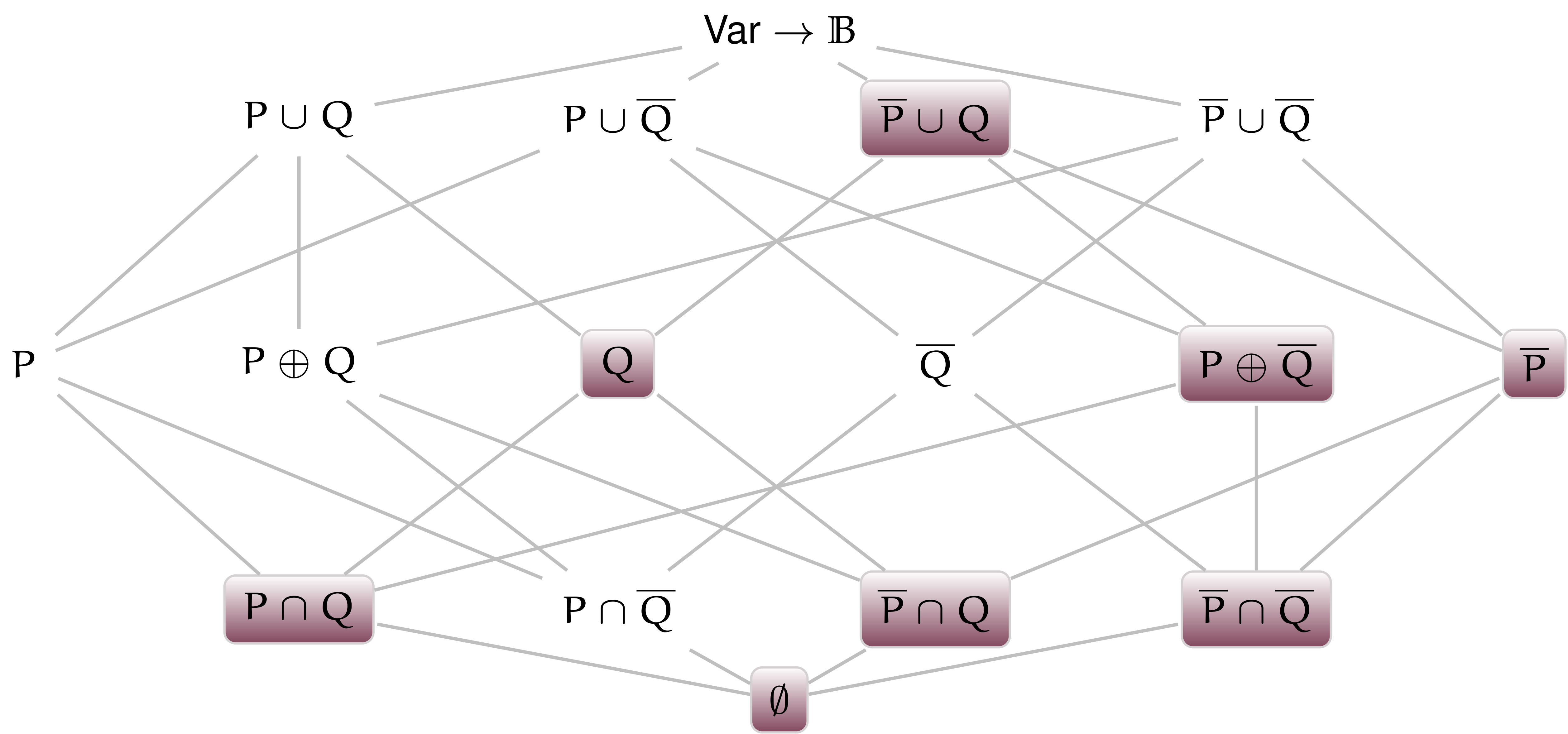
sat.c (/private/tmp) - VIM

```c
int main(void)
{
  bool p,q,r,w;

  if(p && (!p || q) && (q || r || !w) && (q || r || w))
    assert(0);

  return 0;
}
```

/private/tmp/sat.c [POS=0002,0004][16%] [LEN=12]

# Concrete Interpretation

$P = \{\langle p \mapsto \mathsf{t}, q \mapsto \mathsf{t}\rangle, \langle p \mapsto \mathsf{t}, q \mapsto \mathsf{f}\rangle\}$ $Q = \{\langle p \mapsto \mathsf{t}, q \mapsto \mathsf{t}\rangle, \langle p \mapsto \mathsf{f}, q \mapsto \mathsf{t}\rangle\}$




Shaded: Strongest post-condition for `assume(!p || q)`

# Satisfiability as Concrete Analysis

| | |
|---|---|
| $C = \langle \wp(V \to \mathbb{B}), \subseteq, \cap, \cup \rangle$ | Concrete domain |
| $\top = V \to \mathbb{B}$ | All environments |
| $\bot = \emptyset$ | No environment |
| $post_\varphi(X) = \{\varepsilon \in X \mid \varepsilon \text{ satisfies } \varphi\}$ | Strongest post-condition |

## Concrete Satisfiability:

$$\varphi \text{ is satisfiable exactly if } post_\varphi(\top) \neq \emptyset$$

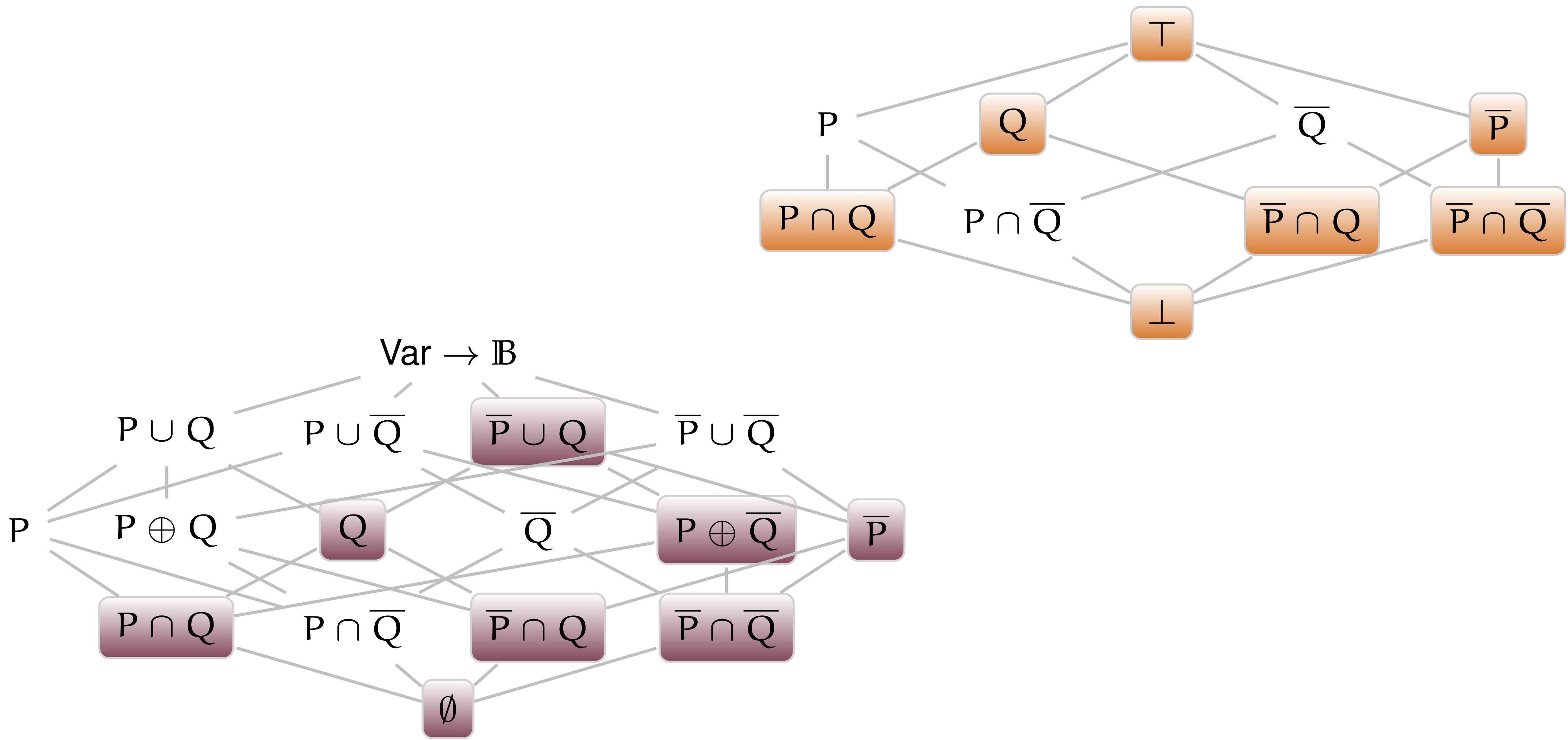
# Cartesian Abstract Domain

$$
\begin{tikzcd}
\wp(V \to \mathbb{B}) \arrow[d, shift right, "\alpha"'] \\
V \to \wp(\mathbb{B}) \arrow[u, shift right, "\gamma"']
\end{tikzcd}
$$

Concrete
Set of environments

Abstract
Environment of sets

# Cartesian Abstract Domain




Shaded: Abstract strongest post-condition for `assume(!p || q)`

# Cartesian Abstract Interpretation

$$C = \langle \wp(V \to \mathbb{B}), \subseteq, \cap, \cup \rangle$$

Concrete domain

$$A = \langle V \to \wp(\mathbb{B}), \sqsubseteq, \sqcap, \sqcup \rangle$$

Abstract domain

$$C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$$

Galois connection

$$apost_\varphi = \alpha \circ post_\varphi \circ \gamma$$

Best abstract transformer

$$P = \{\varepsilon \mid \varepsilon(p) = \mathsf{t}\}$$

$$\alpha(P) = \langle p \mapsto \{\mathsf{t}\}, q \mapsto \mathbb{B}\rangle$$

$$post_{p \wedge q}(\overline{P}) = \emptyset$$

$$apost_{p \wedge q}(\alpha(\overline{P})) = \bot$$

$$post_{p \vee \neg q}(\overline{P}) = \{\langle p \mapsto \mathsf{f}, q \mapsto \mathsf{f}\rangle\}$$

$$apost_{p \vee \neg q}(\alpha(\overline{P})) = \langle p \mapsto \{\mathsf{f}\}, q \mapsto \{\mathsf{f}\}\rangle$$

$$post_{p \,\mathrm{xor}\, q}(\top) = \{\langle p \mapsto \mathsf{f}, q \mapsto \mathsf{t}\rangle \; \langle p \mapsto \mathsf{t}, q \mapsto \mathsf{f}\rangle\}$$

$$apost_{p \,\mathrm{xor}\, q}(\top) = \top$$

# Transformers are sound ...

Computing the best abstract transformer is SAT-hard

Use best abstract transformer only for literals

| | |
|---|---|
| conjunction | meet |
| disjunction | join |

If $apost_{\varphi} = \perp$ then $\varphi$ is unsatisfiable.

(follows from the standard soundness theorem of abstract interpretation)

but they are not complete ...

# ... but not complete

Abbreviate $\langle p \mapsto \{\mathsf{t}\}, q \mapsto \mathbb{B} \rangle$ as $\langle p \mapsto \mathsf{t} \rangle$

$$
\begin{aligned}
\varphi &= p \wedge (\neg p \vee q) \\
apost_\varphi(\top) &= apost_p(\top) \sqcap (apost_{\neg p}(\top) \sqcup apost_q(\top)) \\
&= \langle p \mapsto \mathsf{t} \rangle \sqcap (\langle p \mapsto \mathsf{f} \rangle \sqcup \langle q \mapsto \mathsf{t} \rangle) \\
&= \langle p \mapsto \mathsf{t} \rangle \sqcap \top \\
&= \langle p \mapsto \mathsf{t} \rangle \\
&\neq \\
post_\varphi(\top) &= \{\langle p \mapsto \mathsf{t}, q \mapsto \mathsf{f} \rangle\}
\end{aligned}
$$

# Recovering Precision

**Theorem** (Cousot and Cousot 1979)

$$post(\gamma(a)) \subseteq \gamma(\mathsf{gfp}_x(apost(x \sqcap a))) \subseteq \gamma(apost(a))$$

$$\begin{aligned}
\varphi &= p \wedge (\neg p \vee q) \\
apost_\varphi(\top) &= apost_p(\top) \sqcap (apost_{\neg p}(\top) \sqcup apost_q(\top)) \\
&= \langle p \mapsto \mathsf{t} \rangle \\
apost_\varphi(\langle p \mapsto \mathsf{t} \rangle) &= apost_p(\langle p \mapsto \mathsf{t} \rangle) \sqcap (apost_{\neg p}(\langle p \mapsto \mathsf{t} \rangle) \sqcup apost_q(\langle p \mapsto \mathsf{t} \rangle)) \\
&= \langle p \mapsto \mathsf{t} \rangle \sqcap (\bot \sqcup \langle p \mapsto \mathsf{t}, q \mapsto \mathsf{t} \rangle) \\
&= \langle p \mapsto \mathsf{t} \rangle \sqcap \langle p \mapsto \mathsf{t}, q \mapsto \mathsf{t} \rangle \\
&= \langle p \mapsto \mathsf{t}, q \mapsto \mathsf{t} \rangle
\end{aligned}$$

# Interpreting Logic
## One Line Summaries

Satisfying assignments are fixed points of the semantics

Cartesian abstract interpretation is sound but imprecise

gfp improves precision in the abstract

Conflict Driven Clause Learning

Interpreting Logic

CDCL is Abstract Interpretation

ACDCL(*A*)

# A SAT solver and an abstract interpreter walk into a bar

```
#define l_True  (lbool((uint8_t)0))
#define l_False (lbool((uint8_t)1))
#define l_Undef (lbool((uint8_t)2))

class lbool { [...] };

class Solver {
  [...]
  // FALSE means solver is in a conflicting state
  bool        okay () const;
  vec<lbool>  assigns;  // The current assignments.
  // Enqueue a literal. Assumes value of literal is undefined.
  void        uncheckedEnqueue(Lit p, CRef from = CRef_Undef);
  // Perform unit propagation. Return possibly conflicting clause.
  CRef        propagate ();
};
```

MiniSAT 2.2.0

# Partial assignments

A SAT solver uses partial assignments

$$\begin{array}{ccccc} & & & \mathrm{l\_Undef} & \\ V & \longrightarrow & \mathrm{l\_True} & & \mathrm{l\_False} \\ & & & \neg\,\mathrm{okay} & \end{array}$$

An element of the Cartesian abstraction is:

$$\begin{tikzcd}[column sep=small] & & & \{\mathsf{t},\mathsf{f}\} \arrow[dl, dash] \arrow[dr, dash] & \\ V \arrow[r] & {} & \{\mathsf{f}\} \arrow[dr, dash] & & \{\mathsf{t}\} \arrow[dl, dash] \\ & & & \bot & \end{tikzcd}$$

Partial assignments are order isomorphic to the reduced Cartesian abstraction

# Unit rule

assignment

Clause

$$\mathsf{unit}(\pi, C) = \begin{cases} \text{conflict} & \text{if } \pi \text{ makes all literals in } C \text{ false} \\ \pi[p \mapsto \mathsf{t}] & \text{if } \pi \text{ makes all literals in } C \text{ but } p \text{ false} \\ \pi & \text{otherwise} \end{cases}$$

## Unit rule and abstract transformer

$$h(\mathsf{unit}(\pi, C)) = apost_C(h(\pi))$$

Order isomorphism

**The unit rule is the best abstract transformer**

# BCP

$\mathsf{BCP}(\varphi, \pi)$ {

**repeat**

$\pi' \leftarrow \pi;$

**for** $Clause\ C \in \varphi$ **do** $\pi \leftarrow \mathsf{unit}(C, \pi')$

**until** $\pi' = \pi;$

}

Theorem: BCP as fixed point

$$h(\mathsf{BCP}(\varphi, \pi)) = \mathsf{gfp}_x(apost_\varphi(h(\pi) \sqcap x))$$

BCP is a greatest fixed point

# A SAT solver and an abstract interpreter walk into a bar

```
#define l_True  (lbool((uint8_t)0))
#define l_False (lbool((uint8_t)1))
#define l_Undef (lbool((uint8_t)2))

class lbool { [...] };

class Solver {
   [...]
  // FALSE means solver is in a conflicting state
  bool         okay      () const;
  vec<lbool>   assigns;    // The current assignments.
  // Enqueue a literal. Assumes value of literal is undefined.
  void         uncheckedEnqueue (Lit p, CRef from = CRef_Undef);
  // Perform unit propagation. Return possibly conflicting clause.
  CRef         propagate  ();
};
```

MiniSAT 2.2.0

# Another learning example

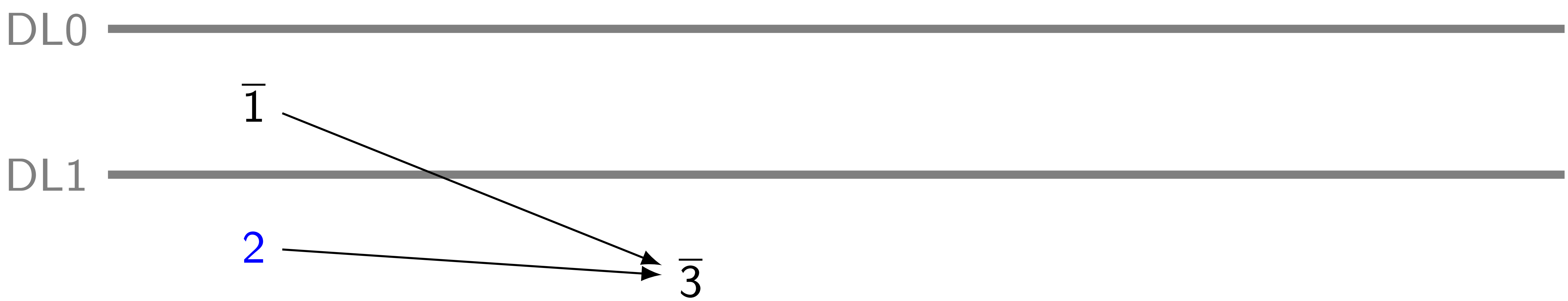
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$

DL0

$\overline{1}$

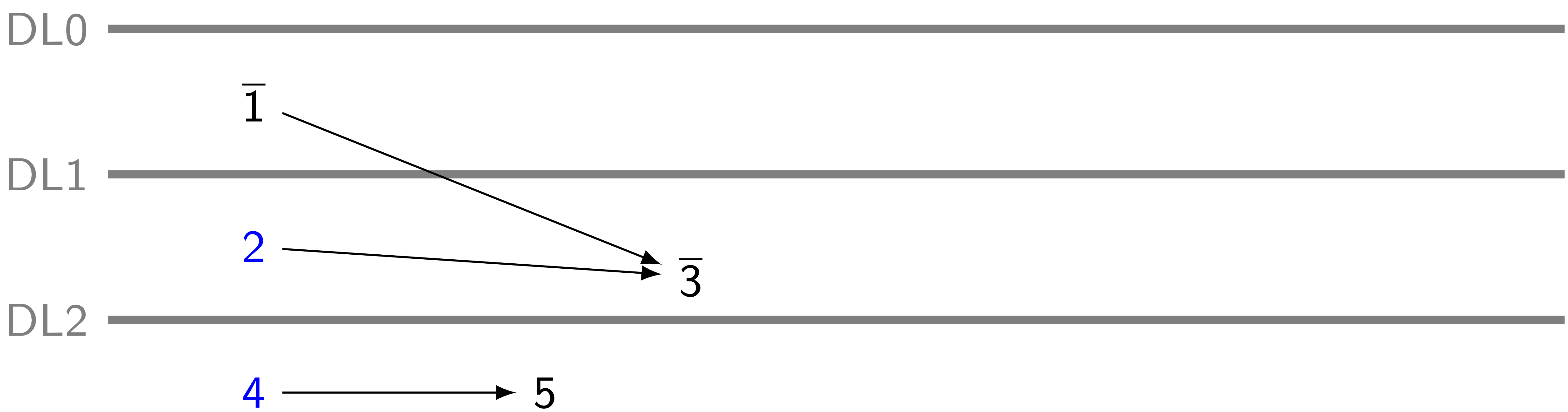
# Another learning example

$$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$$



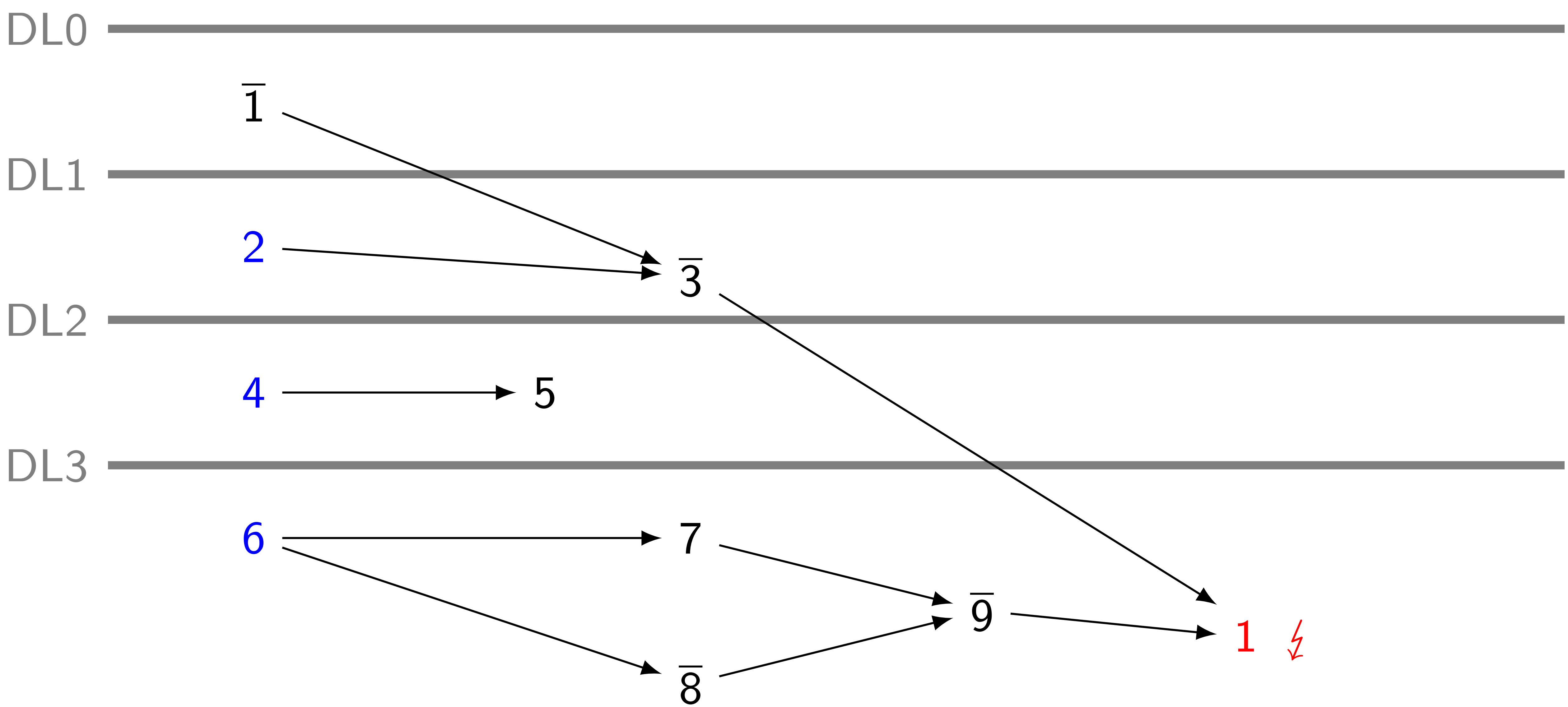
# Another learning example

$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



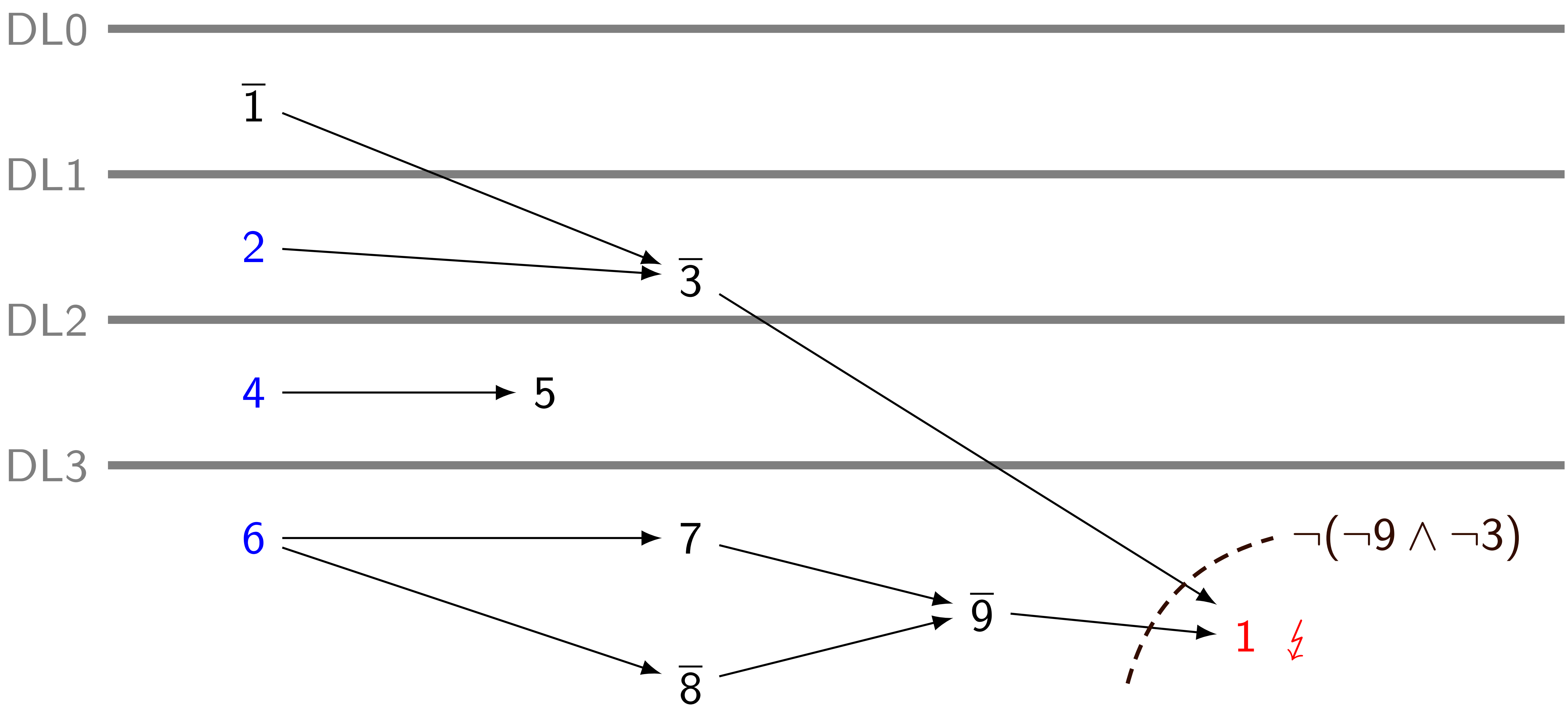
# Another learning example

$$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$$



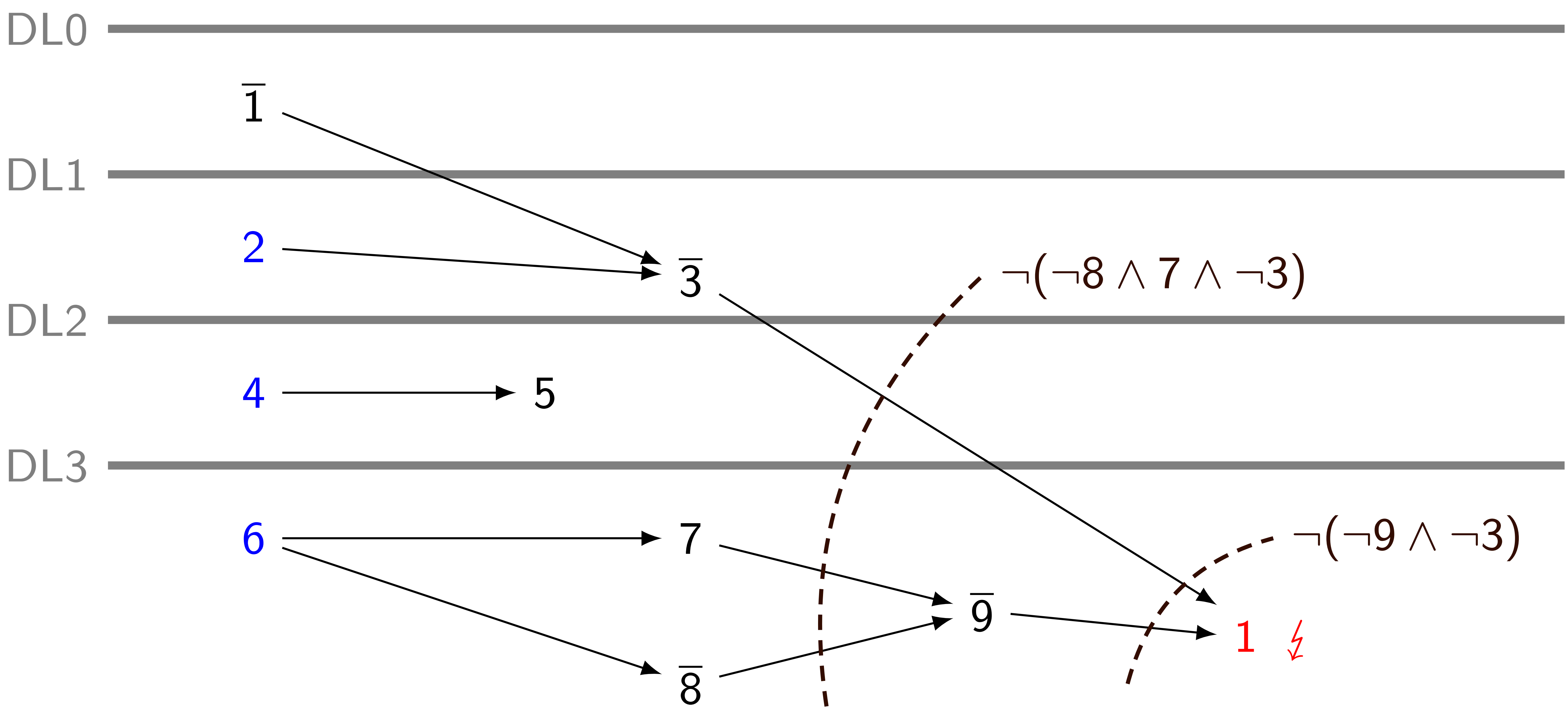
# Another learning example

$$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$$



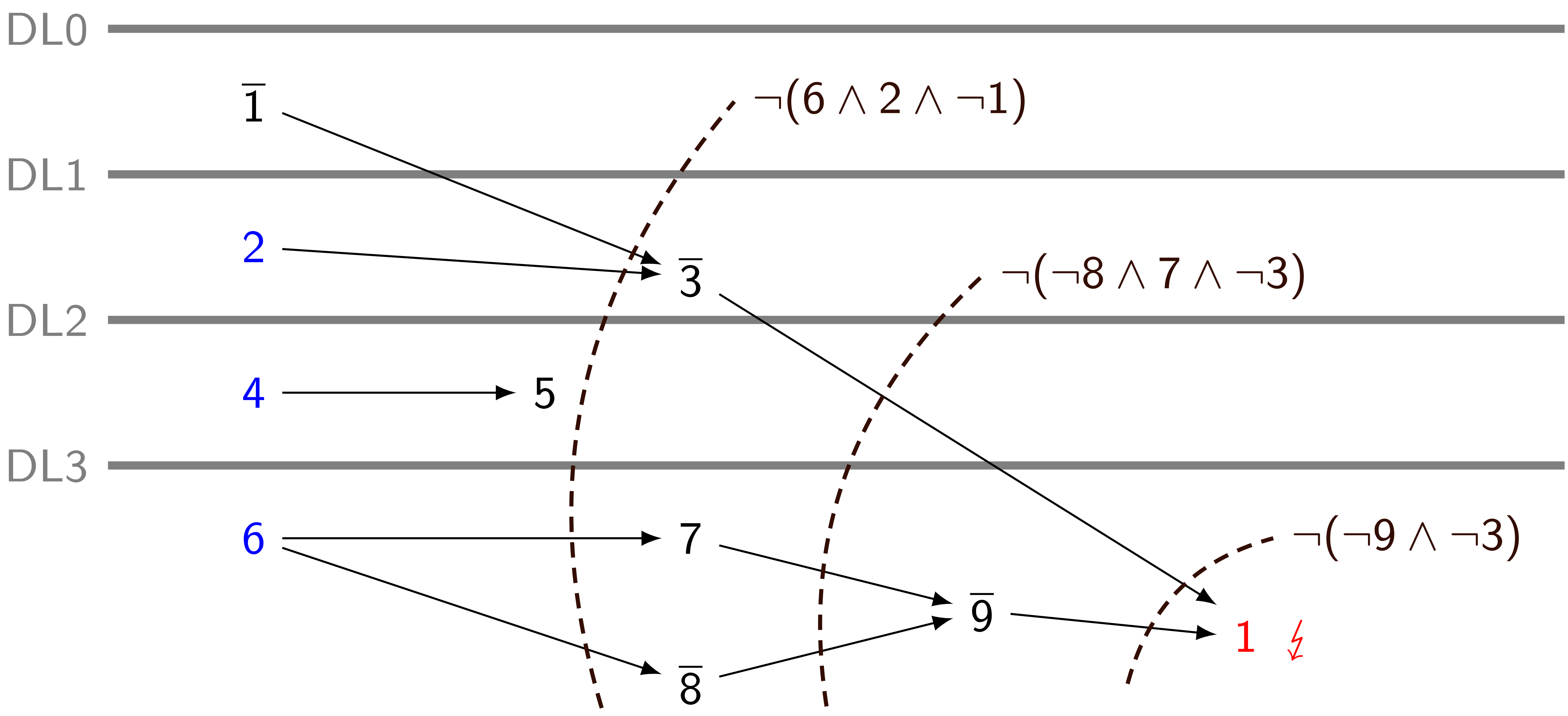
# Another learning example

$$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$$



# Another learning example

$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$

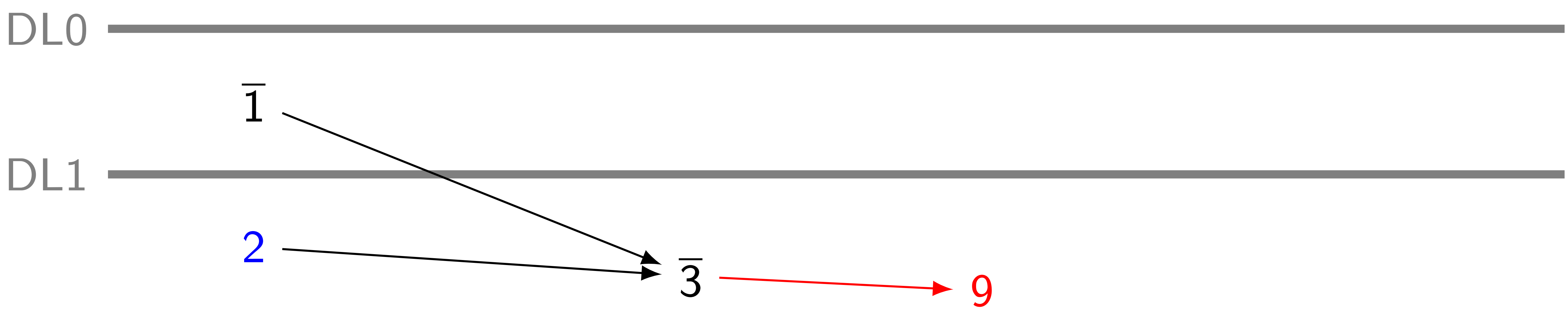



Cuts = Heuristic underapproximation of the weakest pre-condition

# Another learning example

$$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$$

$$\wedge (9 \vee 3)$$



# Trace Partitioning

## (Mauborgne and Rival, 2005)

Domain of Intervals:

$$V \to \mathbb{Z} \times \mathbb{Z}$$

```
int main(void)
{
  int x,y;

  x = y;

  if(x < 5)
    assert(y < 5);

  return 0;
}
```

Analysis too imprecise

Transform program

```
int main(void)
{
  int x,y;

  if(y < 5)
  {
    x = y;

    if(x < 5)
      assert(y < 5);
  }
  else
  {
    x = y;

    if(x < 5)
      assert(y < 5);
  }


  return 0;
}
```

Same analysis is precise

Changing the equation allows one to prove more with the same analysis.

Instance of a power domain (Cousot and Cousot, 1979)

# Learning in SAT

```
if( phi )
  assert(0)
```

Transform formula

Conflict reason

```
if(p = true && q = true)
{
  if( phi )
    assert(0);
}
else
if( !p || !q )
{
  if( phi )
    assert(0);
}
```

Safe

Learned clause

Decisions and learning are dynamic “trace” partitioning

# Learning in SAT

```
if( phi )
  assert(0)
```

Transform formula

Conflict reason

```
if(p = true && q = true)
{
  if( phi )
    assert(0);
}
else
if( !p || !q )
{
  if( phi )
    assert(0);
}
```

Safe

Learned clause

Decisions and learning are dynamic “trace” partitioning

# CDCL is Abstract Interpretation

## One Line Summaries

CDCL implements the Cartesian abstract domain as its main data structure

The unit rule is the application of the best abstract clause transformer

BCP is fixed point computation

Decisions & Learning are discovery of trace partitions

# CDCL is Abstract Interpretation
## Summary of Summaries

| | | | |
|---|---|---|---|
| CDCL = | Partial assignments | = | Cartesian abstract domain |
| | + Unit rule & BCP | | + Abstract transformer & GFP |
| | + Decisions & Learning | | + Trace partitioning |

Not an <u>ANALOGY</u> but an <u>ISOMORPHISM</u>

Precise results using a strict abstraction!

Conflict Driven Clause Learning

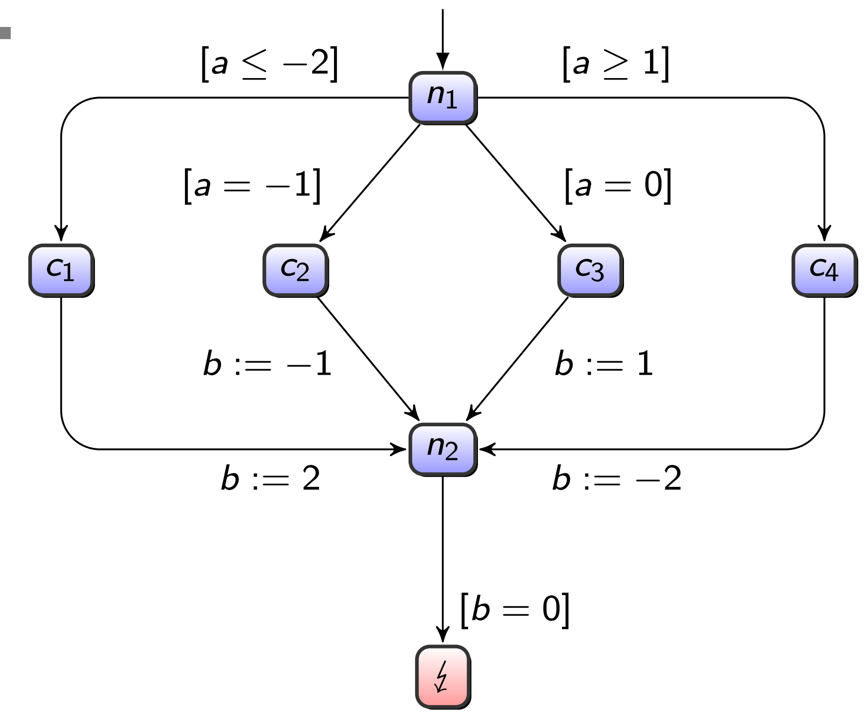
Interpreting Logic

CDCL is Abstract Interpretation

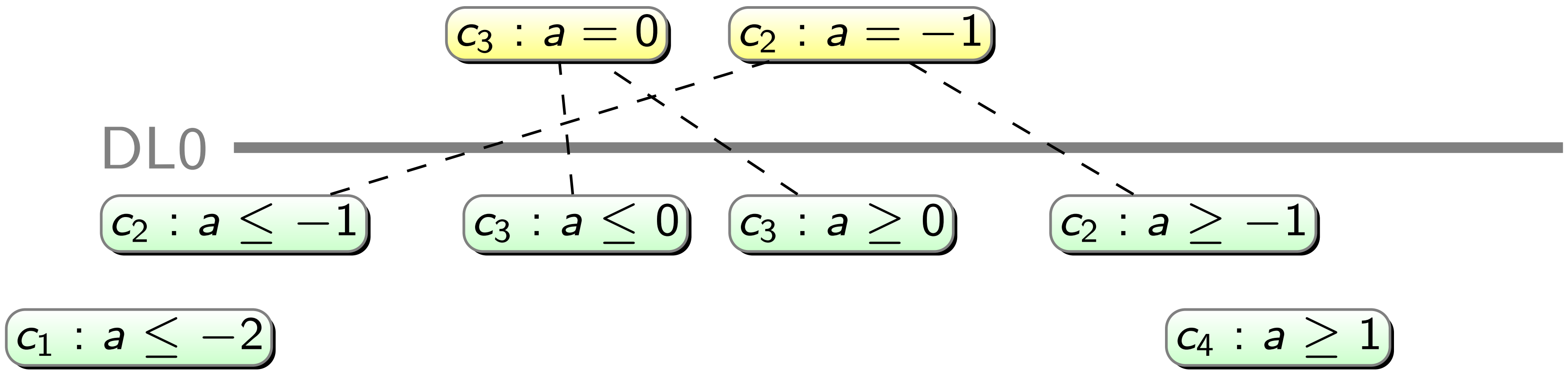
<u>ACDCL(*A*)</u>

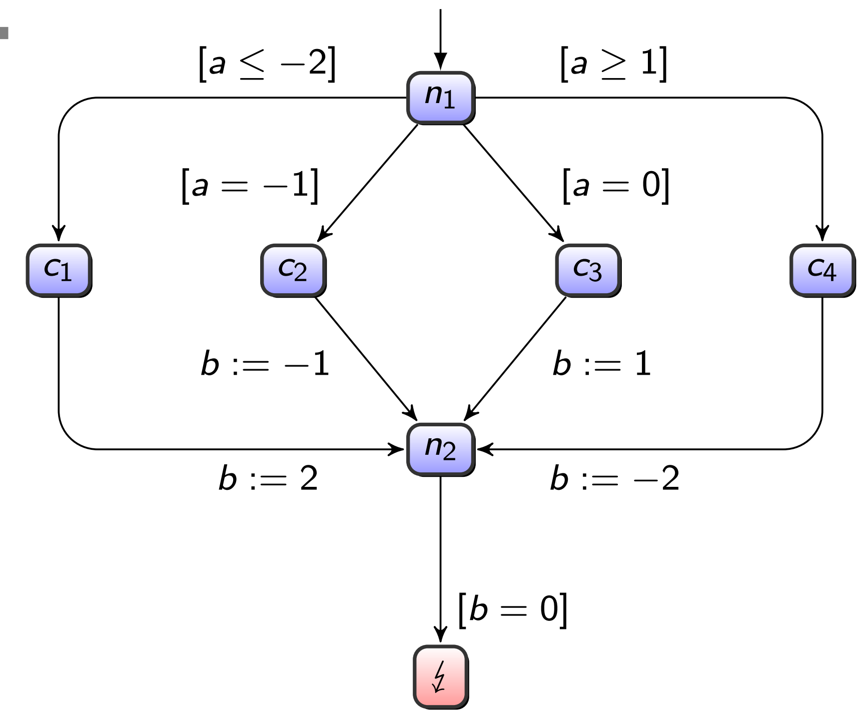
# What about programs?

DL0

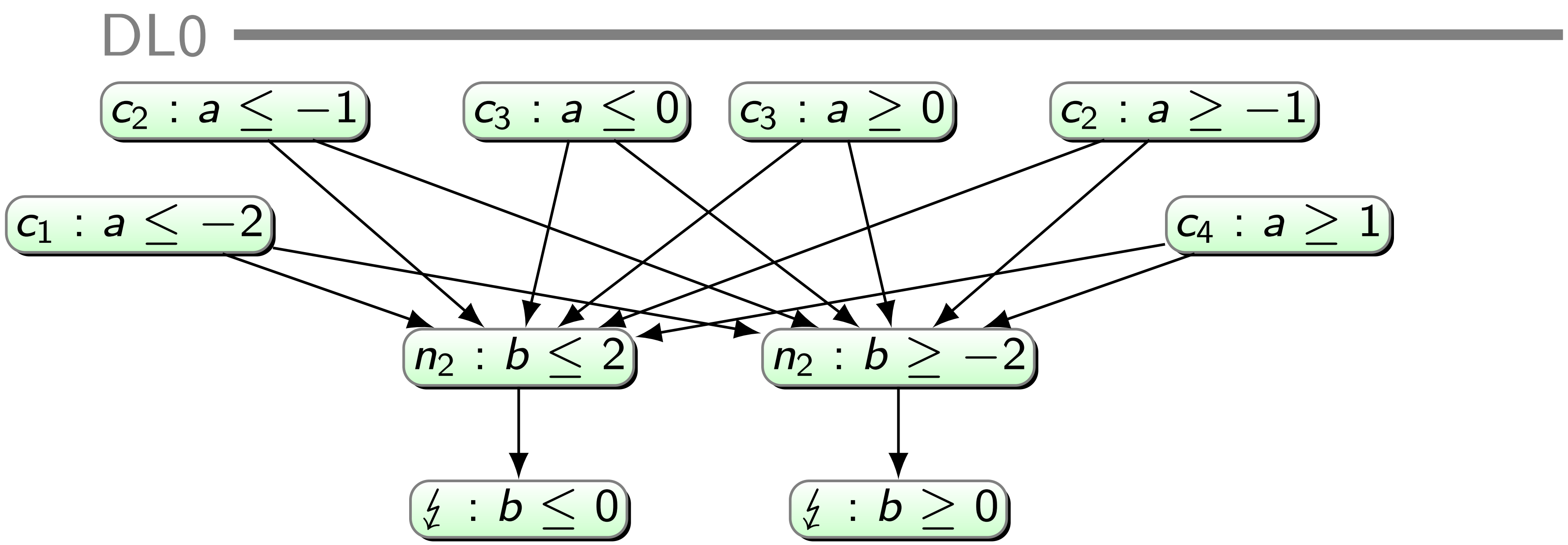


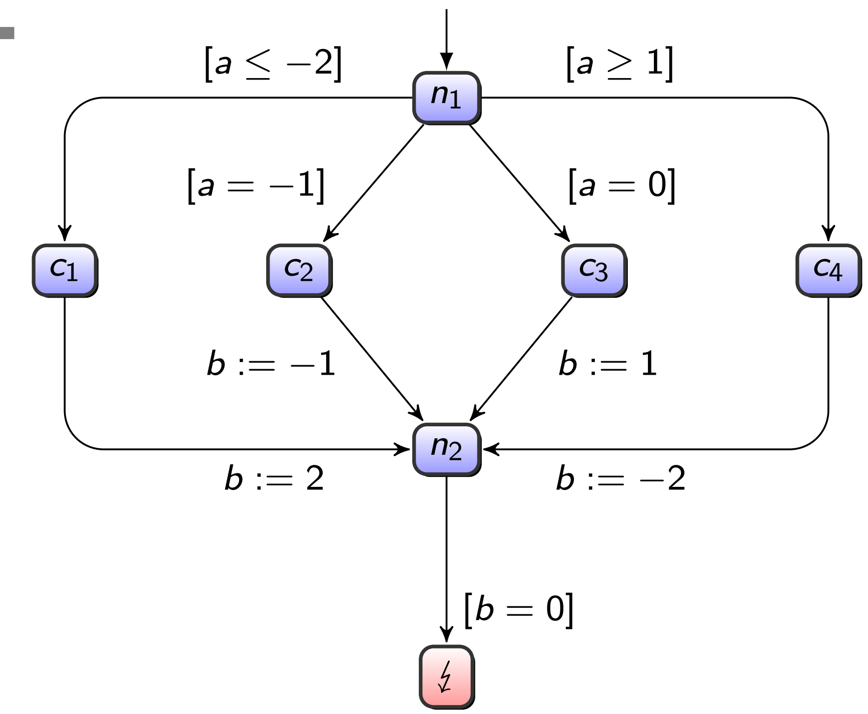
# What about programs?



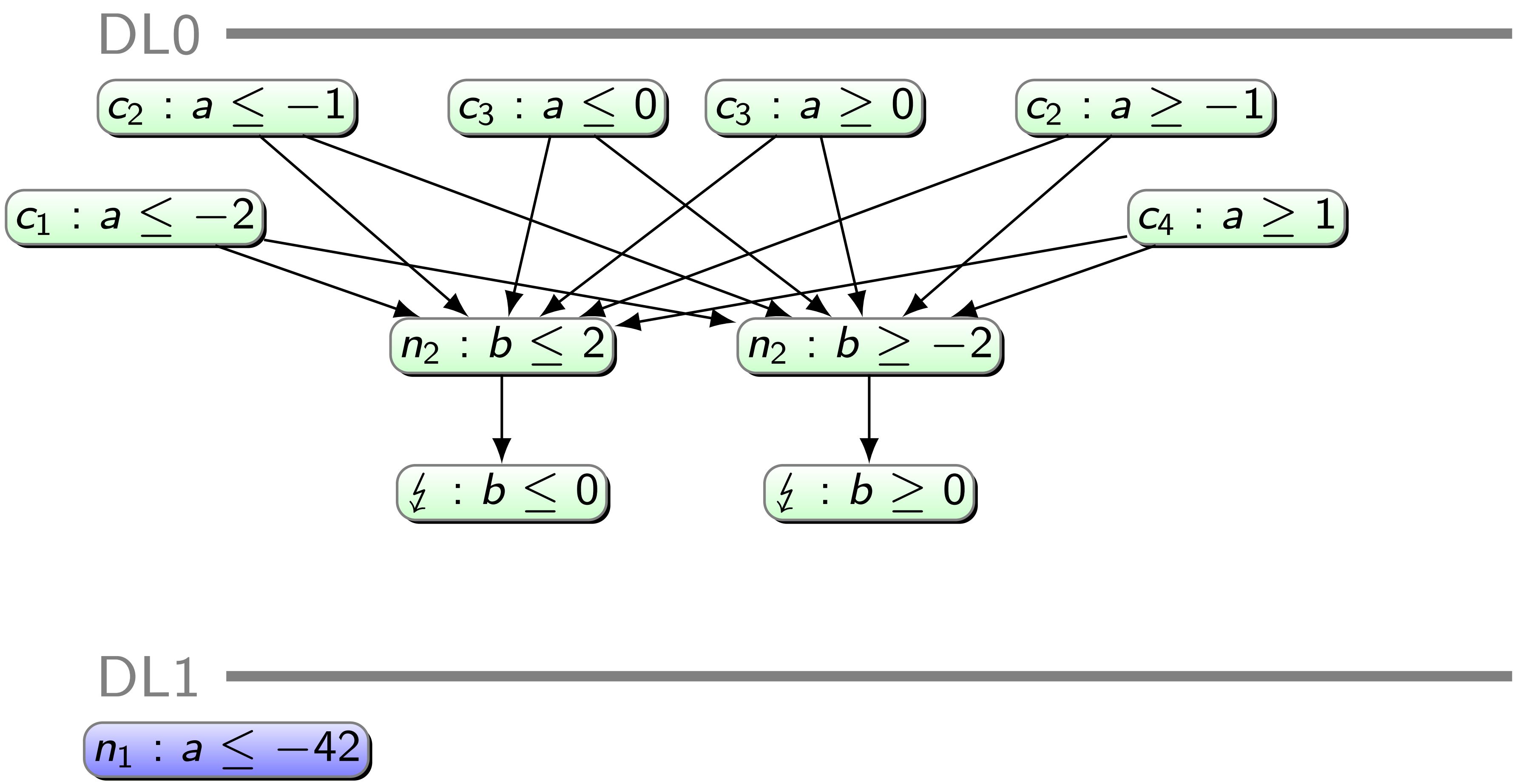



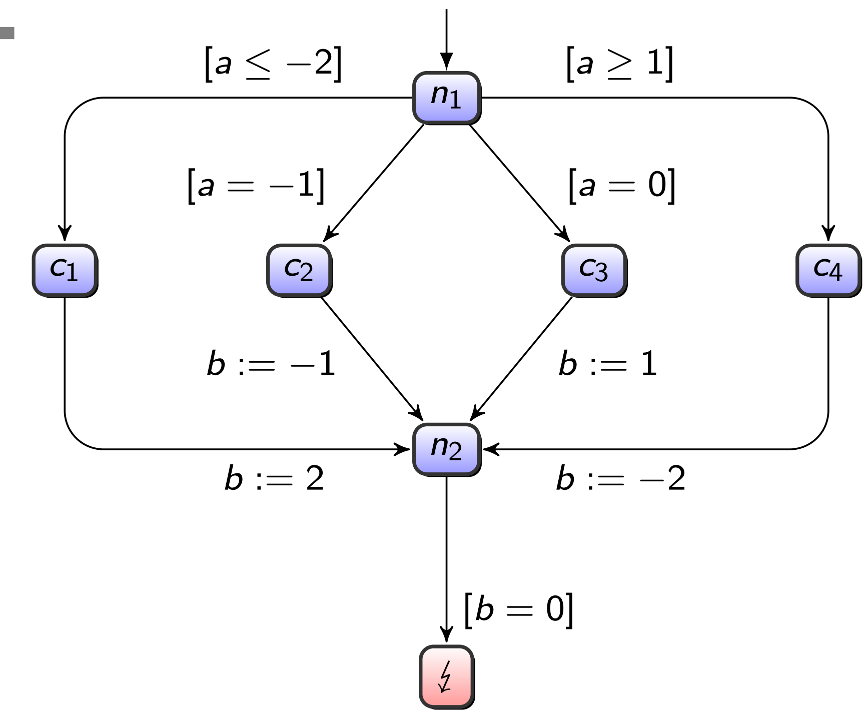
# What about programs?



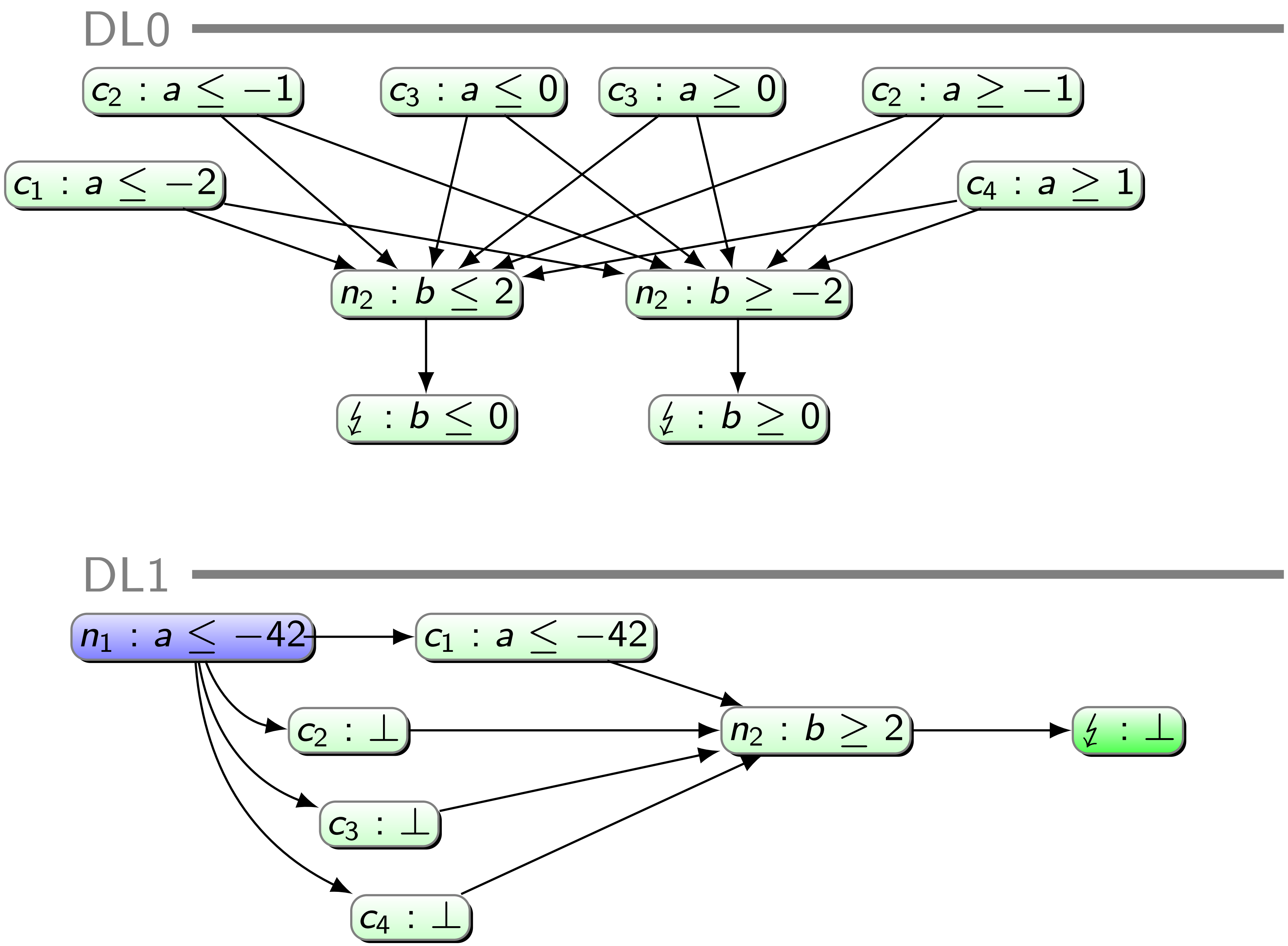



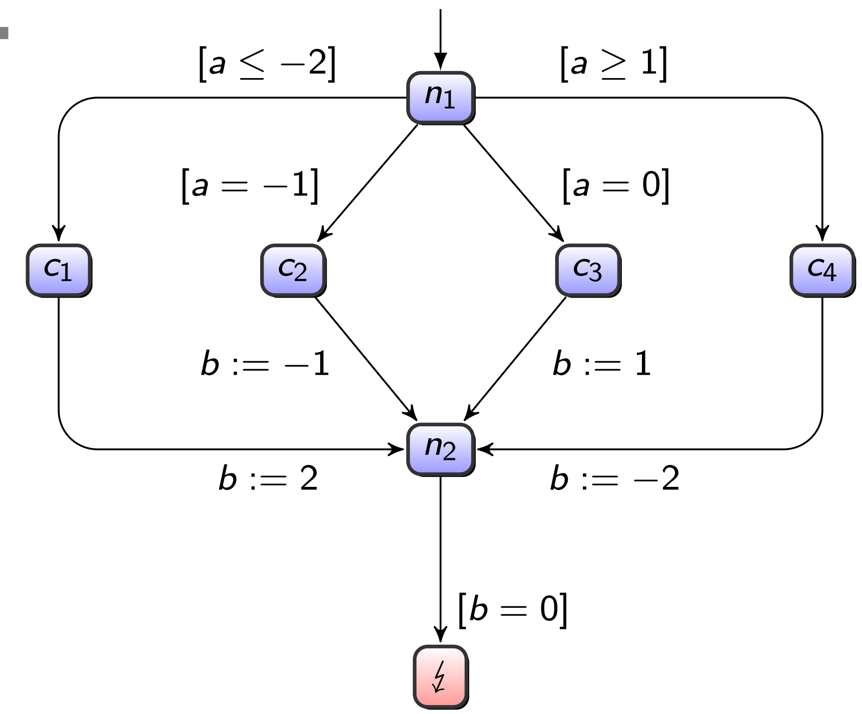
# What about programs?



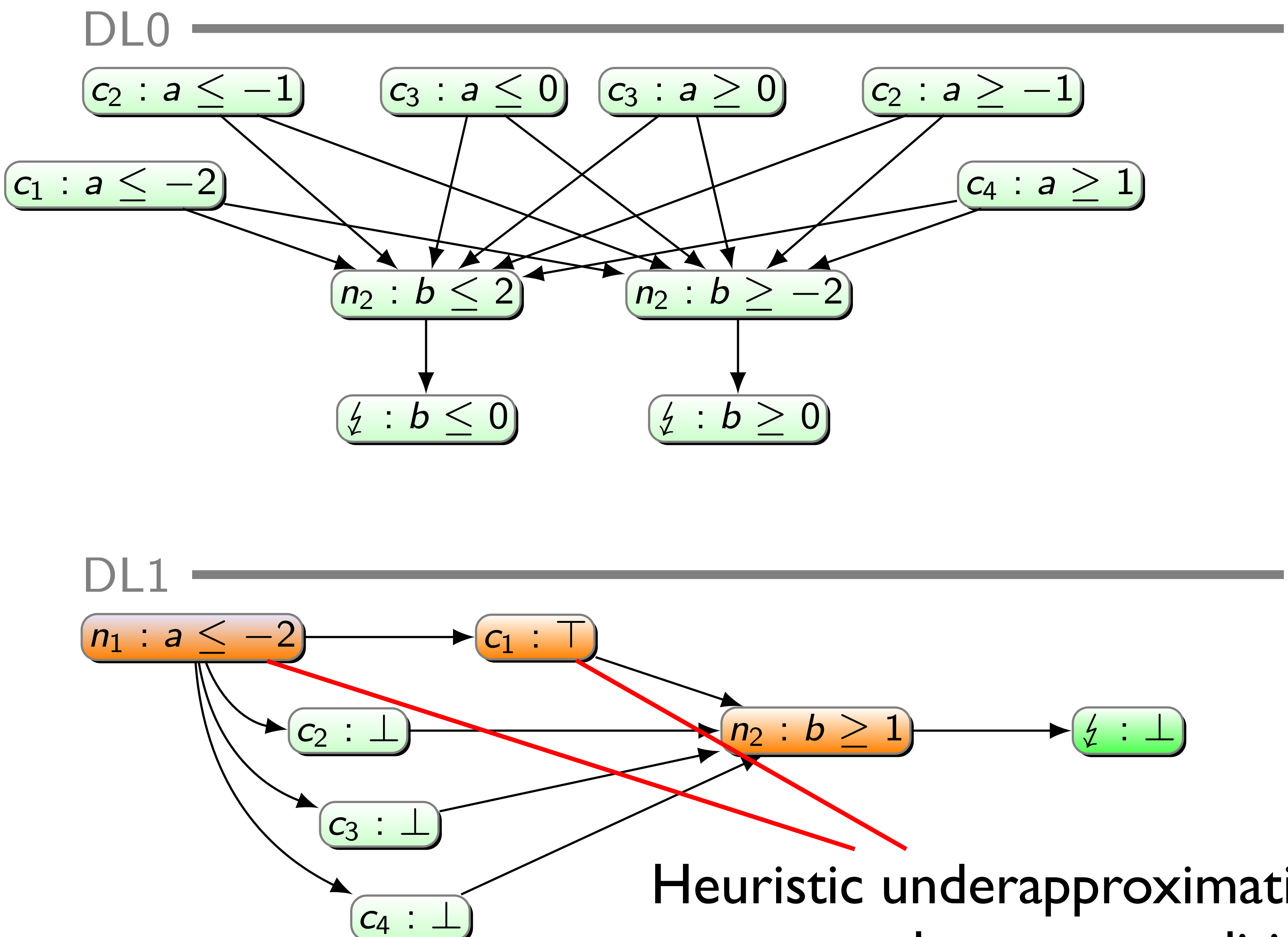
# What about programs?

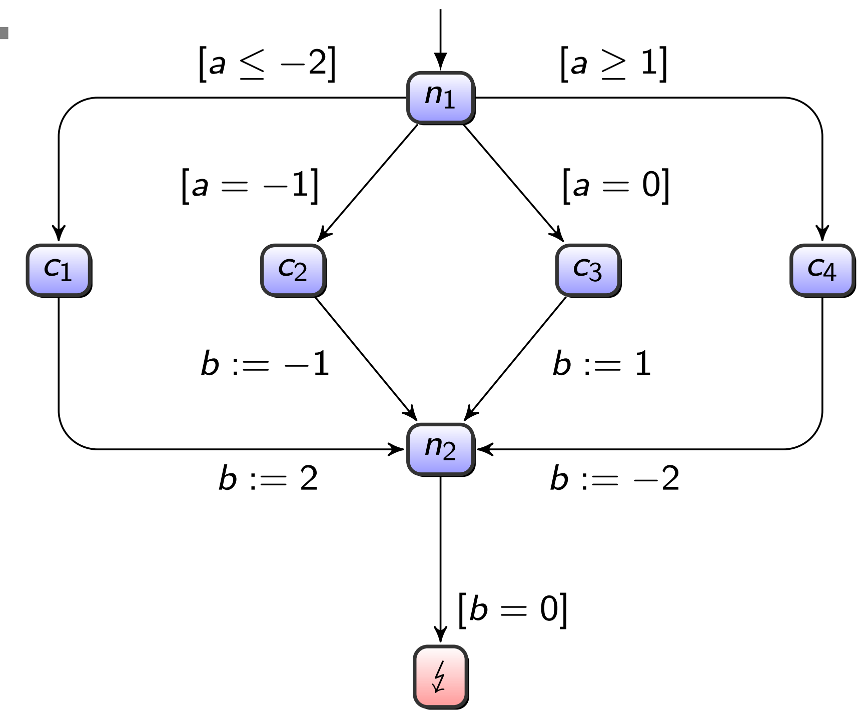



SAFE

# What about programs?



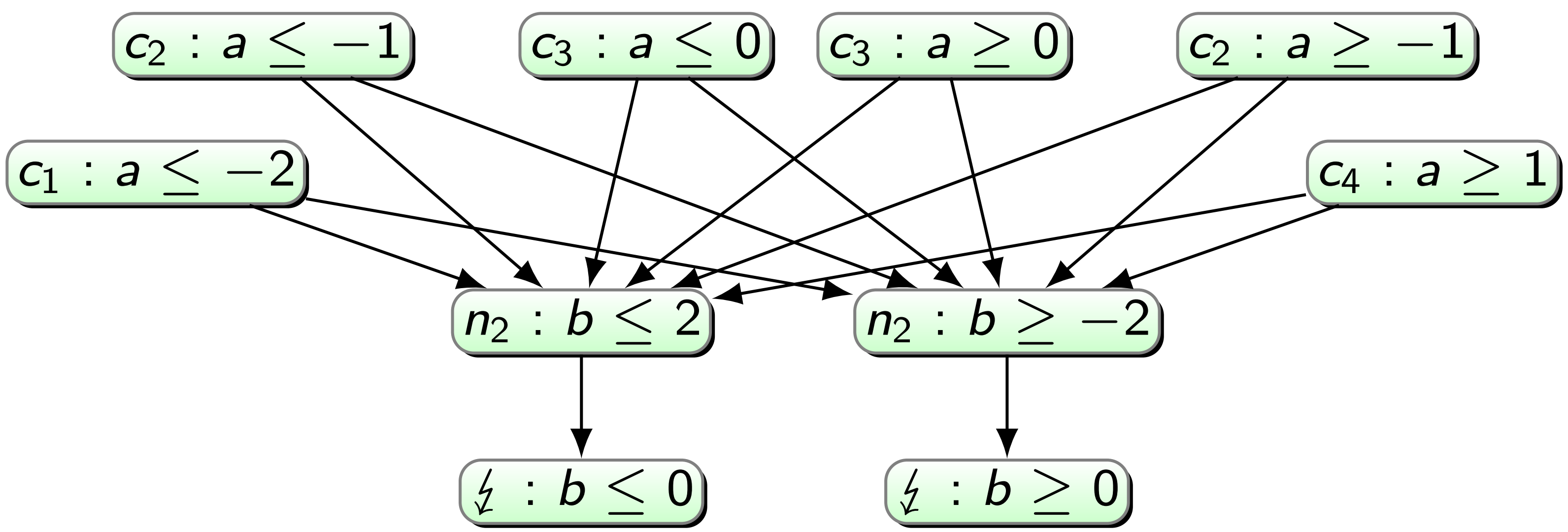



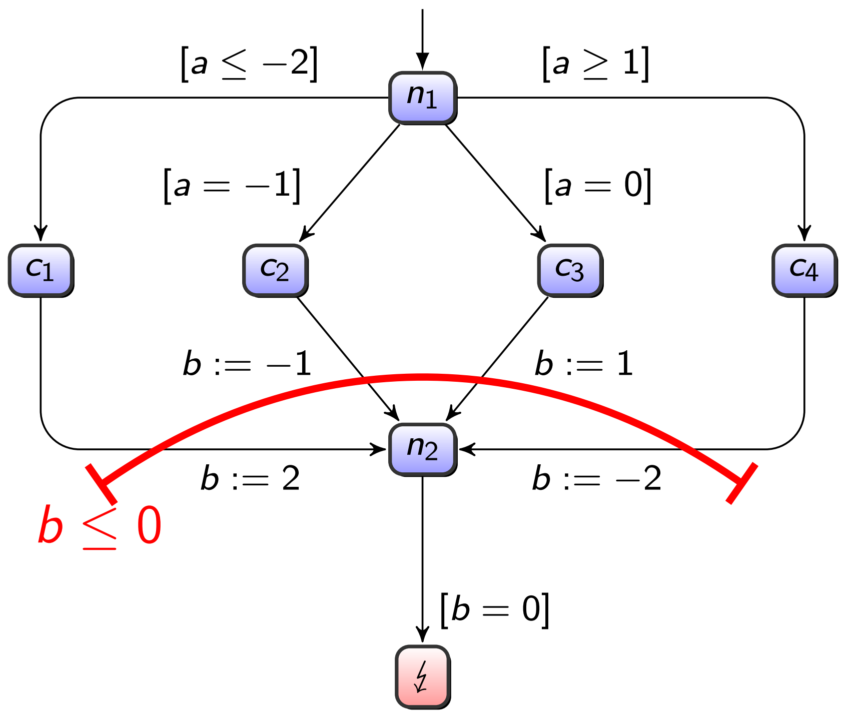

SAFE $\rightarrow$ Generalise!

Heuristic underapproximation of the weakest pre-condition

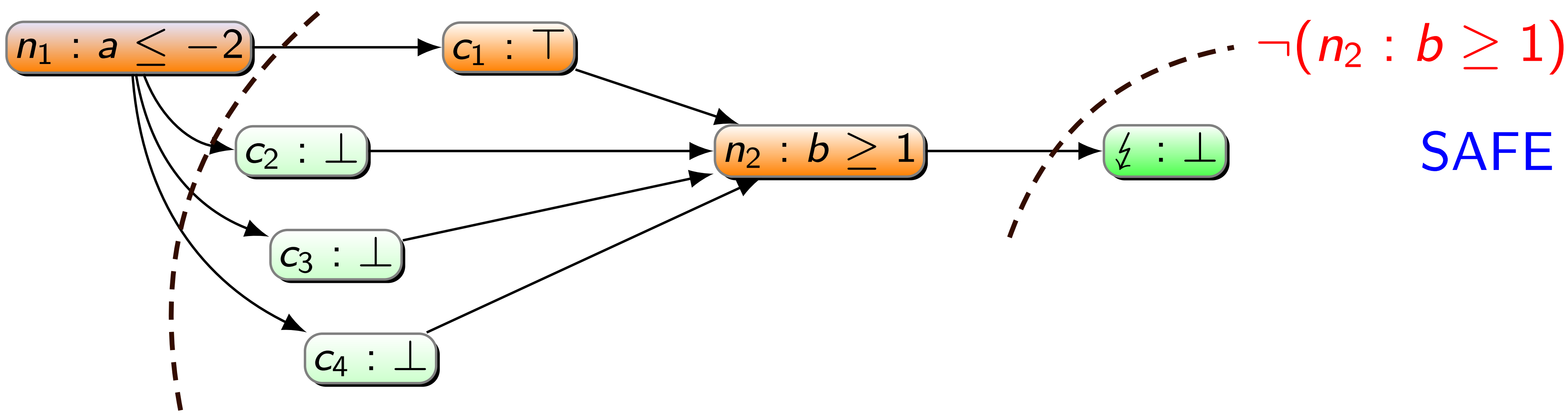
# What about programs?

DL0



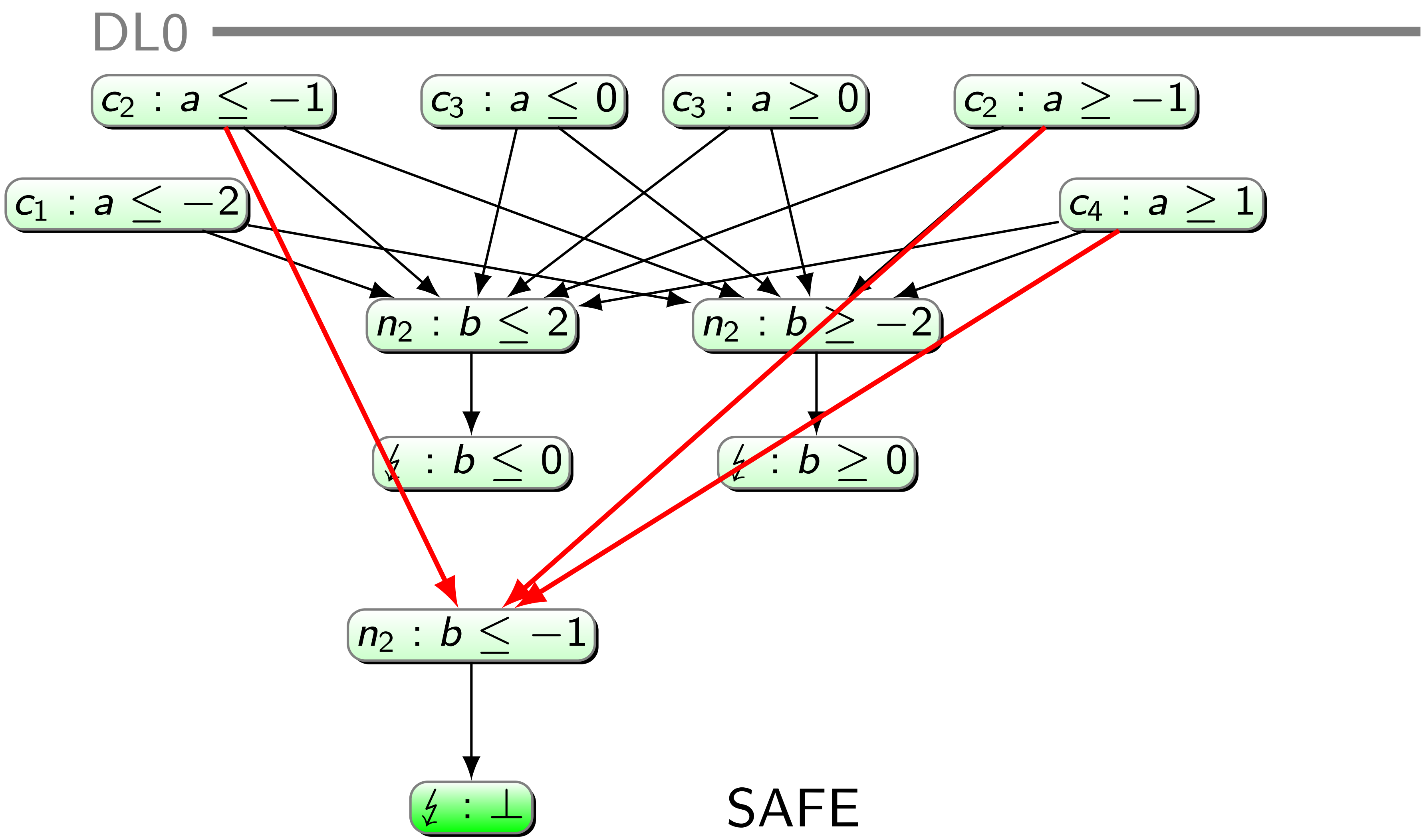




DL1

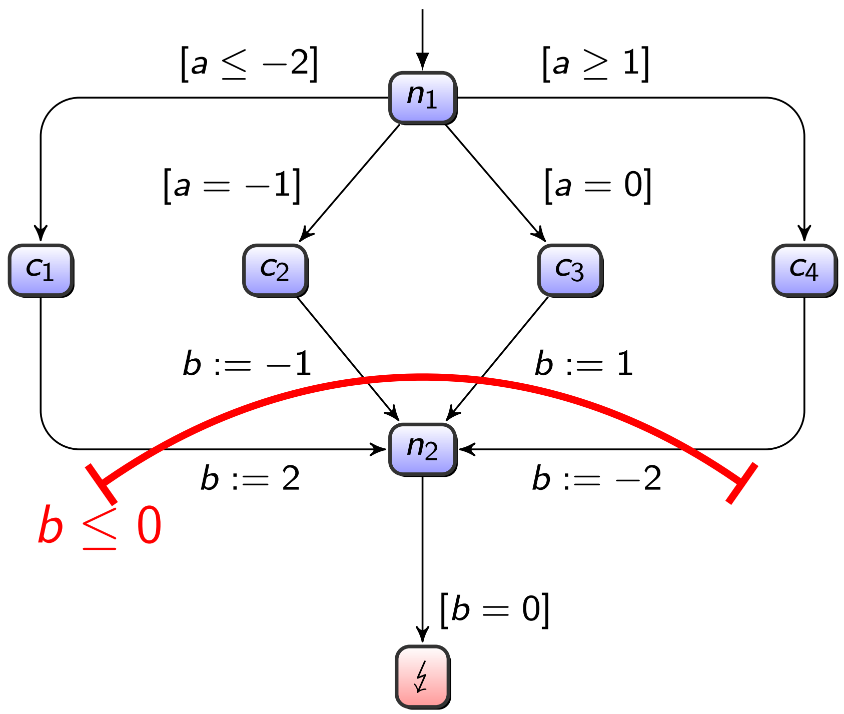



SAFE $\rightarrow$ find cut

# What about programs?




SAFE



# ACDCL(*A*)
## One Line Summaries

ACDCL(*A*) program analysers!

Techniques from SAT translate to programs

ACDCL(*A*) discovers <u>small</u>, <u>property driven</u> refinement

# Something more practical

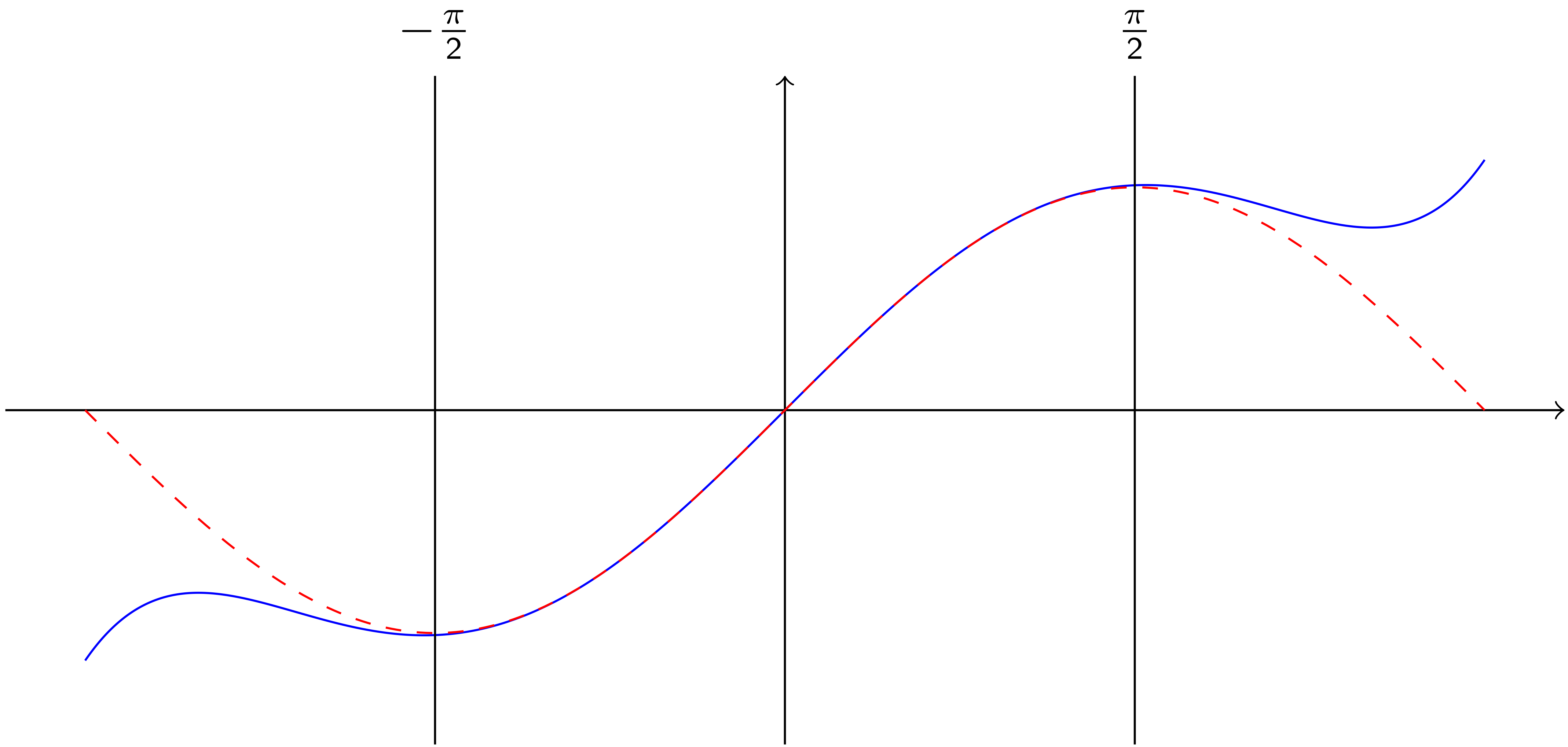
ACDCL(Interval) procedure over floating point and machine integer intervals

Automatically finds <u>property-dependent partitioning</u>

Example: Taylor expansion of sine-function

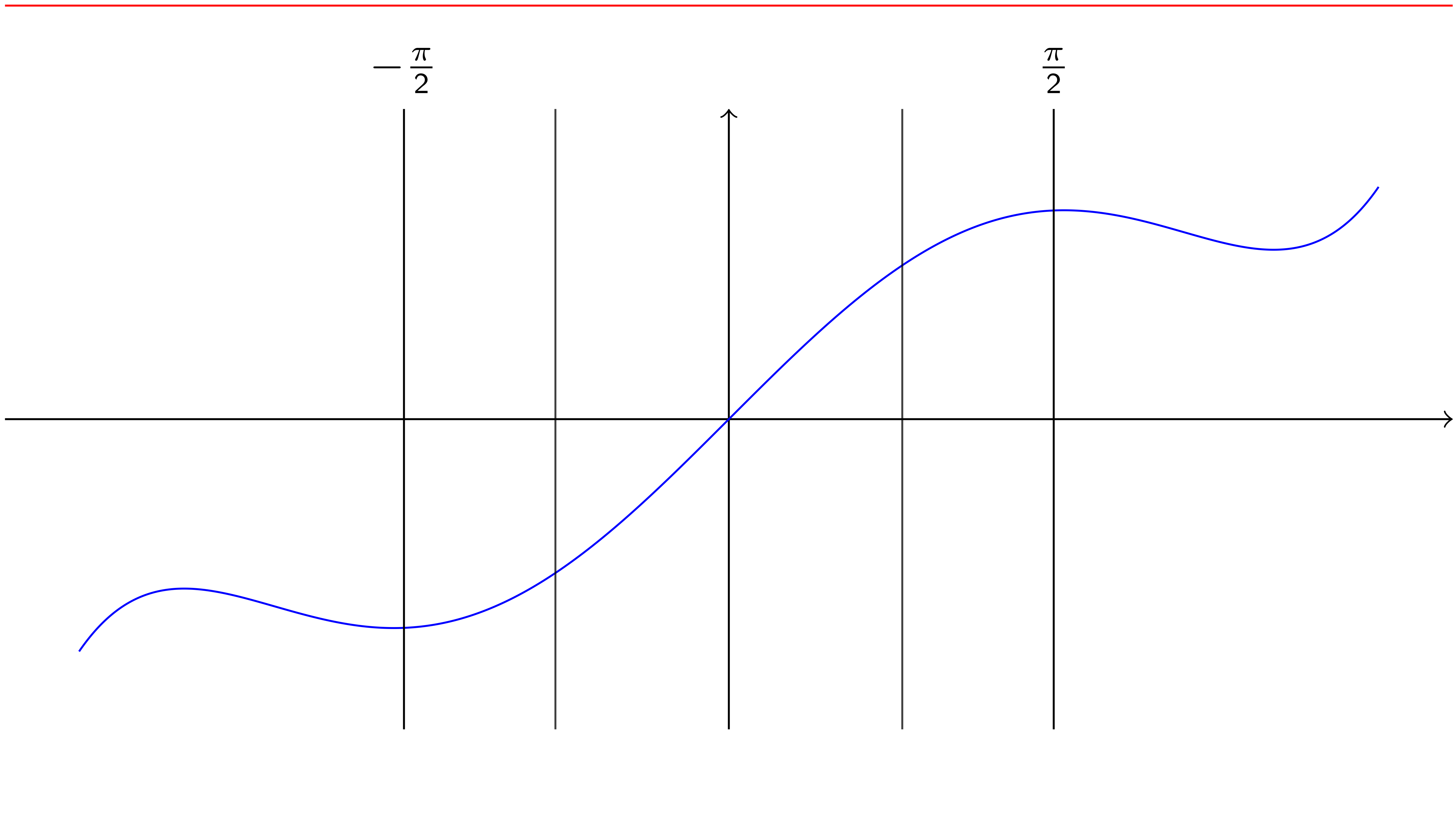
```
int main()
{
  float IN;
  __CPROVER_assume(IN > -HALFPI && IN < HALFPI);

  float x = IN;

  float result = x - (x*x*x)/6.0f + (x*x*x*x*x)/120.0f + (x*x*x*x*x*x*x)/5040.0f;

  assert(result <= VAL && result >= -VAL);

  return 0;
}
```

# Implementation



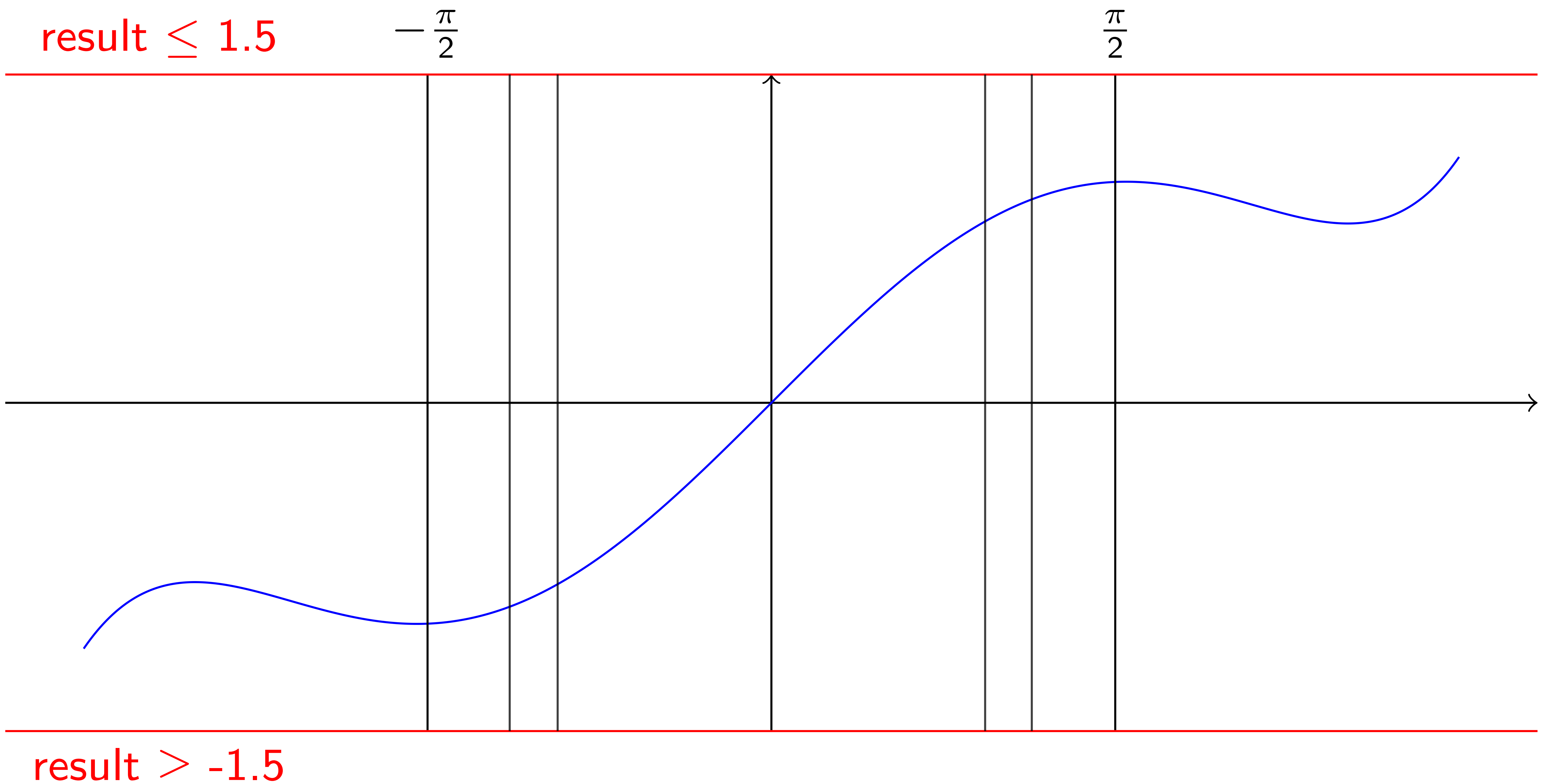
# Number of partitions vs. tightness of bound

result $\leq$ 2.0

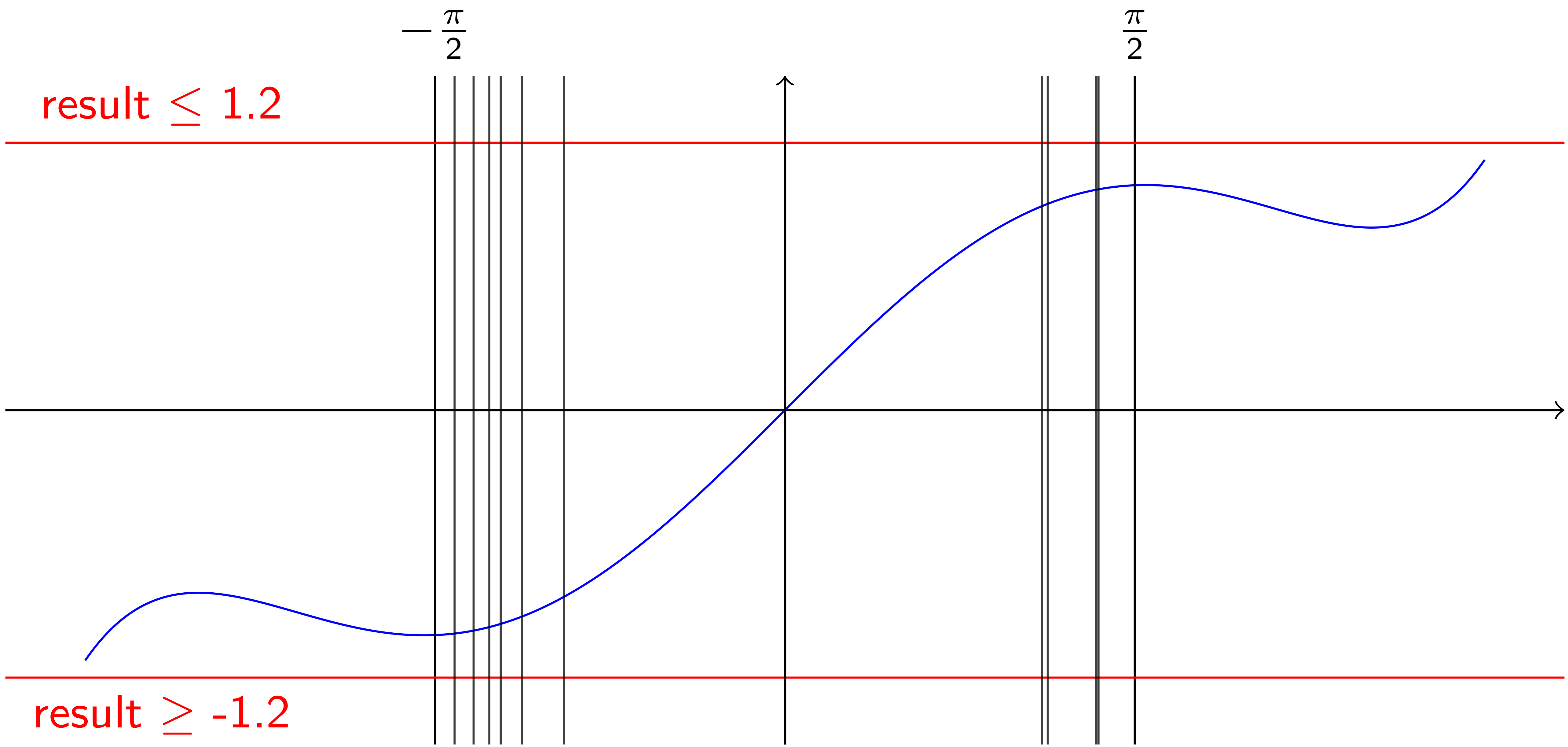



result $\geq$ -2.0

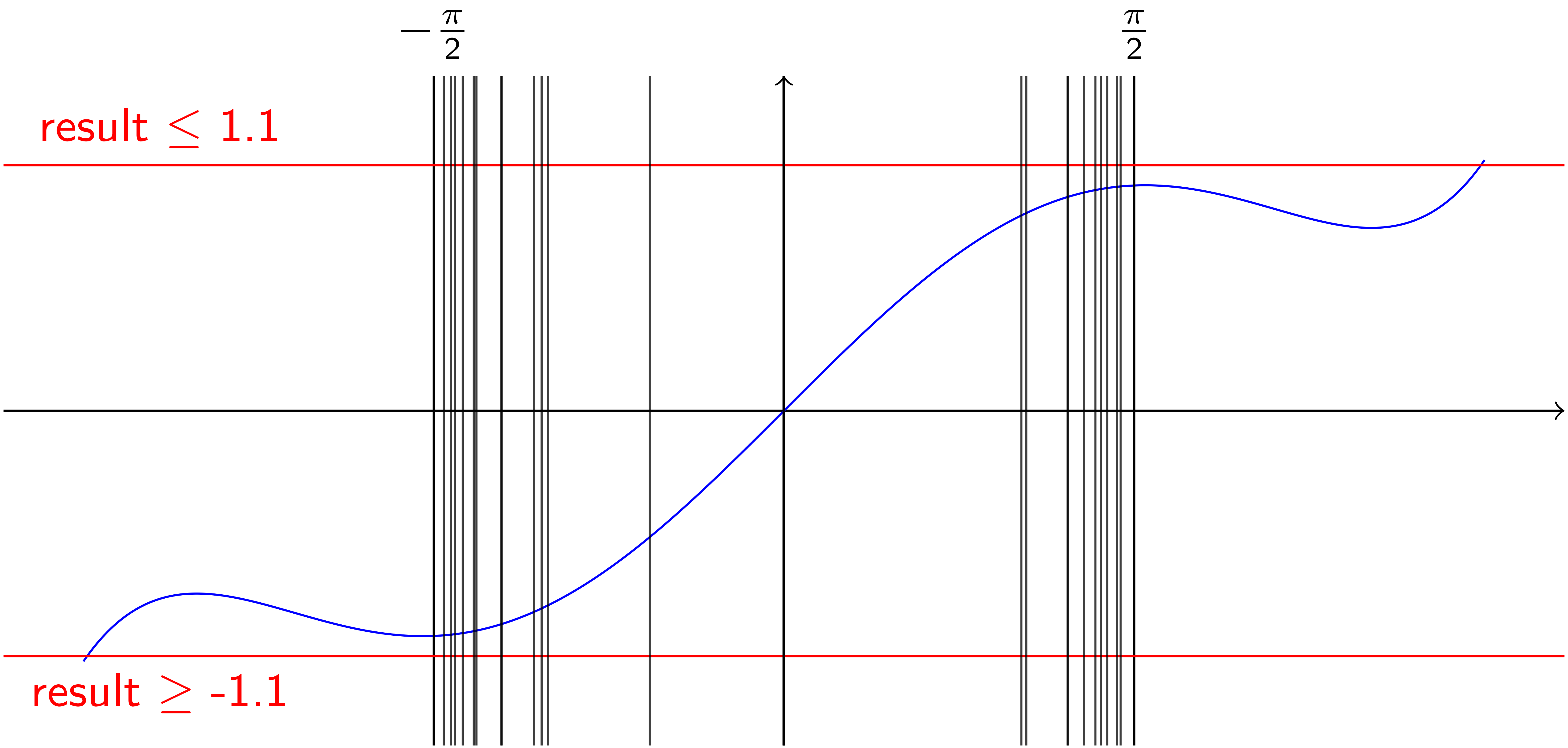
# Number of partitions vs. tightness of bound



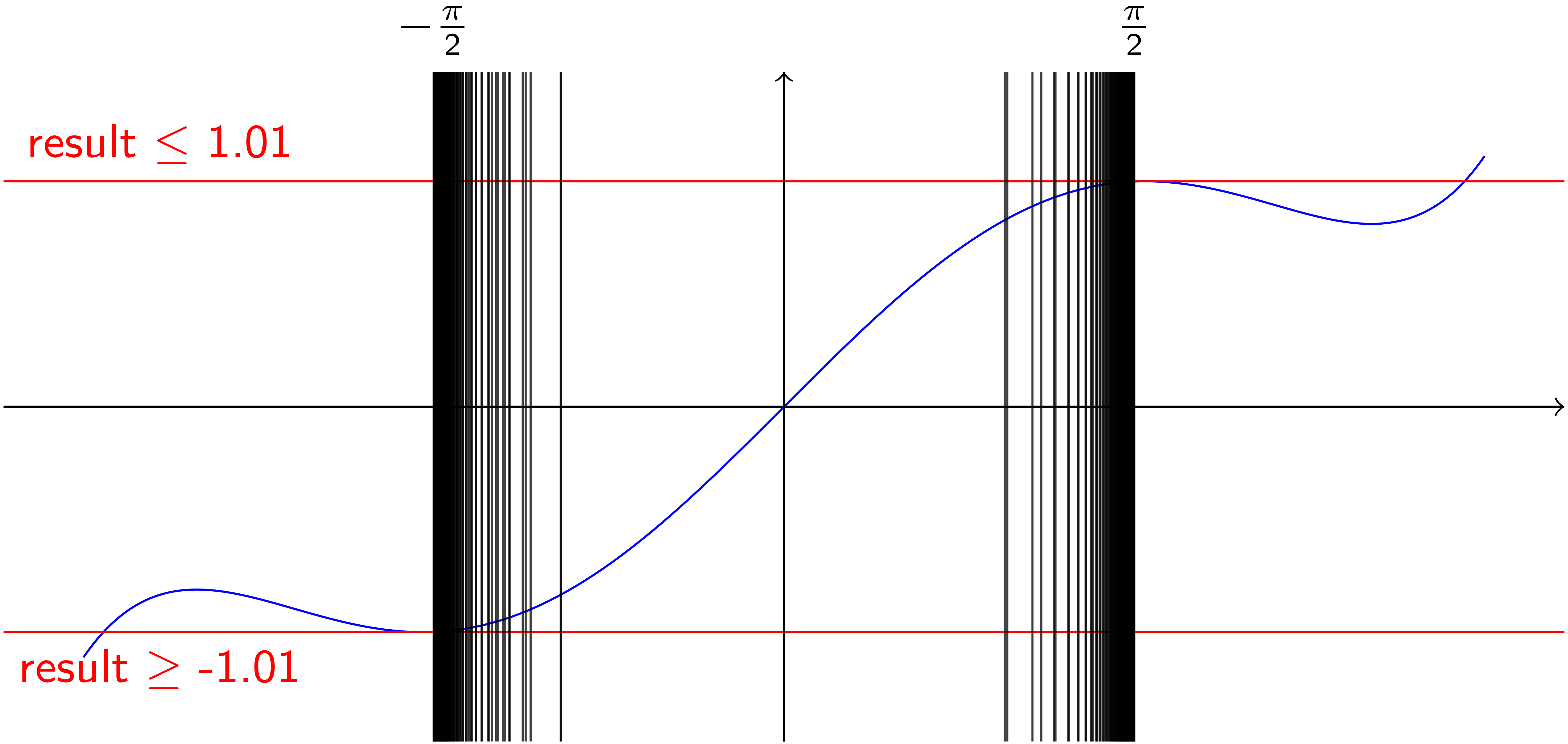
# Number of partitions vs. tightness of bound



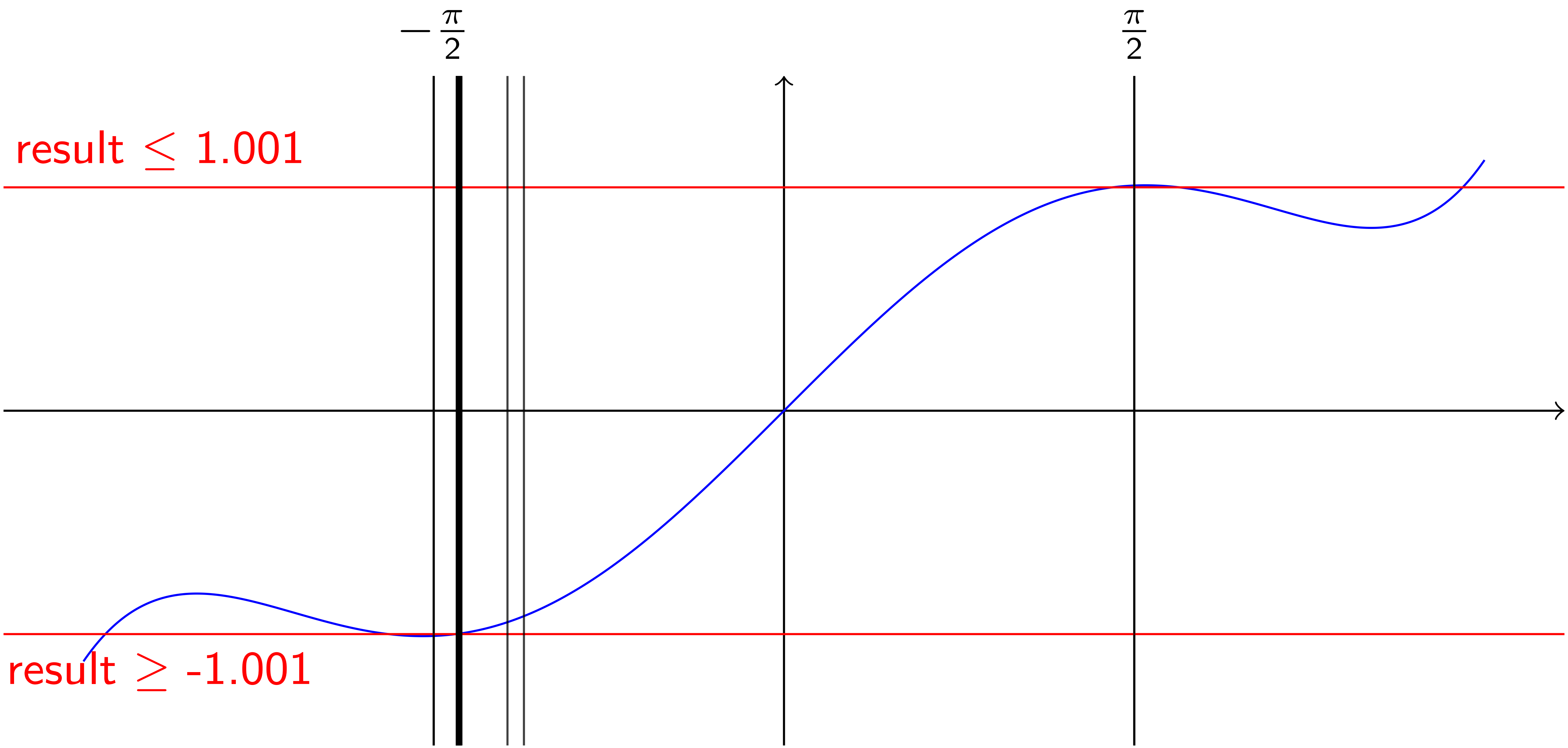
# Number of partitions vs. tightness of bound



# Number of partitions vs. tightness of bound



# Number of partitions vs. tightness of bound




Precise results using a strict abstraction!
Orders of magnitude faster than propositional SAT

# Conclusion

SAT solvers are abstract interpreters

| | |
|---|---|
| partial assignments | Cartesian domain |
| unit rule | abstr. transformer |
| BCP | gfp |
| decisions | meet irreducibles |
| learning | trace partioning |

Abstract interpreters can be SAT solvers

ACDCL(*A*) for program analysis / SMT
precise results in an imprecise abstraction

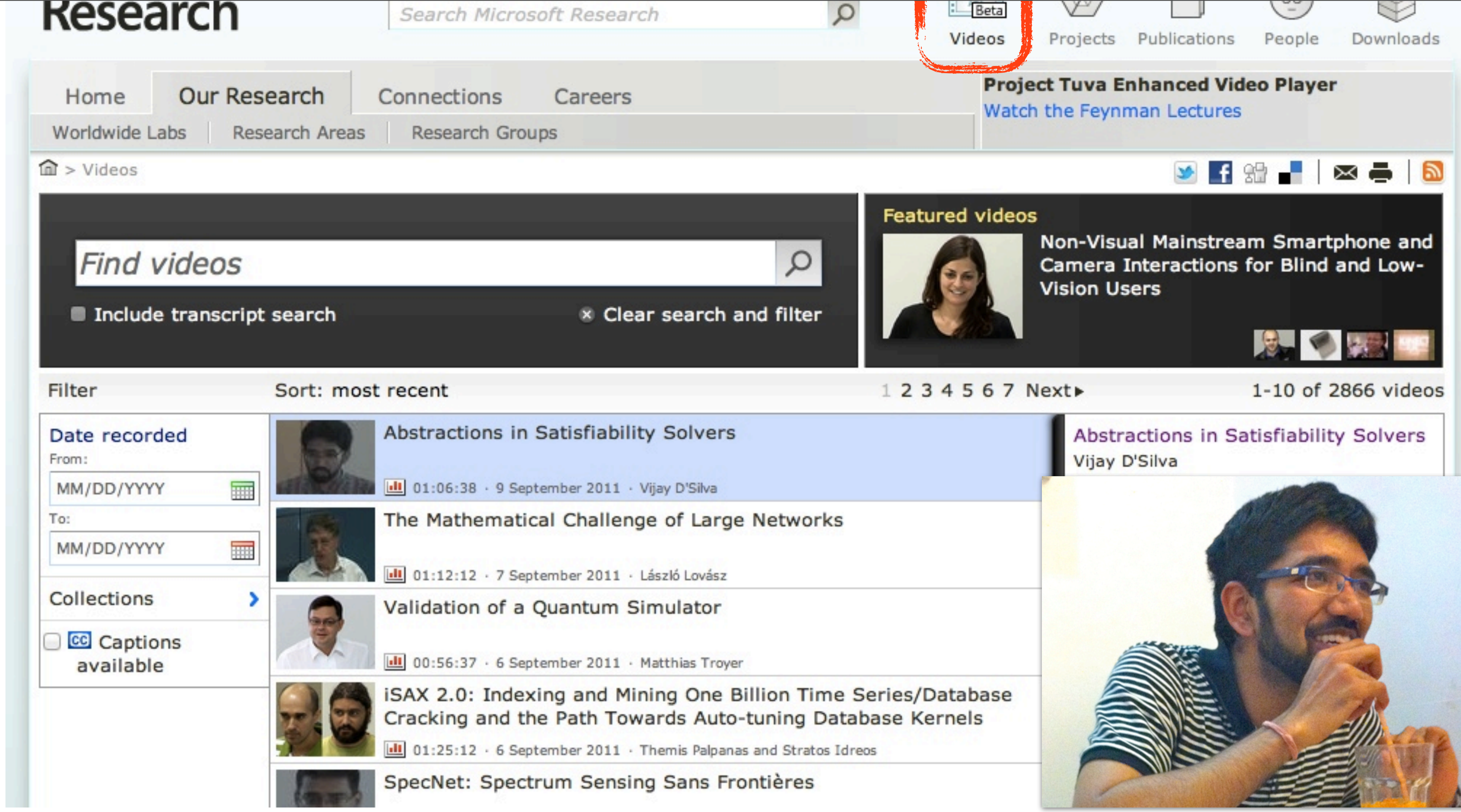
# ... walk into a bar

AI looks toward SMT

post operators, widenings

Precise analysis, efficient handling of disjunction

SMT looks towards AI




# Invited questions

Isn't this just CEGAR?

What if case splits are not enough?

Show me experiments!