Algoritmen & Datastructuren 2

WPO – Intro SAT & DPLL

SAT Solving

- Nagaan of een logische formule vervulbaar (satisfiable) is of niet (unsatisfiable)
  - Propositielogica
- Naïef idee: exhaustief zoeken (vergelijkbaar met waarheidstabellen)
  - 2^n mogelijkheden aftoetsen (n = aantal proposities/variabelen)
  - **Exponentieel in aantal proposities**
  - We gaan proberen beter te doen...

Terminologie

- Atoom/Variabele
- Literal
- Clause
- Formule
- Interpretatie (theorie)

$a, b, c, \dots, x_1, x_2, \dots$

$a, \bar{a}, b, \bar{b}, \dots, x_1, \overline{x_1}, \dots$

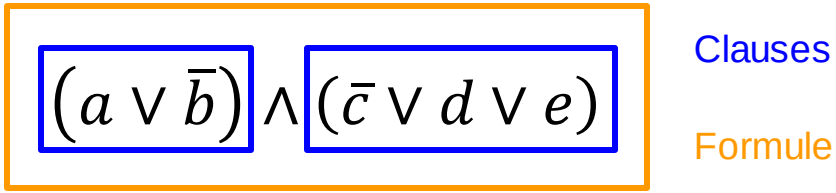
$a \vee \bar{b}$

$(a \vee \bar{b}) \wedge (\bar{c} \vee d \vee e)$

$\{a, \bar{b}, c\}$

Conjunctive Normal Form (CNF)

- Standaard manier om een formule in propositielogica te noteren
- Conjuncties (AND) van disjuncties (OR)
- Elke formule kan geconverteerd worden naar dit formaat!



Enkele belangrijke bestanden/ADTs

Alle code voor hoofdstuk 19 bevindt zich in a-d/sat

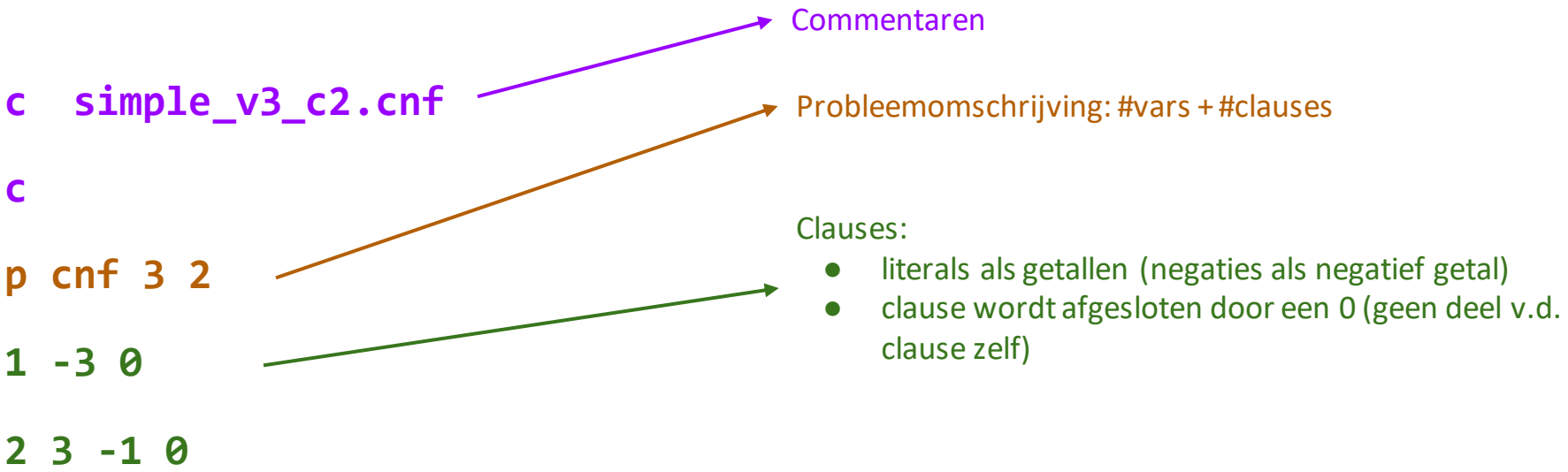
Enkele belangrijke bestanden/ADTs:

- `cnf.rkt`: ADTs die een CNF formule opbouwen
- `interpretation.rkt`: Interpretatie ADT
- `logger.rkt`: logging module om prints uit te voeren
- `dimacs-parser.rkt`: CNF formules uit tekstbestanden inladen

DIMACS Formaat

Zie `a-d/sat/dimacs-parser.rkt`

Tekstbestanden die CNF formules voorstellen



DPLL (Davis–Putnam–Logemann–Loveland)

Depth-first Tree Search: opbouw van een vervullende interpretatie

1. Kiezen van literals
2. Unit-propagatie
3. Pure Literal Elimination

Bestanden: `a-d/sat/dpll`

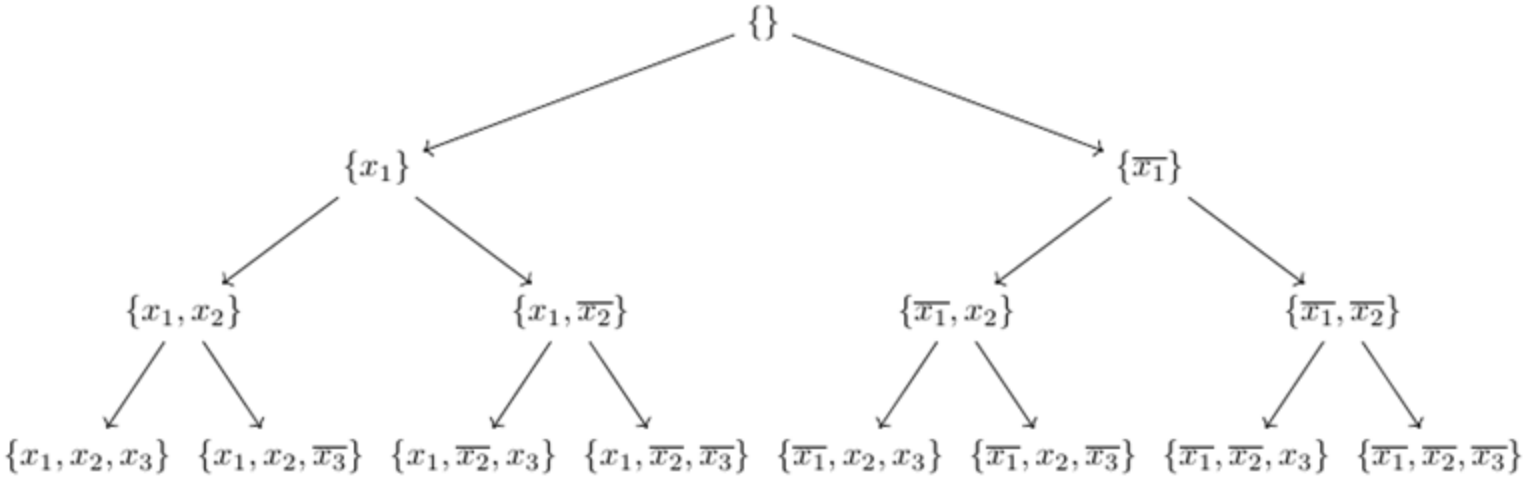
Subfolder `naive`: voor dit WPO

Subfolder `optimized`: voorlopig leeg, voor volgend WPO

Tree Search ≠ Search Tree

DPLL bouwt een interpretatie op d.m.v. een **zoekproces**

We hebben **geen** expliciete zoekboom datastructuur!



Opdracht

Bestudeer de recursieve implementatie(s) van DPLL in de subfolder `naive/`

- **`without-pure-literals.rkt`**: bevat enkel unit-propagatie
- **`with-pure-literals.rkt`**: bevat ook pure literal elimination

Iteratieve versie van DPLL maken

- Voorzie bestand **`a-d/sat/dpll/naive/iterative.rkt`**
- Maak gebruik van een stack om "tree traversal" te doen: zie **`a-d/tree/binary-tree-algorithms.rkt`** (AD1)

Test jouw implementatie via het meegeleverde bestand **`test.rkt`**

Pas **`a-d/sat/dpll/config.rkt`** aan om jouw nieuwe versie te gebruiken!

Zie bijgeleverde folder **`cnf-samples`** voor enkele formules in DIMACS formaat!