Advanced Features of SAT Solvers

IA085: Satisfiability and Automated Reasoning

Martin Jonáš

FI MUNI, Spring 2024

Last Time

- Conflict-Driven Clause Learning (CDCL): DPLL + clause learning + backjumping
- literal decision heuristics
- restarts

Incremental SAT solving

Normal Usage

1. Call `solve(Φ)`.
2. Get the answer (+ possibly a model).
3. ???
4. Profit.

Incremental Usage

Some applications issue incremental queries:

1. Is Φ_1 satisfiable?
2. Is $\Phi_1 \cup \Phi_2$ satisfiable?
3. Is $\Phi_1 \cup \Phi_2 \cup \Phi_3$ satisfiable?
4. ...

Examples

- symbolic execution
- planning

Incremental Usage

Modern solvers support incremental interface:

1. Add clauses Φ_1.
2. Call `solve()`.
3. Do something with the answer.
4. Add clauses Φ_2.
5. Call `solve()`.
6. Do something with the answer.
7. Add clauses Φ_3.
8. ...

Why is this better than calling `solve` for Φ_1, for $\Phi_1 \cup \Phi_2$, for $\Phi_1 \cup \Phi_2 \cup \Phi_3$, ...?

Solving Under Assumptions

What if we need to solve multiple queries that are not incremental, but differ in some literals?

- Is $\Phi \wedge A$ satisfiable?
- Is $\Phi \wedge \neg A \wedge B$ satisfiable?
- Is $\Phi \wedge \neg B \wedge D \wedge E$ satisfiable?
- ...

Examples

- planning (common constraints + individual goals)
- package dependencies (common constraints + individual queries for installed packages)

Solving Under Assumptions

Solving under assumptions (MiniSAT)

- Add clauses Φ.
- Call `solve([A])` and do something with the result.
- Call `solve([¬A, B])` and do something with the result.
- Call `solve([¬B, D, E])` and do something with the result.
- ...

The calls to `solve()` reuse the learnt clauses!

Solving Under Assumptions (alternative API)

Solving under assumptions (CaDiCaL)

- Add clauses Φ.
- Call `assume(A)`.
- Call `solve()` and do something with the result.
- Call `assume(¬A)` and `assume(B)`.
- Call `solve()` and do something with the result.
- Call `assume(¬B)` and `assume(D)` and `assume(E)`.
- Call `solve()` and do something with the result.
- ...

Solving Under Assumptions: Implementation

```
solve([l1, l2, ..., lk])
```

- before the search, decide $l_1, l_2, \ldots, l_k$ on dummy decision levels before decisions level 0
- when backjumping before the real decision level 0, return UNSAT

Nice bonus

- when UNSAT, a slight modification of clause learning (last UIP) can compute a conflict clause $C = \neg\mu$ with $\mu \subseteq \{l_1, l_2, \ldots, l_k\}$
- identifies failed assumptions that contributed to the unsatisfiability

Varying Clauses

What if we need to vary additional clauses, not only literals?

- Is $\Phi \wedge C_1$ satisfiable?
- Is $\Phi \wedge C_2 \wedge C_3$ satisfiable?
- Is $\Phi \wedge C_4$ satisfiable?
- $\ldots$

Activation Literals

Solution

- add a new activation literal to each clause that should be possible to disable

$$\Phi \land C_2 \land C_3 \quad \rightsquigarrow \quad \Phi \land (\neg A_2 \lor C_2) \land (\neg A_3 \lor C_3)$$

- use solving under assumptions to enable clauses
  - `solve([¬A2,¬A3])` $\equiv$ is Φ sat?
  - `solve([ A2,¬A3])` $\equiv$ is $\Phi \land C_2$ sat?
  - `solve([¬A2, A3])` $\equiv$ is $\Phi \land C_3$ sat?
  - `solve([ A2, A3])` $\equiv$ is $\Phi \land C_2 \land C_3$ sat?

Proof generation

Facts

- SAT solvers are used in safety-critical systems
- SAT solvers are pieces of software
- all software has bugs

Proof Generation

Facts

- SAT solvers are used in safety-critical systems
- SAT solvers are pieces of software
- all software has bugs
- ☹

Proof Generation

Facts

- SAT solvers are used in safety-critical systems
- SAT solvers are pieces of software
- all software has bugs
- ☹

Solution

- besides SAT/UNSAT answer, produce an artifact that can be independently checked
- for SAT results = model
- for UNSAT results = unsatisfiability proof

Resolution Proof Generation from DPLL

Recall

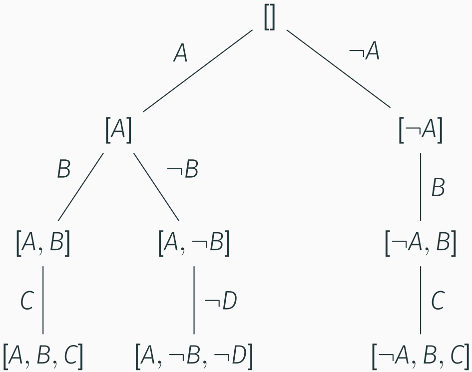
Each UNSAT run of DPLL corresponds to a tree resolution proof of unsatisfiability

Algorithm

- conflicting clauses (leaves) $\rightsquigarrow$ input clauses
- unit propagation steps $\rightsquigarrow$ resolution with the clause that triggered the unit propagation
- decision nodes $\rightsquigarrow$ resolution steps on the decided variable

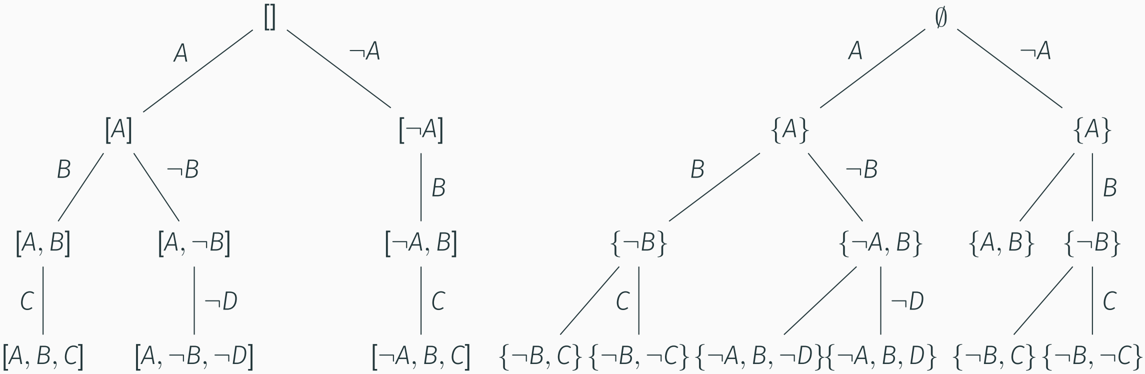
Resolution Proof Generation from DPLL: Example

$$\{\{A, B\}_1, \{\neg B, C\}_2, \{\neg B, \neg C\}_3, \{\neg A, \neg B, \neg D\}_4, \{\neg A, B, \neg D\}_5, \{\neg A, B, D\}_6\}$$



Resolution Proof Generation from DPLL: Example

$$\{\{A, B\}_1, \{\neg B, C\}_2, \{\neg B, \neg C\}_3, \{\neg A, \neg B, \neg D\}_4, \{\neg A, B, \neg D\}_5, \{\neg A, B, D\}_6\}$$



CDCL observations

- the final conflict was achieved by backtracked literals and unit propagated literals (no decisions, why?)
- the final conflict is derived by unit propagation from input clauses and learnt clauses
- the final conflict can be obtained by resolving input clauses and learnt clauses
- each learnt clause was obtained by resolving input clauses and previous learnt clauses

Resolution Proof Generation from CDCL

Algorithm

1. express the final conflict as resolution of input clauses and learnt clauses
2. while the proof contains a leaf that is a learnt clause, replace it by its resolution proof

Practical considerations

- the solver needs to remember for each learnt clause its antecedent clauses from which it was obtained
- might require significant amount of memory and makes the solver more complex

Clausal Proofs

For easier implementation: clausal proofs

- proof is a list of clauses
- each clause has to be entailed by some previous clauses (input or derived)
- SAT solver only outputs the learnt clauses during the search
- proof checker checks the entailment
- examples: DRUP, DRAT

Clausal Proof Formats

$$\{\{A, B\}_1, \{\neg B, C\}_2, \{\neg B, \neg C\}_3, \{\neg A, \neg B, \neg D\}_4, \{\neg A, B, \neg D\}_5, \{\neg A, B, D\}_6\}$$

DIMACS formula

```
p cnf 4 6
 1  2  0
-2  3  0
-2 -3  0
-1 -2 -4 0
-1  2 -4 0
-1  2  4 0
```

Clausal proof

```
-2 0
 1 0
-1 2 0
-1 0
 0
```

Reverse Unit Propagation (RUP)

$$\Phi \models (l_1 \vee l_2 \vee \ldots \vee l_n) \iff \Phi \wedge \neg l_1 \wedge \neg l_2 \wedge \ldots \wedge \neg l_n \models \bot$$

To check clause $C = \{l_1, l_2, \ldots, l_n\}$ using reverse unit propagation (RUP)

1. assign $\neg l_1, \neg l_2, \ldots, \neg l_n$
2. check that unit propagation produces a conflict

Reverse Unit Propagation

- obviously not complete
- sufficient for clauses learnt by CDCL, because it learns clauses that were conflicting by unit propagation
- previous example was RUP proof

Delete Reverse Unit Propagation (DRUP)

- proof checking of RUP requires checking large number of clauses
- some were actually deleted by the solver and are not needed for the proof anymore → express deleting (D) in the proof (DRUP)

DIMACS formula

```
p cnf 4 6
 1  2  0
-2  3  0
-2 -3  0
-1 -2 -4 0
-1  2 -4 0
-1  2  4 0
```

Clausal proof

```
  -2 0
d -2 3 0
d -2 -3 0
   1 0
  -1 2 0
  -1 0
   0
```

Clausal Proof Formats

Multiple clausal proof formats exist besides DRUP

- DRAT
- LRAT
- LPR
- ...

Most of them have efficient proof checkers (some even formally verified).

Clausal Proof Formats

Multiple clausal proof formats exist besides DRUP

- DRAT
- LRAT
- LPR
- ...

Most of them have efficient proof checkers (some even formally verified).

Challenge

- implement (D)RUP proof generation in your solver
- use e.g. DRAT-TRIM for proof checking (`https://www.cs.utexas.edu/~marijn/drat-trim/`)

Unsatisfiable Cores

Unsatisfiable Cores

Definition

For an unsatisfiable formula Φ in CNF, its subset of clauses $\Psi \subseteq \Phi$ is called unsatisfiable core if Ψ is unsatisfiable.

Important

The set Ψ does not have to be minimal.

Applications

- analysis of requirements
- package dependencies
- abstraction refinement

Proof-based algorithm

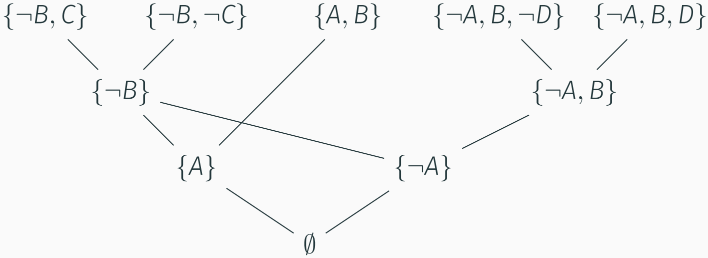
1. Compute a resolution proof of unsatisfiability of Φ.
2. Return the set $\Psi \subseteq \Phi$ of clauses that occur as leaves in the proof.

Unsatisfiable Cores: Proof-based Algorithm

$$\{\{A, B\}, \{D, \neg E\}, \{\neg B, C\}, \{\neg B, \neg C\}, \{B, \neg E, F\}, \{\neg A, \neg B, \neg D\}, \{\neg A, \neg F\}, \{\neg A, B, \neg D\}, \{\neg E, \neg F\}, \{\neg A, B, D\}\}$$

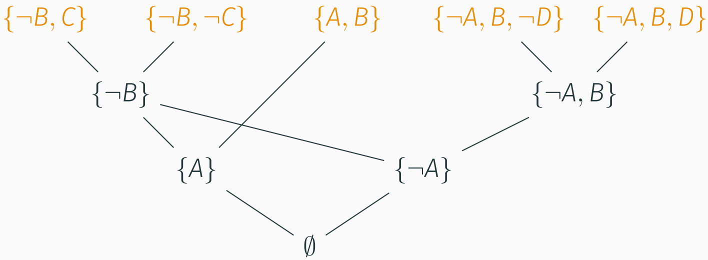
Unsatisfiable Cores: Proof-based Algorithm

$$\{\{A, B\}, \{D, \neg E\}, \{\neg B, C\}, \{\neg B, \neg C\}, \{B, \neg E, F\}, \{\neg A, \neg B, \neg D\}, \{\neg A, \neg F\}, \{\neg A, B, \neg D\}, \{\neg E, \neg F\}, \{\neg A, B, D\}\}$$



Unsatisfiable Cores: Proof-based Algorithm

$$\{\{A, B\}, \{D, \neg E\}, \{\neg B, C\}, \{\neg B, \neg C\}, \{B, \neg E, F\}, \{\neg A, \neg B, \neg D\}, \{\neg A, \neg F\}, \{\neg A, B, \neg D\}, \{\neg E, \neg F\}, \{\neg A, B, D\}\}$$



Assumption-based algorithm

1. Add a new activation literal $\neg A_i$ to each clause C_i of Φ.
2. Solve under assumptions $\texttt{solve}([A_1, A_2, \ldots, A_{|\Phi|}])$.
3. The result will be UNSAT.
4. The set $F \subseteq \{A_1, A_2, \ldots, A_{|\Phi|}\}$ of failed assumption literals corresponds to an unsatisfiable core of Φ.

Unsatisfiable Cores: Assumption-based Algorithm

$$
\begin{aligned}
&\{\{A, B\},\\
&\ \{D, \neg E\},\\
&\ \{\neg B, C\},\\
&\ \{\neg B, \neg C\},\\
&\ \{B, \neg E, F\},\\
&\ \{\neg A, \neg B, \neg D\},\\
&\ \{\neg A, \neg F\},\\
&\ \{\neg A, B, \neg D\},\\
&\ \{\neg E, \neg F\},\\
&\ \{\neg A, B, D\}\}
\end{aligned}
$$

Unsatisfiable Cores: Assumption-based Algorithm

$$
\begin{array}{ll}
\{\{A, B\}, & \{\{\neg A_1, A, B\}, \\
\{D, \neg E\}, & \{\neg A_2, D, \neg E\}, \\
\{\neg B, C\}, & \{\neg A_3, \neg B, C\}, \\
\{\neg B, \neg C\}, & \{\neg A_4, \neg B, \neg C\}, \\
\{B, \neg E, F\}, & \{\neg A_5, B, \neg E, F\}, \\
\{\neg A, \neg B, \neg D\}, & \{\neg A_6, \neg A, \neg B, \neg D\}, \\
\{\neg A, \neg F\}, & \{\neg A_7, \neg A, \neg F\}, \\
\{\neg A, B, \neg D\}, & \{\neg A_8, \neg A, B, \neg D\}, \\
\{\neg E, \neg F\}, & \{\neg A_9, \neg E, \neg F\}, \\
\{\neg A, B, D\}\} & \{\neg A_{10}, \neg A, B, D\}\}
\end{array}
$$

Unsatisfiable Cores: Assumption-based Algorithm

$$\begin{aligned}
&\{\{A, B\},\\
&\ \{D, \neg E\},\\
&\ \{\neg B, C\},\\
&\ \{\neg B, \neg C\},\\
&\ \{B, \neg E, F\},\\
&\ \{\neg A, \neg B, \neg D\},\\
&\ \{\neg A, \neg F\},\\
&\ \{\neg A, B, \neg D\},\\
&\ \{\neg E, \neg F\},\\
&\ \{\neg A, B, D\}\}
\end{aligned}$$

$$\begin{aligned}
&\{\{\neg A_1, A, B\},\\
&\ \{\neg A_2, D, \neg E\},\\
&\ \{\neg A_3, \neg B, C\},\\
&\ \{\neg A_4, \neg B, \neg C\},\\
&\ \{\neg A_5, B, \neg E, F\},\\
&\ \{\neg A_6, \neg A, \neg B, \neg D\},\\
&\ \{\neg A_7, \neg A, \neg F\},\\
&\ \{\neg A_8, \neg A, B, \neg D\},\\
&\ \{\neg A_9, \neg E, \neg F\},\\
&\ \{\neg A_{10}, \neg A, B, D\}\}
\end{aligned}$$

$\texttt{solve}([A_1, A_2, \ldots, A_{10}]) =$

Unsatisfiable Cores: Assumption-based Algorithm

$$\begin{aligned}
\{&\{A, B\}, \\
&\{D, \neg E\}, \\
&\{\neg B, C\}, \\
&\{\neg B, \neg C\}, \\
&\{B, \neg E, F\}, \\
&\{\neg A, \neg B, \neg D\}, \\
&\{\neg A, \neg F\}, \\
&\{\neg A, B, \neg D\}, \\
&\{\neg E, \neg F\}, \\
&\{\neg A, B, D\}\}
\end{aligned}$$

$$\begin{aligned}
\{&\{\neg A_1, A, B\}, \\
&\{\neg A_2, D, \neg E\}, \\
&\{\neg A_3, \neg B, C\}, \\
&\{\neg A_4, \neg B, \neg C\}, \\
&\{\neg A_5, B, \neg E, F\}, \\
&\{\neg A_6, \neg A, \neg B, \neg D\}, \\
&\{\neg A_7, \neg A, \neg F\}, \\
&\{\neg A_8, \neg A, B, \neg D\}, \\
&\{\neg A_9, \neg E, \neg F\}, \\
&\{\neg A_{10}, \neg A, B, D\}\}
\end{aligned}$$

$$\texttt{solve}([A_1, A_2, \ldots, A_{10}]) = \texttt{UNSAT}$$

Unsatisfiable Cores: Assumption-based Algorithm

$$\begin{aligned}
\{&\{A, B\}, \\
&\{D, \neg E\}, \\
&\{\neg B, C\}, \\
&\{\neg B, \neg C\}, \\
&\{B, \neg E, F\}, \\
&\{\neg A, \neg B, \neg D\}, \\
&\{\neg A, \neg F\}, \\
&\{\neg A, B, \neg D\}, \\
&\{\neg E, \neg F\}, \\
&\{\neg A, B, D\}\}
\end{aligned}$$

$$\begin{aligned}
\{&\{\neg A_1, A, B\}, \\
&\{\neg A_2, D, \neg E\}, \\
&\{\neg A_3, \neg B, C\}, \\
&\{\neg A_4, \neg B, \neg C\}, \\
&\{\neg A_5, B, \neg E, F\}, \\
&\{\neg A_6, \neg A, \neg B, \neg D\}, \\
&\{\neg A_7, \neg A, \neg F\}, \\
&\{\neg A_8, \neg A, B, \neg D\}, \\
&\{\neg A_9, \neg E, \neg F\}, \\
&\{\neg A_{10}, \neg A, B, D\}\}
\end{aligned}$$

$\texttt{solve}([A_1, A_2, \ldots, A_{10}]) = \texttt{UNSAT}$

failed literals $\{A_1, A_3, A_4, A_8, A_{10}\}$

Unsatisfiable Cores: Assumption-based Algorithm

$$\{\{A, B\}, \{D, \neg E\}, \{\neg B, C\}, \{\neg B, \neg C\}, \{B, \neg E, F\}, \{\neg A, \neg B, \neg D\}, \{\neg A, \neg F\}, \{\neg A, B, \neg D\}, \{\neg E, \neg F\}, \{\neg A, B, D\}\}$$

$$\{\{\neg A_1, A, B\}, \{\neg A_2, D, \neg E\}, \{\neg A_3, \neg B, C\}, \{\neg A_4, \neg B, \neg C\}, \{\neg A_5, B, \neg E, F\}, \{\neg A_6, \neg A, \neg B, \neg D\}, \{\neg A_7, \neg A, \neg F\}, \{\neg A_8, \neg A, B, \neg D\}, \{\neg A_9, \neg E, \neg F\}, \{\neg A_{10}, \neg A, B, D\}\}$$

$\texttt{solve}([A_1, A_2, \ldots, A_{10}]) = \texttt{UNSAT}$

failed literals $\{A_1, A_3, A_4, A_8, A_{10}\}$

Interpolation

Definition (Craig Interpolant, 1957)
Given a pair of formulas (A, B) such that $A \wedge B \models \bot$, a Craig interpolant is a formula I such that

- $A \models I$
- $B \wedge I \models \bot$
- $Atoms(I) \subseteq Atoms(A) \cap Atoms(B)$

This is the definition used in formal methods, sometimes called reverse Craig interpolant.

Craig Interpolants: Examples

$$
\begin{aligned}
A &= A_1 \land (\neg A_1 \lor C_1) \land A_2 \land (\neg A_2 \lor C_2) \land C_3 \\
B &= (\neg C_1 \lor B_1) \land (\neg C_2 \lor \neg B_1) \land C_3
\end{aligned}
$$

Craig Interpolants: Examples

$$\begin{aligned} A &= A_1 \wedge (\neg A_1 \vee C_1) \wedge A_2 \wedge (\neg A_2 \vee C_2) \wedge C_3 \\ B &= (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge C_3 \\ I &= C_1 \wedge C_2 \end{aligned}$$

Craig Interpolants: Examples

$$
\begin{aligned}
A &= A_1 \land (\neg A_1 \lor C_1) \land A_2 \land (\neg A_2 \lor C_2) \land C_3 \\
B &= (\neg C_1 \lor B_1) \land (\neg C_2 \lor \neg B_1) \land C_3 \\
I &= C_1 \land C_2
\end{aligned}
$$

$$
\begin{aligned}
A &= A_1 \land (\neg A_1 \lor C_1 \lor C_3) \land A_2 \land (\neg A_2 \lor C_2 \lor C_3) \\
B &= (\neg C_1 \lor B_1) \land (\neg C_2 \lor \neg B_1) \land \neg C_3
\end{aligned}
$$

Craig Interpolants: Examples

$$
\begin{aligned}
A &= A_1 \land (\neg A_1 \lor C_1) \land A_2 \land (\neg A_2 \lor C_2) \land C_3 \\
B &= (\neg C_1 \lor B_1) \land (\neg C_2 \lor \neg B_1) \land C_3 \\
I &= C_1 \land C_2
\end{aligned}
$$

$$
\begin{aligned}
A &= A_1 \land (\neg A_1 \lor C_1 \lor C_3) \land A_2 \land (\neg A_2 \lor C_2 \lor C_3) \\
B &= (\neg C_1 \lor B_1) \land (\neg C_2 \lor \neg B_1) \land \neg C_3 \\
I &= (C_1 \lor C_3) \land (C_2 \lor C_3)
\end{aligned}
$$

Craig Interpolants (alternative definition)

Definition (Craig Interpolant: alternative)
Given a pair of formulas (A, B) such that $A \models B$, a Craig interpolant is a formula I such that

- $A \models I$
- $I \models B$
- $Atoms(I) \subseteq Atoms(A) \cap Atoms(B)$

The definitions are dual: (A, B) is a reverse Craig interpolant iff $(A, \neg B)$ is a Craig interpolant in the above sense.

We discuss only reverse Craig interpolants from now on.

Craig Interpolation: Usage

Interpolants widely used in formal verification

- overapproximation of image
- computation of function summaries
- generalization of spurious counterexamples
- refinement of predicate abstraction
- ...

Craig Interpolation: Existence and Size

Theorem (McMillan, 2003)
For every pair of propositional formulas (A, B) such that $A \wedge B \models \bot$, a Craig interpolant can be computed in linear time with respect to the size of a resolution proof of unsatisfiability of $A \wedge B$.

Craig Interpolation: Existence and Size

Theorem (McMillan, 2003)
For every pair of propositional formulas (A, B) such that $A \wedge B \models \bot$, a Craig interpolant can be computed in linear time with respect to the size of a resolution proof of unsatisfiability of $A \wedge B$.

What does it say about the size of interpolant?

Craig Interpolation: Existence and Size

Theorem (McMillan, 2003)
For every pair of propositional formulas (A, B) such that $A \wedge B \models \bot$, a Craig interpolant can be computed in linear time with respect to the size of a resolution proof of unsatisfiability of $A \wedge B$.

What does it say about the size of interpolant?

What does it say about size with respect to $|A| + |B|$?

Computing Craig Interpolants

1. Get resolution proof of unsatisfiability of $A \wedge B$.
2. Label nodes of the proof by preliminary interpolants, starting from leaves.
3. The label of root of the proof is the Craig interpolant of (A, B).

Preliminary Interpolants

Definition

A formula f is a preliminary interpolant of the resolution proof node C (written $c\ [f]$) if

1. $A \models f$
2. $B \wedge f \models C$
3. $Atoms(C) \subseteq Atoms(A) \cup Atoms(B)$
4. $Atoms(f) \subseteq Atoms(A) \cap (Atoms(B) \cup Atoms(C))$

Preliminary interpolant f of the root $C = \bot$ is the real Craig interpolant of (A, B).

Leaves

$$\frac{}{C\ [C]}\ C \in A \qquad\qquad \frac{}{C\ [\top]}\ C \in B$$

where $\varphi\big|_l$ replaces all l in φ by $\top$ and $\neg l$ by $\bot$

Leaves

$$\frac{}{C\ [C]}\ C \in A \qquad\qquad \frac{}{C\ [\top]}\ C \in B$$

Inner nodes

$$\frac{(l \vee C)\ [f] \quad (\neg l \vee D)\ [g]}{(C \vee D)\ [\qquad]}\ var(l) \in Atoms(B) \qquad \frac{(l \vee C)\ [f] \quad (\neg l \vee D)\ [g]}{(C \vee D)\ [\qquad\qquad]}\ var(l) \notin Atoms(B)$$

where $\varphi\big|_l$ replaces all l in φ by $\top$ and $\neg l$ by $\bot$

Leaves

$$\frac{}{C\ [C]}\ C \in A \qquad \frac{}{C\ [\top]}\ C \in B$$

Inner nodes

$$\frac{(l \vee C)\ [f] \quad (\neg l \vee D)\ [g]}{(C \vee D)\ [f \wedge g]}\ var(l) \in Atoms(B) \qquad \frac{(l \vee C)\ [f] \quad (\neg l \vee D)\ [g]}{(C \vee D)\ [\qquad\qquad]}\ var(l) \notin Atoms(B)$$

where $\varphi\big|_l$ replaces all l in φ by $\top$ and $\neg l$ by $\bot$

Leaves

$$\frac{}{C\ [C]}\ C \in A \qquad \frac{}{C\ [\top]}\ C \in B$$

Inner nodes

$$\frac{(l \vee C)\ [f] \quad (\neg l \vee D)\ [g]}{(C \vee D)\ [f \wedge g]}\ var(l) \in Atoms(B) \quad \frac{(l \vee C)\ [f] \quad (\neg l \vee D)\ [g]}{(C \vee D)\ [f\big|_{\neg l} \vee g\big|_{l}]}\ var(l) \notin Atoms(B)$$

where $\varphi\big|_l$ replaces all l in φ by $\top$ and $\neg l$ by $\bot$

Interpolation Algorithm: Example

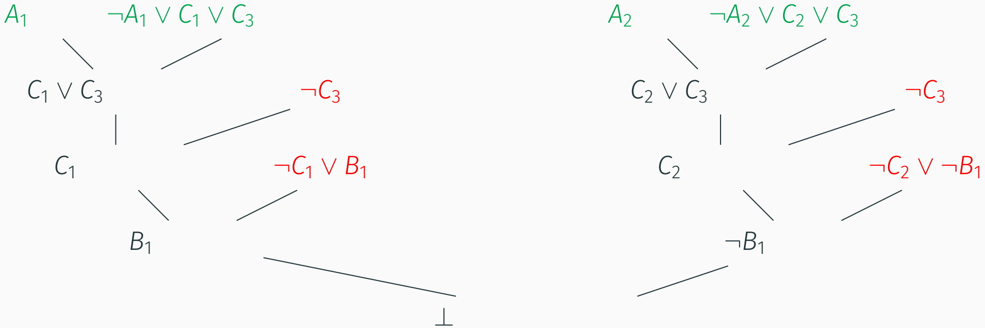
$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$

Interpolation Algorithm: Example

$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

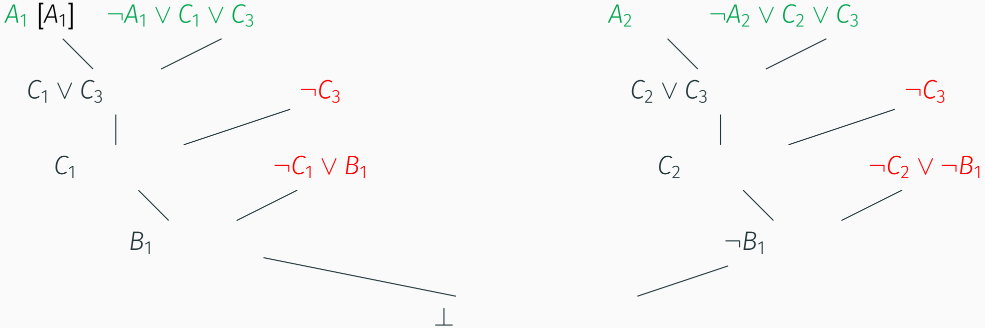
$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



Interpolation Algorithm: Example

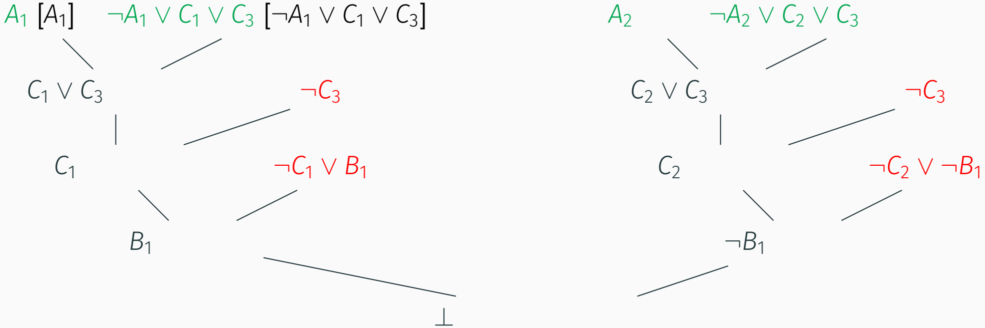
$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



Interpolation Algorithm: Example

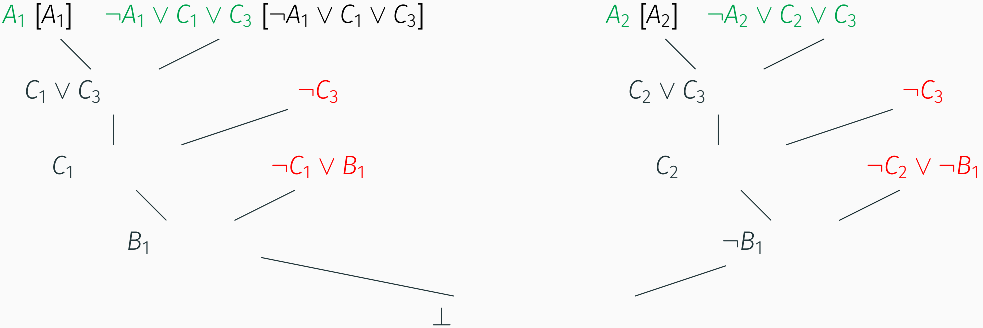
$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$
$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



Interpolation Algorithm: Example

$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

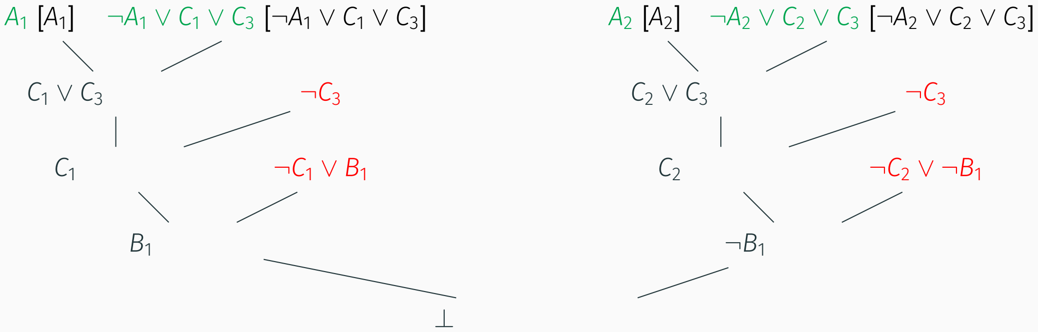
$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



Interpolation Algorithm: Example

$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

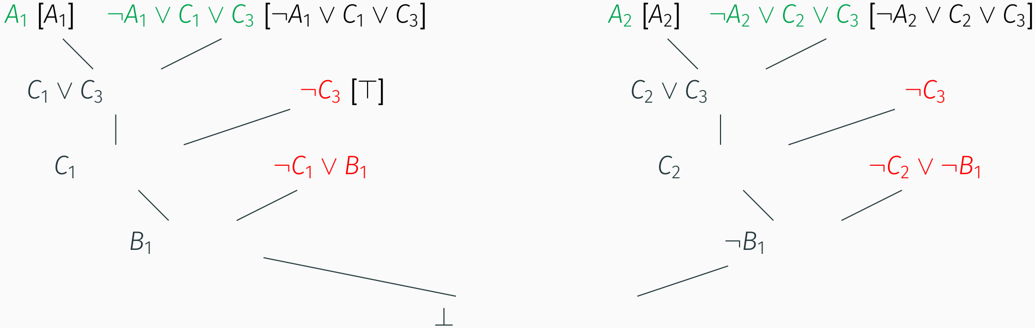
$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



Interpolation Algorithm: Example

$$A = A_1 \land (\neg A_1 \lor C_1 \lor C_3) \land A_2 \land (\neg A_2 \lor C_2 \lor C_3)$$

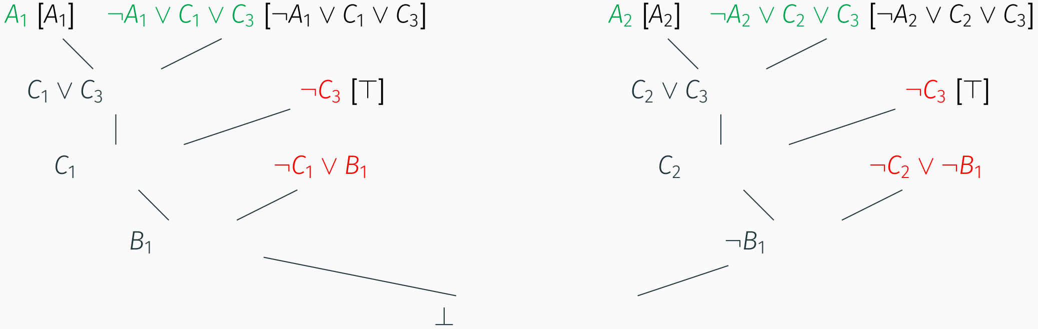
$$B = (\neg C_1 \lor B_1) \land (\neg C_2 \lor \neg B_1) \land \neg C_3$$



Interpolation Algorithm: Example

$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

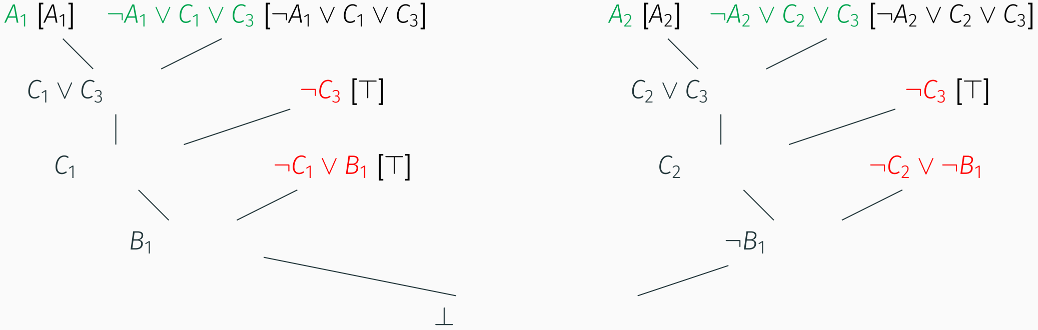
$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



Interpolation Algorithm: Example

$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

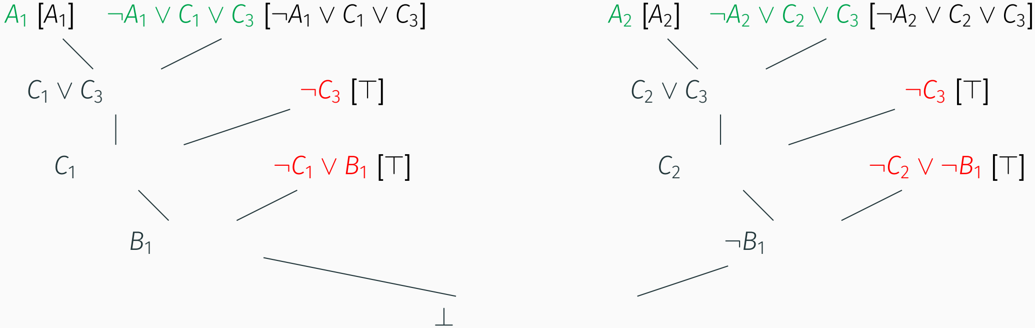
$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



Interpolation Algorithm: Example

$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

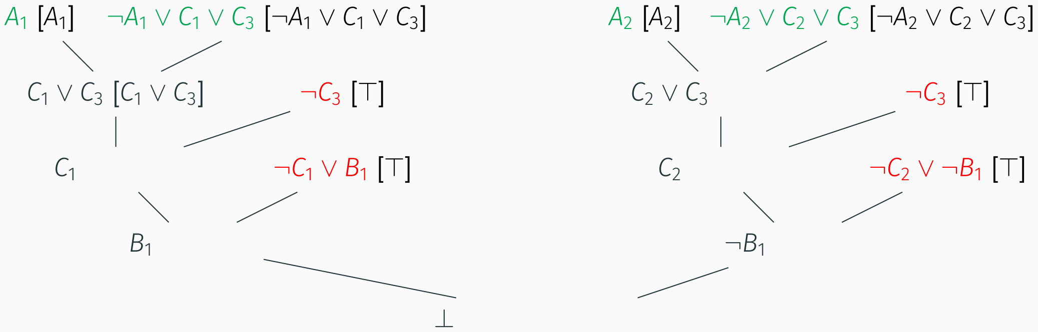
$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



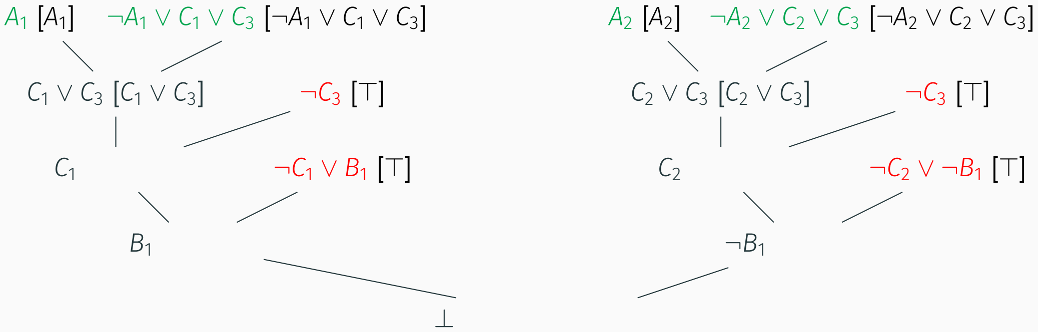
Interpolation Algorithm: Example

$$A = A_1 \land (\neg A_1 \lor C_1 \lor C_3) \land A_2 \land (\neg A_2 \lor C_2 \lor C_3)$$

$$B = (\neg C_1 \lor B_1) \land (\neg C_2 \lor \neg B_1) \land \neg C_3$$



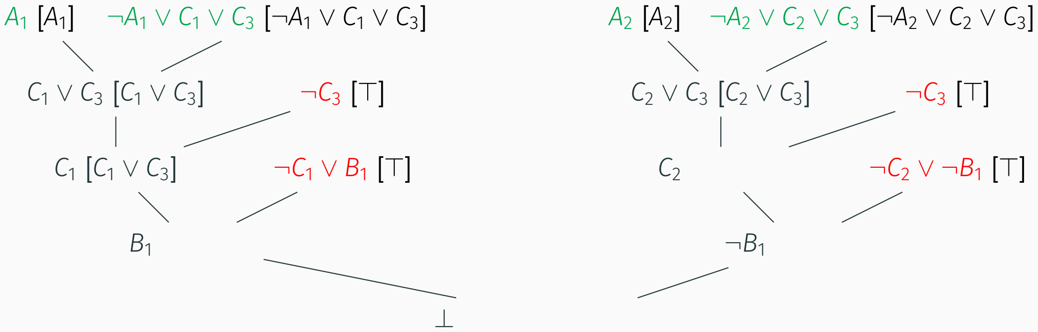
$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$
$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



Interpolation Algorithm: Example

$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

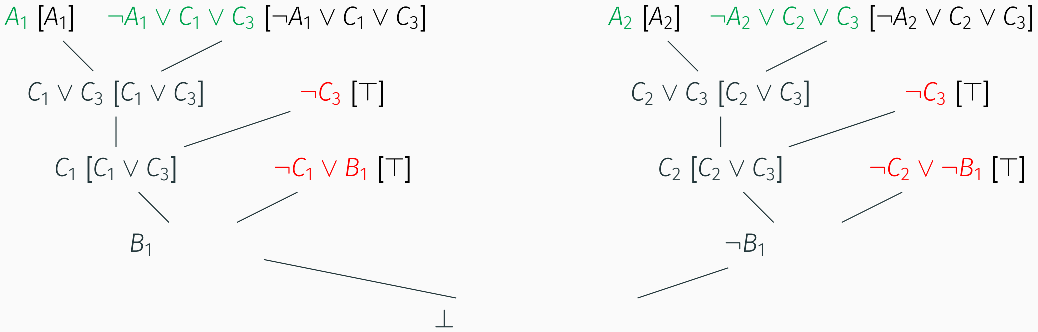
$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



Interpolation Algorithm: Example

$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

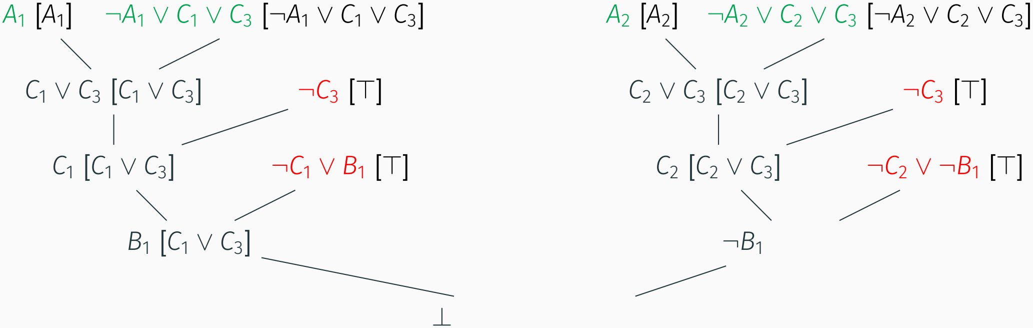
$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



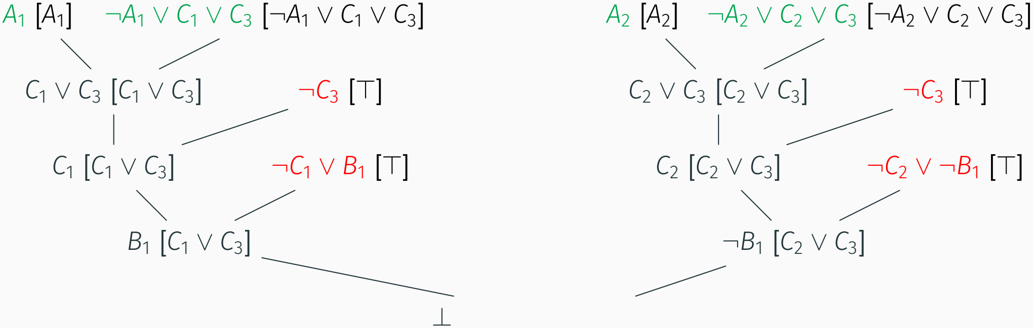
Interpolation Algorithm: Example

$$A = A_1 \land (\neg A_1 \lor C_1 \lor C_3) \land A_2 \land (\neg A_2 \lor C_2 \lor C_3)$$

$$B = (\neg C_1 \lor B_1) \land (\neg C_2 \lor \neg B_1) \land \neg C_3$$



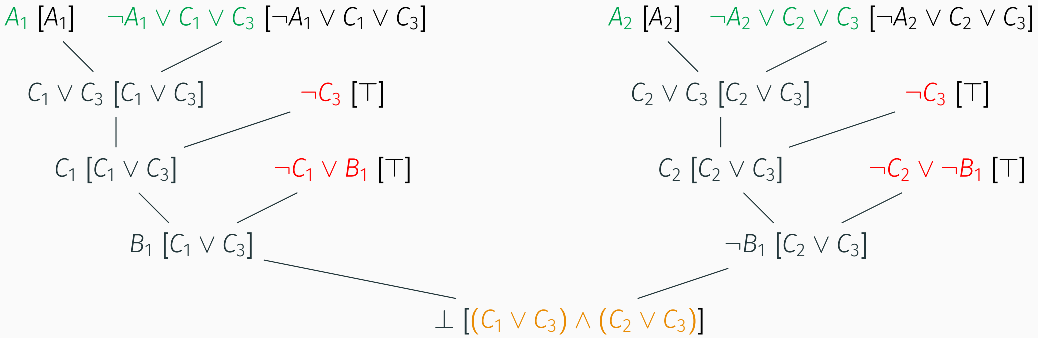
$$A = A_1 \land (\neg A_1 \lor C_1 \lor C_3) \land A_2 \land (\neg A_2 \lor C_2 \lor C_3)$$
$$B = (\neg C_1 \lor B_1) \land (\neg C_2 \lor \neg B_1) \land \neg C_3$$



Interpolation Algorithm: Example

$$A = A_1 \wedge (\neg A_1 \vee C_1 \vee C_3) \wedge A_2 \wedge (\neg A_2 \vee C_2 \vee C_3)$$

$$B = (\neg C_1 \vee B_1) \wedge (\neg C_2 \vee \neg B_1) \wedge \neg C_3$$



Interpolation Algorithm: Correctness

We can prove that

1. if

$$\frac{}{C\,[f]}\,,$$

   then f is a preliminary interpolant of C

2. if

$$\frac{C\,[f] \quad D\,[g]}{E\,[h]}$$

   and f is a preliminary interpolant of C
   and g is preliminary interpolant of D,
   then h is preliminary interpolant of E

Where are we?

Contents

Propositional satisfiability (SAT)

- $(A \lor \neg B) \land (\neg A \lor C)$
- is it satisfiable?

- $\leftarrow$ YOU ARE STANDING HERE

Satisfiability modulo theories (SMT)

- $x = 1 \;\land\; x = y + y \;\land\; y > 0$
- is it satisfiable over reals?
- is it satisfiable over integers?

Automated theorem proving (ATP)

- axioms: $\forall x\,(x + x = 0)$, $\forall x \forall y\,(x + y = y + x)$
- do they imply $\forall x \forall y\,((x + y) + (y + x) = 0)$?

We already know

- normal forms of propositional logic (CNF)
- efficient conversions (Tseitin encoding)
- resolution method and Davis-Putnam algorithm
- DPLL
- two watched literal scheme for unit propagation and conflict detection
- CDCL (clause learning and backjumping)
- literal decision heuristics, restarts
- incremental solving, proof generation, unsat core generation, interpolant generation

Next time

- first-order logic
- first-order theories
- satisfiability modulo theories (SMT)
- theories of interest (integer arithmetic, real arithmetic, uninterpreted functions, arrays, bit-vectors, . . .)