Docker setup

You can start

```
docker pull registry.gitlab.inria.fr/soliman/inf555/td2
```

now

Boolean satisfiability, SAT solvers (part 1)

Sylvain Soliman

September 25th, 2019

Thanks to Lintao Zhang and David L. Dill for inspiration

Propositional Logic

Propositional formulae are built from the following:

- Constants: 1 and 0
- Propositional variables: $a, b, p, q, x, y, \ldots$
- Logical Connectives: $\vee, \wedge, \neg$
  (others too, e.g., $(a \Rightarrow b) \Leftrightarrow (\neg a \vee b)$)

No quantifiers, no functions

A formula itself is a function $\mathbb{B}^n \mapsto \mathbb{B}$ of its n variables

Satisfiable *iff* $\exists$ mapping of variables to constants such that the formula becomes true (tautology)

Boolean Satisfiability problem

Definition

The Boolean satisfiability problem (SAT) is the decision problem whether a propositional formula ϕ is satisfiable, i.e., $\phi^{-1}(1) \neq \varnothing$

First **NP-complete** problem (Cook 1971)

- solving SAT efficiently provides a way to solve any other NP problem rather efficiently

Combinatorial Problems

Most combinatorial problems (AI planning, theorem proving, model checking, resource allocation, etc.) are in NP

Ubiquitous in the industry

Solver competitions in the academia

- SAT was well studied for the last 40 years and now very efficient (millions of variables)

We will see how SAT-solvers work and how to use one

Formula Representation

Representing formulae by the state space, i.e., the Boolean hypercube $\mathbb{B}^n$ is exponential

Same for truth tables

Better reprentations:

- Boolean circuit
- Binary Decision Diagram (BDD)
- Boolean formula in **normal form**

Conjunctive Normal Form — CNF

Used in most modern SAT solvers

Functions represented as a big conjunction ($\wedge$ sometimes noted multiplicatively)

of **clauses**, i.e., disjunctions ($\vee$ sometimes noted $+$)

of **literals** (a variable x or its negation $\neg x$ or x')

$$(x'(y' + z'))' = x + yz$$

Normalization

Eliminate constants:

- $x1 = x$
- $x0 = 0$

Optionally do more simplifications

- $xx = x$
- $xx' = 0$
- etc.

If the formula is a constant, no need for CNF

Careful!
applying distribution leads to an exponential blowup

Tseytin's Transformation

Add **new variables** for subformulae of the original formula, e.g.

$$\phi = pq + \psi$$

Introduce $x \Leftrightarrow p \wedge q$, i.e., $(x \Rightarrow (p \wedge q)) \wedge ((p \wedge q) \Rightarrow x)$

$(x' + pq)((pq)' + x)$

$(x' + p)(x' + q)(p' + q' + x)$

$$\phi = (x + \psi)(x' + p)(x' + q)(p' + q' + x)$$

Example — The N-Queens revisited

Given an integer N, write a (mathematical) Boolean satisfiability problem representing the N-Queens problem that you already saw last week

All variables need to be Boolean

Try to write the constraints in CNF

Clauses Optimizations

Once in clause form:

$(p + p + \dots) = (p + \dots)$

$(p + p' + \dots)(\psi) = \psi$

Avoid introducing two variables in Tseytin's transformation for ϕ and $\neg\phi$

Now, we have to solve:
$(p + q + r)(p' + q' + r)(p' + q + r')(p + q' + r')$

Clauses Optimizations

Once in clause form:

$(p + p + \dots) = (p + \dots)$

$(p + p' + \dots)(\psi) = \psi$

Avoid introducing two variables in Tseytin's transformation for ϕ and $\neg\phi$

Now, we have to solve:
$(p + q + r)(p' + q' + r)(p' + q + r')(p + q' + r')$

Naïve Solver

A naïve solver could try possible assignments by backtracking

Until either all clauses have a true literal (return SAT)

or some clause has all false literals (return UNSAT)

Very slow, unless early SAT or UNSAT result

M. **Davis**, H. **Putnam**, "A computing procedure for quantification theory", J. of ACM, Vol. 7, pp. 201–214, 1960

Iterative variable elimination and clause rewriting for clauses with **one** incompatible variable (p and p')

$$(a+b+c)(b+c'+f')(b'+e) \longrightarrow (a+c+e)(c'+e+f') \longrightarrow (a+e+f') \longrightarrow \text{SAT}$$

$$(a+b)(a+b')(a'+c)(a'+c') \longrightarrow (a)(a'+c)(a'+c') \longrightarrow (c)(c') \longrightarrow \text{UNSAT}$$

If some literal is always positive (or negative), i.e., ***pure***, don't try the other valuation

Potential memory explosion issue

M. **Davis**, G. **Logemann**, and D. **Loveland**, "A machine program for theorem-proving". Communications of the ACM, Vol. 5, No. 7, pp. 394–397, 1962

Avoids EXPSPACE issue of Davis-Putman

Replace DP with **Depth First Search (DFS) and BCP**
to obtain DPLL, the current basis of all SAT solvers

Unit propagation a.k.a. Boolean Constraint Propagation (**BCP**):

- if only one literal of a clause doesn't currently evaluate to $0 \Rightarrow$ set it to 1

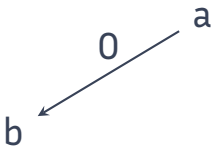
DPLL example

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$

a

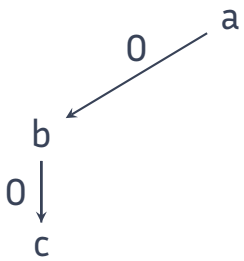
DPLL example

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$



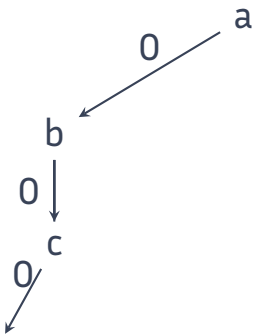
DPLL example

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$



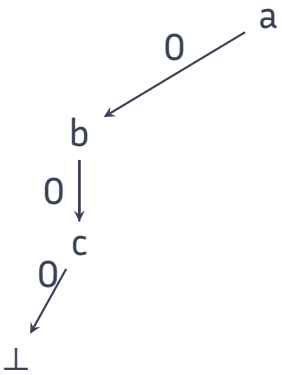
DPLL example

$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$
$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$



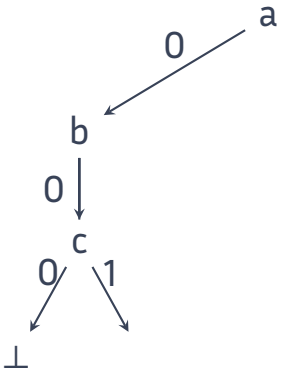
DPLL example

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$



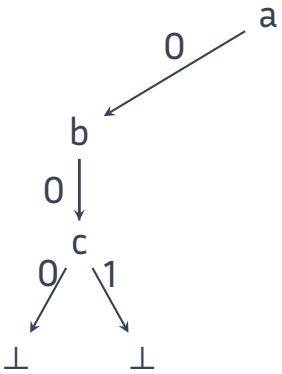
DPLL example

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$



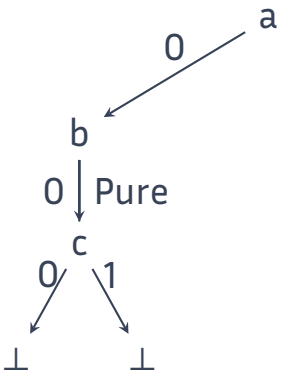
DPLL example

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$



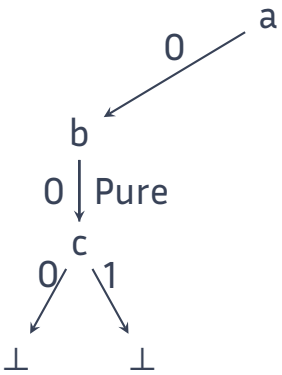
DPLL example

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$



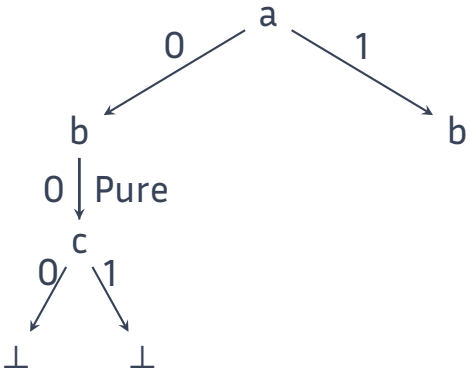
DPLL example

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$



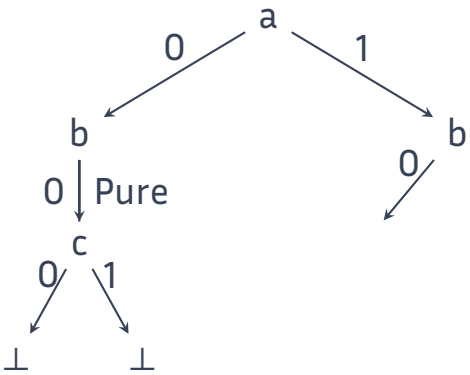
DPLL example

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$



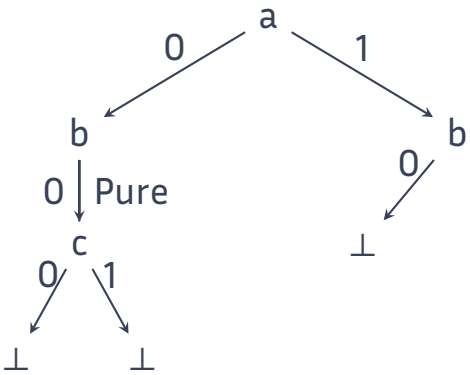
DPLL example

$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$
$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$



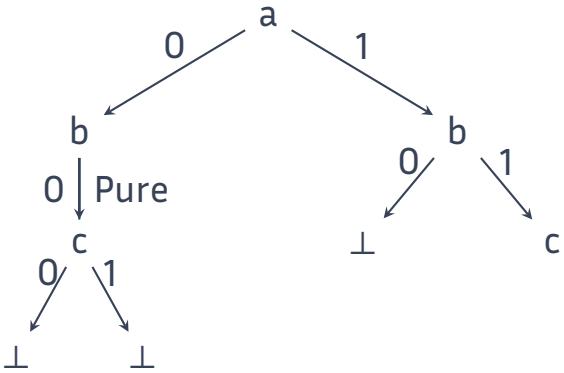
DPLL example

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$



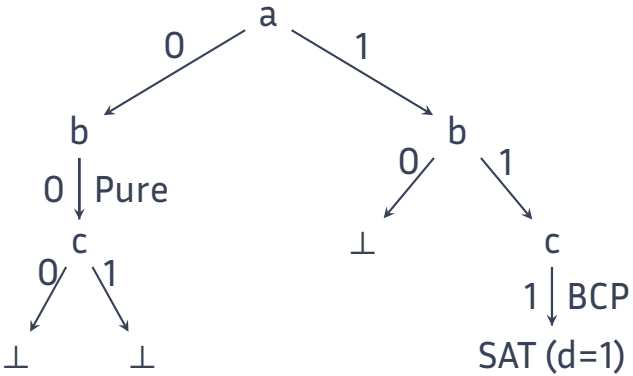
DPLL example

$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$
$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$



DPLL example

$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$
$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$



C'est bien, mais pas suffisant !

J. P. Marques-Silva and K. A. Sakallah, "GRASP — A New Search Algorithm for Satisfiability", Proc. ICCAD 1996.

Learning from conflicts:
$a \wedge b \wedge c \Rightarrow$ conflict

is equivalent to:
no conflict $\Rightarrow \neg a \vee \neg b \vee \neg c$

When a conflict occurs, add a new clause to the initial problem!

DPLL example revisited

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$

DPLL example revisited

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$

$$a = 0$$

DPLL example revisited

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$

$$a = 0, b = 0$$

DPLL example revisited

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$$

$a = 0, b = 0, c = 0$

Conflict: $a = 0 \wedge c = 0$, i.e., $a'c'$

DPLL example revisited

$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$
$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)$

$a = 0, b = 0, c = 0$

Conflict: $a = 0 \wedge c = 0$, i.e., $a'c'$

We can add the new non-conflict clause: $(a'c')' = (a + c)$

Backjumping

We can use conflicts to backtrack better: **Conflict-driven backtracking**

A conflict clause will lead to BCP when the second from last variable gets instantiated

Sometimes you can jump even more (cf. Unique implication point)

In our example, adding $(a + c)$ will instantiate c to 1 by BCP as soon as $a = 0$ is chosen: we can backtrack to that level and proceed from here...

DPLL example revisited

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)(a + c)$$

DPLL example revisited

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)(a + c)$$

$$a = 0$$

DPLL example revisited

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)(a + c)$$

$a = 0$ BCP

DPLL example revisited

$$(a' + b + c)(a + c + d)(a + c + d')(a + c' + d)$$
$$(a + c' + d')(b' + c' + d)(a' + b + c')(a' + b' + c)(a + c)$$

$a = 0$ BCP: $c = 1$

We get the d conflict without needing to go through the choices for b or c, only **unit propagation**

There's more...

Most of the computation time is taken by BCP: **optimize unit propagation**

The procedure remains a search procedure:
use **heuristics** to choose variables and assignments

Variable/Value selection heuristics

There are many heuristics

For values (try True or False first?)

But mostly for variables, e.g.,

- RAND (random)
- MOM (Maximum Occurrences on clauses of Minimum Size)
- DLIS/DLCS (Dynamic Largest Individual/Combined Sum in unresolved clauses)
- etc.

But there will be a course dedicated to heuristics in CSPs in general

Optimizing BCP

BCP takes a lot of time because of bookkeeping:

- Find all unit clauses;
- Detect conflicts;
- Detect satisfied clauses;
- Undo all of this on backtracking!

Many algorithms (SATO, Chaff, etc.) optimize this process with **significant impact** on the solver performance

You will see the cost of naive BCP in TD

Literal Counting Scheme

Basic optimization

Maintain a count of *non-false* literals in each clause

Unit clause when $k = 1$
The remaining literal has to be found and instantiated to 1

Conflict when $k = 0$

Chaff

Idea: a clause can become Unit when

Chaff

Idea: a clause can become Unit when *all but one* of its literals are 0

Chaff

Idea: a clause can become Unit when *all but one* of its literals are 0

Just pick two literals to **watch** and ignore the rest
Maintain this as an *invariant*

Handle clauses with a single literal as a special case

Maintain the state and pending assignments as stacks

Undoing assignments on backtrack maintains our invariant!

Example

$$(b + c + a + d + e)$$
$$(a + b + c')$$
$$(a + b')$$
$$(a' + d)$$
$$(a')$$

Example

$$(b + c + a + d + e)$$
$$(a + b + c')$$
$$(a + b')$$
$$(a' + d)$$
$$(a')$$

Watched literals

Example

$$(b + c + a + d + e)$$
$$(a + b + c')$$
$$(a + b')$$
$$(a' + d)$$
$$\Rightarrow (a') \text{ Special case of BCP}$$

Watched literals

Example

$$(b + c + a + d + e)$$
$$(a + b + c')$$
$$(a + b')$$
$$(a' + d)$$
$\Rightarrow (a')$ Special case of BCP

Watched literals

Current assignment: $a = 0$
Pending:

Example

$$
\begin{aligned}
& (b + c + a + d + e) \\
\Rightarrow\ & (a + b + c') \\
& (a + b') \\
& (a' + d) \\
& (a')
\end{aligned}
$$

Watched literals

Current assignment: $a = 0$
Pending:

Maintain watched literals!

Example

$(b + c + a + d + e)$

$\Rightarrow (a + b + c')$ Replace $a = 0$ by an unassigned literal

$(a + b')$

$(a' + d)$

(a')

Watched literals

Current assignment: $a = 0$

Pending:

Maintain watched literals!

Example

$$
\begin{aligned}
& (b + c + a + d + e) \\
\Rightarrow\ & (a + b + c') \\
& (a + b') \\
& (a' + d) \\
& (a')
\end{aligned}
$$

Watched literals

Current assignment: $a = 0$
Pending:

Maintain watched literals!

Example

$$
\begin{aligned}
& (b + c + a + d + e) \\
& (a + b + c') \\
\Rightarrow\ & (a + b') \\
& (a' + d) \\
& (a')
\end{aligned}
$$

Watched literals

Current assignment: $a = 0$
Pending:

Maintain watched literals!

Example

$(b + c + a + d + e)$
$(a + b + c')$
$\Rightarrow (a + b')$ Unit! Record assignment
$(a' + d)$
(a')

Watched literals

Current assignment: $a = 0$
Pending:

Maintain watched literals!

Example

$$
\begin{aligned}
& (b + c + a + d + e) \\
& (a + b + c') \\
\Rightarrow\ & (a + b') \\
& (a' + d) \\
& (a')
\end{aligned}
$$

Watched literals

Current assignment: $a = 0$
Pending: $b = 0$

Maintain watched literals!

Example

$$
\begin{aligned}
&(b + c + a + d + e)\\
&(a + b + c')\\
&(a + b')\\
\Rightarrow\ &(a' + d)\\
&(a')
\end{aligned}
$$

Watched literals

Current assignment: $a = 0$
Pending: $b = 0$

Maintain watched literals!

Example

$(b + c + a + d + e)$
$(a + b + c')$
$(a + b')$
$\Rightarrow (a' + d)$ Satisfied
(a')

Watched literals

Current assignment: $a = 0$
Pending: $b = 0$

Maintain watched literals!

Example

$$\Rightarrow (b + c + a + d + e)$$
$$(a + b + c')$$
$$(a + b')$$
$$(a' + d)$$
$$(a')$$

Watched literals

Current assignment: $a = 0\ b = 0$
Pending:

Maintain watched literals!

Example

$$\Rightarrow (b + c + a + d + e)$$
$$(a + b + c')$$
$$(a + b')$$
$$(a' + d)$$
$$(a')$$

Watched literals

Current assignment: $a = 0\ b = 0$
Pending:

Maintain watched literals!

Example

$$
\begin{aligned}
& (b + c + a + d + e) \\
\Rightarrow\ & (a + b + c') \\
& (a + b') \\
& (a' + d) \\
& (a')
\end{aligned}
$$

Watched literals

Current assignment: $a = 0\ b = 0$
Pending:

Maintain watched literals!

Example

$$
\begin{aligned}
& (b + c + a + d + e) \\
\Rightarrow\ & (a + b + c') \\
& (a + b') \\
& (a' + d) \\
& (a')
\end{aligned}
$$

Watched literals

Current assignment: $a = 0\ b = 0$
Pending: $c = 0$

Maintain watched literals!

Example

$$\Rightarrow (b + c + a + d + e)$$
$$(a + b + c')$$
$$(a + b')$$
$$(a' + d)$$
$$(a')$$

Watched literals

Current assignment: $a = 0\ b = 0\ c = 0$
Pending:

Maintain watched literals!

Example

$\Rightarrow (b + c + a + d + e)$
$(a + b + c')$
$(a + b')$
$(a' + d)$
(a')

Watched literals

Current assignment: $a = 0\ b = 0\ c = 0$
Pending:

Maintain watched literals!

Example

$(b + c + a + d + e)$
$(a + b + c')$
$(a + b')$
$(a' + d)$
(a')

Watched literals

Current assignment: $a = 0\ b = 0\ c = 0$
Pending: Nothing to do, done with BCP!

Maintain watched literals!

“You don’t have to think too hard when you talk to teachers.”

— J. D. Salinger

Docker setup

You can start

```
docker pull registry.gitlab.inria.fr/soliman/inf555/td2
```

now

and then

```
docker run -p 8888:8888 -p 8080:8080 \
-v "$PWD":/home/jovyan/work \
registry.gitlab.inria.fr/soliman/inf555/td2
```