SAT

Etienne Renault & Souheib Baarir

Avril 2015

https://www.lrde.epita.fr/~renault/teaching/imc/

Avant-propos 1/2

Problème

On souhaite organiser une réunion entre John, Catherine, Anne et Pierre. Leurs disponibilités sont les suivantes :

- John est libre lundi, mercredi et jeudi
- Catherine doit garder sa fille mercredi
- Anne est en vacances vendredi
- Pierre a cours tous les jeudi et tous les mardi il n'est donc pas disponible ces jours là
- Personne ne travaille le weekend

Avant-propos 1/2

Problème

On souhaite organiser une réunion entre John, Catherine, Anne et Pierre. Leurs disponibilités sont les suivantes :

- John est libre lundi, mercredi et jeudi
- Catherine doit garder sa fille mercredi
- Anne est en vacances vendredi
- Pierre a cours tous les jeudi et tous les mardi il n'est donc pas disponible ces jours là
- Personne ne travaille le weekend

Question

Quand peut-on planifier la réunion ?

Avant-propos 2/2

On peut se focaliser sur les jours qui ne sont pas dans le weekend.

Avant-propos 2/2

On peut se focaliser sur les jours qui ne sont pas dans le weekend.

On peut alors exprimer la relation suivante :
$(Mon \vee Wed \vee Thu) \wedge (\neg Wed) \wedge (\neg Fri) \wedge (\neg Tue \wedge \neg Thu)$

Avant-propos 2/2

On peut se focaliser sur les jours qui ne sont pas dans le weekend.

On peut alors exprimer la relation suivante :
$(Mon \vee Wed \vee Thu) \wedge (\neg Wed) \wedge (\neg Fri) \wedge (\neg Tue \wedge \neg Thu)$

Pour qu'il y ait une solution chaque clause doit être vraie. On a nécessairement :

- $Wed = \bot \implies \neg Wed = \top$
- $Frid = \bot \implies \neg Frid = \top$
- $Tues = \bot \implies \neg Tues = \top$
- $Thu = \bot \implies \neg Thu = \top$

Avant-propos 2/2

On peut se focaliser sur les jours qui ne sont pas dans le weekend.

On peut alors exprimer la relation suivante :
$(Mon \vee Wed \vee Thu) \wedge (\neg Wed) \wedge (\neg Fri) \wedge (\neg Tue \wedge \neg Thu)$

Pour qu'il y ait une solution chaque clause doit être vraie. On a nécessairement :

- $Wed = \bot \implies \neg Wed = \top$
- $Frid = \bot \implies \neg Frid = \top$
- $Tues = \bot \implies \neg Tues = \top$
- $Thu = \bot \implies \neg Thu = \top$

On a : $(Mon \vee Wed \vee Thu) \wedge (\neg Wed) \wedge (\neg Fri) \wedge (\neg Tue \wedge \neg Thu)$

Avant-propos 2/2

On peut se focaliser sur les jours qui ne sont pas dans le weekend.

On peut alors exprimer la relation suivante :
$(Mon \vee Wed \vee Thu) \wedge (\neg Wed) \wedge (\neg Fri) \wedge (\neg Tue \wedge \neg Thu)$

Pour qu'il y ait une solution chaque clause doit être vraie. On a nécessairement :

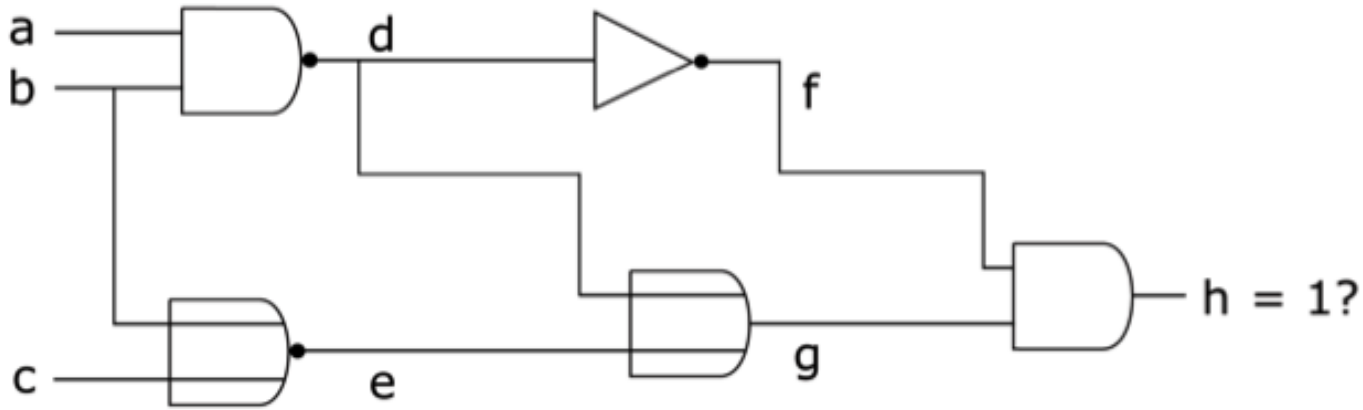
- $Wed = \bot \implies \neg Wed = \top$
- $Frid = \bot \implies \neg Frid = \top$
- $Tues = \bot \implies \neg Tues = \top$
- $Thu = \bot \implies \neg Thu = \top$

On a : $(Mon \vee Wed \vee Thu) \wedge (\neg Wed) \wedge (\neg Fri) \wedge (\neg Tue \wedge \neg Thu)$

La réunion aura lieu Lundi !

Motivations

- Planification et diagnostiques
- Conception de circuits
- Preuve automatique de théorèmes
- model-checking



Un problème SAT peut être vu comme la représentation d'un circuit combinatoire

SAT

Boolean Satisfiability Problem

Étant donné une formule booléenne, on cherche une affectation des variables qui évalue la formule à $\top$, sinon on veut prouver qu'une telle affectation n'existe pas.

Exemple

- $F = ab + cd$ est satisfiable pour $c = 1$ et $d = 1$ par exemple.
- $G = abc(b \text{ xor } c)$ n'est pas statisfiable

a	b	c	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0

a	b	c	
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Représentation du problème

Forme normale conjonctive (CNF)

La formule est toujours représentée sous une forme CNF, c'est-à-dire comme une conjonction de disjonction de littéraux

- la formule est une conjonction de *clauses*
- les *clauses* sont des disjonctions de *littéraux*
- les *littéraux* sont des variables ou leur négation

SATisfiabilité

Pour qu'une formule soit satisfiable, il faut que chaque clause le soit.

Remarques & définitions

Une clause est satisfiable si au moins un de ses littéraux est affecté à la valeur $\top$. Par exemple si $a = \bot$, $\neg c = \top$ et $\neg b$ non affecté, alors :

- $(a \vee \neg b \vee \neg c)$ est satisfiable

Une clause est non-satisfiable si tous ses littéraux sont affectés à la valeur $\bot$. Par exemple si $a = \neg b = \neg c = \bot$ alors :

- $(a \vee \neg b \vee \neg c)$ est non-satisfiable

Une clause est unitaire si elle contient un littéral non affecté tandis que tous les autres littéraux sont à $\bot$. Par exemple, si $a = \bot$, $\neg b = \bot$ et $\neg c$ non affecté, alors :

- $(a \vee \neg b \vee \neg c)$ est unitaire

Complexité

Forme normale disjonctive (DNF)

Une formule est sous forme normale disjonctive si elle est représentée comme une disjonction de conjonction de littéraux

Complexité

- Si le problème est donné sous forme DNF, le problème est trivial à résoudre, chaque terme est une solution
- Si le problème est donné sous forme CNF, le problème est NP-complet (en général), et l'on est obligé de trouver la bonne affectation des variables

Complexité

Forme normale disjonctive (DNF)

Une formule est sous forme normale disjonctive si elle est représentée comme une disjonction de conjonction de littéraux

Complexité

- Si le problème est donné sous forme DNF, le problème est trivial à résoudre, chaque terme est une solution
- Si le problème est donné sous forme CNF, le problème est NP-complet (en général), et l'on est obligé de trouver la bonne affectation des variables

Dans la pratique la forme CNF est utilisée car elle est exponentiellement plus petite que la forme DNF !

Algorithme pour la SATisfiabilité

Deux types d'algorithmes existent :

- les algorithmes incomplets : peuvent converger très vite mais peuvent seulement prouver la satisfiabilité ou la non-satisfiabilité :
  - Local Search
  - Simulation
  - algorithmes génétiques, ...

- les algorithmes exacts : peuvent prendre des années de calcul, mais offrent une procédure de décision exacte
  - Systèmes de preuve
  - BDD
  - Backtrack Search/DPLL/CDCL
  - ...

Local Search

Ne peux *généralement* prouver que la non-satisfiabilité

Exemple : $(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$

On commence par choisir une affectation aléatoire des variables

Repéter N fois :

- Si toutes les clauses ne sont pas satisfaites, on inverse la valeur d'une variable au hasard (par exemple x_4)
- Sinon on a terminé (toutes les clauses sont satisfaites)

Exemple

$$(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$$

Initialement $x_1 = \top$, $x_2 = \top$, $x_3 = \top$, $x_1 = \bot$

Exemple

$$(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$$

Initialement $x_1 = \top$, $x_2 = \top$, $x_3 = \top$, $x_1 = \bot$

On inverse la valeur de x_3, on a donc $x_3 = \bot$

Exemple

$$(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$$

Initialement $x_1 = \top$, $x_2 = \top$, $x_3 = \top$, $x_1 = \bot$

On inverse la valeur de x_3, on a donc $x_3 = \bot$

On inverse la valeur de x_4, on a donc $x_4 = \top$

Remarques

Cet algorithme est vraiment efficace dans certains contextes ...

Remarques

Cet algorithme est vraiment efficace dans certains contextes . . .
. . . Mais a trop de limitations !

Remarques

Cet algorithme est vraiment efficace dans certains contextes ...
...Mais a trop de limitations !

Les SAT solvers modernes utilisent :

- des algorithmes exacts de branchement ou de backtrack
- avec des procédures d'élagage
  - cela permet d'avoir une complexité meilleure que le 2^n attendu par une exploration exhaustive

Définitions & Heuristique

Un littéral est dit pur si il n'apparaît que sous sa forme positive ou que sous sa forme négative dans une formule :

- $(\neg x1 \vee x2)(x3 \vee \neg x2)(x4 \vee \neg x5)(x5 \vee \neg x4)$
- dans cette formule x_1 et $x3$ sont des littéraux purs

Règle des littéraux purs : Les clauses contenant des littéraux purs peuvent être supprimées de la formule, i.e. on affecte la variable de manière à rendre la clause satisfiable.

- L'exemple ci-dessus peut alors être réduit à $(x4 \vee \neg x5)(x5 \vee \neg x4)$

Définitions & Heuristique

Règle des clauses unitaires : Étant donné une clause unitaire, son unique littéral non affecté doit prendre la valeur $\top$ pour que la clause soit satisfaite :

- Par exemple, pour la clause $(x_1 \vee \neg x_2 \vee x_3)$ si $x_1 = \bot$ et $\neg x_2 = \bot$ alors x_3 doit prendre la valeur $\top$

Propagation unitaire : On itère l'application de la règle ci-dessus

- $(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$

Définitions & Heuristique

Règle des clauses unitaires : Étant donné une clause unitaire, son unique littéral non affecté doit prendre la valeur $\top$ pour que la clause soit satisfaite :

- Par exemple, pour la clause $(x_1 \vee \neg x_2 \vee x_3)$ si $x_1 = \bot$ et $\neg x_2 = \bot$ alors x_3 doit prendre la valeur $\top$

Propagation unitaire : On itère l'application de la règle ci-dessus

- $(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$
- $(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$

Définitions & Heuristique

Règle des clauses unitaires : Étant donné une clause unitaire, son unique littéral non affecté doit prendre la valeur $\top$ pour que la clause soit satisfaite :

- Par exemple, pour la clause $(x_1 \vee \neg x_2 \vee x_3)$ si $x_1 = \bot$ et $\neg x_2 = \bot$ alors x_3 doit prendre la valeur $\top$

Propagation unitaire : On itère l'application de la règle ci-dessus

- $(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$
- $(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$
- $(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$

Définitions & Heuristique

Règle des clauses unitaires : Étant donné une clause unitaire, son unique littéral non affecté doit prendre la valeur $\top$ pour que la clause soit satisfaite :

- Par exemple, pour la clause $(x_1 \vee \neg x_2 \vee x_3)$ si $x_1 = \bot$ et $\neg x_2 = \bot$ alors x_3 doit prendre la valeur $\top$

Propagation unitaire : On itère l'application de la règle ci-dessus

- $(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$
- $(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$
- $(x_1 \vee \neg x_2 \vee \neg x_3)(\neg x_1 \vee \neg x_3 \vee x_4)(\neg x_1 \vee \neg x_2 \vee x_4)$

Cette propagation permet de montrer la satisfiabilité en trouvant une affectation ou la non-satisfiabilité en trouvant des conflits (par exemple si l'on remplace x_4 par $\neg x_4$ dans la dernière clause)

Résolution

Règle de résolution : si une formule φ contient $(x \vee \alpha)$ et $(\neg x \vee \beta)$ alors on peut inférer $(\alpha \vee \beta)$:

- $(x \vee \alpha)(\neg x \vee \beta) \models (\alpha \vee \beta)$

Algorithme de résolution [Davis et al, 1960]

Répéter :

- Sélectionner une variable x
- Appliquer la méthode de résolution pour toutes les paires $(x \vee \alpha)$ et $(\neg x \vee \beta)$
- supprimer toutes les clauses contenant x ou $\neg x$
- appliquer les règles des littéraux pure et de propagation unitaire

Jusqu'à ce que l'on tombe sur une clause vide (unsat) ou sur une formule ne contenant que des littéraux purs

Résolution – exemple

$$(x1 \vee \neg x2 \vee \neg x3)(\neg x1 \vee \neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$$

Résolution – exemple

$(x1 \vee \neg x2 \vee \neg x3)(\neg x1 \vee \neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(\neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

Résolution – exemple

$(x1 \vee \neg x2 \vee \neg x3)(\neg x1 \vee \neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(\neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(\neg x3 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

Résolution – exemple

$(x1 \vee \neg x2 \vee \neg x3)(\neg x1 \vee \neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(\neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(\neg x3 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(x3 \vee x4)(x3 \vee \neg x4)$

Résolution – exemple

$(x1 \vee \neg x2 \vee \neg x3)(\neg x1 \vee \neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(\neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(\neg x3 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(x3 \vee x4)(x3 \vee \neg x4)$

$(x3)$

Résolution – exemple

$(x1 \vee \neg x2 \vee \neg x3)(\neg x1 \vee \neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(\neg x2 \vee \neg x3)(x2 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(\neg x3 \vee x3)(x3 \vee x4)(x3 \vee \neg x4)$

$(x3 \vee x4)(x3 \vee \neg x4)$

$(x3)$

La formule est satisfiable !

DPLL

En 1962, M. Davis, G. Logemann and D. Loveland ont proposé un algorithme alternatif : il s'agit en réalité d'un algorithme de backtracking !

Idée générale :

- Plutôt que de supprimer une variable à chaque étape, l'ensemble des variables est divisé à chaque étape
- Application des règles de propagation unitaire et de la règle des littéraux

DPLL - algorithme

À chaque étape :

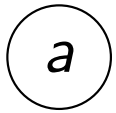
- Choisir une affectation d'une variable
- Appliquer les règles de propagation unitaire et (optionnellement) la règle des pure litéraux
- si un conflit est identifié, on effectue un backtrack
  - si on ne peut plus backtracker retourner unsat
  - sinon on effectue une propagation unitaire
- Si la formule est satisfaite, on a terminé
- Sinon recommencer !

DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$

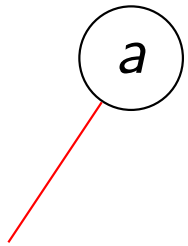
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



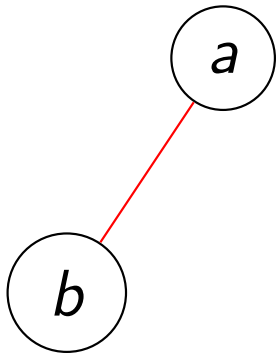
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



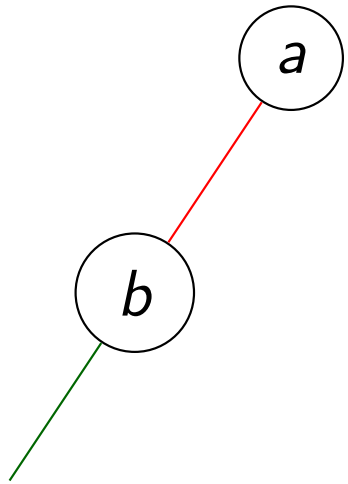
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



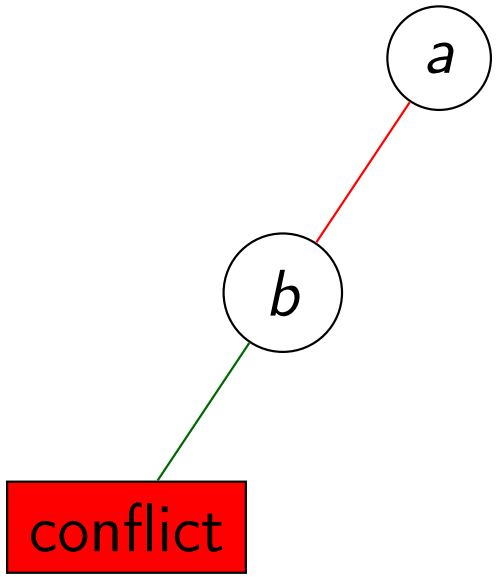
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



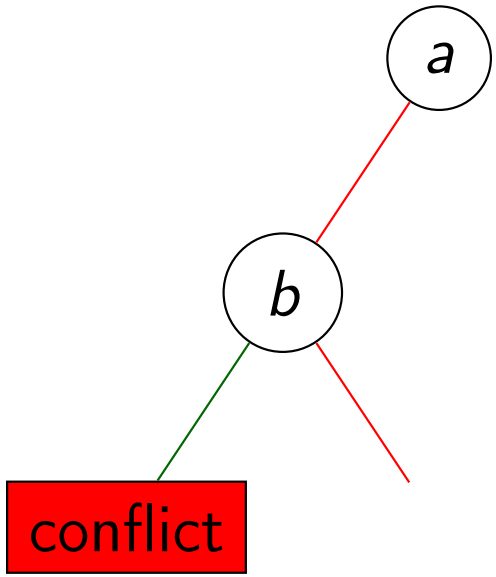
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



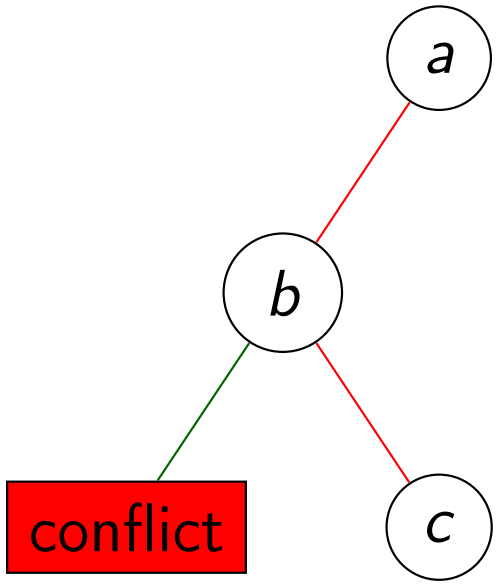
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



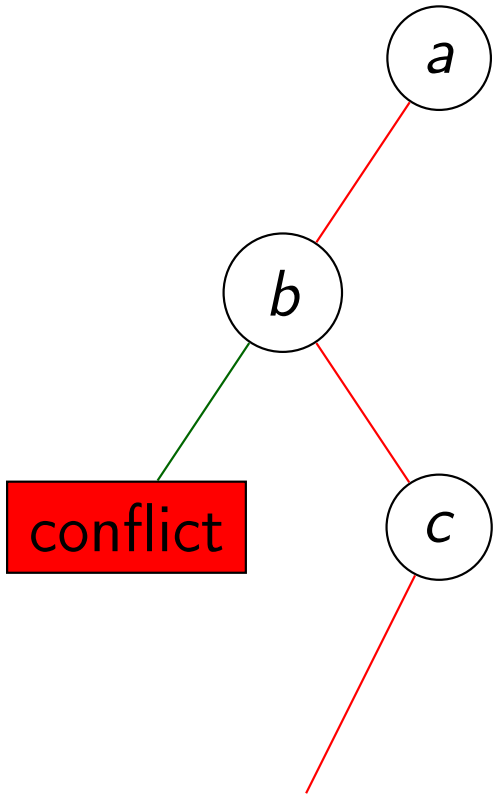
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



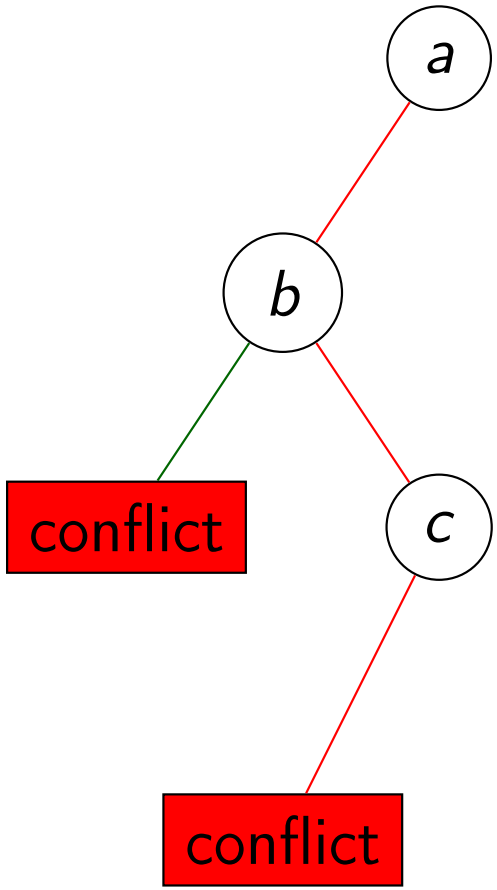
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



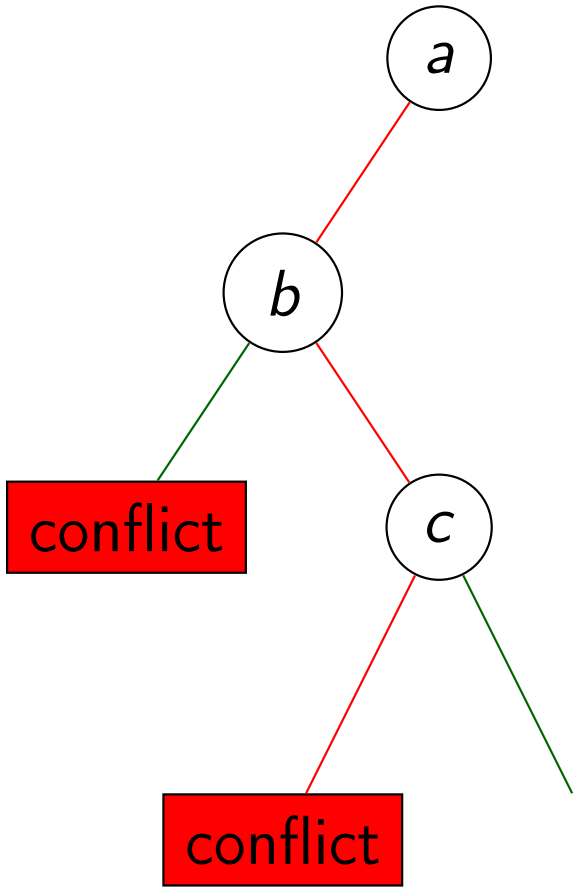
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



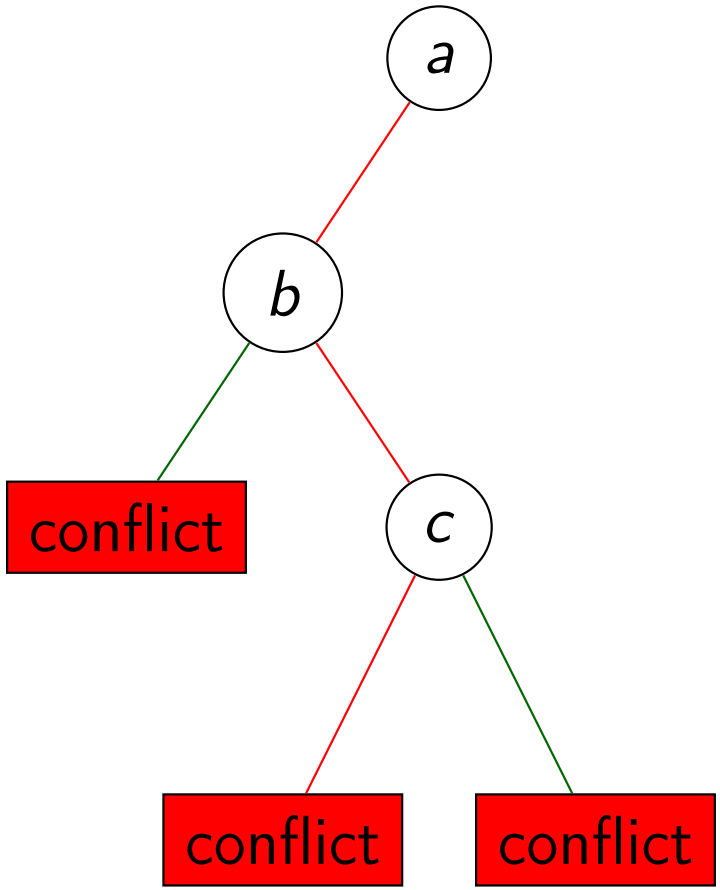
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



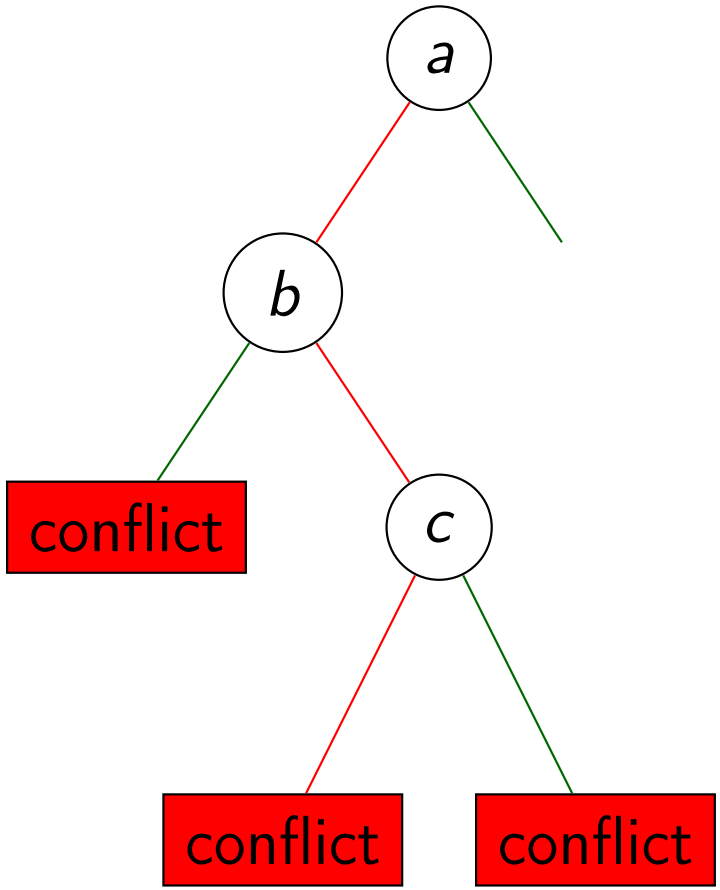
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



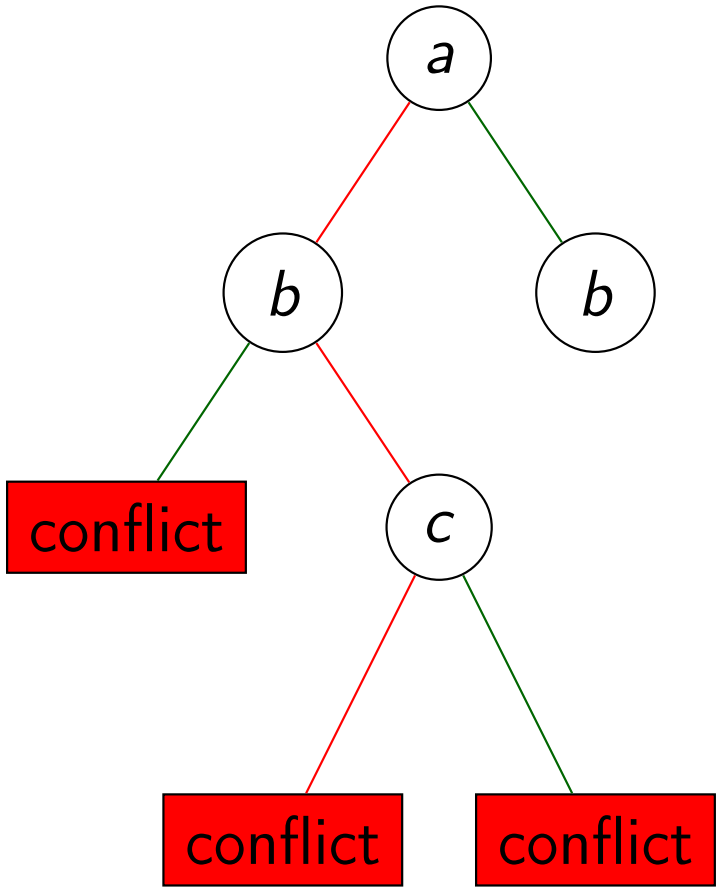
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



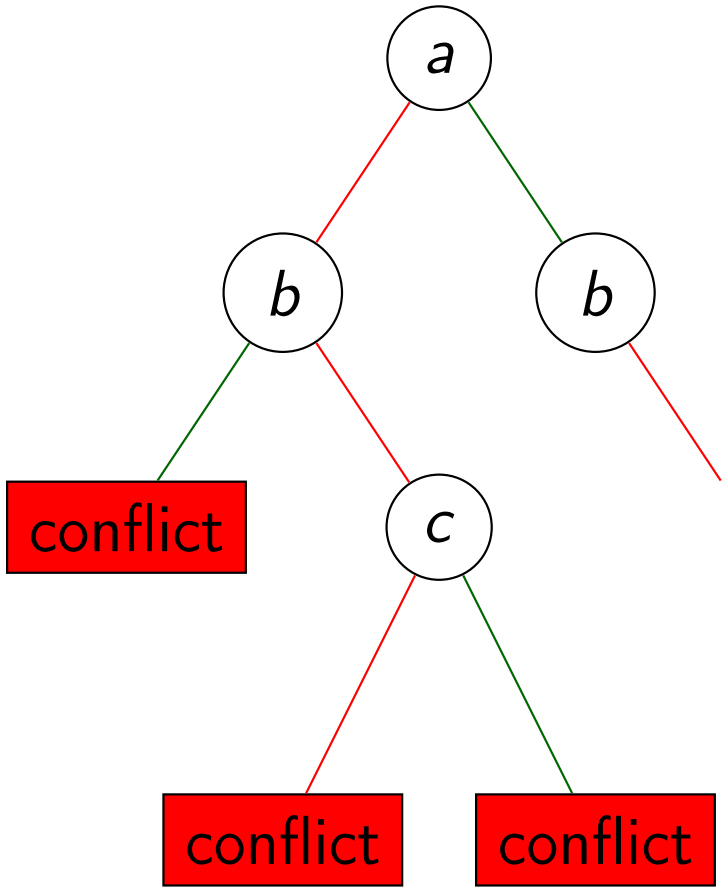
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



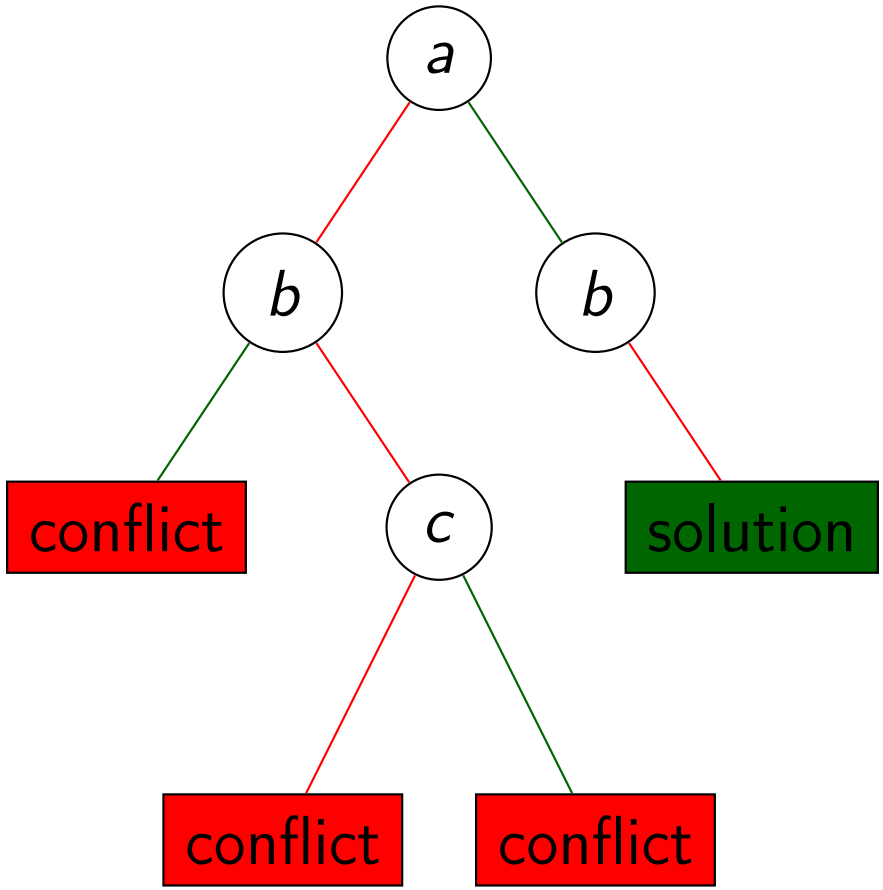
DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



DPLL - exemple

$(a \vee \neg b \vee d)(a \vee \neg b \vee e)(\neg b \vee \neg d \vee \neg e)(a \vee b \vee c \vee d)(a \vee b \vee c \vee \neg d)(a \vee b \vee \neg c \vee e)(a \vee b \vee \neg c \vee \neg e)$



Facteurs d'efficacité

- Heuristiques de décision
  - Sous ensemble de clauses à analyse
  - Prochaines variables à affecter
- Analyse des backtrack
  - Pas nécessairement chronologiques ...
- Analyse de conflits
  - Injection de nouvelles clauses dans l'instance

CDCL SAT Solvers

- Inspiré de l'algorithme précédent
  - Doit prouver la satisfiabilité et la non-satisfiabilité
- De nouvelles clauses sont apprises en fonction des conflits détectés
- La structure de ces conflits peut être exploitée
- Une phase de backtrack est régulièrement relancée (et pas nécessairement dans un ordre chronologique)

CDCL SAT Solvers

- Inspiré de l'algorithme précédent
  - Doit prouver la satisfiabilité et la non-satisfiabilité
- De nouvelles clauses sont apprises en fonction des conflits détectés
- La structure de ces conflits peut être exploitée
- Une phase de backtrack est régulièrement relancée (et pas nécessairement dans un ordre chronologique)

Cette technique introduite dans le milieu des années 90 permet de traiter des centaines de milliers de variables et des dizaines de millions de clauses

Clause Learning

Pendant la recherche, pour chaque conflit on apprend une nouvelle clause qui explique et préviens la répétition du même conflit

Clause Learning

Pendant la recherche, pour chaque conflit on apprend une nouvelle clause qui explique et préviens la répétition du même conflit

Exemple : $\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$

Clause Learning

Pendant la recherche, pour chaque conflit on apprend une nouvelle clause qui explique et préviens la répétition du même conflit

Exemple : $\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$

Supposons la décision $c = f = a = \bot$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

Clause Learning

Pendant la recherche, pour chaque conflit on apprend une nouvelle clause qui explique et préviens la répétition du même conflit

Exemple : $\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$

Supposons la décision $c = f = a = \bot$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

Étude des implications : $b = \top$,

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

Clause Learning

Pendant la recherche, pour chaque conflit on apprend une nouvelle clause qui explique et préviens la répétition du même conflit

Exemple : $\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$

Supposons la décision $c = f = a = \bot$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

Étude des implications : $b = \top$,

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

. . . donc $d = e = \top$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

Clause Learning

Pendant la recherche, pour chaque conflit on apprend une nouvelle clause qui explique et préviens la répétition du même conflit

Exemple : $\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$

Supposons la décision $c = f = a = \bot$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

Étude des implications : $b = \top$,

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

. . . donc $d = e = \top$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f) \Rightarrow \text{CONFLIT !}$$

Clause Learning

Pendant la recherche, pour chaque conflit on apprend une nouvelle clause qui explique et préviens la répétition du même conflit

Exemple : $\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$

Supposons la décision $c = f = a = \bot$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

Étude des implications : $b = \top$,

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

. . . donc $d = e = \top$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f) \Rightarrow \text{CONFLIT !}$$

$$(a = \bot) \wedge (c = \bot) \wedge (f = \bot) \implies (\varphi = \bot)$$

Clause Learning

Pendant la recherche, pour chaque conflit on apprend une nouvelle clause qui explique et préviens la répétition du même conflit

Exemple : $\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$

Supposons la décision $c = f = a = \bot$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

Étude des implications : $b = \top$,

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

. . . donc $d = e = \top$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f) \Rightarrow \text{CONFLIT !}$$

$$(a = \bot) \wedge (c = \bot) \wedge (f = \bot) \implies (\varphi = \bot)$$
$$(\varphi = \top) \implies (a = \top) \vee (c = \top) \vee (f = \top)$$

Clause Learning

Pendant la recherche, pour chaque conflit on apprend une nouvelle clause qui explique et préviens la répétition du même conflit

Exemple : $\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$

Supposons la décision $c = f = a = \bot$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

Étude des implications : $b = \top$,

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$

. . . donc $d = e = \top$

$$(a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f) \Rightarrow \text{CONFLIT !}$$

$$(a = \bot) \wedge (c = \bot) \wedge (f = \bot) \implies (\varphi = \bot)$$
$$(\varphi = \top) \implies (a = \top) \vee (c = \top) \vee (f = \top)$$
Nouvelle clause $(a = \top \vee c = \top \vee f = \top)$

Remarques

- Chaque conflit peut amener une nouvelle clause, on gère donc une base de donnée de clauses conflits
- Cette base devient rapidement large :
  - Pose des problèmes de stockage
  - Engendre un surcoût important pour la propagation unitaire
- Différentes techniques de la BD
  - **Relevance Bounded Learning** : maintenir les clauses les plus importantes pour le sous-espace de recherche courant.
  - **Size Bounded Learning** maintenir que les clauses faisant intervenir i variables (ou moins)
  - Elimination régulière des clauses en se basant sur les deux stratégies précédentes

Conclusion partielle

- SAT vs. BDD
  - BDD : toutes les solutions $\Longrightarrow$ ennemi : taille mémoire
  - SAT : une seule solution $\Longrightarrow$ ennemi : temps de calcul
- Résolution de systèmes avec plusieurs milliers (voir millions) de variables et clauses
- SAT solveurs disponibles : MiniSAT, Glucose, zCHAFF...
- Outils de résolution de problèmes SAT inclus dans les chaînes d'outils de synthèse et de vérification de circuits (vis, cadence...)
- Domaine encore très ouvert :
  - Heuristiques
  - Stratégies de gestion de la BD des clauses
  - Parallélisation

SAT & Model checking

- Pour prouver $Sys \models \varphi$
  - Construction de l'espace d'états
  - CTL : Parcours récursif de l'ensemble d'états en fonction de la relation de transition
  - LTL : Produit du graphe d'états avec un automate reconnaisseur (Büchi), puis recherche de cycles !
- Recherche d'un contre-exemple : encodage en problème SAT
  - Support de LTL [Biere, 1999], [Clarke, 2005]
    Mais ici on s'intéresse aux propriétés de sûreté
  - Recherche d'un conte-exemple de longueur k
    - ⋆ Existe-t-il un chemin dans l'espace (borné par k) menant vers un état ne satisfaisant pas φ
    - ⋆ Exploration de l'arbre d'exécution complet en largeur mais sur une profondeur de taille k

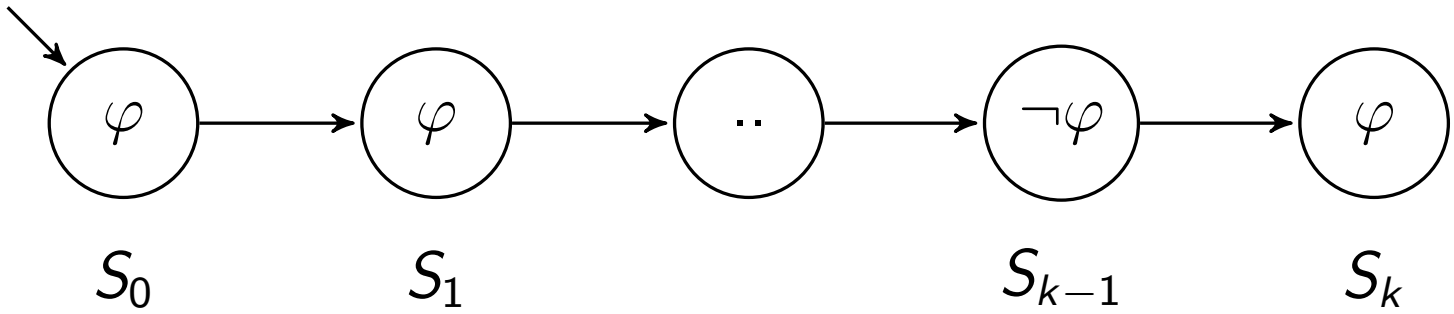
Bounded Model Checking

- Soit φ une formule de sûreté
- Construire le problème SAT représentant :
  - Un contre-exemple de la propriété
  - Qui puisse être exécuté par le circuit (donc compatible avec la relation de transition du circuit)
- Rechercher une solution via un appel au SAT-solver

Exemple

Soit φ une propriété (par exemple "signal_a = signal_b")

Existe-t-il un état accessible en k itération qui satisfait $\neg\varphi$



Exploration bornée de l'espace d'états

L'espace d'état accessible sur k étapes est capturé par :

$$I(S_0) \wedge R(S_0, S_1) \wedge R(S_1, S_2) \wedge \ldots \wedge R(S_{k-1}, S_k)$$

où :

- $R(S_i, S_j)$ est la fonction caractéristique de la relation de transition entre les états S_i et S_j
- $I(S_0)$ est la fonction caractéristique de l'ensemble des états initiaux

Vérification d'une propriété de sureté

La propriété φ est violé sur une des k étapes si :

$$\neg\varphi(S_1) \vee \neg\varphi(S_2) \vee \ldots \neg\varphi(S_{k-1}) \vee \neg\varphi(S_k)$$

Vérification d'une propriété de sureté

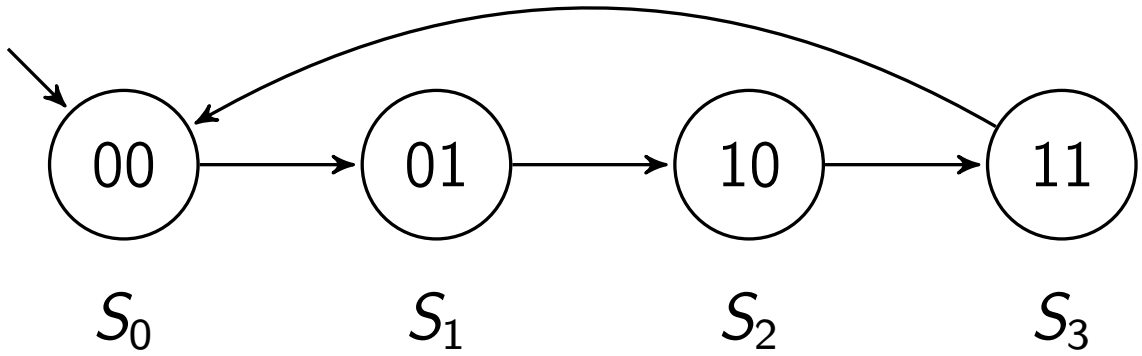
La propriété φ est violé sur une des k étapes si :

$$\neg\varphi(S_1) \vee \neg\varphi(S_2) \vee \ldots \neg\varphi(S_{k-1}) \vee \neg\varphi(S_k)$$

Vérifier la propriété de sûreté φ jusqu'à l'étape k est équivalent à montrer que $\Omega(k)$ est satisfiable :

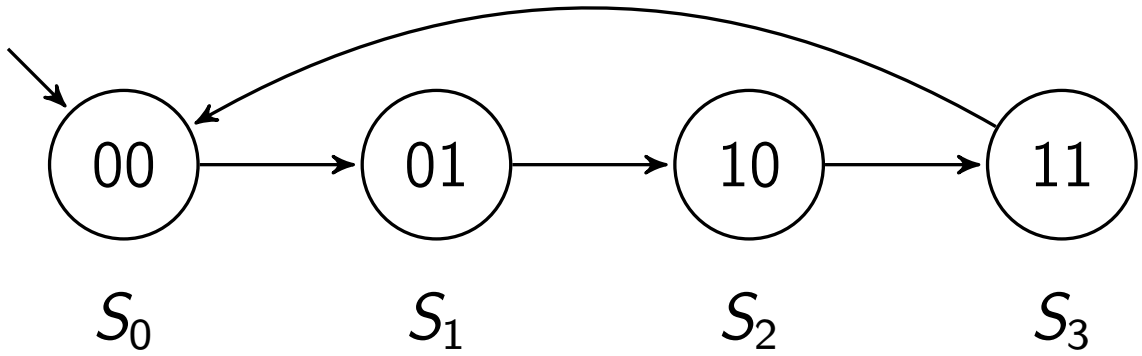
$$\Omega(k) = I(S_0) \wedge \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1}) \wedge \bigvee_{i=0}^{k-1} \neg\varphi(S_i)$$

Exemple - compteur à deux bits 1/2



- Chaque état S est représenté par deux variables $S[1]$ et $S[0]$
- Initialement la valeur est 00, donc $I(S_0) = (\neg S_0[1] \wedge \neg S_0[0])$
- $T(S_0, S_1) = ((S_1[0] \leftrightarrow \neg S_0[0]) \wedge (S_1[1] \leftrightarrow (S_0[0] \textbf{ xor } S_0[0])))$
- $T(S_1, S_2) = ((S_2[0] \leftrightarrow \neg S_1[0]) \wedge (S_2[1] \leftrightarrow (S_1[0] \textbf{ xor } S_1[0])))$
- ...

Exemple - compteur à deux bits 1/2



- Chaque état S est représenté par deux variables $S[1]$ et $S[0]$
- Initialement la valeur est 00, donc $I(S_0) = (\neg S_0[1] \wedge \neg S_0[0])$
- $T(S_0, S_1) = ((S_1[0] \leftrightarrow \neg S_0[0]) \wedge (S_1[1] \leftrightarrow (S_0[0] \textbf{ xor } S_0[0])))$
- $T(S_1, S_2) = ((S_2[0] \leftrightarrow \neg S_1[0]) \wedge (S_2[1] \leftrightarrow (S_1[0] \textbf{ xor } S_1[0])))$
- ...
- Il ne reste plus qu'a ajouter la négation de la propriété !

Exemple - compteur à deux bits 2/2

Si $\varphi = G(S[1] \vee S[0])$, i.e.

- pour $k = 2$:
  $(S_0[1] \wedge S_0[0]) \vee (S_1[1] \wedge S_1[0]) \vee (S_2[1] \wedge S_2[0])$
- pour $k = 3$:
  $(S_0[1] \wedge S_0[0]) \vee (S_1[1] \wedge S_1[0]) \vee (S_2[1] \wedge S_2[0]) \vee (S_3[1] \wedge S_1[0])$

Pour $k = 2$, la propriété est vérifiée, i.e. le problème est unsatisfiable

Pour $k = 3$, la propriété est violé, i.e. le problème est statisfiable.

Extension à tout LTL

- On encode le problème en problème SAT
- On effectue du bounded model-checking
- Lorsque le diamètre du graphe est atteint, on a l'assurance qu'aucun contre-exemple n'existe

Extension à tout LTL

- On encode le problème en problème SAT
- On effectue du bounded model-checking
- Lorsque le diamètre du graphe est atteint, on a l'assurance qu'aucun contre-exemple n'existe
- Cela est très coûteux !

Extension à tout LTL

- On encode le problème en problème SAT
- On effectue du bounded model-checking
- Lorsque le diamètre du graphe est atteint, on a l'assurance qu'aucun contre-exemple n'existe
- Cela est très coûteux !

Comment être plus efficace ?

Intuition

Le model-checking borné et l'interpolation de Craig permettent de calculer des sur-approximation des images (successeurs) par rapport à une propriété.

Cela évite le calcul de l'image exact (qui est coûteux) !

Cela permet de tirer parti de la capacité des SAT-solveur de filtrer les faits non-importants

Lemme d'interpolation [Craig, 1957]

Si $UNSAT(A \wedge B)$ alors il existe un interpolant A' pour (A, B) tel que :

- $A \implies A'$
- $UNSAT(A' \wedge B)$
- A' se compose de variables communes à A et B

Exemple :

- $A = p \wedge q$
- $B = \neg q \wedge r$
- $A' = q$

Model checking basé sur les interpolants

Remarque

Pour calculer l'interpolation, on peut poser $A = I(S_0)$ et B le reste de $\Omega(k)$

Intuition

1. On construit $\Omega(k)$
2. Si $UNSAT(\Omega(k))$ alors, au lieu d'augmenter k, on sur-approxime l'état initial tout en respectant la propriété (via de l'interpolation)
3. Si la surapproximation est différente, retourner à l'étape 2

Conclusion

- Le model checking via SAT permet de combattre l'explosion combinatoire . . . au prix d'un cout temporel important.
- Les ingrédients pour construire un *bon* SAT-solver
  1. Les erreurs ne sont pas des problèmes : clause leearning, backtrack non chronologique
  2. Etre fainéant : structures de données et heuristiques.

The end !