Algoritmen & Datastructuren 2

WPO – Two-watched Literal Scheme

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

Unit-propagatie kan versneld worden d.m.v. een speciale datastructuur

Two-watched Literal Scheme (TWLS)

Snel proberen navigeren tussen clauses en literal:

Welke literal zit waar?

Welke clause is unit?

} **Pointers to the rescue!**

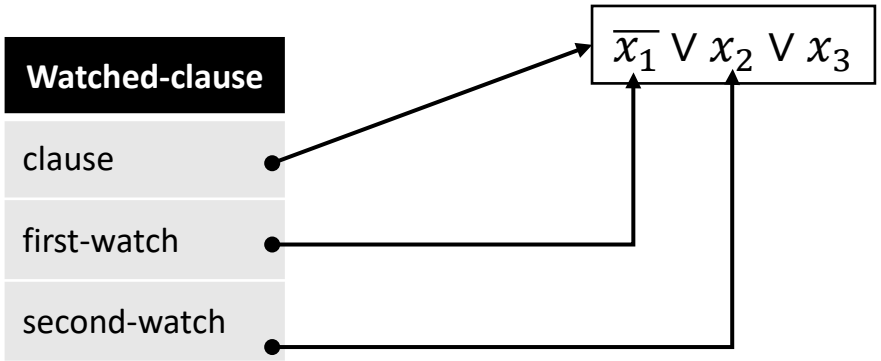
Hoeveel pointers zijn er nodig?

Watched Clause ADT

Wrapper rond Clause ADT waarvan **2 literals** "bekeken" worden

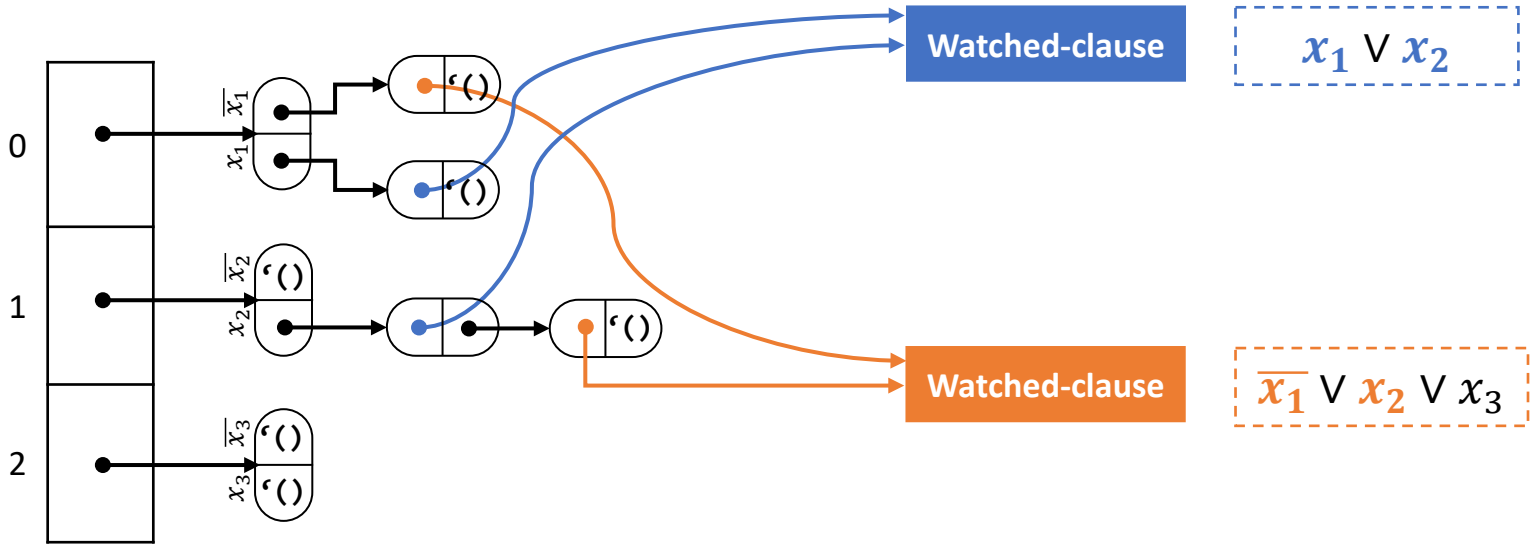
Legt relatie `clause → literal` voor TWLS vast

Omgekeerde link is ook nodig: **welke literal wordt waar bekeken?**



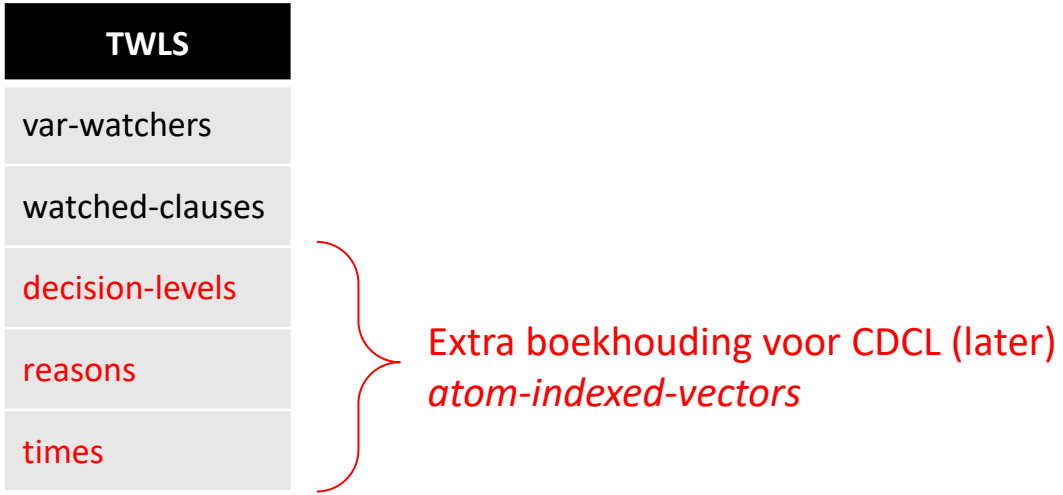
Variable watchers

Legt relatie `literal → watched-clause` voor **literals die bekeken worden** vast in een *atom-indexed-vector*



TWLS ADT

Zie `a-d/sat/twls.rkt`



Unit-propagatie met TWLS

- Pseudo-code in cursustekst
- Propagatie-queue
  - Literals waarmaken
  - Invloed op meerdere clauses

Algorithm 3: Unit propagation and conflict detection with the two-watched literal scheme

```
1  UNIT-PROP():
   Output: Either a conflicting clause c or ⊥ if no clause is conflicting
2  for Clause c = ℓ of length one in 𝒯 do
3      if ℓ.value = f then return c;
4      if ℓ.value = u then MAKETRUE(ℓ, c);
5  end
6  while propagationQueue is non-empty do
7      ℓ ← propagationQueue.DEQUEUE();
8      ℓ.value ← t;
9      for c ∈ ℓ̄.watchers do
10         if Some ℓ' ∈ c \ c.watches satisfies ℓ'.value ≠ f then
11             ℓ̄.watchers ← ℓ̄.watchers \ {c};
12             ℓ'.watchers ← ℓ'.watchers ∪ {c};
13             c.watches ← c.watches \ {ℓ̄} ∪ {ℓ'}
14         else
15             ℓ' ← the other watch of c;
               // If ℓ'.value = t the clause is satisfied and
                  nothing has to happen.
16             if ℓ'.value = u then
17                 MAKETRUE(ℓ', c);
18             else if ℓ'.value = f then
19                 propagationQueue.CLEAR();
20                 return c;
21             end
22         end
23     end
24 end
25 return ⊥;
```

Clauses met 1 literal zijn standaard unit

```
(define (propagate-single-lit-clauses! interpret)
    (for-each (lambda (clause)
                (if (= (clause-size clause) 1)
                    (let ((lit (car (literals clause))))
                       (log-debug "We have a single lit clause:" (clause->string clause))
                       (cond ((false? interpret lit)
                              (exit clause))
                             ((unknown? interpret lit)
                              (make-true! lit interpret clause))))))
              (clauses formula))
    interpret)
```

```
2 for Clause c = ℓ of length one in T do
3 |   if ℓ.value = f then return c;
4 |   if ℓ.value = u then MAKETRUE(ℓ, c);
5 end
```

maak literal waar + extra boekhouding

Literals propageren vereist boekhouding

```
(define (make-true! lit interpret reason)
    (log-debug "Making things come TRUE:" (literal->string lit))
    (true! interpret lit)
    (twls:reason! twls lit reason)
    (twls:level! twls lit current-decision-level)
    (queue:enqueue! prop-q lit))
```

De clause die propagatie toeliet (nodig voor CDCL)

Enkel in CDCL-code!

```
1  MAKETRUE(ℓ, reason):
   Input  : A literal ℓ to be made true and its reason reason
2  if ℓ is a negative literal ℓ = x̄ then
3  |  x.value ← f;
4  |  x.reason ← c;
5  |  x.level ← currentDecisionLevel;
6  else
7  |  ℓ.value ← t;
8  |  ℓ.reason ← c;
9  |  ℓ.level ← currentDecisionLevel;
10 end
11 propagationQueue.ENQUEUE(ℓ);
12 assignmentStack.PUSH(ℓ);
```

Unit-propagatie met TWLS

```
6  while propagationQueue is non-empty do
7      ℓ ← propagationQueue.DEQUEUE();
8      ℓ.value ← t;
9      for c ∈ ℓ̄.watchers do
10         if Some ℓ′ ∈ c \ c.watches satisfies ℓ′.value ≠ f then
11             ℓ̄.watchers ← ℓ̄.watchers \ {c};
12             ℓ′.watchers ← ℓ′.watchers ∪ {c};
13             c.watches ← c.watches \ {ℓ̄} ∪ {ℓ′}
14         else
```

Unit-propagatie met TWLS

```
15          ℓ′ ← the other watch of c;
            // If ℓ′.value = t the clause is satisfied and
               nothing has to happen.
16          if ℓ′.value = u then
17            makeTrue(ℓ′, c);
18          else if ℓ′.value = f then
19            propagationQueue.clear();
20            return c;
21          end
22        end
23      end
24    end
25    return ⊥;
```

Opdracht

Implementeer unit-propagatie in DPLL met TWLS

- Inspecteer `a-d/sat/twls.rkt`
- Vul bestand `iterative.rkt` van de opgave aan
  - Procedure `unit-prop!` afwerken
  - Wanneer propagatie-queue opvullen?
  - Unit clauses propageren telkens opnieuw?
- Plaats dit bestand in `a-d/sat/dpll/optimized`

Test jouw implementatie via `test.rkt`

- Pas `a-d/sat/dpll/config.rkt` aan
- Zie bijgeleverde folder `cnf-samples`

Denkvraagjes (extra oefeningen)

- In DPLL wordt het `current-decision-level` niet aangepast
  - Hoe correct wijzigen?
- Interpretatie wordt telkens gekopieerd en op stack geplaatst?
  - Kunnen we een enkel object gebruiken? (m.a.w. zonder kopieën)