Informatics 1A

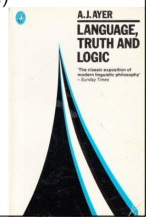
Computation and Logic 9

DPLL (an implementation)

Michael P. Fourman

@mp4man

searching
for satisfaction



$$\frac{\Gamma \vDash \neg B, C}{A, \Gamma \vDash \neg B, C} \quad \frac{\Gamma \vDash D, F}{A, \Gamma \vDash D, F}$$

if A

today we will turn this idea into an algorithm

$$\frac{}{A, \Gamma \vDash \neg A, \neg B, C} \quad \frac{}{A, \Gamma \vDash \neg A, D, F} \quad \frac{}{A, \Gamma \vDash A, B, E} \quad \frac{}{A, \Gamma \vDash A, B, \neg C}$$

Can you find a model with A true?

Can you find a model with ¬A true?

Can you find all models for the original cnf?

$$\frac{}{\neg A, \Gamma \vDash \neg A, \neg B, C} \quad \frac{}{\neg A, \Gamma \vDash \neg A, D, F} \quad \frac{\Gamma \vDash B, E}{\neg A, \Gamma \vDash B, E} \quad \frac{\Gamma \vDash B, \neg C}{\neg A, \Gamma \vDash B, \neg C}$$

if ¬A

$$\frac{}{\neg A, \Gamma \vDash A, B, E} \quad \frac{}{\neg A, \Gamma \vDash A, B, \neg C}$$

$$\frac{\Gamma \vDash \neg B, C}{A, \Gamma \vDash \neg B, C} \quad \frac{\Gamma \vDash D, F}{A, \Gamma \vDash D, F}$$

if A

today we will turn this idea into an algorithm

$$\frac{}{A, \Gamma \vDash \neg A, \neg B, C} \quad \frac{}{A, \Gamma \vDash \neg A, D, F} \quad \frac{}{A, \Gamma \vDash A, B, E} \quad \frac{}{A, \Gamma \vDash A, B, \neg C}$$

Can you find a model with A true?

Find all valuations Γ (consistent sets of literals) such that Γ satisfies the CNF:

$$\Gamma \vDash \neg A, \neg B, C \quad \Gamma \vDash \neg A, D, F \quad \Gamma \vDash A, B, E \quad \Gamma \vDash A, B, \neg C$$

and no smaller set of literals satisfies this CNF.

Can you find a model with ¬A true?

$$\frac{}{\neg A, \Gamma \vDash \neg A, \neg B, C} \quad \frac{}{\neg A, \Gamma \vDash \neg A, D, F} \quad \frac{\Gamma \vDash B, E}{\neg A, \Gamma \vDash B, E} \quad \frac{\Gamma \vDash B, \neg C}{\neg A, \Gamma \vDash B, \neg C}$$

if ¬A

$$\frac{}{\neg A, \Gamma \vDash A, B, E} \quad \frac{}{\neg A, \Gamma \vDash A, B, \neg C}$$

$$\frac{\Gamma \vDash \neg B, C}{A, \Gamma \vDash \neg B, C} \quad \frac{A, \Gamma \vDash \neg B, C}{A, \Gamma \vDash \neg A, \neg B, C} \quad \frac{\Gamma \vDash D, F}{A, \Gamma \vDash D, F} \quad \frac{A, \Gamma \vDash D, F}{A, \Gamma \vDash \neg A, D, F} \quad \frac{}{A, \Gamma \vDash A, B, E} \quad \frac{}{A, \Gamma \vDash A, B, \neg C}$$

if A

today we will turn this idea into an algorithm

Can you find a model with A true?

[A, ¬B, D]
[A, ¬B, F]
[A, C, D]
[A, C, F]

Can you find a model with ¬A true?

[¬A, B]
[¬A, ¬C, E]

$$\frac{}{\neg A, \Gamma \vDash \neg A, \neg B, C} \quad \frac{}{\neg A, \Gamma \vDash \neg A, D, F} \quad \frac{\Gamma \vDash B, E}{\neg A, \Gamma \vDash B, E} \quad \frac{}{\neg A, \Gamma \vDash A, B, E} \quad \frac{\Gamma \vDash B, \neg C}{\neg A, \Gamma \vDash B, \neg C} \quad \frac{}{\neg A, \Gamma \vDash A, B, \neg C}$$

if ¬A

---

cs [Or xs | Or xs <- cs, not (A `elem` xs)] [Or (delete (neg A) xs) | Or xs <- cs, not (A `elem` xs)]

¬A∨C∨D
¬B∨F∨D
¬B∨¬F∨¬C
¬D∨¬B
B∨¬C∨¬A
B∨F∨C
B∨¬F∨¬D
A∨E
A∨F
¬F∨C∨¬E
A∨¬C∨¬E

How can we simplify if we choose to make A True?

---

cs [Or xs | Or xs <- cs, not (A `elem` xs)] [Or (delete (neg A) xs) | Or xs <- cs, not (A `elem` xs)]

¬A∨C∨D	¬A∨C∨D
¬B∨F∨D	¬B∨F∨D
¬B∨¬F∨¬C	¬B∨¬F∨¬C
¬D∨¬B	¬D∨¬B
B∨¬C∨¬A	B∨¬C∨¬A
B∨F∨C	B∨F∨C
B∨¬F∨¬D	B∨¬F∨¬D
A∨E	
A∨F	
¬F∨C∨¬E	¬F∨C∨¬E
A∨¬C∨¬E	

cs	[Or xs | Or xs <- cs, not (A `elem` xs)]	[Or (delete (neg A) xs) | Or xs <- cs, not (A `elem` xs)]
¬A∨C∨D	¬A∨C∨D	C∨D
¬B∨F∨D	¬B∨F∨D	¬B∨F∨D
¬B∨¬F∨¬C	¬B∨¬F∨¬C	¬B∨¬F∨¬C
¬D∨¬B	¬D∨¬B	¬D∨¬B
B∨¬C∨¬A	B∨¬C∨¬A	B∨¬C
B∨F∨C	B∨F∨C	B∨F∨C
B∨¬F∨¬D	B∨¬F∨¬D	B∨¬F∨¬D
A∨E		
A∨F		
¬F∨C∨¬E	¬F∨C∨¬E	¬F∨C∨¬E
A∨¬C∨¬E		

---

cs

¬A∨C∨D
¬B∨F∨D
¬B∨¬F∨¬C
¬D∨¬B
B∨¬C∨¬A
B∨F∨C
B∨¬F∨¬D
A∨E
A∨F
¬F∨C∨¬E
A∨¬C∨¬E

```
import Data.List(elem, delete)
data Literal a = N a | P a deriving (Eq, Show)
neg :: Literal a -> Literal a
neg (P a) = N a
neg (N a) = P a
data Clause  a = Or  [Literal a]
data Form    a = And [Clause a]

(<<) :: Eq a => [Clause a] -> Literal a -> [Clause a]
-- reduces the clauses with x set true
cs  << x = [ Or (delete (neg x) xs)
           |  Or xs <- cs, not (x `elem` xs) ]
```

cs << A

C∨D
¬B∨F∨D
¬B∨¬F∨¬C
¬D∨¬B
B∨¬C
B∨F∨C
B∨¬F∨¬D
¬F∨C∨¬E

---

cs	cs << neg A	cs << A
¬A∨C∨D		C∨D
¬B∨F∨D	?	¬B∨F∨D
¬B∨¬F∨¬C		¬B∨¬F∨¬C
¬D∨¬B	?	¬D∨¬B
B∨¬C∨¬A		B∨¬C
B∨F∨C	?	B∨F∨C
B∨¬F∨¬D		B∨¬F∨¬D
A∨E	?	
A∨F		
¬F∨C∨¬E		¬F∨C∨¬E
A∨¬C∨¬E		

```
cs  << x = [ Or (delete (neg x) xs)
           |  Or xs <- cs, not (x `elem` xs) ]
```

cs [Or xs | xs <- cs, not (neg A `elem` xs)] cs << neg A cs << A

¬AvCvD			CvD
¬BvFvD	¬BvFvD		¬BvFvD
¬Bv¬Fv¬C	¬Bv¬Fv¬C		¬Bv¬Fv¬C
¬Dv¬B	¬Dv¬B		¬Dv¬B
Bv¬Cv¬A			Bv¬C
BvFvC	BvFvC		BvFvC
Bv¬Fv¬D	Bv¬Fv¬D		Bv¬Fv¬D
AvE	AvE		
AvF	AvF		
¬FvCv¬E	¬FvCv¬E		¬FvCv¬E
Av¬Cv¬E	Av¬Cv¬E		

```
cs  << x = [ Or (delete (neg x) xs)
           |  Or xs <- cs, not (x `elem` xs) ]
```

---

cs [Or xs | xs <- cs, not (neg A `elem` xs)] cs << neg A cs << A

¬AvCvD			CvD
¬BvFvD	¬BvFvD	¬BvFvD	¬BvFvD
¬Bv¬Fv¬C	¬Bv¬Fv¬C	¬Bv¬Fv¬C	¬Bv¬Fv¬C
¬Dv¬B	¬Dv¬B	¬Dv¬B	¬Dv¬B
Bv¬Cv¬A			Bv¬C
BvFvC	BvFvC	BvFvC	BvFvC
Bv¬Fv¬D	Bv¬Fv¬D	Bv¬Fv¬D	Bv¬Fv¬D
AvE	AvE	E	
AvF	AvF	F	
¬FvCv¬E	¬FvCv¬E	¬FvCv¬E	¬FvCv¬E
Av¬Cv¬E	Av¬Cv¬E	¬Cv¬E	

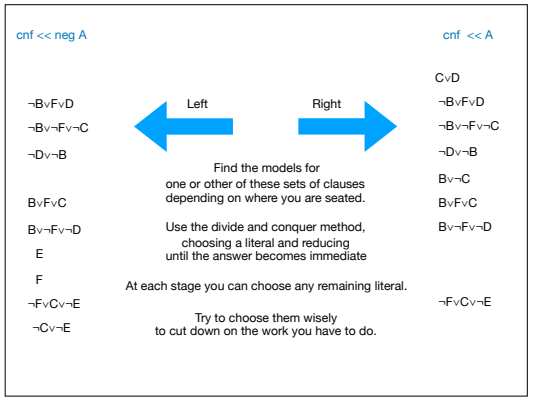
```
cs  << x = [ Or (delete (neg x) xs)
           |  Or xs <- cs, not (x `elem` xs) ]
```

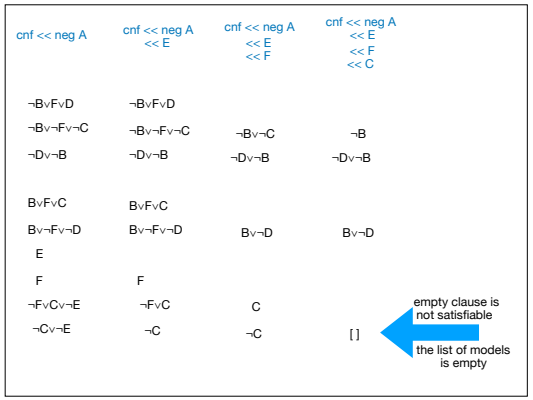
---

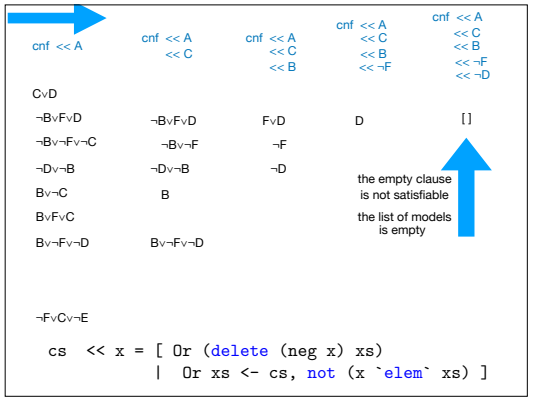
cs cs << neg A cs << A

¬AvCvD	if we choose a literal x then we can consider		CvD
¬BvFvD	the models in which x is true	¬BvFvD	¬BvFvD
¬Bv¬Fv¬C	and	¬Bv¬Fv¬C	¬Bv¬Fv¬C
¬Dv¬B	the models in which neg x is true	¬Dv¬B	¬Dv¬B
Bv¬Cv¬A			Bv¬C
BvFvC	models cnf =	BvFvC	BvFvC
Bv¬Fv¬D	[A : m | m <- models (cnf << A)]	Bv¬Fv¬D	Bv¬Fv¬D
AvE	++	E	
AvF	[neg A : m | m <- models (cnf << neg A)]	F	
¬FvCv¬E		¬FvCv¬E	¬FvCv¬E
Av¬Cv¬E		¬Cv¬E	

```
cs  << x = [ Or (delete (neg x) xs)
           |  Or xs <- cs, not (x `elem` xs) ]
```








```
cs  << x = [ Or (delete (neg x) xs)
           |  Or xs <- cs, not (x `elem` xs) ]
```

```
cs  << x = [ Or (delete (neg x) xs)
           |  Or xs <- cs, not (x `elem` xs) ]

models :: Eq a => [Clause a] -> [[Literal a]]
-- returns the list of satisfying valuations
models (Or (x : xs): cs) =
     [     x : m | m <- models (cs << x) ]
     ++
     [ neg x : m | m <- models (Or xs :(cs << neg x)) ]
```

Which patterns are missing?

```
*Main> :set -W
*Main> :load DPLL.hs
[1 of 1] Compiling Main    ( DPLL.hs, interpreted )

Tutorial6.hs:26:1: warning: [-Wincomplete-patterns]
    Pattern match(es) are non-exhaustive
    In an equation for 'models':
        Patterns not matched:
            []
            (Or []:_)
   |
26 | models (Or (x : xs): cs) =
   | ^^^^^^^^^^^^^^^^^^^^^^^^^^...
```

```
models :: Eq a => Form a -> [[Literal a]]
-- returns the list of satisfying valuations
models     []             = undefined -- null form
models (Or []      : _  )) = undefined -- null clause
models (Or (x : xs): cs )) =
  [     x : m | m <- models (cs << x) ]
  ++
  [ neg x : m | m <- models (Or xs: (cs << neg x)) ]
```

```haskell
models :: Eq a => Form a -> [[Literal a]]
-- returns the list of satisfying valuations
models     []              = undefined -- null form
models (Or []      : _  )) = undefined -- null clause
models (Or [x]     : cs  ) = undefined -- unit clause
models (Or (x : xs): cs )) =
  [     x : m | m <- models (cs << x) ]
  ++
  [ neg x : m | m <- models (Or xs: (cs << neg x)) ]
```

```haskell
models :: Eq a => [Clause a] -> [[Literal a]]
-- returns the list of satisfying valuations
models clauses =
   case clauses of
   [ ]             -> undefined -- empty form
   Or [ ]     : _  -> undefined -- empty clause
   Or [x]     : _  -> undefined -- unit clause
   Or (x : xs): cs ->
     [     x : m | m <- models (cs << x) ]
     ++
     [ neg x : m | m <- models (Or xs :(cs << neg x)) ]
```

```haskell
models clauses =
   case prioritise clauses of
   [ ]             -> undefined -- empty form
   Or [ ]     : _  -> undefined -- empty clause
   Or [x]     : _  -> undefined -- unit clause
   Or (x : xs): cs ->
     [     x : m | m <- models (cs << x) ]
     ++
     [ neg x : m | m <- models (Or xs :(cs << neg x)) ]
  where prioritise = id -- for the time being
```

```haskell
models :: Eq a => [Clause a] -> [[Literal a]]
-- returns the list of satisfying valuations
models clauses =
   case prioritise clauses of
   [ ]             -> [[]] -- trivially satisfied
   Or [ ]     : _  -> [ ]  --     never satisfied
   Or [x]     : _  ->      --         unit clause
     [     x : m | m <- models (cs << x) ]
   Or (x : xs): cs ->
     [     x : m | m <- models (cs << x) ]
     ++
     [ neg x : m | m <- models (Or xs :(cs << neg x)) ]
  where prioritise = id -- for the time being
```

```haskell
models :: Eq a => [Clause a] -> [[Literal a]]
-- returns the list of satisfying valuations
models clauses =
   case prioritise clauses of
   [ ]             -> [[]] -- trivially satisfied
   Or [ ]     : _  -> [ ]  --     never satisfied
   Or [x]     : _  ->      --         unit clause
     [     x : m | m <- models (cs << x) ]
   Or (x : xs): cs ->
     [     x : m | m <- models (cs << x) ]
     ++
     [ neg x : m | m <- models (Or xs :(cs << neg x)) ]
  where prioritise =
          sortOn (\(Or xs) -> (min 3 . length) xs)
```
