DPLL

① Essentially DPLL is intelligent search over the $O(2^N)$ space of models, given $\Sigma = \{\sigma_1, \ldots, \sigma_N\}$.

We'll demonstrate its basics through an extended example:

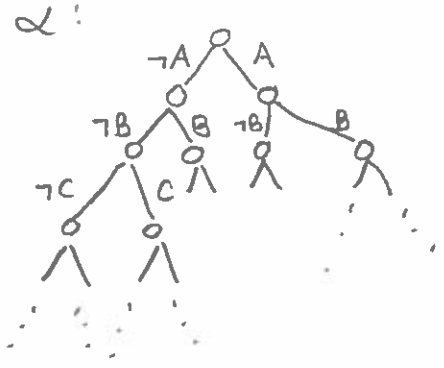
DPLL is short for Davis-Putnam-Logemann-Loveland

$$
\begin{aligned}
&(D \vee E) \\
\wedge\ &(B \vee \neg D \vee E) \\
\wedge\ &(D \vee \neg E) \\
\wedge\ &(\neg B \vee E) \\
\wedge\ &(A \vee \neg B) \\
\wedge\ &(\neg C \vee \neg E \vee \neg F) \\
\wedge\ &(\neg B \vee D) \\
\wedge\ &(C \vee \neg E) \\
\wedge\ &(\neg E \vee F)
\end{aligned}
$$

Let's see how DPLL shows this CNF sentence α is unsatisfiable.

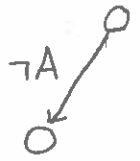
---

② The naive search-based algorithm just searches through all models, and checks each one to see if it is a model of sentence α:



DPLL

3) The main optimization of DPLL is to apply a limited amount of resolution at each node of the search tree. For instance, suppose we begin by assuming $m(A)=0$ for our model:



Note that this is the same as asking:

is $\alpha \wedge \neg A$ satisfiable?

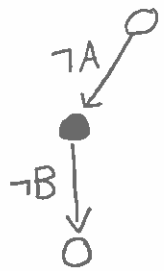
Rather than applying "full-strength" resolution to this subquestion, we apply a lighterweight version that only resolves the new clause $\neg A$ against each of the existing clauses. We get:

$$\neg A \wedge (A \vee \neg B) \vdash \neg B$$

4) Because resolution is sound, this means:

$$\alpha \wedge \neg A \models \neg B$$

which means we don't have to make any choices at our next search node:

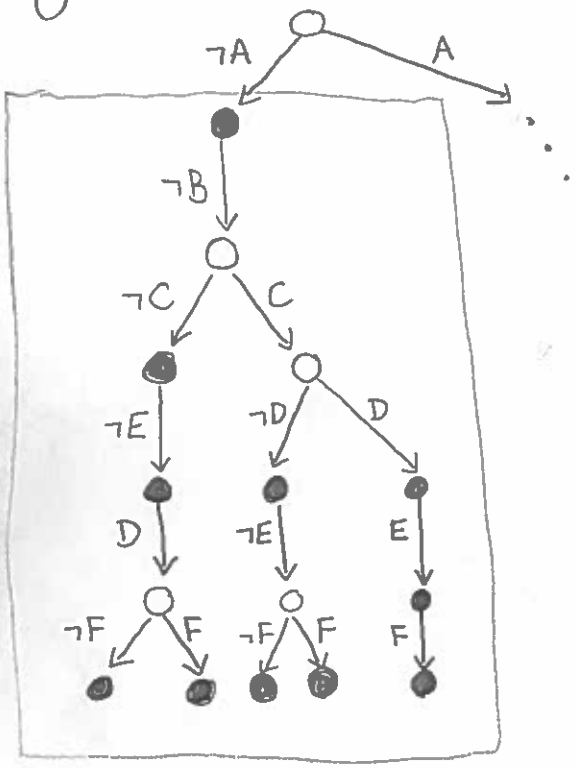


DPLL

(5) This optimization is called unit resolution or unit propagation, because we only resolve when one of the clauses is a single literal, and our goal is to derive more single-literal clauses, so we can reduce our search space.

(6) We can proceed like this, making assumptions only when necessary:

this subtree has 5 leaves instead of $2^5 = 32$ leaves



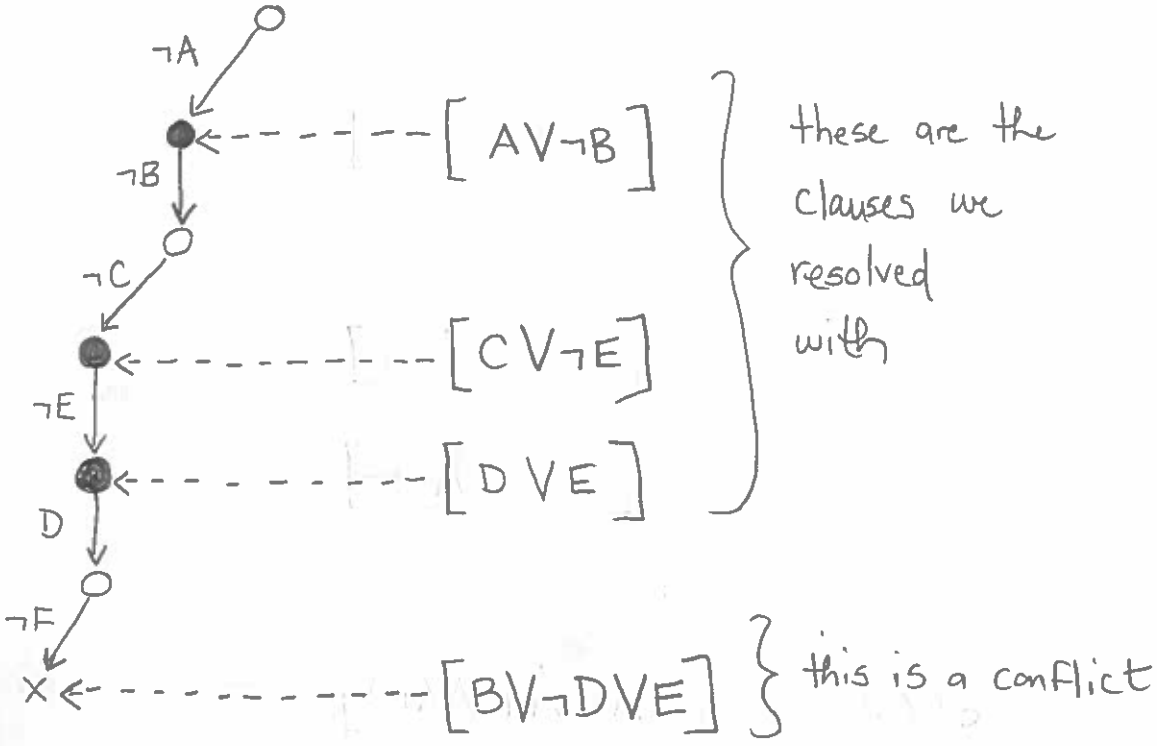
(7) Because we only prune subtrees when we have proven (by resolution) that they only contain unsatisfying models, DPLL is complete.

It is also, conveniently, linear space.

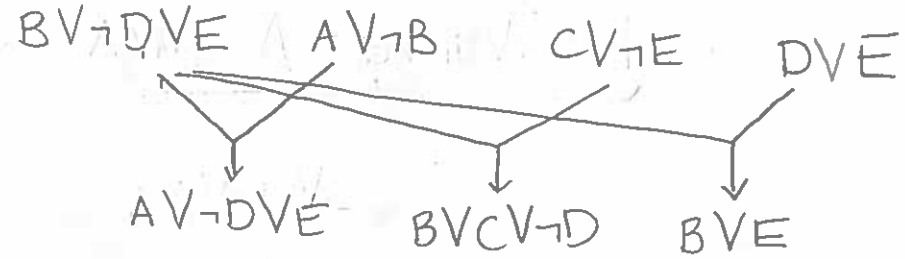
DPLL

(8) An important modern optimization of DPLL is called conflict-driven learning.

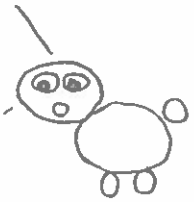
Let's pause when we reach the first leaf of our search space:



(9) The observation is that we have been driven towards a conflict by a subset of the clauses in our CNF sentence. Maybe we can learn from our mistakes?



b/c $\alpha \models \beta$
means $I(\alpha) \subseteq I(\beta)$
means $I(\alpha) = I(\alpha) \cap I(\beta)$
means $\alpha \equiv \alpha \wedge \beta$



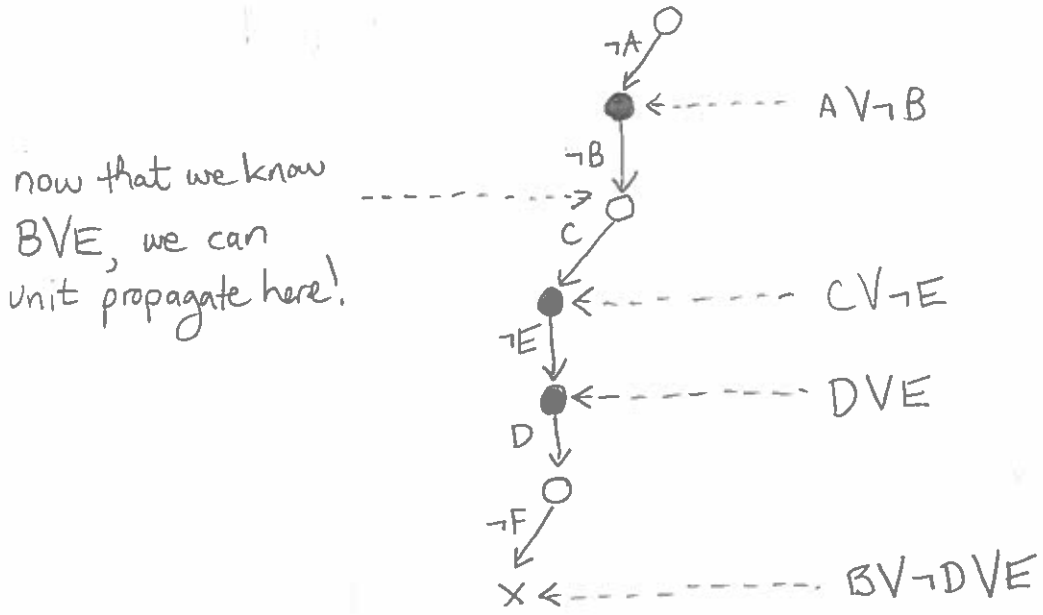
This "resolution-in-retrospect" has shown us that:

$$\alpha \models (A \vee D \vee E) \wedge (B \vee C \vee \neg D) \wedge (B \vee E)$$

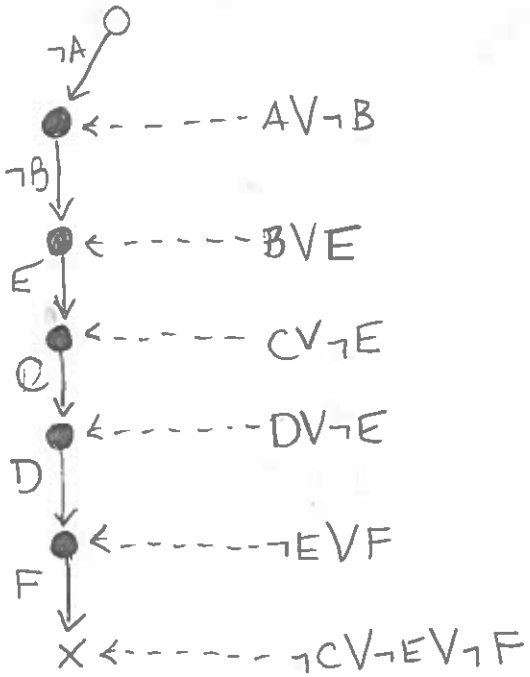
Which means: $\alpha \equiv \alpha \wedge (A \vee D \vee E) \wedge (B \vee C \vee \neg D) \wedge (B \vee E)$

DPLL

(10) If we had known these additional clauses earlier, we could have been more efficient:



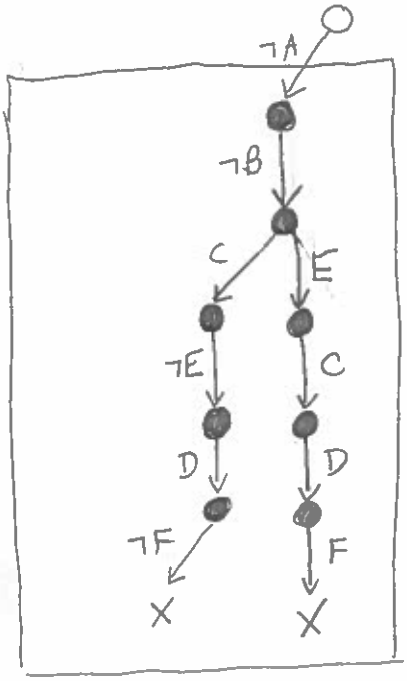
(11) We can't turn back time exactly, but we can go back and correct our mistakes, now that we know more:



DPLL

(2) With conflict-driven learning, we reduced the number of leaves of the search tree (after choosing ¬A) from 5 to 2!

this subtree has 2 leaves instead of $2^5 = 32$ leaves



(13) DPLL with conflict-driven learning is the foundation of state-of-the-art satisfiability solvers.

It should be clear that it's not a fully specified algorithm, but an algorithmic schema with parameters like:

- in what order do we choose literals?
- Which clauses do we store ("learn") when conflicts arise?

Also, with conflict-driven learning, the space requirements are no longer linear in the number of variables, since the number of clauses in the formula is always growing. Because of this, often DPLL solvers use heuristics to determine when to "forget" learned clauses.