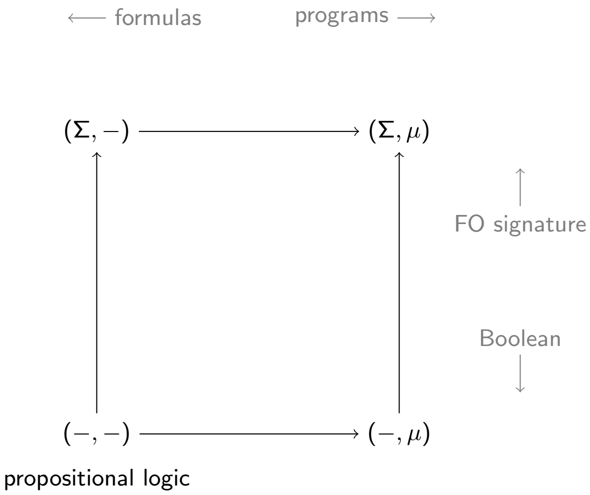
DPLL is Abstract Interpretation

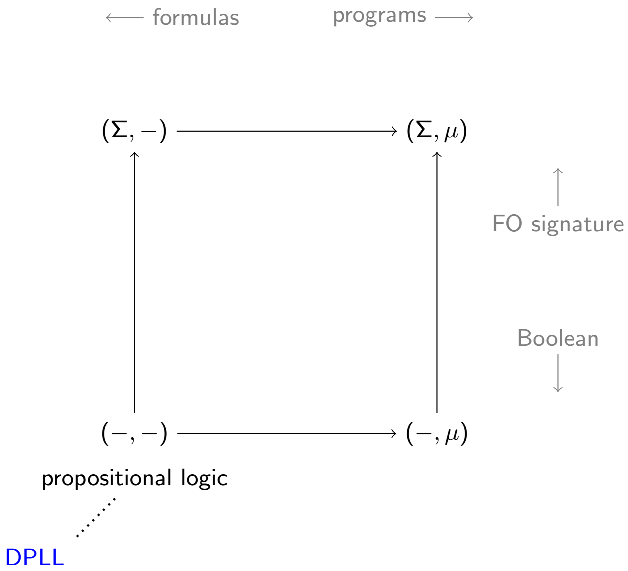
Leopold Haller
Vijay D'Silva

UNIVERSITY OF OXFORD

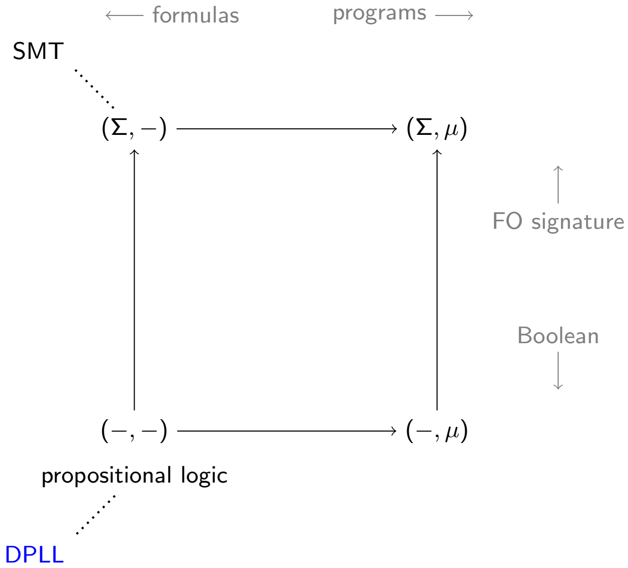
Analysing Logic and Programs



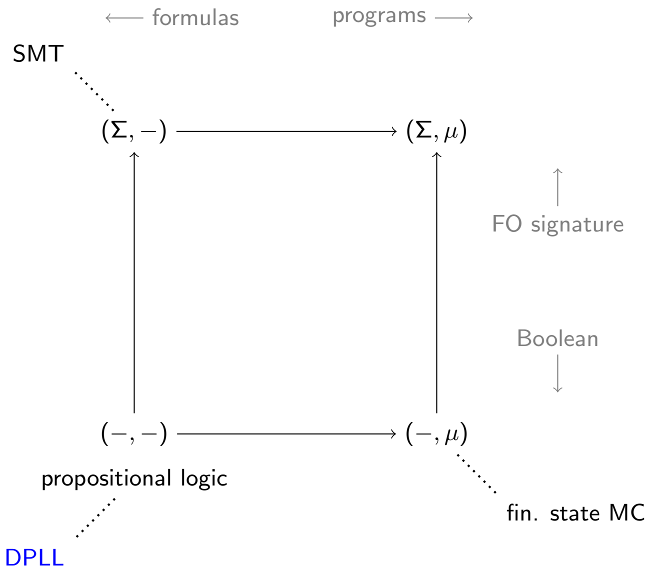
Analysing Logic and Programs



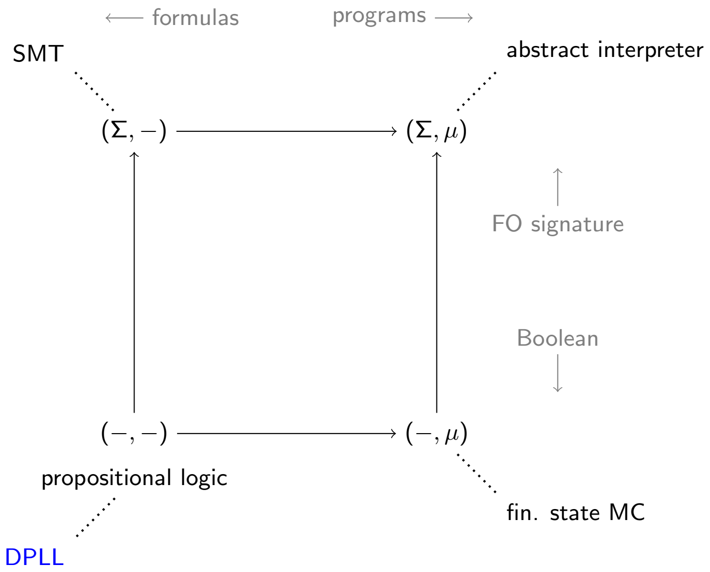
Analysing Logic and Programs



Analysing Logic and Programs



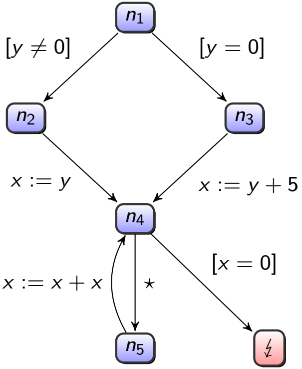
Analysing Logic and Programs



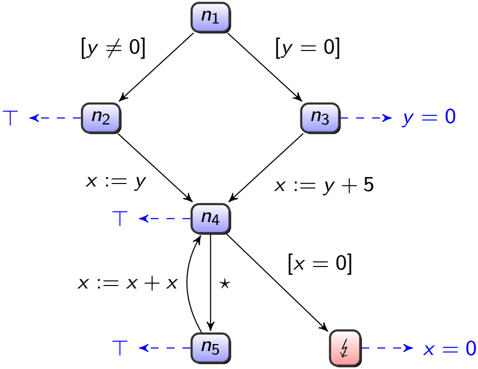
SAT Solvers are Efficient

(Malik and Zhang 2009)

Prove the following program safe using the domain of intervals:

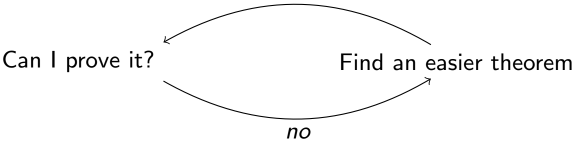


Prove the following program safe using the domain of intervals:



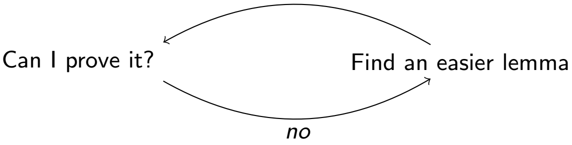
Flowchart for Proving a Theorem

Prove theorem T



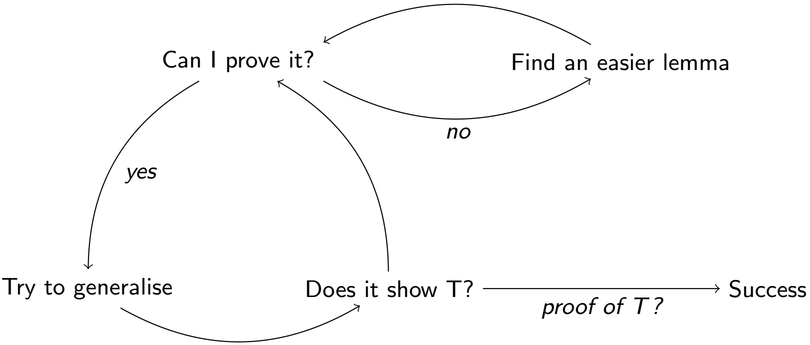
Flowchart for Proving a Theorem

Prove theorem T



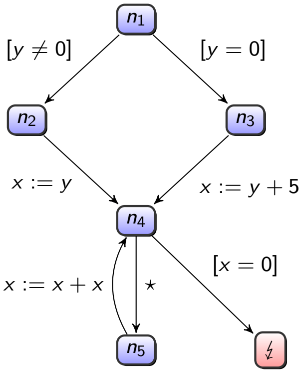
Flowchart for Proving a Theorem

Prove theorem T



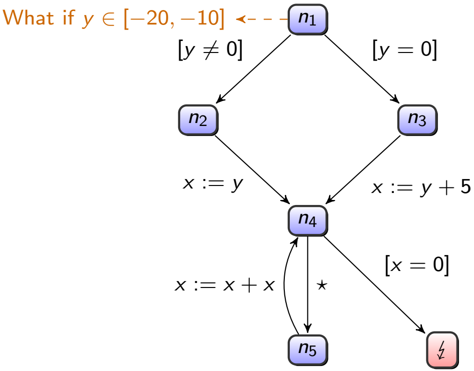
Refining the analysis

Let's try this on the program:



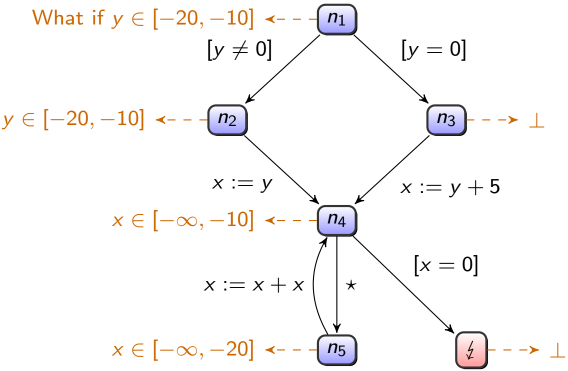
Refining the analysis

Let's try this on the program:



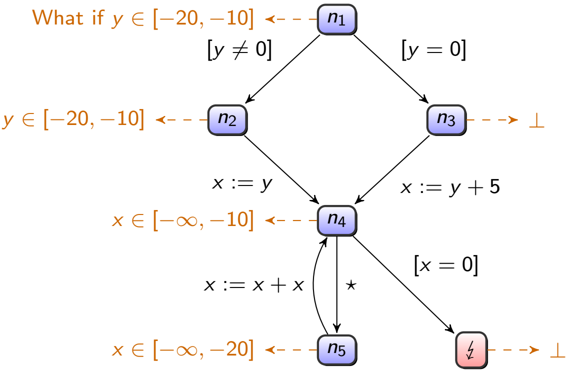
Refining the analysis

Let's try this on the program:



Refining the analysis

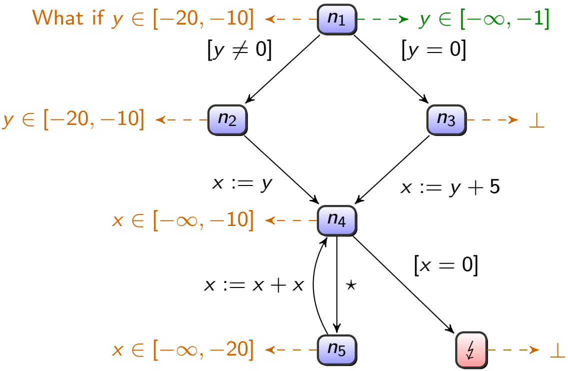
Let's try this on the program:



Analyse proof

Refining the analysis

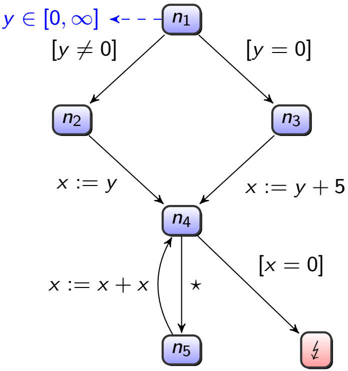
Let's try this on the program:



Analyse proof

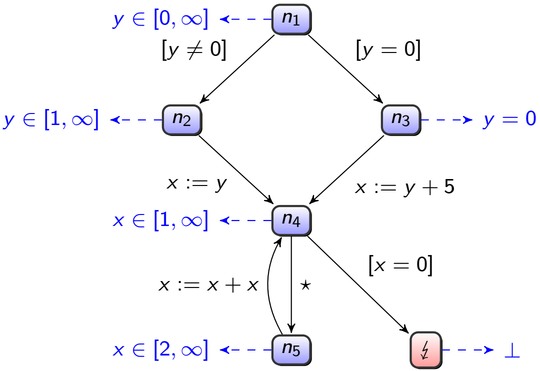
Refining the analysis

Let's try this on the program:



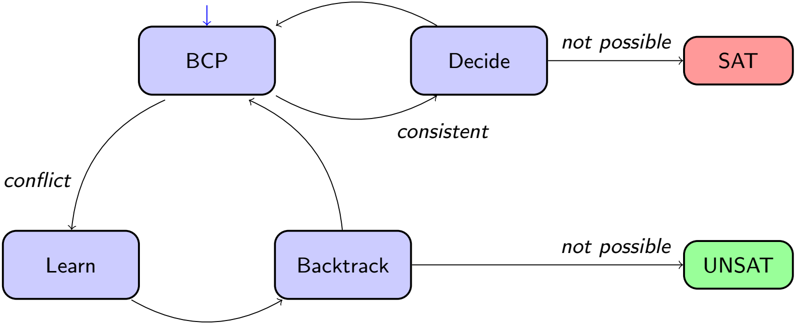
Refining the analysis

Let's try this on the program:



The Modern DPLL algorithm

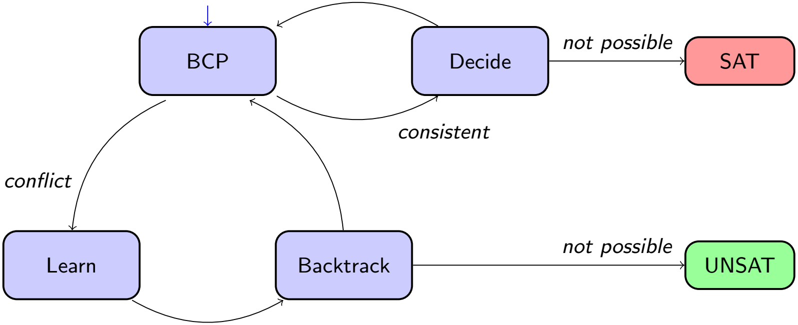
Decide satisfiability of propositional formula ϕ in conjunctive normal form (CNF)



Preliminaries

- Set of variables Var
- $v, \neg v$ are literals for $v \in$ Var
- Disjunction of literals is a clause $v_1 \vee \neg v_2 \vee v_3$
- CNF formula is a conjunction of clauses

The Modern DPLL algorithm



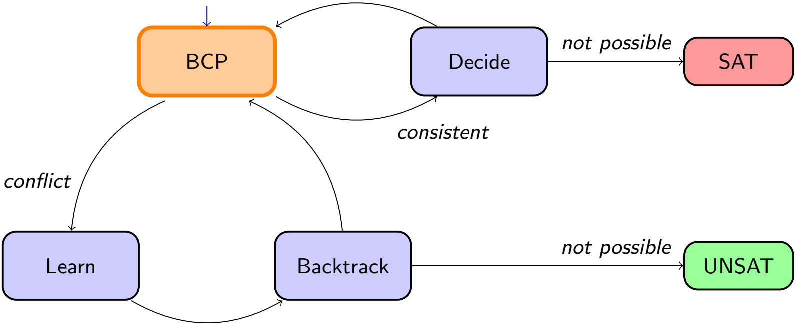
Partial assignments

SAT solver explores the space of partial functions:

$$\mathsf{Var} \longrightarrow \{\mathsf{t}, \mathsf{f}\}$$

Uses deduction and search to find a satisfying assignment or exhaustively search space.

The Modern DPLL algorithm

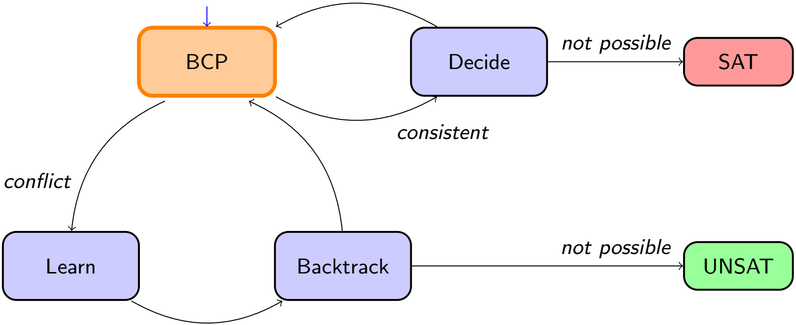


$$\phi = {\color{red}x} \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

BCP example

BCP: $\emptyset \rightarrow$

The Modern DPLL algorithm

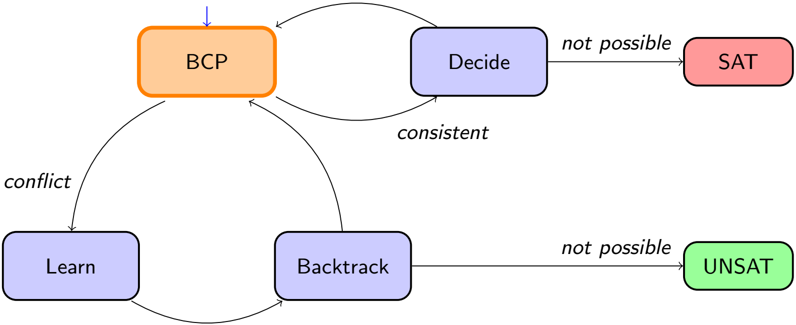


$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

BCP example

BCP: $\emptyset \rightarrow \{x \mapsto \mathrm{t}\} \rightarrow$

The Modern DPLL algorithm

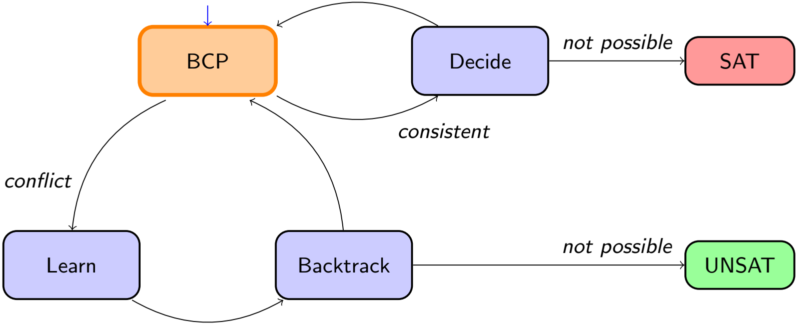


$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

BCP example

BCP: $\emptyset \rightarrow \{x \mapsto \mathrm{t}\} \rightarrow \{x \mapsto \mathrm{t}, y \mapsto \mathrm{f}\}$

The Modern DPLL algorithm

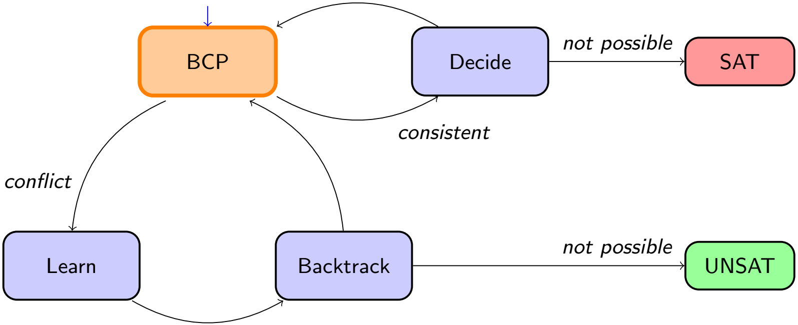


Unit rule

Deduction uses the unit rule:

$$\text{unit}(\rho, l_1 \vee \ldots l_i \ldots \vee l_k) = \begin{cases} \text{conflict} & \text{all literals are contradicted by } \rho \\ \rho[\mathrm{l_i} \mapsto \mathrm{t}] & \text{all literals but } l_i \text{ contradicted } \rho \\ \rho & \text{otherwise} \end{cases}$$

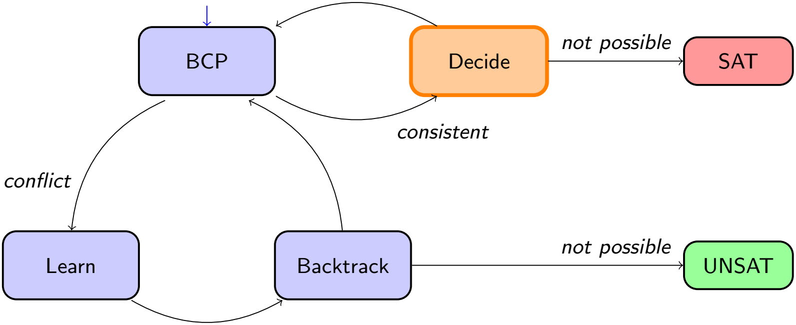
The Modern DPLL algorithm



Boolean Constraint Propagation

BCP(ϕ, ρ) {
    repeat
        $\rho' \leftarrow \rho$;
        for *Clause* $c \in \phi$ **do** $\rho \leftarrow \text{unit}(c, \rho)$;
    until $\rho = \rho'$;
}

The Modern DPLL algorithm



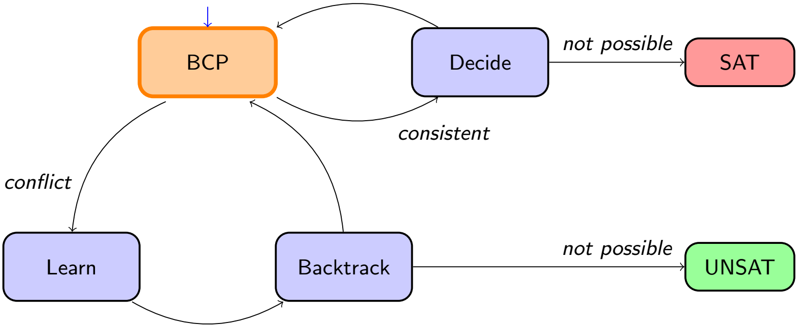
$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

Decision

BCP: $\emptyset \rightarrow \{x \mapsto \mathsf{t}\} \rightarrow \{x \mapsto \mathsf{t}, y \mapsto \mathsf{f}\} \rightarrow$

Add assumption to partial assignment:
Decision: $\{x \mapsto \mathsf{t}, y \mapsto \mathsf{f}, z \mapsto \mathsf{f}\}$

The Modern DPLL algorithm



$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

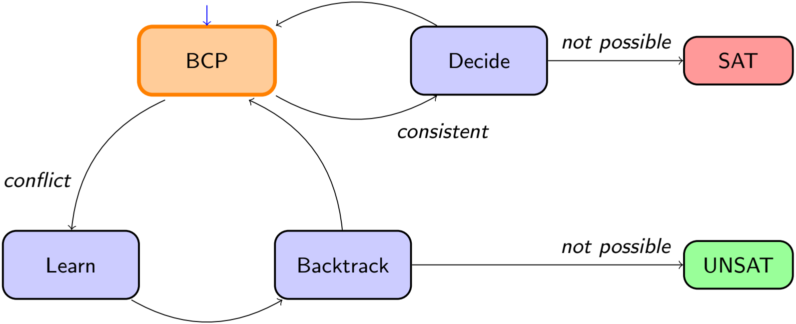
BCP

BCP: $\emptyset \rightarrow \{x \mapsto t\} \rightarrow \{x \mapsto t, y \mapsto f\} \rightarrow$

Decision: $\{x \mapsto t, y \mapsto f, z \mapsto f\} \rightarrow$

BCP:

The Modern DPLL algorithm

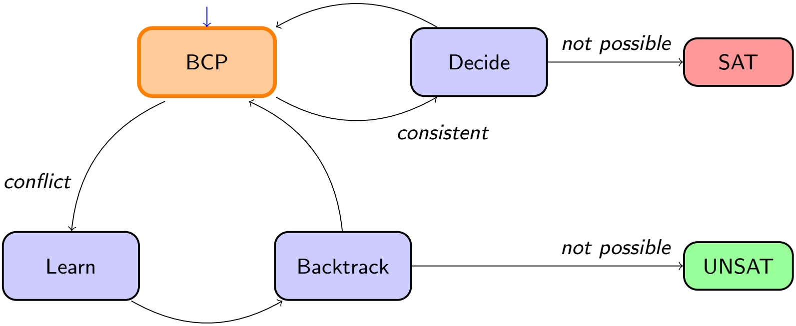


$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge \color{red}{(y \vee z \vee w)}$$

BCP

BCP:	$\emptyset \to \{x \mapsto \mathrm{t}\} \to \{x \mapsto \mathrm{t}, y \mapsto \mathrm{f}\} \to$
Decision:	$\{x \mapsto \mathrm{t}, y \mapsto \mathrm{f}, \color{blue}{z \mapsto \mathrm{f}}\} \to$
BCP:	$\{x \mapsto \mathrm{t}, y \mapsto \mathrm{f}, z \mapsto \mathrm{f}, w \mapsto \mathrm{f}\}$

The Modern DPLL algorithm

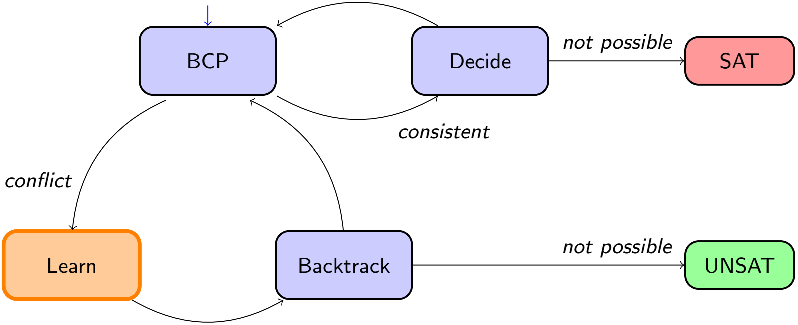


$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

BCP

BCP:	$\emptyset \rightarrow \{x \mapsto \mathrm{t}\} \rightarrow \{x \mapsto \mathrm{t}, y \mapsto \mathrm{f}\} \rightarrow$
Decision:	$\{x \mapsto \mathrm{t}, y \mapsto \mathrm{f}, z \mapsto \mathrm{f}\} \rightarrow$
BCP:	$\{x \mapsto \mathrm{t}, y \mapsto \mathrm{f}, z \mapsto \mathrm{f}, w \mapsto \mathrm{f}\} \rightarrow$ conflict

The Modern DPLL algorithm



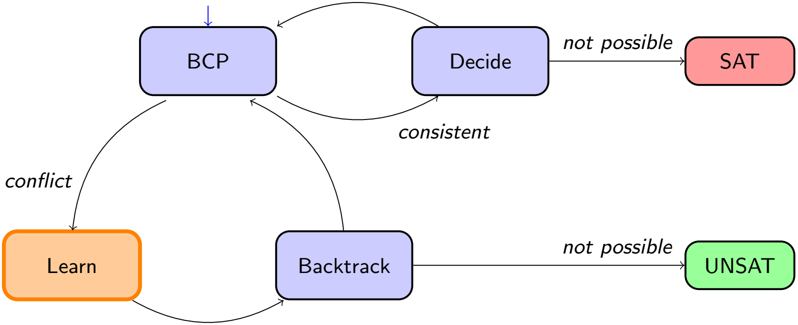
$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

Learn

BCP: $\{x \mapsto \text{t}, y \mapsto \text{f}, z \mapsto \text{f}, w \mapsto \text{f}\} \rightarrow$ conflict

Find reason for the conflict,

The Modern DPLL algorithm



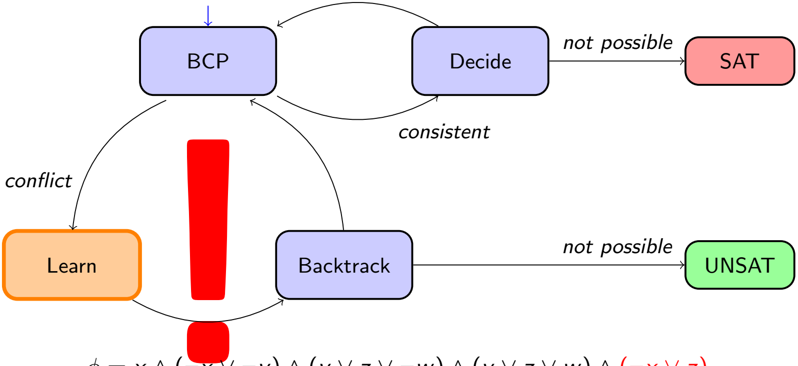
$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

Learn

BCP: $\{x \mapsto \mathrm{t}, y \mapsto \mathrm{f}, z \mapsto \mathrm{f}, w \mapsto \mathrm{f}\} \rightarrow$ conflict

Find reason for the conflict, e.g., $x \wedge \neg z$ can never be true:

The Modern DPLL algorithm



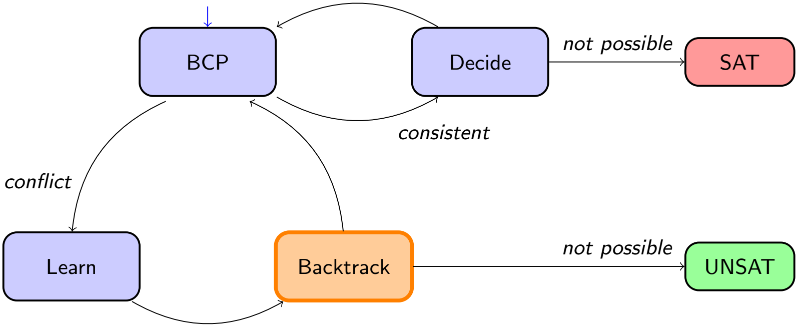
$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w) \wedge (\neg x \vee z)$$

Learn

BCP: $\{x \mapsto \text{t}, y \mapsto \text{f}, z \mapsto \text{f}, w \mapsto \text{f}\} \rightarrow$ conflict

Find reason for the conflict, e.g., $x \wedge \neg z$ can never be true:
Learn $\neg(x \wedge \neg z) = \neg x \vee z$.

The Modern DPLL algorithm

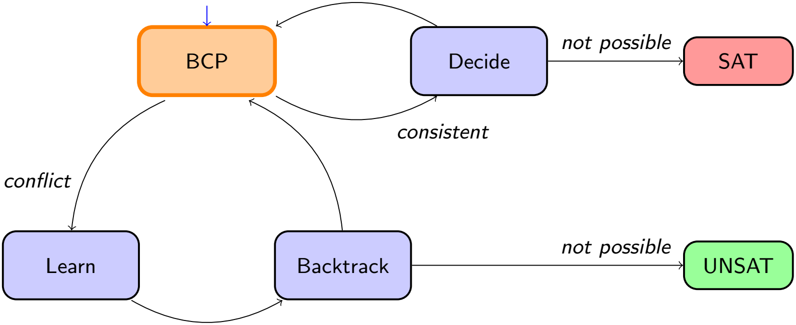


$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w) \wedge (\neg x \vee z)$$

Backtrack

Undo assumptions that contradict learned clause:
Backtrack: $\{x \mapsto \mathrm{t}, y \mapsto \mathrm{t}\}$

The Modern DPLL algorithm



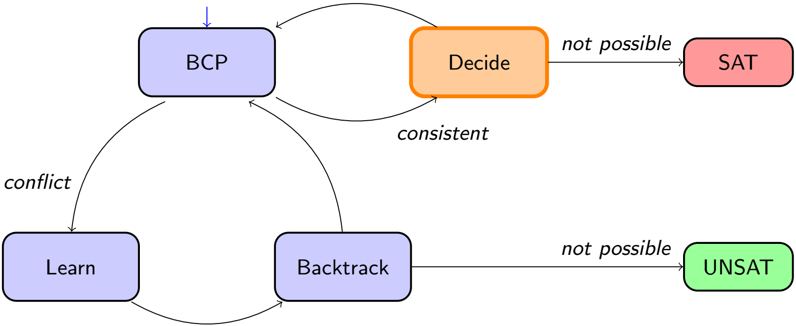
$$\phi = x \land (\neg x \lor \neg y) \land (y \lor z \lor \neg w) \land (y \lor z \lor w) \land (\neg x \lor z)$$

BCP

Backtrack: $\{x \mapsto \mathrm{t}, y \mapsto \mathrm{t}\} \rightarrow$

BCP automatically takes us to a new part of the search space:
BCP: $\{x \mapsto \mathrm{t}, y \mapsto \mathrm{t}, z \mapsto \mathrm{t}\}$

The Modern DPLL algorithm

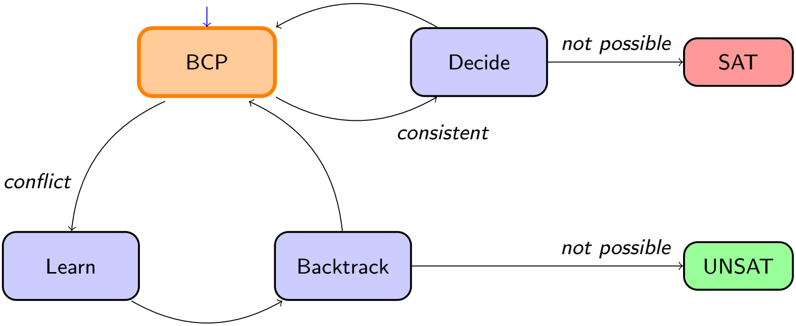


$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w) \wedge (\neg x \vee z)$$

Decide

Backtrack:	$\{x \mapsto \mathrm{t}, y \mapsto \mathrm{t}\} \rightarrow$
BCP:	$\{x \mapsto \mathrm{t}, y \mapsto \mathrm{t}, z \mapsto \mathrm{t}\} \rightarrow$
Decide:	$\{x \mapsto \mathrm{t}, y \mapsto \mathrm{t}, z \mapsto \mathrm{t}, w \mapsto \mathrm{f}\}$

The Modern DPLL algorithm



$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w) \wedge (\neg x \vee z)$$

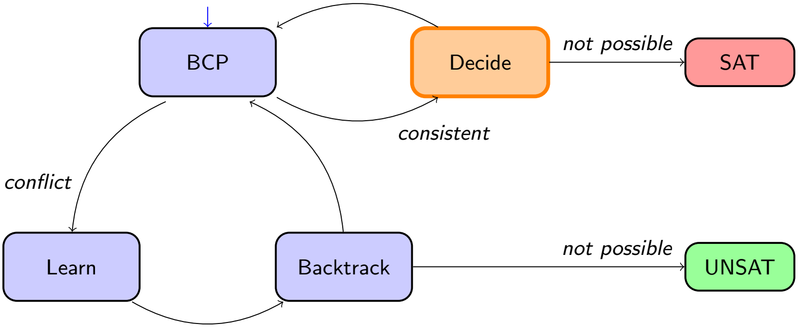
Decide

Backtrack: $\{x \mapsto \mathrm{t}, y \mapsto \mathrm{t}\} \rightarrow$

BCP: $\{x \mapsto \mathrm{t}, y \mapsto \mathrm{t}, z \mapsto \mathrm{t}\} \rightarrow$

Decide: $\{x \mapsto \mathrm{t}, y \mapsto \mathrm{t}, z \mapsto \mathrm{t}, w \mapsto \mathrm{f}\}$

The Modern DPLL algorithm

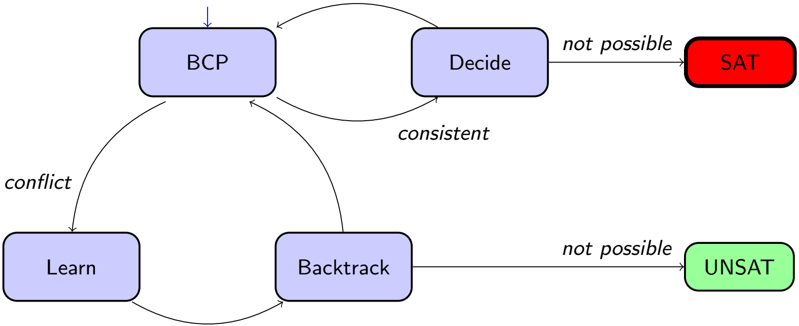


$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w) \wedge (\neg x \vee z)$$

Decide

Backtrack:	$\{x \mapsto t, y \mapsto t\} \rightarrow$
BCP:	$\{x \mapsto t, y \mapsto t, z \mapsto t\} \rightarrow$
Decide:	$\{x \mapsto t, y \mapsto t, z \mapsto t, w \mapsto f\}$

The Modern DPLL algorithm



$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w) \wedge (\neg x \vee z)$$

Decide

Backtrack:	$\{x \mapsto \mathsf{t}, y \mapsto \mathsf{t}\} \rightarrow$
BCP:	$\{x \mapsto \mathsf{t}, y \mapsto \mathsf{t}, z \mapsto \mathsf{t}\} \rightarrow$
Decide:	$\{x \mapsto \mathsf{t}, y \mapsto \mathsf{t}, z \mapsto \mathsf{t}, w \mapsto \mathsf{f}\}$

```
bool v_1, ..., v_k;
if(φ)
  assert(0);
// program safe iff φ UNSAT
```

Logic for Static Analysis People

$$C = \langle \wp(\mathrm{Var} \to \{\mathrm{t}, \mathrm{f}\}), \subseteq, \cap, \cup \rangle$$
$$post_{\phi}^{C}(X) = \{\varepsilon \in X \mid \phi \text{ is true under } \varepsilon\}$$

Logic for Static Analysis People

$$C = \langle \wp(\mathsf{Var} \to \{\mathsf{t}, \mathsf{f}\}), \subseteq, \cap, \cup \rangle$$
$$post^C_\phi(X) = \{\varepsilon \in X \mid \phi \text{ is true under } \varepsilon\}$$

Examples

$$post^C_{a \wedge b}(\top) = \{\langle a \mapsto \mathsf{t}, b \mapsto \mathsf{t}\rangle\}$$
$$post^C_{a \vee \neg b}(\top) = \{\langle a \mapsto \mathsf{t}, b \mapsto \mathsf{t}\rangle, \langle a \mapsto \mathsf{t}, b \mapsto \mathsf{f}\rangle, \langle a \mapsto \mathsf{f}, b \mapsto \mathsf{f}\rangle\}$$

Logic for Static Analysis People

$$C = \langle \wp(\mathrm{Var} \to \{\mathrm{t}, \mathrm{f}\}), \subseteq, \cap, \cup \rangle$$
$$post^C_\phi(X) = \{\varepsilon \in X \mid \phi \text{ is true under } \varepsilon\}$$

Examples

$$post^C_{a \wedge b}(\top) = \{\langle a \mapsto \mathrm{t}, b \mapsto \mathrm{t}\rangle\}$$
$$post^C_{a \vee \neg b}(\top) = \{\langle a \mapsto \mathrm{t}, b \mapsto \mathrm{t}\rangle, \langle a \mapsto \mathrm{t}, b \mapsto \mathrm{f}\rangle, \langle a \mapsto \mathrm{f}, b \mapsto \mathrm{f}\rangle\}$$

ϕ is satisfiable exactly if $post^C_\phi(\top) \neq \emptyset$.

Abstraction and DPLL

DPLL datastructure

Partial assignment $\text{Var} \mapsto \{t, f\}$ and additional conflict states.
Each variable is:

unknown

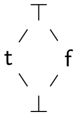
true false

conflicting

Abstraction and DPLL

DPLL operates over an abstract domain

$$\mathsf{Var} \to \wp(\{\mathsf{t}, \mathsf{f}\}) \quad = \quad \mathsf{Var} \longrightarrow$$



$$A = \langle \mathsf{Var} \to \wp(\{\mathsf{t}, \mathsf{f}\}), \sqsubseteq, \sqcap, \sqcup \rangle$$

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

DPLL datastructure

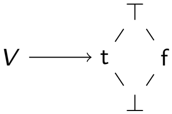
Partial assignment $\mathsf{Var} \mapsto \{\mathsf{t}, \mathsf{f}\}$ and additional conflict states.
Each variable is:

unknown

true false

conflicting

Best Abstract Transformers over the Cartesian Abstraction



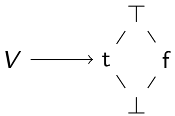
$$post^A_\phi(\top) = \bot \quad \Longrightarrow \quad \phi \text{ is unsatisfiable}$$

Best abstract transformer for a clause $a \vee b$

We synthesize the transformer $post^A_{a\vee b}$ for the argument $(a \mapsto \mathsf{f}, b \mapsto \top)$:

$$post^A_{a\vee b}(\langle a \mapsto \mathsf{f}, b \mapsto \top\rangle) =$$

Best Abstract Transformers over the Cartesian Abstraction



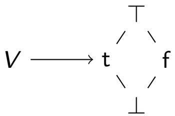
$$post^A_\phi(\top) = \bot \quad \Longrightarrow \quad \phi \text{ is unsatisfiable}$$

Best abstract transformer for a clause $a \vee b$

We synthesize the transformer $post^A_{a\vee b}$ for the argument $(a \mapsto \mathsf{f}, b \mapsto \top)$:

$$post^A_{a\vee b}(\langle a \mapsto \mathsf{f}, b \mapsto \top\rangle) = \alpha \circ post^C_{a\vee b} \circ \gamma(\langle a \mapsto \mathsf{f}, b \mapsto \top\rangle)$$

Best Abstract Transformers over the Cartesian Abstraction



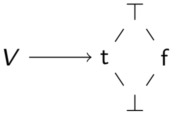
$$post^A_\phi(\top) = \bot \quad \Longrightarrow \quad \phi \text{ is unsatisfiable}$$

Best abstract transformer for a clause $a \vee b$

We synthesize the transformer $post^A_{a\vee b}$ for the argument $(a \mapsto \mathrm{f}, b \mapsto \top)$:

$$\begin{aligned} post^A_{a\vee b}(\langle a \mapsto \mathrm{f}, b \mapsto \top\rangle) &= \alpha \circ post^C_{a\vee b} \circ \gamma(\langle a \mapsto \mathrm{f}, b \mapsto \top\rangle) \\ &= \alpha \circ post^C_{a\vee b}(\{\langle a : \mathrm{f}, b : \mathrm{f}\rangle, \langle a : \mathrm{f}, b : \mathrm{t}\rangle\}) \end{aligned}$$

Best Abstract Transformers over the Cartesian Abstraction



$$post^A_\phi(\top) = \bot \quad \Longrightarrow \quad \phi \text{ is unsatisfiable}$$

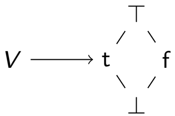
Best abstract transformer for a clause $a \vee b$

We synthesize the transformer $post^A_{a\vee b}$ for the argument $(a \mapsto \mathsf{f}, b \mapsto \top)$:

$$\begin{aligned} post^A_{a\vee b}(\langle a \mapsto \mathsf{f}, b \mapsto \top\rangle) &= \alpha \circ post^C_{a\vee b} \circ \gamma(\langle a \mapsto \mathsf{f}, b \mapsto \top\rangle) \\ &= \alpha \circ post^C_{a\vee b}(\{\langle a : \mathsf{f}, b : \mathsf{f}\rangle, \langle a : \mathsf{f}, b : \mathsf{t}\rangle\}) \\ &= \alpha(\{\langle a : \mathsf{f}, b : \mathsf{t}\rangle\}) = \langle a \mapsto \mathsf{f}, b \mapsto \mathsf{t}\rangle \end{aligned}$$

Best Abstract Transformers over the Cartesian Abstraction

DPLL operates over an abstract domain.
The unit rule is the best abstract transformer.



$$post^{A}_{\phi}(\top) = \bot \quad \Longrightarrow \quad \phi \text{ is unsatisfiable}$$

Best abstract transformer for a clause $a \vee b$

We synthesize the transformer $post^{A}_{a\vee b}$ for the argument $(a \mapsto \mathrm{f}, b \mapsto \top)$:

$$\begin{aligned} post^{A}_{a\vee b}(\langle a \mapsto \mathrm{f}, b \mapsto \top\rangle) &= \alpha \circ post^{C}_{a\vee b} \circ \gamma(\langle a \mapsto \mathrm{f}, b \mapsto \top\rangle) \\ &= \alpha \circ post^{C}_{a\vee b}(\{\langle a : \mathrm{f}, b : \mathrm{f}\rangle, \langle a : \mathrm{f}, b : \mathrm{t}\rangle\}) \\ &= \alpha(\{\langle a : \mathrm{f}, b : \mathrm{t}\rangle\}) = \langle a \mapsto \mathrm{f}, b \mapsto \mathrm{t}\rangle \end{aligned}$$

Abstract Transformer Semantics for CNF

Literals:

$$post^A_v(a) = a \sqcap \langle v \mapsto \mathrm{t} \rangle$$
$$post^A_{\neg v}(a) = a \sqcap \langle v \mapsto \mathrm{f} \rangle$$

Disjunction and Conjunction:

$$post^A_{\phi \vee \psi}(a) = post^A_\phi(a) \sqcup post^A_\psi(a)$$
$$post^A_{\phi \wedge \psi}(a) = post^A_\phi(a) \sqcap post^A_\psi(a)$$

Abstract Transformer Semantics for CNF

Literals:

$$post^A_v(a) = a \sqcap \langle v \mapsto \mathrm{t} \rangle$$
$$post^A_{\neg v}(a) = a \sqcap \langle v \mapsto \mathrm{f} \rangle$$

Disjunction and Conjunction:

$$post^A_{\phi \vee \psi}(a) = post^A_\phi(a) \sqcup post^A_\psi(a)$$
$$post^A_{\phi \wedge \psi}(a) = post^A_\phi(a) \sqcap post^A_\psi(a)$$

Imprecision example:

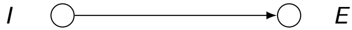
$$\phi = a \wedge (\neg a \vee b)$$

$$\begin{aligned} post^A_\phi(\top) &= post^A_a(\top) \sqcap post^A_{\neg a \vee b}(\top) \\ &= \langle a \mapsto \mathrm{t} \rangle \sqcap (\langle a \mapsto \mathrm{f} \rangle \sqcup \langle b \mapsto \mathrm{t} \rangle) \\ &= \langle a \mapsto \mathrm{t} \rangle \sqcap \top = \langle a \mapsto \mathrm{t} \rangle \end{aligned}$$

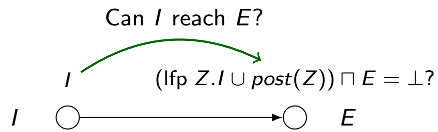
→analysis too imprecise

Refined Abstract Analyses through gfp Iteration

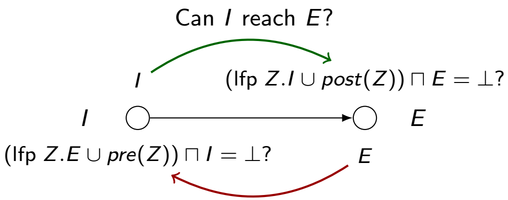
Can I reach E?



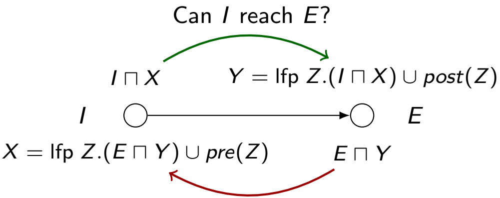
Refined Abstract Analyses through gfp Iteration



Refined Abstract Analyses through gfp Iteration

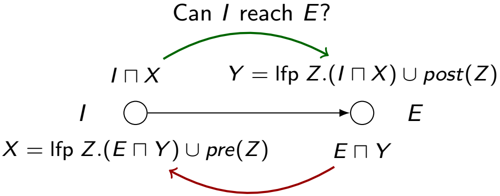


Refined Abstract Analyses through gfp Iteration



(P. Cousot 1978)

Refined Abstract Analyses through gfp Iteration



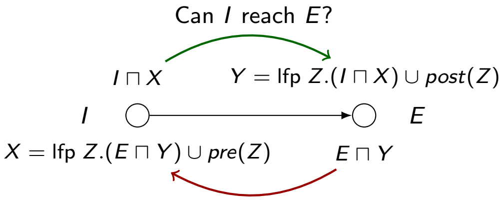
(P. Cousot 1978)

Iterating abstractions for increased precision

Iterate forward and backward analysis

$$\text{gfp}\langle X, Y\rangle. \quad \langle\ \text{lfp } Z.(E \sqcap Y) \sqcup \mathit{pre}(Z),\ \ \text{lfp } Z.(I \sqcap X) \sqcup \mathit{post}(Z)\ \rangle$$

Refined Abstract Analyses through gfp Iteration



(P. Cousot 1978)

Iterating abstractions for increased precision

Iterate forward and backward analysis

$$\text{gfp}\langle X, Y\rangle. \quad \langle \ \text{lfp } Z.(E \sqcap Y) \sqcup pre(Z), \ \ \text{lfp } Z.(I \sqcap X) \sqcup post(Z) \ \rangle$$

In propositional logic, it is the case that $pre = post$

$$\text{gfp}\langle X, Y\rangle. \quad \langle \ post^A_\phi(\top \sqcap Y), \ \ post^A_\phi(\top \sqcap X) \ \rangle$$

Fixed point semantics: gfp $post^A_\phi$

Fixed Point Semantics

Fixed point semantics example: $\phi = a \wedge (\neg a \vee b)$

$$
\begin{aligned}
post^A_\phi(\top) &= \langle a \mapsto \mathrm{t} \rangle \sqcap \top & &= \langle a \mapsto \mathrm{t} \rangle \\
post^A_\phi(\langle a \mapsto \mathrm{t} \rangle) &= \langle a \mapsto \mathrm{t} \rangle \sqcap (\bot \sqcup \langle a \mapsto \mathrm{t}, b \mapsto \mathrm{t} \rangle) & &= \langle a \mapsto \mathrm{t}, b \mapsto \mathrm{t} \rangle
\end{aligned}
$$

Fixed Point Semantics

Fixed point semantics example: $\phi = a \wedge (\neg a \vee b)$

$$post^A_\phi(\top) = \langle a \mapsto \mathrm{t} \rangle \sqcap \top = \langle a \mapsto \mathrm{t} \rangle$$

$$post^A_\phi(\langle a \mapsto \mathrm{t} \rangle) = \langle a \mapsto \mathrm{t} \rangle \sqcap (\bot \sqcup \langle a \mapsto \mathrm{t}, b \mapsto \mathrm{t} \rangle) = \langle a \mapsto \mathrm{t}, b \mapsto \mathrm{t} \rangle$$

Boolean Constraint Propagation

```
BCP(φ, ρ) {
    repeat
        ρ′ ← ρ;
        for Clause c ∈ φ do  ρ ← unit(c, ρ);
    until ρ = ρ′ ;
}
```

Fixed Point Semantics

DPLL operates over an abstract domain.
The unit rule is the best abstract transformer.
BCP is the gfp semantics of the abstract transformer

Fixed point semantics example: $\phi = a \wedge (\neg a \vee b)$

$$
\begin{aligned}
post^{A}_{\phi}(\top) &= \langle a \mapsto \mathrm{t} \rangle \sqcap \top && = \langle a \mapsto \mathrm{t} \rangle \\
post^{A}_{\phi}(\langle a \mapsto \mathrm{t} \rangle) &= \langle a \mapsto \mathrm{t} \rangle \sqcap (\bot \sqcup \langle a \mapsto \mathrm{t}, b \mapsto \mathrm{t} \rangle) && = \langle a \mapsto \mathrm{t}, b \mapsto \mathrm{t} \rangle
\end{aligned}
$$

Boolean Constraint Propagation

BCP(ϕ, ρ) {
 repeat
  $\rho' \leftarrow \rho$;
  for *Clause* $c \in \phi$ **do** $\rho \leftarrow \mathrm{unit}(c, \rho)$;
 until $\rho = \rho'$;
}

Partitioning of gfp Semantics

Example

$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

$$\text{gfp } post_{\phi}^{A} = \langle x \mapsto \mathrm{t}, y \mapsto \mathrm{f} \rangle \rightarrow \text{too imprecise}$$

Partitioning of gfp Semantics

Example

$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

$$\text{gfp } post_\phi^A = \langle x \mapsto \text{t}, y \mapsto \text{f} \rangle \rightarrow \text{too imprecise}$$

Trace partitioning (Mauborgne and Rival 2005)

Partition gfp $post_\phi^A(X)$ using transformers F and F'

1. $$\text{gfp } X.(post_\phi^A(X) \sqcap F(X)) \qquad \rightarrow F(X) = post_{\neg z}^A(X)$$
2. $$\text{gfp } X.(post_\phi^A(X) \sqcap F'(X)) \qquad \rightarrow F'(X) = post_{\neg x \vee z}^A(X)$$

Partitioning of gfp Semantics

DPLL operates over an abstract domain.
The unit rule is the best abstract transformer.
BCP is the gfp semantics of the abstract transformer
Decisions and learning are dynamic construction of trace partitionings

Example

$$\phi = x \wedge (\neg x \vee \neg y) \wedge (y \vee z \vee \neg w) \wedge (y \vee z \vee w)$$

$$\text{gfp } post_{\phi}^{A} = \langle x \mapsto \text{t}, y \mapsto \text{f} \rangle$$ → too imprecise

Trace partitioning (Mauborgne and Rival 2005)

Partition gfp $post_{\phi}^{A}(X)$ using transformers F and F'

1. $$\text{gfp } X.(post_{\phi}^{A}(X) \sqcap F(X)) \qquad \rightarrow F(X) = post_{\neg z}^{A}(X)$$
2. $$\text{gfp } X.(post_{\phi}^{A}(X) \sqcap F'(X)) \qquad \rightarrow F'(X) = post_{\neg x \vee z}^{A}(X)$$

DPLL is Abstract Interpretation

1. DPLL operates over an abstract domain
2. The unit rule is the best abstract transformer.
3. BCP is the gfp semantics of the abstract transformer
4. Decisions and learning are dynamic construction of trace partitionings

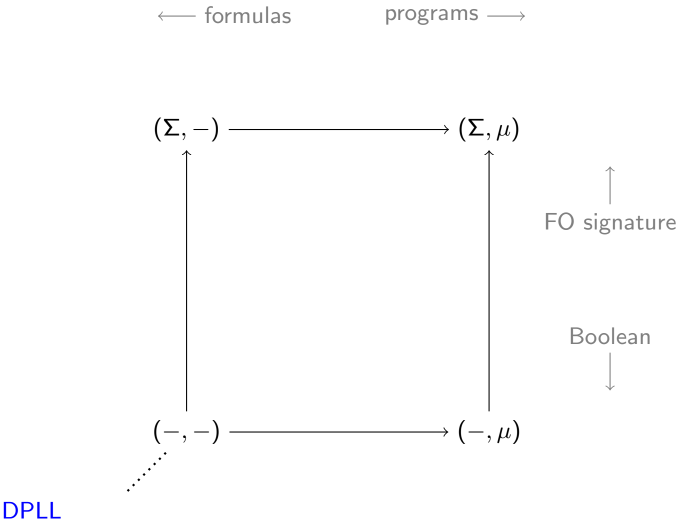
DPLL is Abstract Interpretation

1. DPLL operates over an abstract domain
2. The unit rule is the best abstract transformer.
3. BCP is the gfp semantics of the abstract transformer
4. Decisions and learning are dynamic construction of trace partitionings

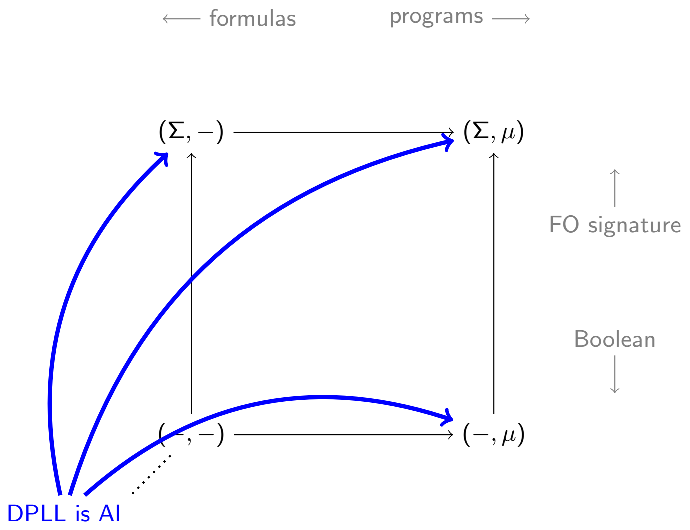
DPLL = AI + gfp semantics + dynamic trace partioning

- DPLL operates over an abstraction to compute a precise concrete result
- DPLL iteratively computes a minimal "error-preserving" transformer

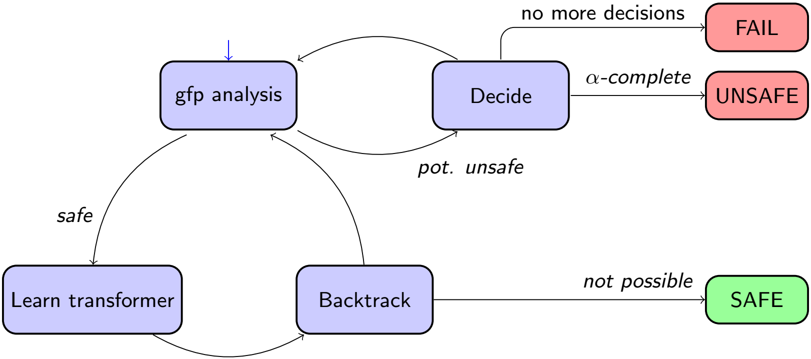
World Domination through Abstract Interpretation



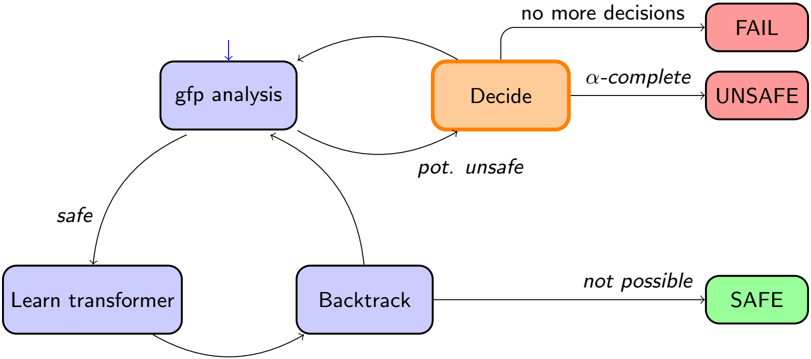
World Domination through Abstract Interpretation



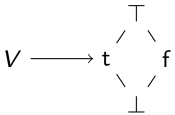
Generalising DPLL



Generalising DPLL



Generalising Decisions



Complementable decompositions

Take $a = \langle x \mapsto \mathrm{t}, y \mapsto \mathrm{f} \rangle \in \mathrm{Var} \to \wp(\{\mathrm{t}, \mathrm{f}\})$.

- Complement: $\neg a = \langle x \mapsto \mathrm{f} \rangle \vee \langle y \mapsto \mathrm{t} \rangle$
  Not precisely expressible in $\mathrm{Var} \to \wp(\{\mathrm{t}, \mathrm{f}\})$.
- Decomposition: $a = \langle x \mapsto \mathrm{t} \rangle \sqcap \langle y \mapsto \mathrm{f} \rangle$
  Each element of the decomposition has a precise complement.

Decisions are made over complementable meet-irreducible elements.

Decisions in Numeric Domains

Intervals

$a = \langle x \in [0, 10], y \in [-\infty, 0] \rangle$ has no precise complements

$$a = \underbrace{\langle x \geq 0 \rangle}_{\neg\langle x<0 \rangle} \sqcap \underbrace{\langle x \leq 10 \rangle}_{\neg\langle x>10 \rangle} \sqcap \underbrace{\langle y \leq 0 \rangle}_{\neg\langle y>0 \rangle}$$

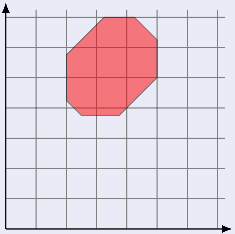
Decisions in Numeric Domains

Intervals

$a = \langle x \in [0, 10], y \in [-\infty, 0] \rangle$ has no precise complements

$$a = \underbrace{\langle x \geq 0 \rangle}_{\neg \langle x < 0 \rangle} \sqcap \underbrace{\langle x \leq 10 \rangle}_{\neg \langle x > 10 \rangle} \sqcap \underbrace{\langle y \leq 0 \rangle}_{\neg \langle y > 0 \rangle}$$

Octagons



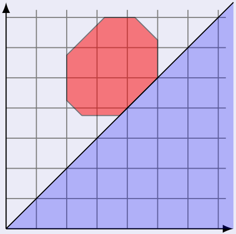
Decisions in Numeric Domains

Intervals

$a = \langle x \in [0, 10], y \in [-\infty, 0] \rangle$ has no precise complements

$$a = \underbrace{\langle x \geq 0 \rangle}_{\neg\langle x<0 \rangle} \sqcap \underbrace{\langle x \leq 10 \rangle}_{\neg\langle x>10 \rangle} \sqcap \underbrace{\langle y \leq 0 \rangle}_{\neg\langle y>0 \rangle}$$

Octagons



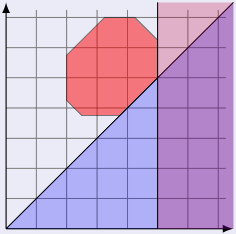
Decisions in Numeric Domains

Intervals

$a = \langle x \in [0, 10], y \in [-\infty, 0] \rangle$ has no precise complements

$$a = \underbrace{\langle x \geq 0 \rangle}_{\neg\langle x<0 \rangle} \sqcap \underbrace{\langle x \leq 10 \rangle}_{\neg\langle x>10 \rangle} \sqcap \underbrace{\langle y \leq 0 \rangle}_{\neg\langle y>0 \rangle}$$

Octagons



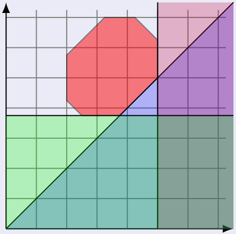
Decisions in Numeric Domains

Intervals

$a = \langle x \in [0, 10], y \in [-\infty, 0] \rangle$ has no precise complements

$$a = \underbrace{\langle x \geq 0 \rangle}_{\neg\langle x<0 \rangle} \sqcap \underbrace{\langle x \leq 10 \rangle}_{\neg\langle x>10 \rangle} \sqcap \underbrace{\langle y \leq 0 \rangle}_{\neg\langle y>0 \rangle}$$

Octagons



Decisions in Trace Abstractions

Control-flow abstraction

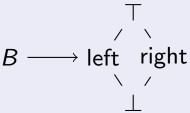
Set of control-flow branches B

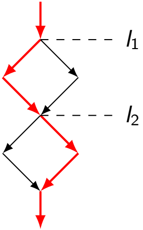
$$B \longrightarrow \begin{array}{ccc} & \top & \\ / & & \backslash \\ \text{left} & & \text{right} \\ \backslash & & / \\ & \bot & \end{array}$$

Decisions in Trace Abstractions

Control-flow abstraction

Set of control-flow branches B

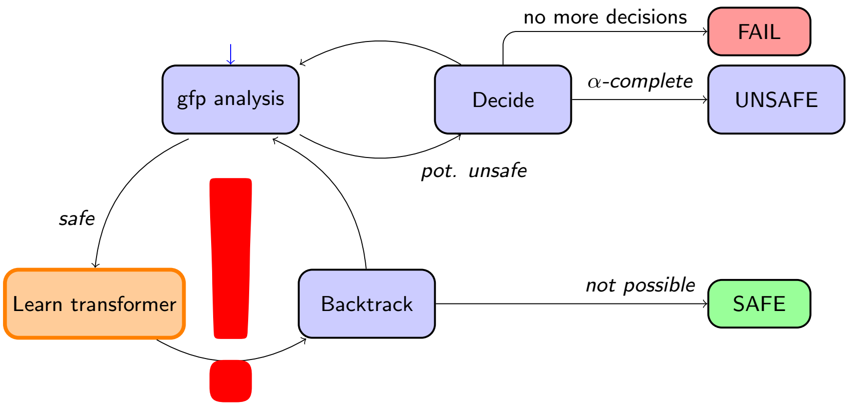




Decomposition

$$
\begin{aligned}
a &= \langle l_1 \mapsto \text{left}, l_2 \mapsto \text{right} \rangle \\
&= \underbrace{\langle l_1 \mapsto \text{left} \rangle}_{\neg \langle l_1 \mapsto \text{right} \rangle} \sqcap \underbrace{\langle l_2 \mapsto \text{right} \rangle}_{\neg \langle l_2 \mapsto \text{left} \rangle}
\end{aligned}
$$

Generalising DPLL



DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

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

DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

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

DL0

$\overline{1}$

DPLL Learning Example

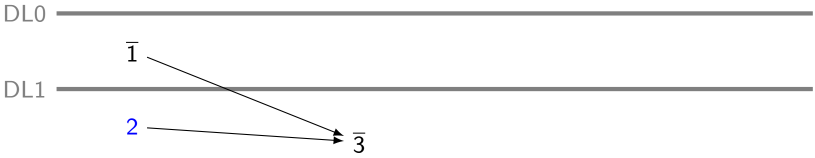
Learn deeper reason for a conflict using an implication graph

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

DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

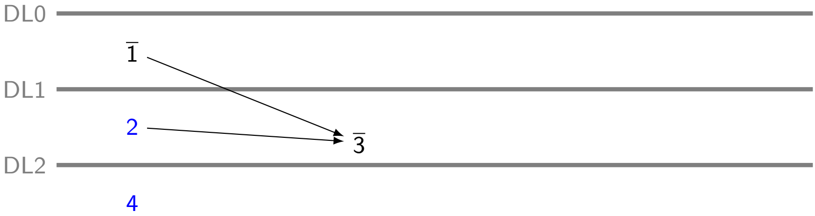
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

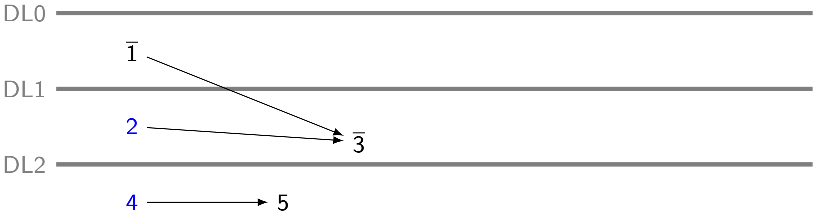
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

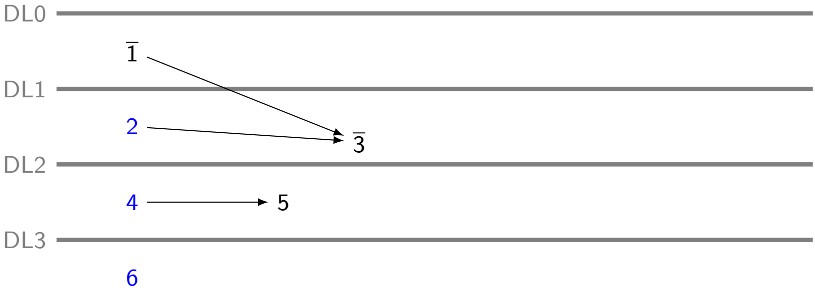
$$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$$



DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

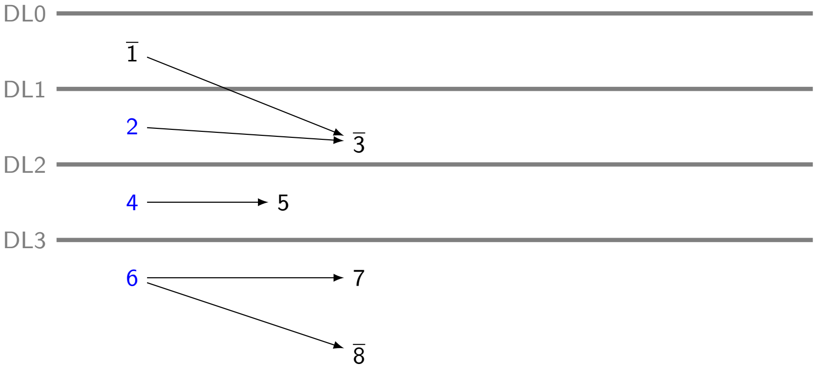
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

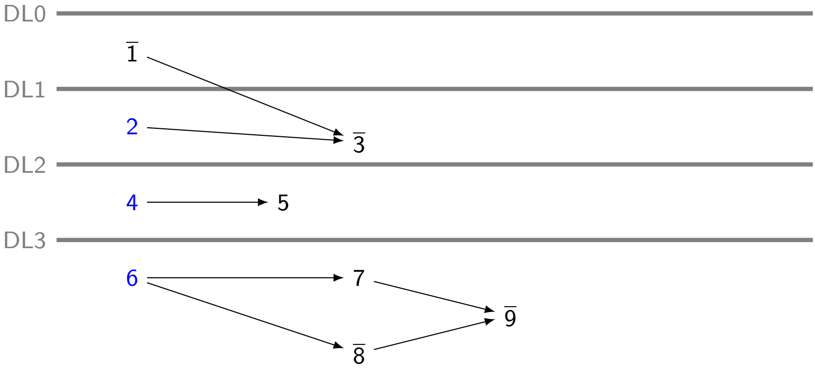
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

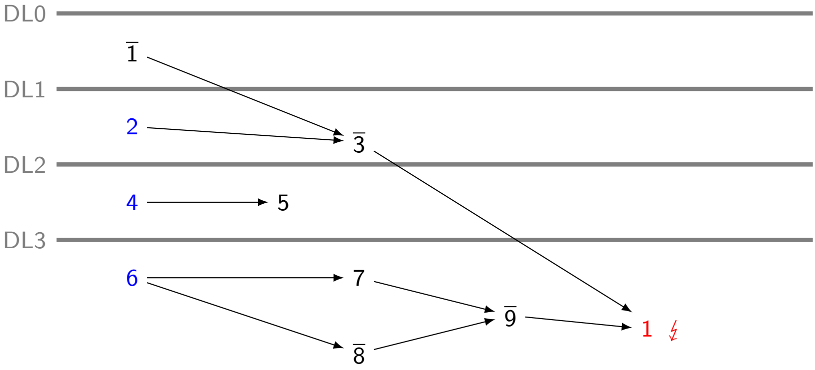
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

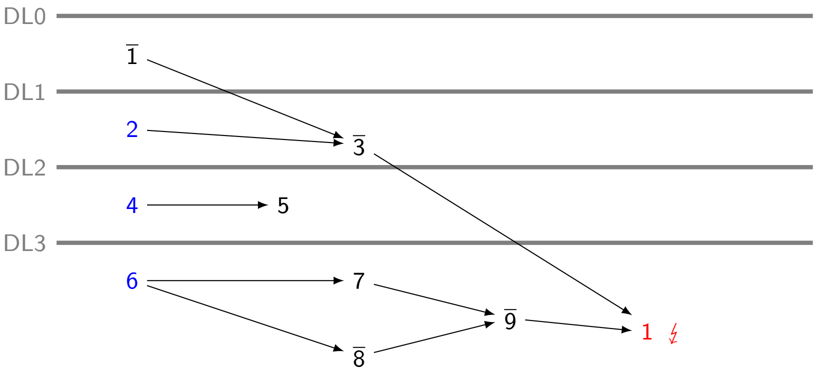
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$



DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

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

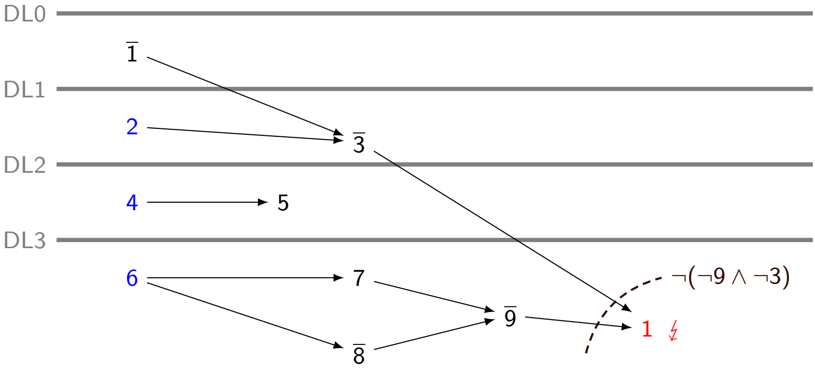


Every cut that disconnects the roots from the error is a reason

DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

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

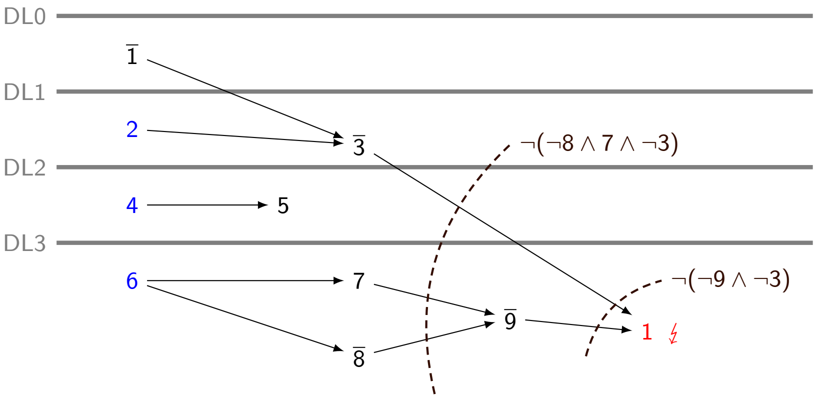


Every cut that disconnects the roots from the error is a reason

DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

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

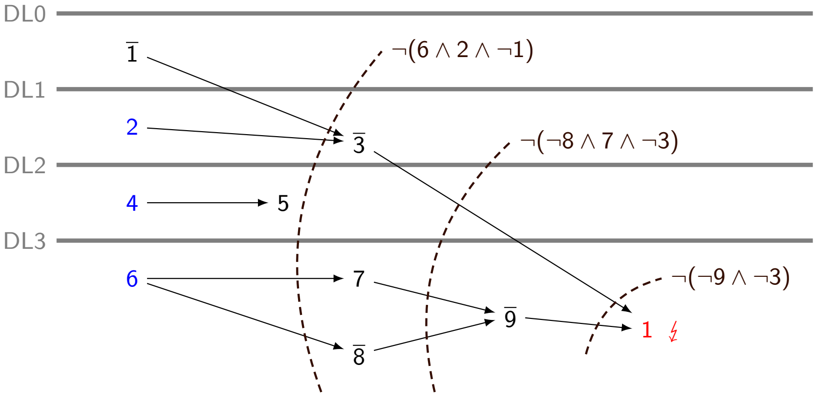


Every cut that disconnects the roots from the error is a reason

DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

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

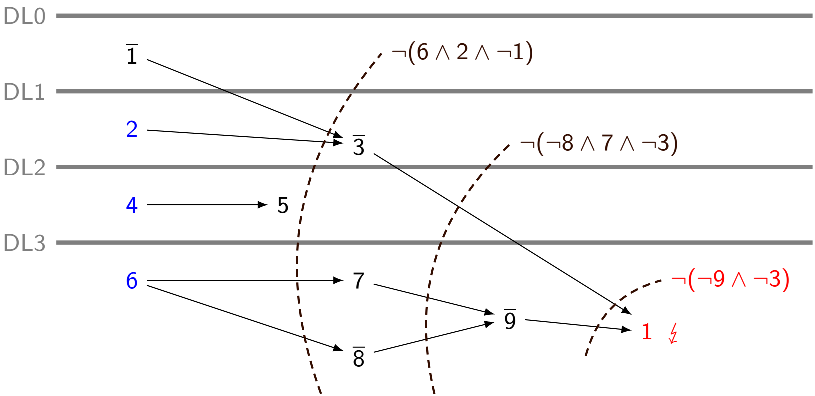


Every cut that disconnects the roots from the error is a reason

DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

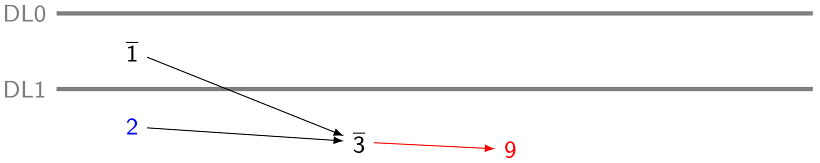
$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$
$\wedge (9 \vee 3)$



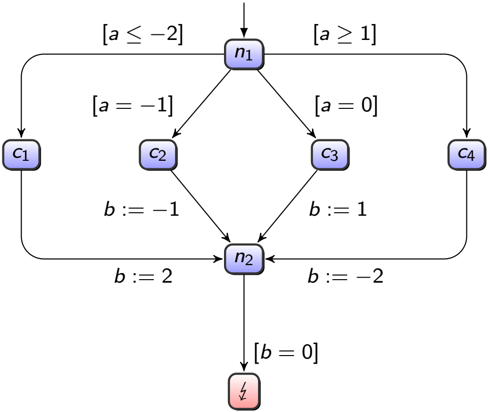
DPLL Learning Example

Learn deeper reason for a conflict using an implication graph

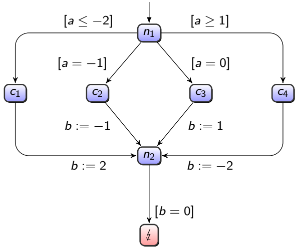
$$\neg 1 \wedge (1 \vee \neg 2 \vee \neg 3) \wedge (\neg 4 \vee 5) \wedge (\neg 6 \vee 7) \wedge (\neg 6 \vee \neg 8) \wedge (\neg 7 \vee 8 \vee \neg 9) \wedge (3 \vee 9 \vee 1)$$
$$\wedge (9 \vee 3)$$



Putting it All Together: Simple Program Example

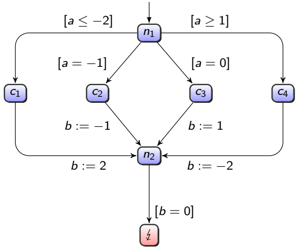


Abstract Implication Graph

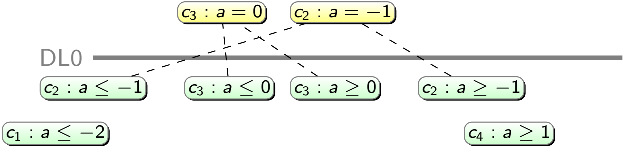


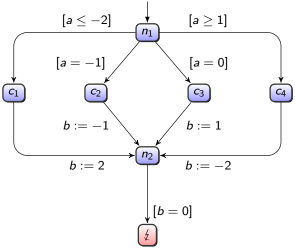
Abstract Implication Graph

DL0

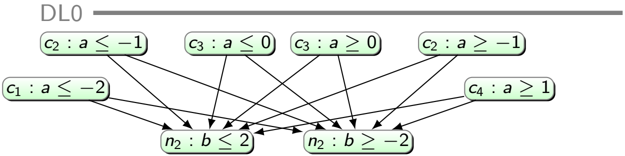


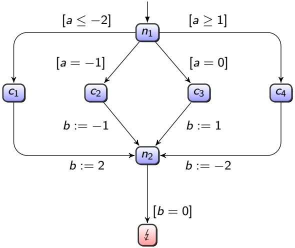
Abstract Implication Graph



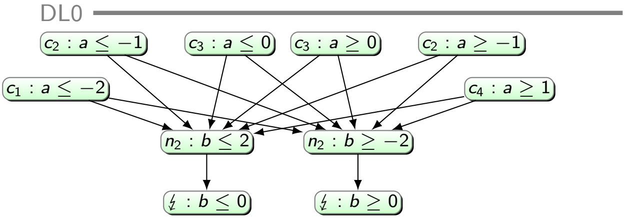


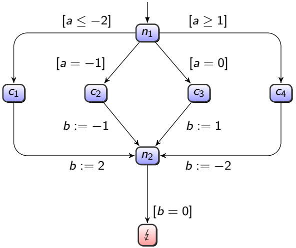
Abstract Implication Graph



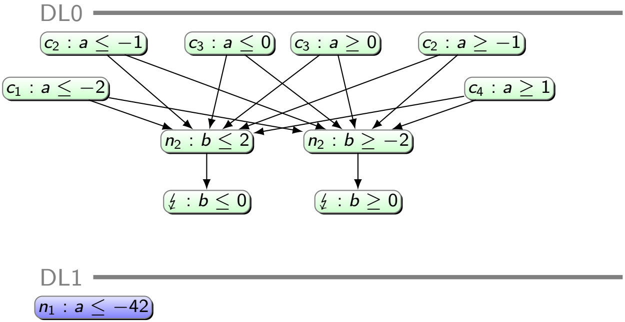


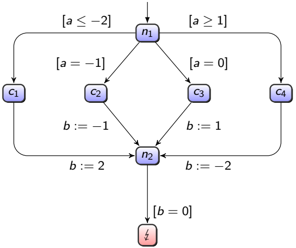
Abstract Implication Graph



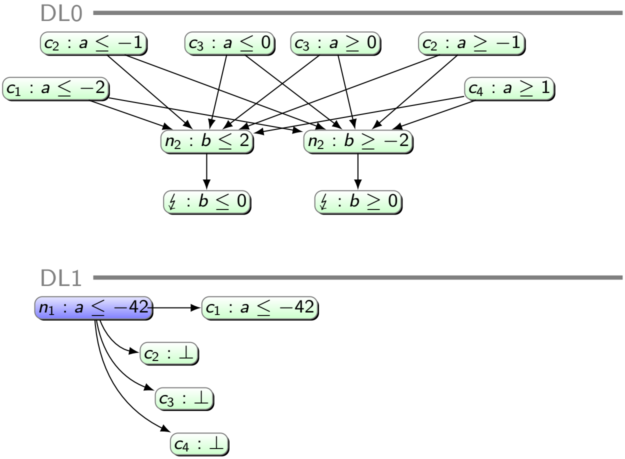


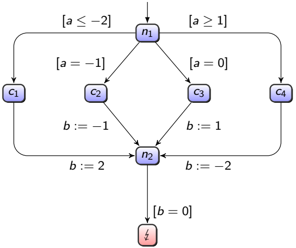
Abstract Implication Graph



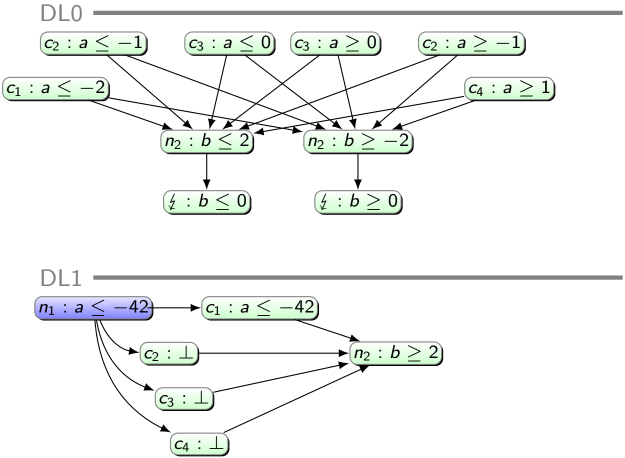


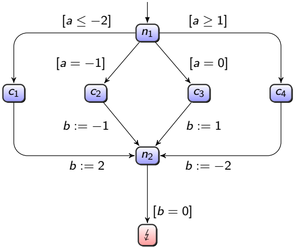
Abstract Implication Graph



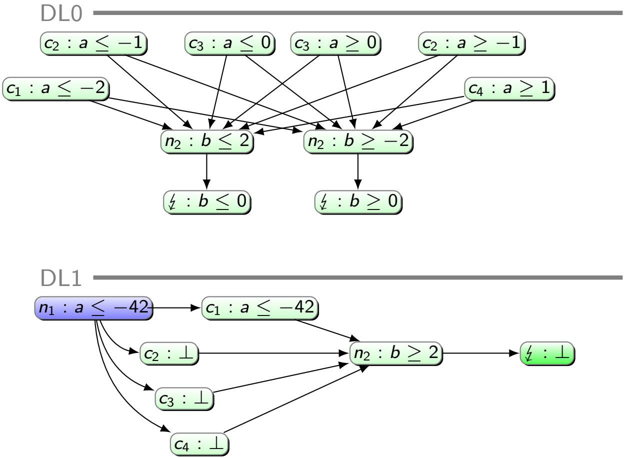


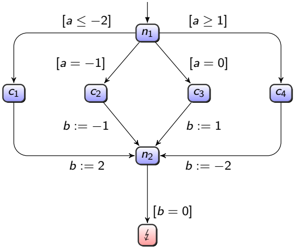
Abstract Implication Graph





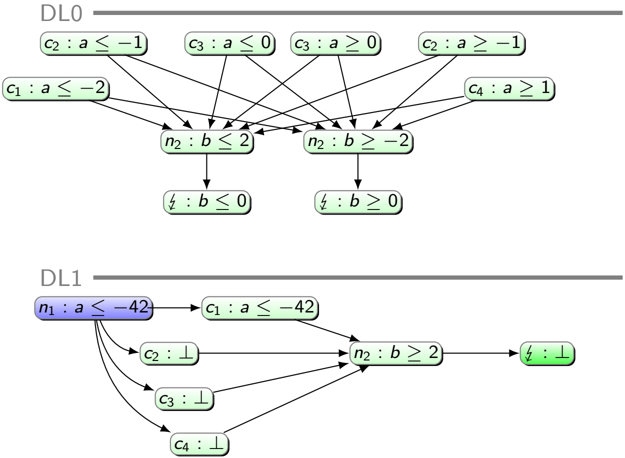
Abstract Implication Graph

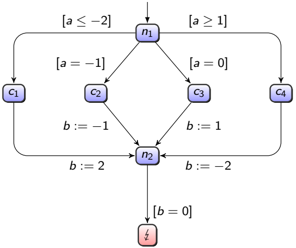




SAFE

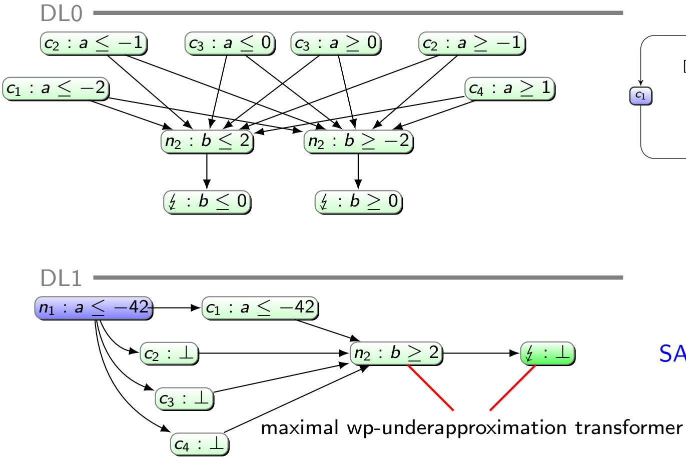
Abstract Implication Graph

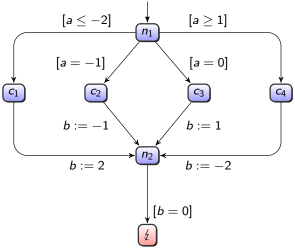




SAFE $\rightarrow$ Generalise!

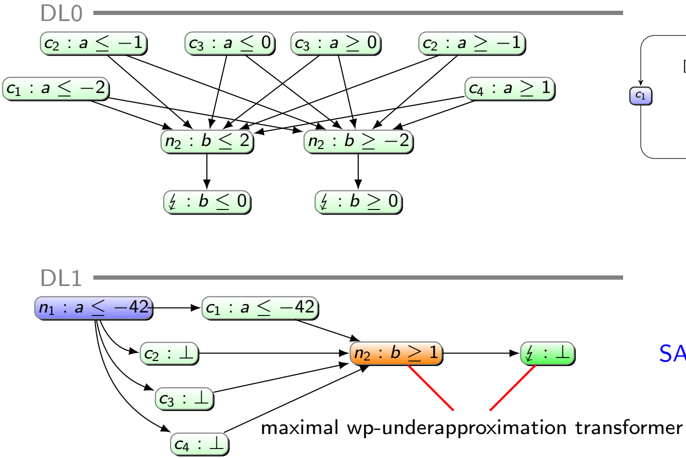
Abstract Implication Graph

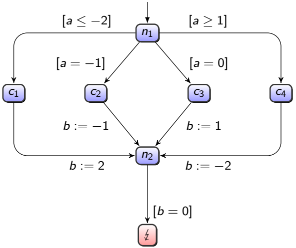




SAFE $\rightarrow$ Generalise!

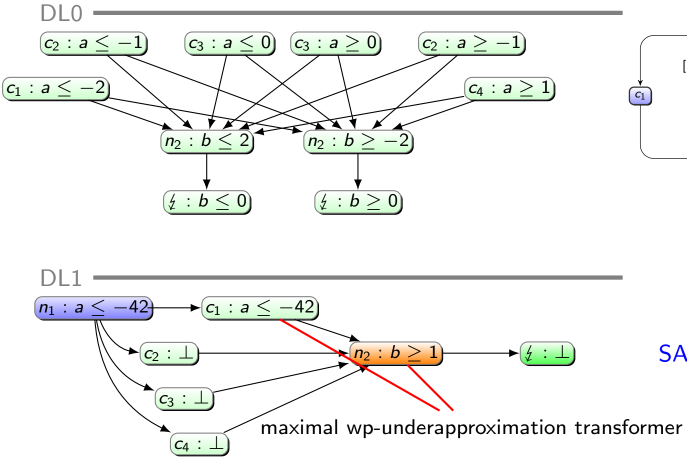
Abstract Implication Graph

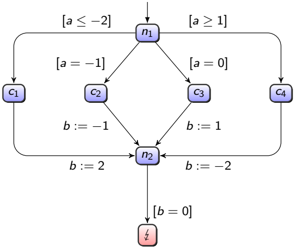




SAFE $\rightarrow$ Generalise!

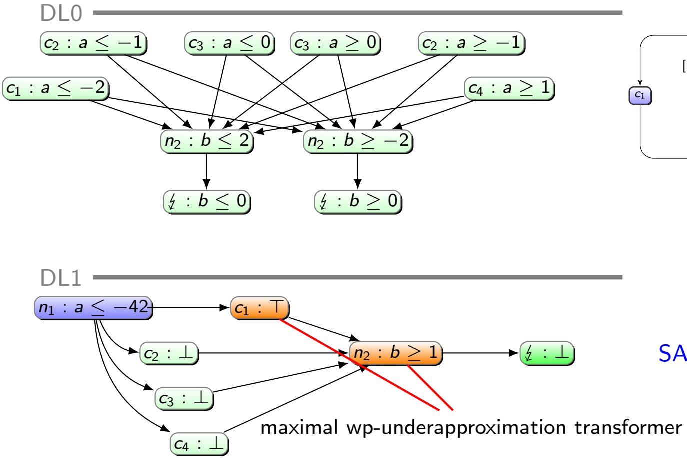
Abstract Implication Graph

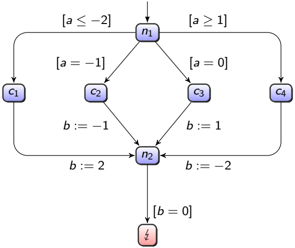




SAFE $\rightarrow$ Generalise!

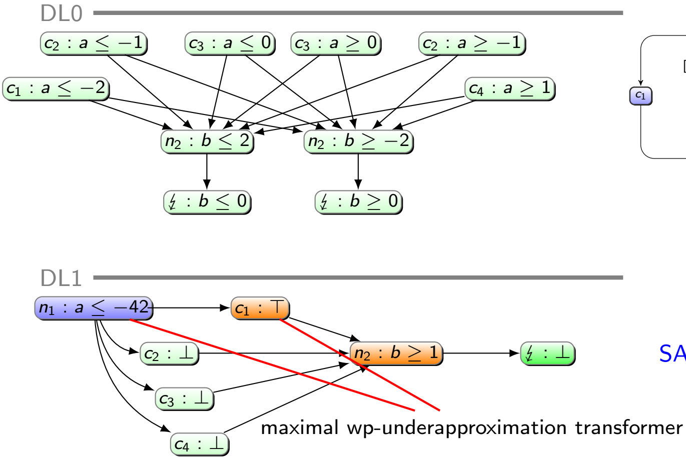
Abstract Implication Graph

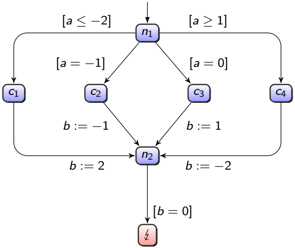




SAFE $\rightarrow$ Generalise!

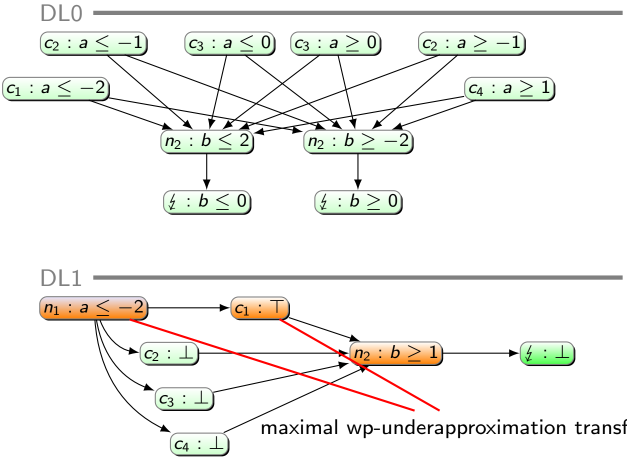
Abstract Implication Graph

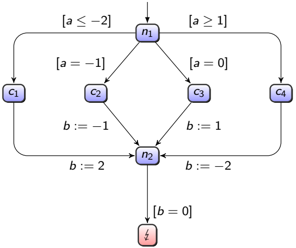




SAFE $\rightarrow$ Generalise!

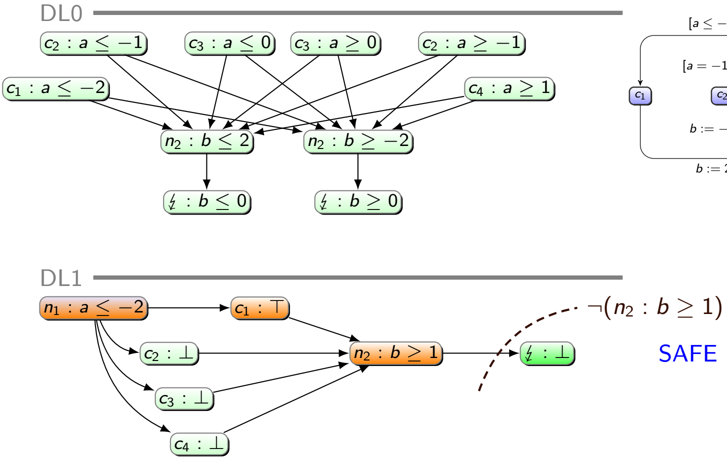
Abstract Implication Graph

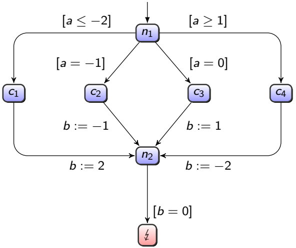




SAFE $\rightarrow$ Generalise!

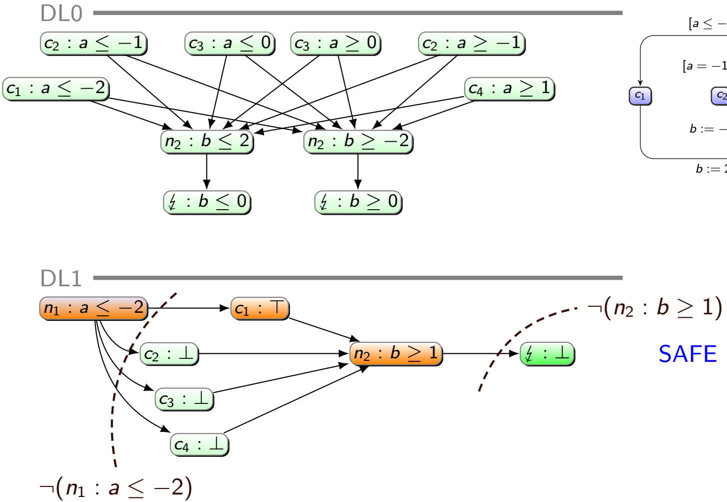
Abstract Implication Graph

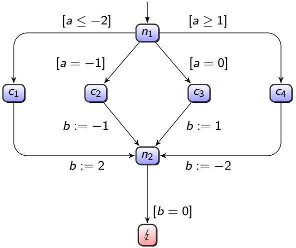




SAFE $\rightarrow$ find cut

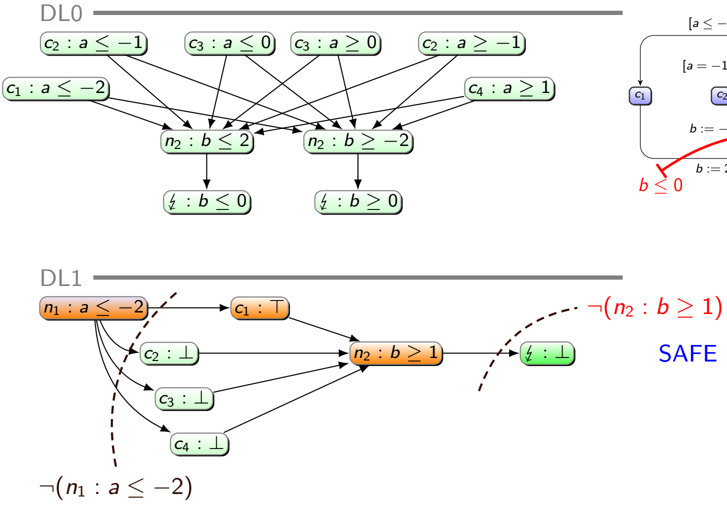
Abstract Implication Graph

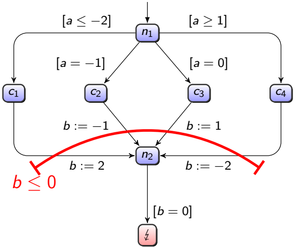




SAFE $\rightarrow$ find cut

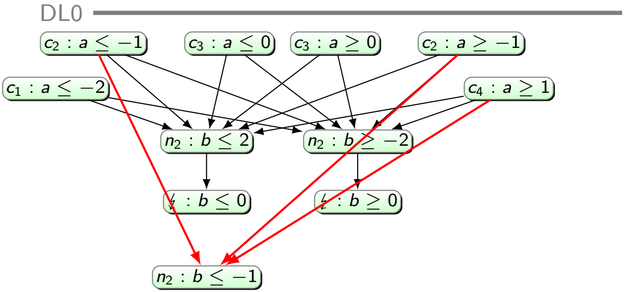
Abstract Implication Graph

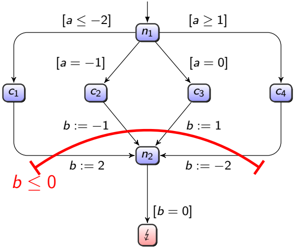




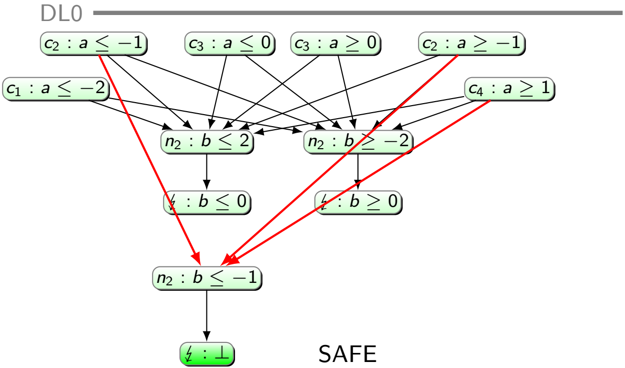
SAFE $\rightarrow$ find cut

Abstract Implication Graph

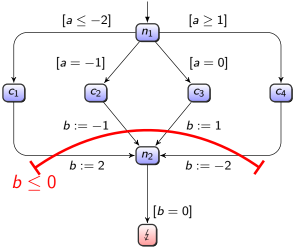




Abstract Implication Graph

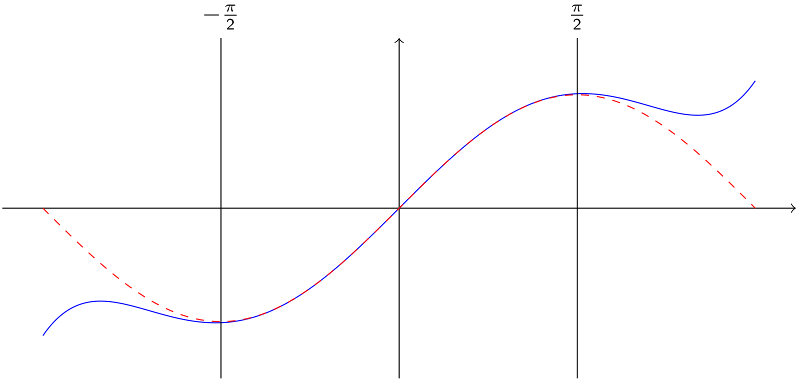


SAFE

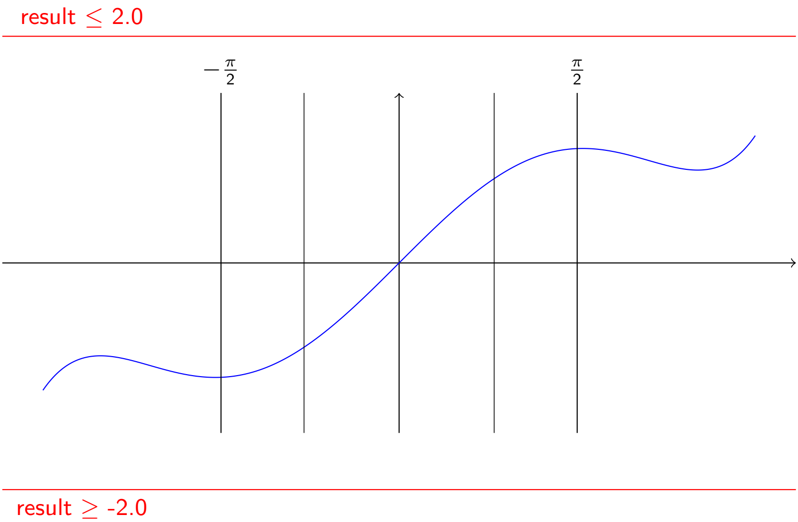


Demo

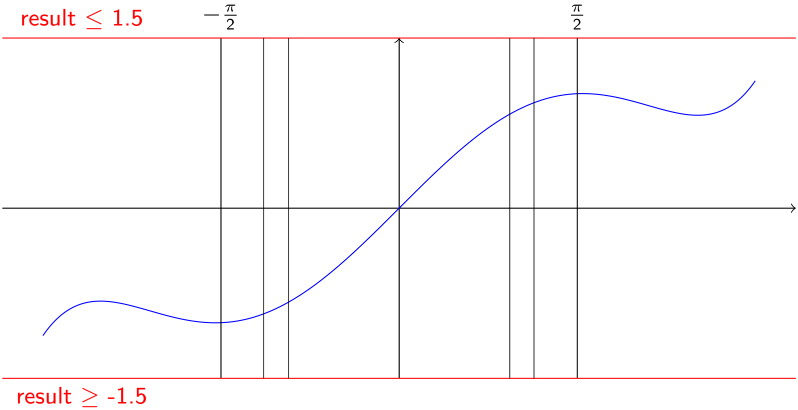
Property-dependent Trace Partitioning



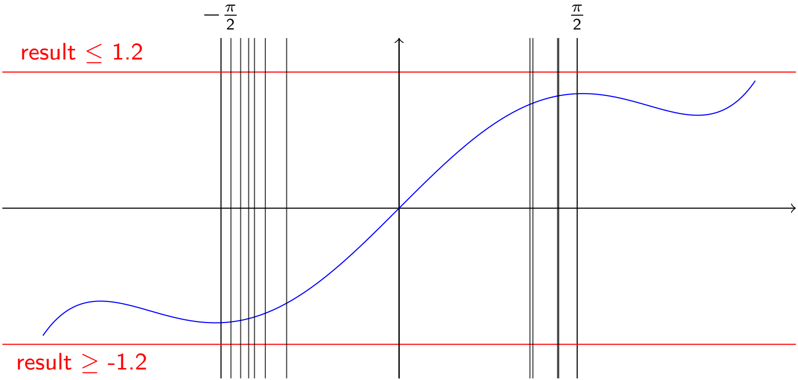
Property-dependent Trace Partitioning



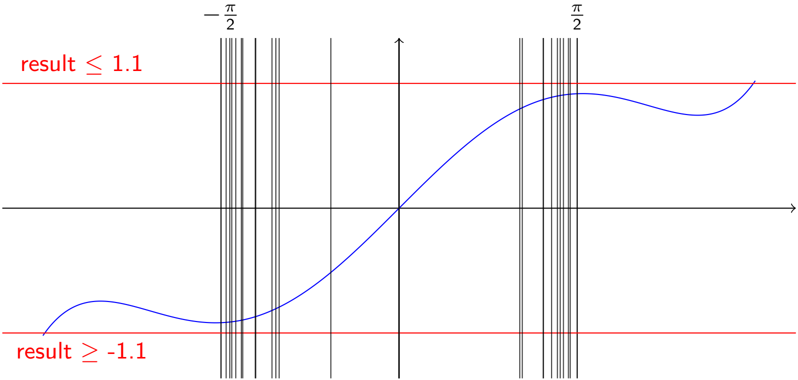
Property-dependent Trace Partitioning



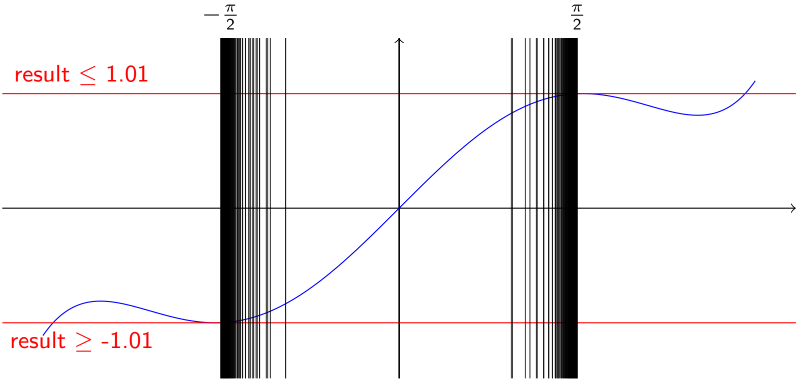
Property-dependent Trace Partitioning



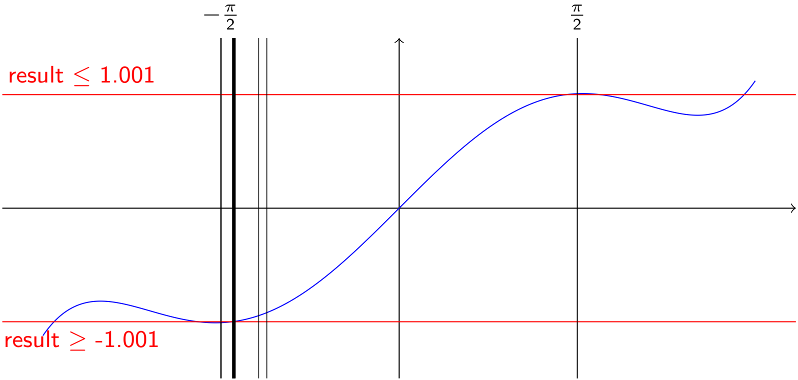
Property-dependent Trace Partitioning



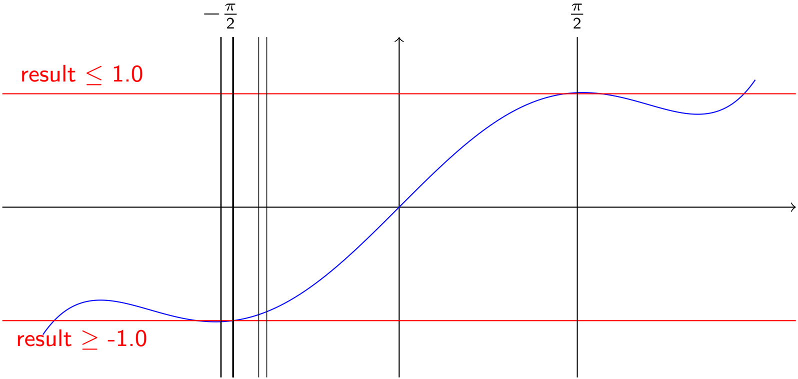
Property-dependent Trace Partitioning



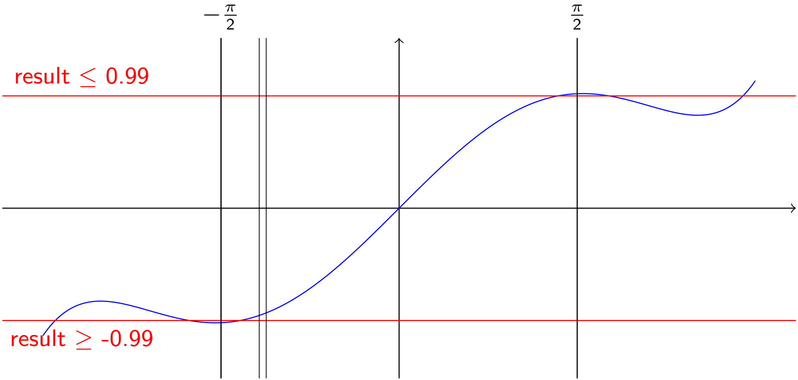
Property-dependent Trace Partitioning



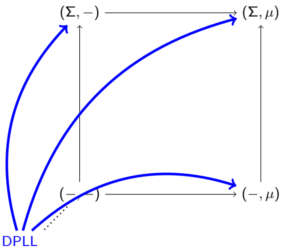
Property-dependent Trace Partitioning



Property-dependent Trace Partitioning



Conclusion



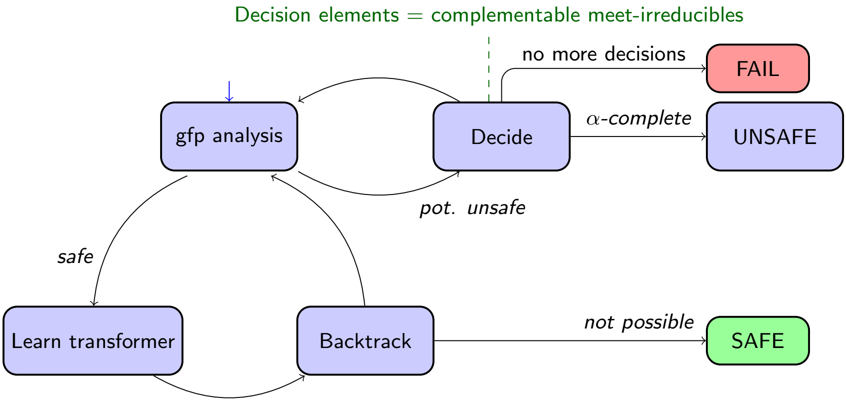
- A DPLL solver . . .
  - is an abstract interpreter over the Cartesian Boolean abstraction
  - AI + gfp-iteration + dynamic trace partitioning
  - iteratively computes a minimal property-preserving abstract transformer

- We can simply change the domain to lift DPLL to richer logics and programs

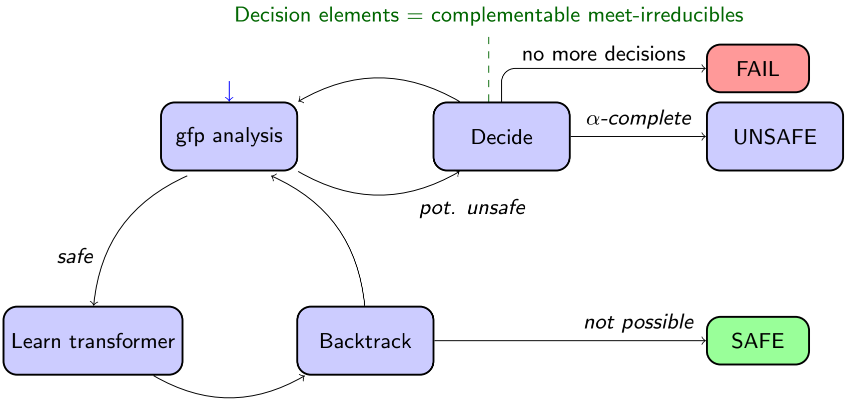
Thank you for your attention

Appendix

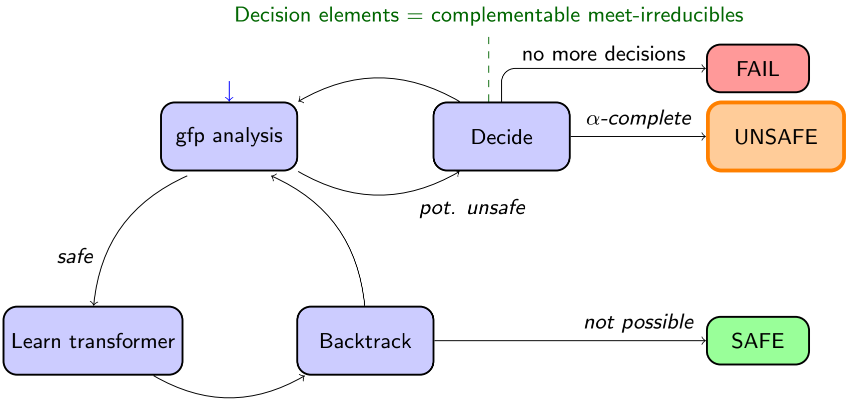
Generalising DPLL



Generalising DPLL



Generalising DPLL



Satisfying Assignments and Proofs

Satisfying assignments

DPLL constructs restricted transformers. When can we stop?

$$\phi = (a \vee b) \wedge (\neg b \vee c)$$

$$r = \langle a \mapsto \mathrm{t}, c \mapsto \mathrm{t} \rangle$$

Satisfying Assignments and Proofs

Satisfying assignments

DPLL constructs restricted transformers. When can we stop?

$$\phi = (a \vee b) \wedge (\neg b \vee c)$$

$$r = \langle a \mapsto \mathrm{t}, c \mapsto \mathrm{t} \rangle$$

Construct restricted transformer

$$\widehat{F}_r(X) = post^A_\phi(X \sqcap r)$$

α-completeness: $\widehat{F}_r \circ \alpha = \alpha \circ F_r$

γ-completeness: $\gamma \circ \widehat{F}_r = F_r \circ \gamma$

Satisfying Assignments and Proofs

Satisfying assignments

DPLL constructs restricted transformers. When can we stop?

$$\phi = (a \vee b) \wedge (\neg b \vee c)$$

$$r = \langle a \mapsto \mathrm{t}, c \mapsto \mathrm{t} \rangle$$

Construct restricted transformer

$$\widehat{F}_r(X) = post^A_\phi(X \sqcap r)$$

α-completeness: $\widehat{F}_r \circ \alpha = \alpha \circ F_r$

γ-completeness: $\gamma \circ \widehat{F}_r = F_r \circ \gamma$

$\widehat{F}_r$ is a sufficient to show SAT $\iff$ $\widehat{F}_r$ is α-complete and $\widehat{F}_r(\top) \sqsupset \bot$

Satisfying Assignments and Proofs

Satisfying assignments

DPLL constructs restricted transformers. When can we stop?

$$\phi = (a \vee b) \wedge (\neg b \vee c)$$

$$r = \langle a \mapsto \mathrm{t}, c \mapsto \mathrm{t} \rangle$$

Construct restricted transformer

$$\widehat{F}_r(X) = post^A_\phi(X \sqcap r)$$

$$\alpha\text{-completeness: } \widehat{F}_r \circ \alpha = \alpha \circ F_r$$

$$\gamma\text{-completeness: } \gamma \circ \widehat{F}_r = F_r \circ \gamma$$

$\widehat{F}_r$ is a sufficient to show SAT $\iff$ $\widehat{F}_r$ is α-complete and $\widehat{F}_r(\top) \sqsupset \bot$

$\widehat{F}_r$ is witness to SAT assignments $\iff$ $\widehat{F}_r$ is γ-complete and $\widehat{F}_r(\top) \sqsupset \bot$

Satisfying Assignments and Proofs

Interval counter-example

When can we stop refining in a program?

Satisfying Assignments and Proofs

Interval counter-example

When can we stop refining in a program?

```
int x, y; x = y; assert(x != 0);
```

$$\langle y = 0 \rangle$$

Let $\widehat{F}$ be the transformer from the initial state to the error location.

Satisfying Assignments and Proofs

Interval counter-example

When can we stop refining in a program?

```
int x, y; x = y; assert(x != 0);
```

$$\langle y = 0 \rangle$$

Let $\widehat{F}$ be the transformer from the initial state to the error location.

Witness for counter-example:

$$\widehat{F}_{\langle y=0 \rangle}(X) = \widehat{F}(X \sqcap \langle y = 0 \rangle)$$

$$\widehat{F}_{\langle y=0 \rangle} \text{ is } \gamma\text{-complete and } \widehat{F}_{\langle y=0 \rangle}(\top) \sqsupset \bot$$