Problèmes SAT
et
Vérification formelle

Souheib Baarir
Souheib.baarir@lip6.fr

Plan du cours

- **Définition d'un problème SAT**
  - Motivation
  - Rappels algèbre Booléenne
  - Définition d'un problème SAT
  - Exemples et exercices
- **Résolution d'un problème SAT**
  - Procédure Davis-Putnam-Logemann-Loveland (DPLL)
  - Facteurs d'efficacité.
  - Les SAT-solveurs modernes…
- **Les problèmes SAT et la vérification formelle**
  - Bounded Model-checking
  - UnBounded Model-checking
    - K-induction
    - Interpolation

Définition d’un problème SAT

Motivation

Le problèmes SAT apparaissent dans plusieurs disciplines mathématiques et informatiques :

- La planification et le diagnostic (en I.A.)
- La conception de circuits
- La preuve automatique de théorèmes
- ***Le model-checking***
- …

Rappels algèbre Booléenne

- a,b,c $\in$ IB={0,1}
- A = < IB, `not` (¬,!), `and` (∧) , `or`(∨) , 0, 1> forme une algèbre de Boole :
  - `And` et `or` sont associatives et commutatives
  - 0 el. neutre de `or`, 1 el. neutre de `and`
  - `And` et `or` sont distributives l'une par rapport à l'autre
  - la somme d'un élément et son complément est 1, le produit d'un élément et son complément est 0
- Lois De Morgan :
  - `not` (a `and` b) = `not` a or `not` b ; `not` (a `or` b) = `not` a `and` `not` b
- Simplifications :
  - Involution : `not not` x = x
  - Absorption : x `and` 0 = 0, x`or` 1 = 1
  - Idempotence : x `and` (x `or` y) = x, x or (x `and` y) = x
  - Autres …

Fonctions booléennes – Représentation

- f : $IB^n \rightarrow IB$ une fonction booléenne à n variables

- Table de vérité

- Sum-of-products :

  F = !x1x2x3 + x1!x2x3 + x1x2x3

- Product-of-sums :

  F = (x1+x2+x3) (x1+x2+!x3) (x1+!x2+x3) (!x1+x2+x3) (!x1+!x2+x3)

- **CNF** et DNF sont les PoS et SoP avant simplification.
  - x1 et !x1 sont des **littéraux**
  - (x1+x2+x3) et (x1+!x2+x3) sont des **clauses**

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Problème SAT

- f : $IB^n \rightarrow IB$.

- $v \in IB^n$, $f(v) = 1 \Leftrightarrow v \in SAT(f)$

- **Problème SAT : étant donnée *f*,**
  - **Trouver un élément $v \in SAT(f)$ / Prouver que $SAT(f)$ est vide.**

- Si *f* est donnée sous forme DNF, le problème est trivial :
  - f(<a,b,c>) = a.b.c + a.b.c $\rightarrow$ chaque terme est une solution

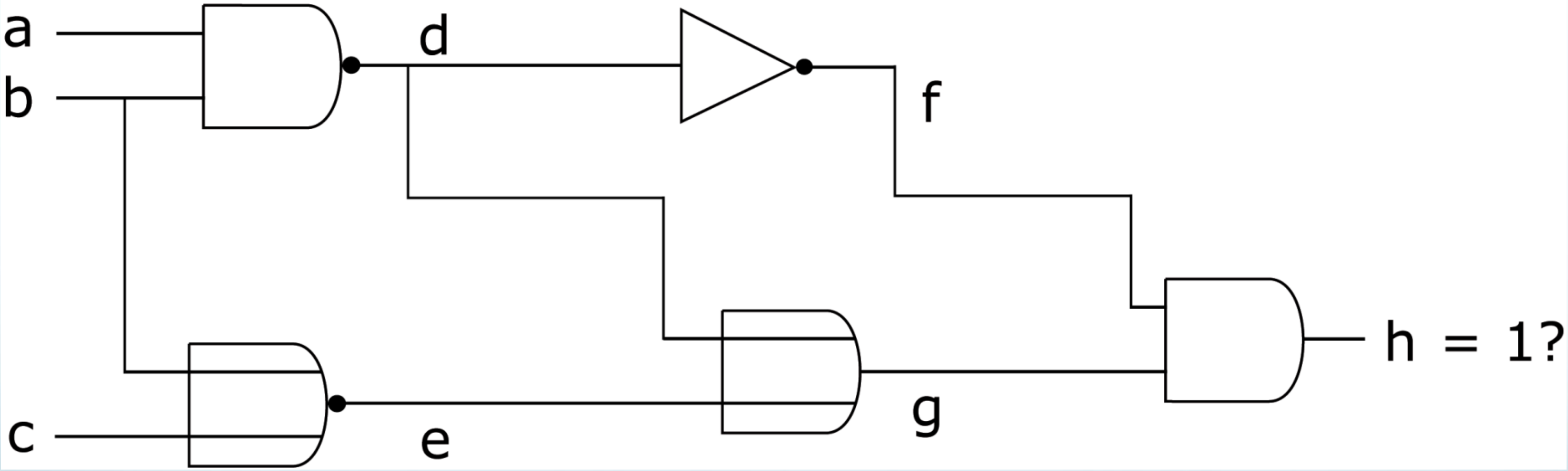
- Si *f* est donnée sous forme CNF, le problème est **NP-complet (en général)**
  - f(<a,b,c>) = a.(b+c).(a+b+c) $\rightarrow$ trouver une combinaison de a,b,c pour laquelle chaque terme est vrai

La forme **CNF** est utilisée car **exponentiellement plus petite** que la DNF

Exemple de problème SAT : représentation d'un circuit combinatoire (1)

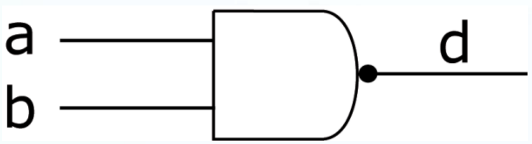
Construction de l'instance SAT :

- conjonction de termes associés à chaque porte
- variables intermédiaires permettant une démarche modulaire (à utiliser avec parcimonie !)



$\varphi = h\ [d = \neg(ab)]\ [e = \neg(b + c)]\ [f = \neg d]\ [g = d + e]\ [h = fg]$

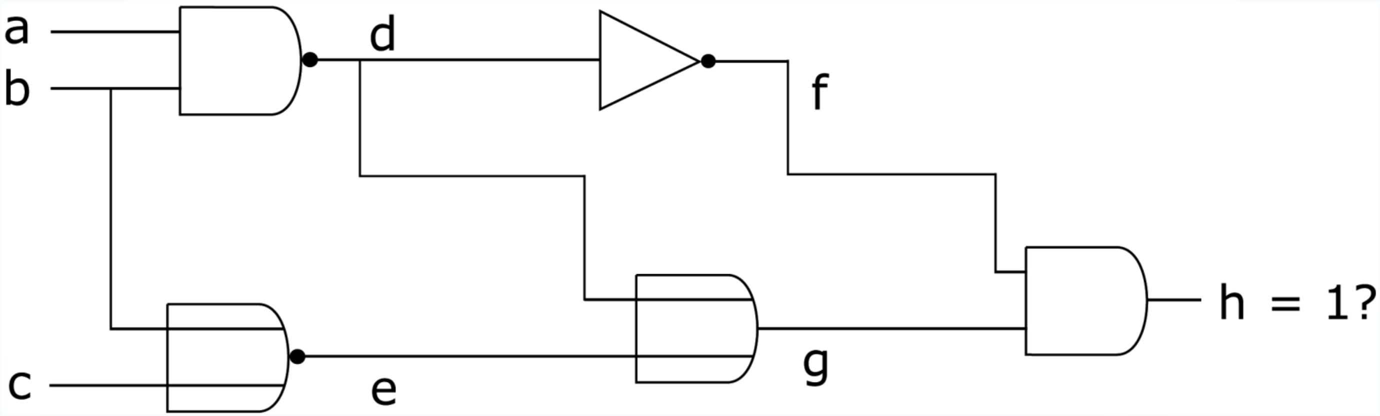
Exemple de problème SAT : représentation d'un circuit combinatoire (2)



$$
\begin{aligned}
\varphi_d &= [d = \neg(a\ b)] \\
&= \neg[d \oplus \neg(a\ b)] \\
&= \neg[\neg(a\ b)\neg d + a\ b\ d] \\
&= \neg[\neg a\ \neg d + \neg b\ \neg d + a\ b\ d] \\
&= (a + d)(b + d)(\neg a + \neg b + \neg d)
\end{aligned}
$$

Représentation CNF

Exemple de problème SAT : représentation d'un circuit combinatoire (3)



$\varphi = h\,[d = (ab)]\,[e = \neg(b + c)]\,[f = \neg d]\,[g = d + e]\,[h = fg]$

$= h$

$(a + d)(b + d)(\neg a + \neg b + \neg d)$

$(\neg b + \neg e)(\neg c + \neg e)(b + c + e)$

$(\neg d + \neg f)(d + f)$

$(\neg d + g)(\neg e + g)(d + e + \neg g)$

$(f + \neg h)(g + \neg h)(\neg f + \neg g + h)$

Conjonction de toutes les clauses

Existe-t-il une configuration de a,b,…h satisfaisant tous les termes ?

Exemple d’utilisation : equivalence de circuits

- Soient 2 circuits combinatoires C1 et C2, de même ensembles d’entrées et sorties primaires.
- Construire l’instance SAT décidant de l’équivalence de C1 et C2.
- **Application :**
  - Circuit C1 : s1 = c and (b xor a)
  - Circuit C2 : s2 = c xor (a and b)

Résolution d'un problème SAT

Algorithmes de résolutions

- Algorithmes stochastiques :
  - Peuvent converger très vite !
  - Semi-décision

- Algorithmes exactes :
  - Peuvent prendre des années (voire des siècles) de calcul !
  - Décision

Algorithme de résolution d'un problème SAT (1)

- Procédure Davis-Putnam-Logemann-Loveland (DPLL)[Davis'62]
  - Affecte successivement des valeurs aux variables
    - *Configuration courante : certaines variables sont affectées à une valeur mais pas forcément toutes*
  - État de chaque clause pour la configuration courante
    - Non décidé / Satisfaite (=1) / Non Satisfiable (=0)
    - Non décidé : l'affectation courante n'est pas assez complète

      → il faut affecter plus de variables
    - Non satisfiable : configuration courante non acceptable

      → il faut modifier l'état d'affectation d'une ou plusieurs variables (et recommencer → Backtrack)

Algorithme de résolution d'un problème SAT (2)

F = la formule à résoudre
α = l'assignation courante (initialement vide)

procedure DPLL(F,α)

F=UnitPropagation (F,α); //Assignation des clauses unitaires

if empty clause in F **then return** UNSAT;

if $|\alpha|$ = nb. variables **then return** SAT;

p = choose a decision literal occurring in F;

return DPLL(F U {(p)}, α U {p}) or DPLL(F U {($\neg p$)}, α U {$\neg p$});

end procedure

Exemple de résolution

(h)
$(a + d)(b + d)(\neg a + \neg b + \neg d)$
$(\neg b + \neg e)(\neg c + \neg e)(b + c + e)$
$(\neg d + \neg f)(d + f)$
$(\neg d + g)(\neg e + g)(d + e + \neg g)$
$(f + \neg h)(g + \neg h)(\neg f + \neg g + h)$

1. Propagation unitaire :

(h) → h=1

$(f + 0)(g + 0)(\neg f + \neg g + 1)$ → f=1 et g=1

$(\neg d + 1)(\neg e + 1)(d + e + 0)\ (\neg d + 0)(d + 1)$ → d=0

$(0 + e + 0)$ → e=1

$(a + 0)(b + 0)(\neg a + \neg b + 1)\ (\neg b + 0)(\neg c + 0)(b + c + 1)$ → b=1 et b=0

() → Empty clause

→ UNSAT

Une autre présentation… *Backtrack search*

1 $(a + b + c)$

2 $(a + b + \neg c)$

3 $(\neg a + b + \neg c)$

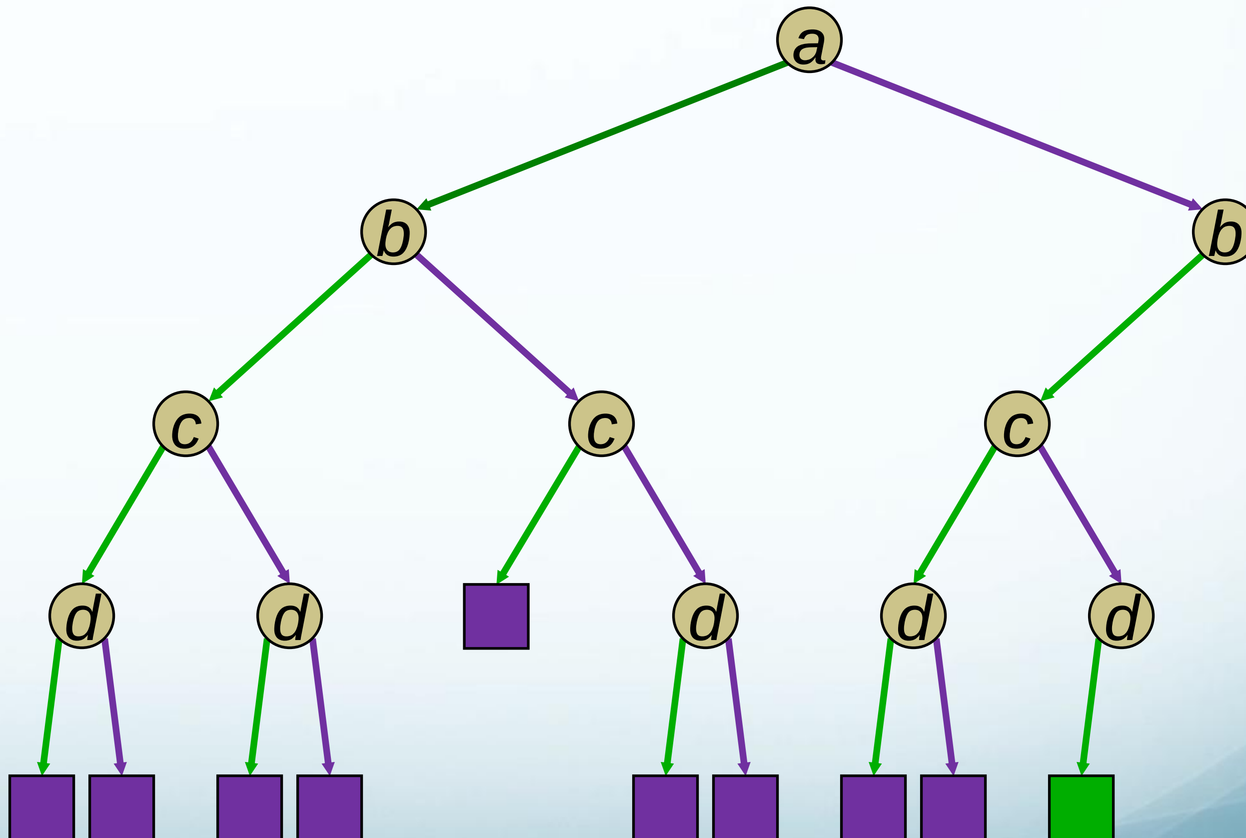
4 $(a + c + d)$

5 $(\neg a + c + d)$

6 $(\neg a + c + \neg d)$

7 $(\neg b + \neg c + \neg d)$

8 $(\neg b + \neg c + d)$



Facteurs d’efficacité

- **Décisions : heuristiques**
  - **Sous ensemble de clauses à analyser**
  - **Prochaine variable à affecter**

- **Analyse des conflits**
  - **Injection de nouvelles clauses dans l’instance (clauses learning)**

- **Analyse des backtracks**
  - **Pas nécessairement chronologiques**

- **…**

Facteurs d'efficacité : heuristiques

- Heuristique de BOHM :
  - choisir la variable *x* dont le vecteur ($H_1(x),\ldots,H_n(x)$) est maximal.
  - Ou, $H_i(x) = \max(h_i(x), h_i(!x)) + 2*\min(h_i(x), h_i(!x))$.
  - Et, $h_i(x)$ est le nombre de clauses non décidées contenant x et dont la taille est i.

- Heuristique MOM (Maximum Occurrences of clauses of Minimum size) :
  - choisir la variable qui maximise la fonction : $[f(x) + f(!x)] * 2^k + f(x) * f(!x)$
  - Ou, *f(x) est* le nombre d'occurrences de *x* dans les plus petites clauses non décidées.

- Heuristique JW (Jeroslow-Wang) :
  - choisir la variable *x* qui maximise la fonction J(x), tel que :

$$J(x) = \sum_{x \in cl} 2^{-|cl|}$$

…

- Heuristique DLCS (Dynamic Largest Combined Sum) :
  - choisir la variable x qui maximise : C(x) + C(!x)
  - Ou, C(x) est le nombre de clauses dans lesquels x apparaît.

- **Heuristique VSIDS (Variable State Independent Decaying Sum)**
  - **Basée sur les *Clauses de Conflits*….**

Conflict Directed Clause Learning (CDCL)

```
procedure CDCL( F )
  Γ = F , α = Ø, level = 0, conflict=false;
  repeat :
      conflict = UnitPropagation(Γ , α) ;
      if (conflict)
          if level = 0 return UNSAT ;
          C = Get Conflict Clause ;
          p = the sole literal of C set at the conflict level ;
          level = max{ level(x) : x is an element of C \ p} ;
          α = α less all assignments made at levels greater than level ;
          (Γ, α) = (Γ ∪ {C}, α.p);
      else
          if (α is total) return SAT ;
          p = choose a decision literal occurring in Γ\α;
          α = α.p;
          increment level;
      end if
end repeat
```

Conflict Directed Clause Learning (CDCL) : un exemple

$\omega_1 = (x_1 \vee x_2)$
$\omega_2 = (x_1 \vee x_3 \vee x_7)$
$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$
$\omega_4 = (\neg x_4 \vee x_5 \vee x_8)$
$\omega_5 = (\neg x_4 \vee x_6 \vee x_9)$
$\omega_6 = (\neg x_5 \vee \neg x_6)$

$x_7 = 0@1$ PU = Ø

$x_8 = 0@2$ PU = Ø

$x_9 = 0@3$ PU = Ø

$x_1 = 0@4$ PU = $\{x_2=1, x_3=1, x_4=1, \mathbf{x_5=1}, x_6=1, \mathbf{x_5=0}\}$

- La propagation unitaire ne rajoute rien aux niveaux de décisions 1, 2 and 3.
- La propagation du niveau *4* mène à un conflit !

Conflict Directed Clause Learning (CDCL) : analyse du conflit

Assignement partiel courant: $\{ x_7=0@1 ,x_8=0@2, x_9=0@3\}$

Assignement du niveau courant : $\{x_1=0@4\}$

$\omega_1 = (x_1 \vee x_2)$

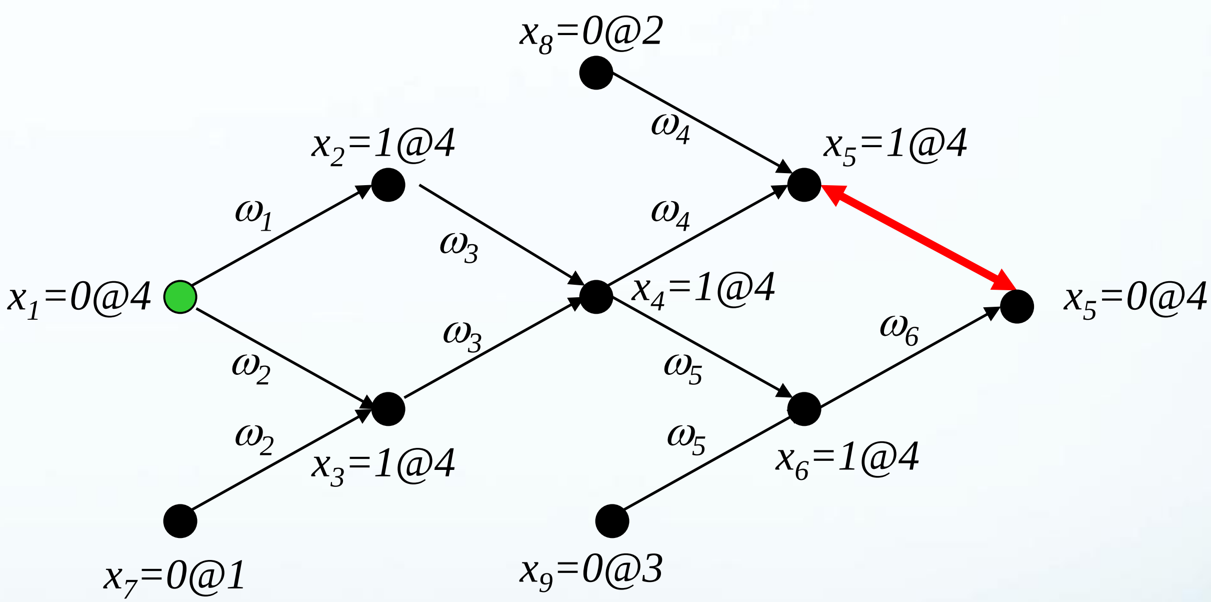
$\omega_2 = (x_1 \vee x_3 \vee x_7)$

$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$\omega_4 = (\neg x_4 \vee x_5 \vee x_8)$

$\omega_5 = (\neg x_4 \vee x_6 \vee x_9)$

$\omega_6 = (\neg x_5 \vee \neg x_6)$



Graphe d'implications

Conflict Directed Clause Learning (CDCL) : apprentissage (1)

$\omega_1 = (x_1 \vee x_2)$

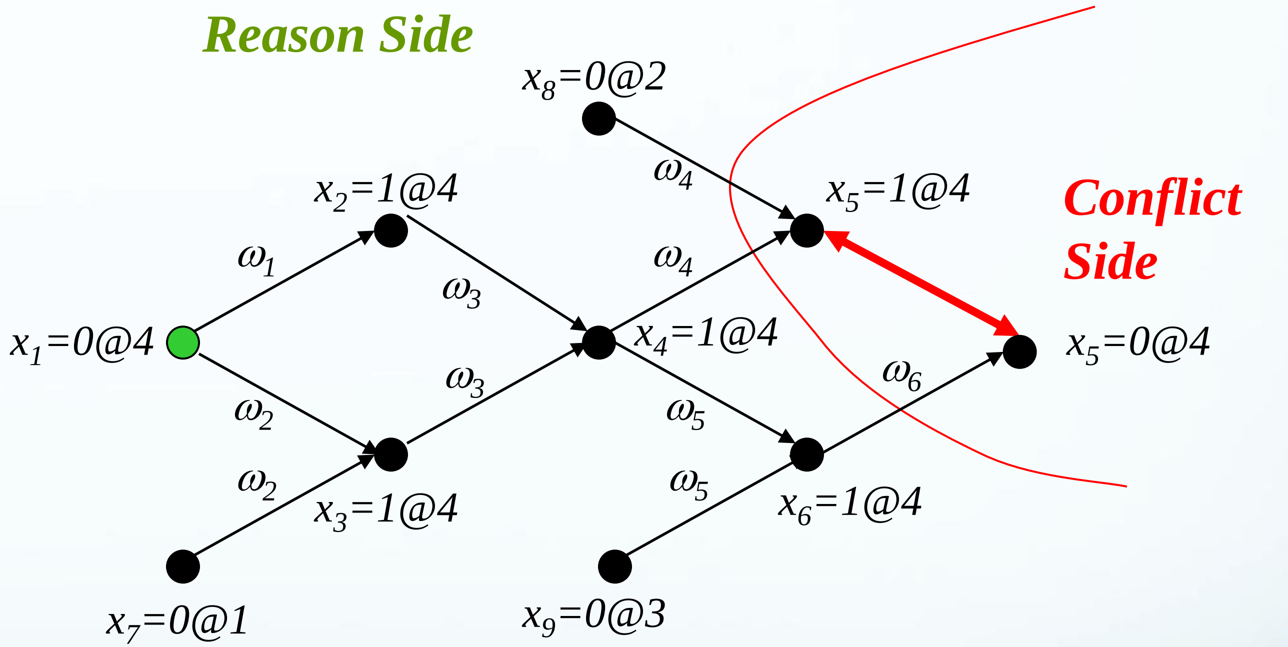
$\omega_2 = (x_1 \vee x_3 \vee x_7)$

$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$\omega_4 = (\neg x_4 \vee x_5 \vee x_8)$

$\omega_5 = (\neg x_4 \vee x_6 \vee x_9)$

$\omega_6 = (\neg x_5 \vee \neg x_6)$



Conflict Directed Clause Learning (CDCL) : apprentissage (2)

$\omega_1 = (x_1 \vee x_2)$

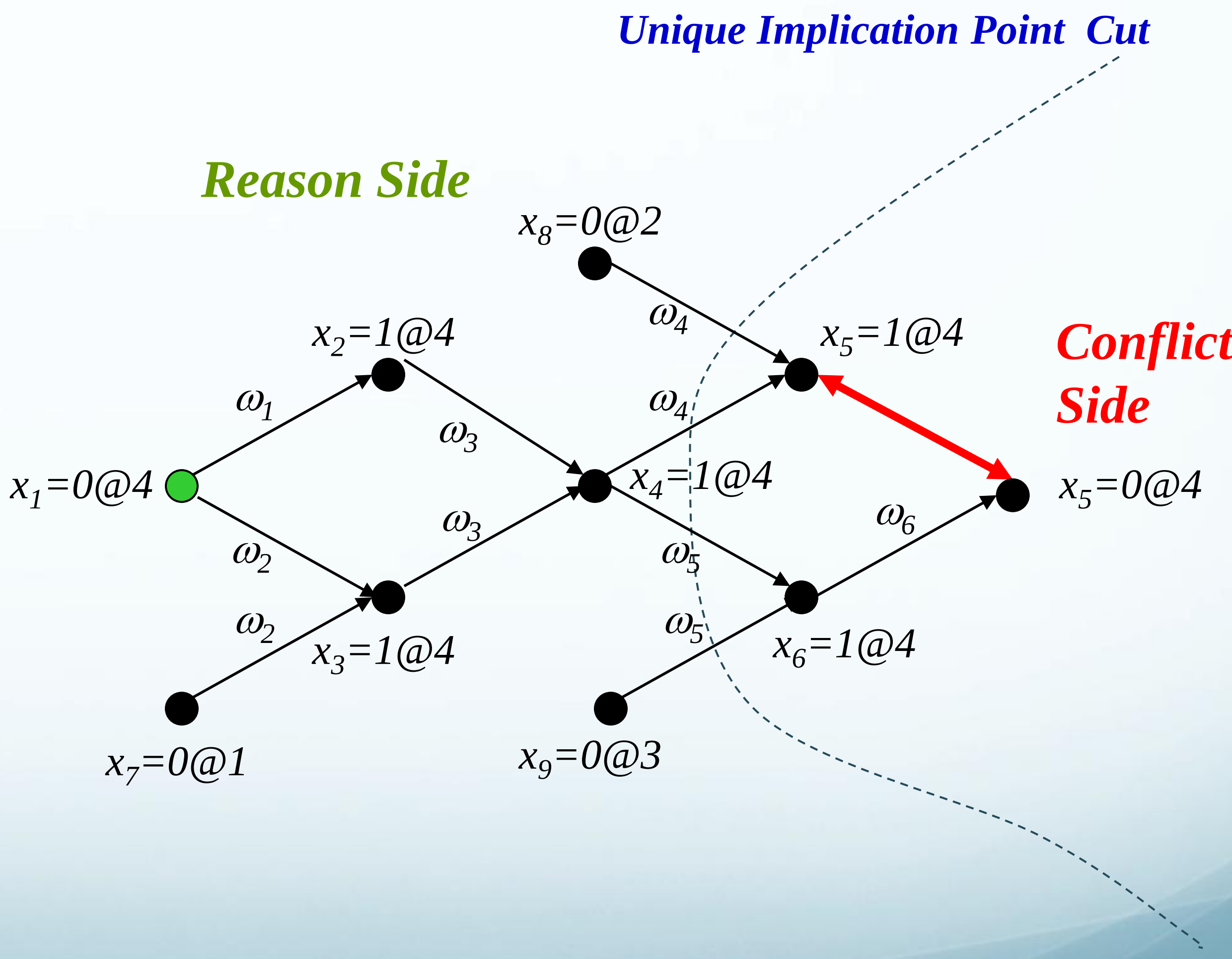
$\omega_2 = (x_1 \vee x_3 \vee x_7)$

$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$\omega_4 = (\neg x_4 \vee x_5 \vee x_8)$

$\omega_5 = (\neg x_4 \vee x_6 \vee x_9)$

$\omega_6 = (\neg x_5 \vee \neg x_6)$



Conflict Directed Clause Learning (CDCL) : apprentissage (3)

$\omega_1 = (x_1 \vee x_2)$

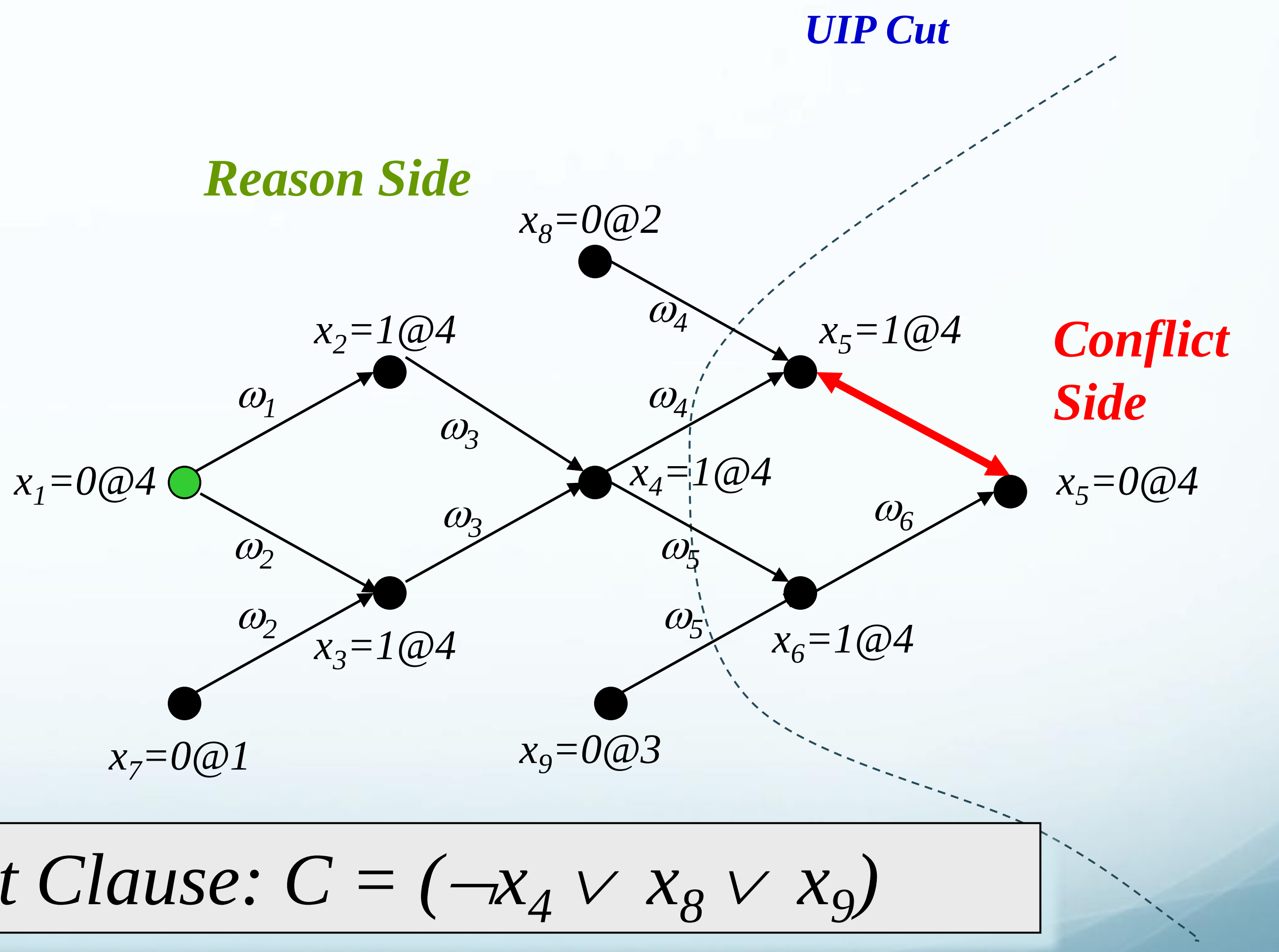
$\omega_2 = (x_1 \vee x_3 \vee x_7)$

$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$\omega_4 = (\neg x_4 \vee x_5 \vee x_8)$

$\omega_5 = (\neg x_4 \vee x_6 \vee x_9)$

$\omega_6 = (\neg x_5 \vee \neg x_6)$



Conflict Clause: $C = (\neg x_4 \vee x_8 \vee x_9)$

Conflict Directed Clause Learning (CDCL) : backtrack (1)

$\omega_1 = (x_1 \vee x_2)$

$\omega_2 = (x_1 \vee x_3 \vee x_7)$

$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$\omega_4 = (\neg x_4 \vee x_5 \vee x_8)$

$\omega_5 = (\neg x_4 \vee x_6 \vee x_9)$

$\omega_6 = (\neg x_5 \vee \neg x_6)$

$C = (\neg x_4 \vee x_8 \vee x_9)$

$x_7 = 0@1$ PU = Ø

$x_8 = 0@2$ PU = Ø

$x_9 = 0@3$ PU = Ø

$x_1 = 0@4$ PU = {x_2=1, x_3=1, x_4=1, **x_5=1**, x_6=1, **x_5=0**}

Backtrack au niveau= max{ level(x) : *est un element de C – p*}

p = x_4 ⇨ niveau= 3

Conflict Directed Clause Learning (CDCL) : backtrack (2)

$\omega_1 = (x_1 \vee x_2)$

$\omega_2 = (x_1 \vee x_3 \vee x_7)$

$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$\omega_4 = (\neg x_4 \vee x_5 \vee x_8)$

$\omega_5 = (\neg x_4 \vee x_6 \vee x_9)$

$\omega_6 = (\neg x_5 \vee \neg x_6)$

$C = (\neg x_4 \vee x_8 \vee x_9)$

$x_7 = 0@1$ UP = Ø

$x_8 = 0@2$ UP = Ø

$x_9 = 0@3$

Assignement partiel courant : $\{x_7=0, x_8=0, x_9=0, x_4=0\}$

Conflict Directed Clause Learning (CDCL) : Maintenance de la BD des clauses de conflits

- Chaque conflit peut ajouter une nouvelle clause
  ➔ BD de clauses de conflits

- La BD des clauses de conflits devient large…
  - Pose le problème de stockage.
  - Engendre un surcout important pour la propagation unitaire.

- Techniques de gestion de la BD.
  - ***Relevance Bounded Learning***: maintenir les clauses les importantes pour le sous-espace de recherche courant.
  - ***Size-Bounded Learning***: maintenir les clauses dont la taille est $\leq$ i variables.
  - Eliminer périodiquement des clauses en se basant sur les stratégies ci-dessus.

Conflict Directed Clause Learning (CDCL) : Redémarrage

- La recherche peut entrer dans des régions où les clauses de conflits produites ne sont plus intéressantes .

- Le redémarrage jette l'assignement courant et repart de 0.

- Retient les clauses apprises lors du dernier démarrage et ceci produit une différent parcourt.

- Avec le redémarrage, la procédure de résolution n'est plus complète...!

Parallélisation

- Les problèmes SAT sont consommateurs de temps :
  ➔ La parallélisation est un bon moyen d'améliorer les performances.

- **Porte Folio.**
  •Plusieurs SAT solveurs séquentiels lancés en parallèle, exécutant :
    - le même algorithme mais configurer différemment ou
    - des algorithmes différents.

- **Diviser pour mieux régner.**
  •Un SAT solveur multi-threadés parcourant l'arbre de recherche de manière parallèle.

Quelle technique est meilleur ?

Première conclusion

- SAT vs. BDD :
  - BDDs : toutes les solutions (ennemi = taille mémoire)
  - SAT : 1 seule solution (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…)
- Problèmes SAT annexes : max-SAT, all-SAT, **sharp-SAT,**….
- **Domaines encore très ouverts :**
  - Heuristiques
  - Stratégies de gestions de la BD des clauses
  - **Parallélisation**

Problèmes SAT
et Vérification

Bounded Model Checking (1)

- Prouver S |= φ (Model-Checking)
  - Construction de l’espace d’états
  - CTL : Parcours récursifs d’ensembles d’états selon la relation de transitions
  - LTL : Produit du graphe d’états avec automate reconnaisseur (Büchi), puis recherche de cycles / composantes fortement connexes dans le produit

- Recherche d’un contre-exemple : encodage en problème SAT
  - Supporte :
    - LTL [Biere’99] [Clark’05], ACTL [Penczek’02],…
    - **Mais, ici, on va s’intéresser aux propriétés de sureté !**

  - Recherche d’un contre-exemple de longueur *k* :
    - *Existe-t-il un chemin dans l’espace d’état (borné à l’horizon k) menant vers un état ne satisfaisant pas φ ?*
    - Exploration de l’arbre d’exécution (complet en largueur) mais sur une « tranche » de profondeur *k* seulement

Bounded Model Checking (2)

- Soit une formule φ (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 (appel au SAT-solveur)

Construction du problème SAT (1)

1. **Définir les configurations initiales**
   - Init : S →IB , $\mathbf{S_0}$ ∈ Init

2. **Dérouler la relation de transitions et la fonction de sortie sur k cycles**
   - Soit T ∈ (S × I × S) la relation de transition sur un cycle et G : S × I → O la fonction de génération.
   - On introduit *k+1* jeux de variables $S_0, \ldots S_k, I_0, \ldots I_k$, représentant les états et les configurations d'entrées aux instants 0,… k.
   - Alors $T_{0..k} : (S \times I)^{k+1} \rightarrow IB$, $\mathbf{T_{0..k} = \bigwedge_{j\in [0..k]} T(S_j,I_j,S_{j+1})}$
   - De même, $G_{0..k} : (S \times I \times O)^{k+1} \rightarrow IB$, $\mathbf{G_{0..k} = \bigwedge_{j\in [0..k+1]} (Oj \Leftrightarrow G(S_j,I_j))}$

3. **Construire *la négation* de la propriété φ**
   - φ représente une propriété d'état accessible par un chemin de longueur bornée
   - $\mathbf{Prop_{0..k}}$ **,** $\mathbf{NegProp_{0..k} : (S \times I \times O)^{k+1} \rightarrow IB; NegProp_{0..k} = \neg Prop_{0..k}}$

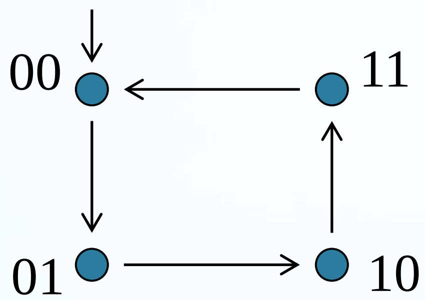
Construction du problème SAT (2)

4. **Existence d'un contre-exemple de longueur k :**

   $$CE_k = Init(s_0) \wedge T_{0..k} \wedge G_{0..k} \wedge NegProp_{0..k}$$

S'il existe au moins une solution à ce problème alors la propriété φ est violée par le système et la solution forme un exemple à cette violation.

Exemple : compteur à deux bits



Etat initial : $I : \neg l \wedge \neg r$

Transition : $T : \begin{pmatrix} l' = (l \neq r) \wedge \\ r' = \neg r \end{pmatrix}$

Propriété : $\varphi : G(\neg l \vee \neg r)$

$$CE_2 : (\neg l_0 \wedge \neg r_0) \wedge \begin{pmatrix} l_1 = (l_0 \neq r_0) \wedge r_1 = \neg r_0 \wedge \\ l_2 = (l_1 \neq r_1) \wedge r_2 = \neg r_1 \end{pmatrix} \wedge \begin{pmatrix} (l_0 \wedge r_0) \vee \\ (l_1 \wedge r_1) \vee \\ (l_2 \wedge r_2) \end{pmatrix}$$

CE_2 : est unsatisfiable.

CE_3 : est satisfiable.

Du BMC au « Unbounded » MC

Existence d'un contre-exemple de longueur k :

$$CE_k = Init(s_0) \bigwedge T_{0..k} \bigwedge G_{0..k} \bigwedge NegProp_{0..k}$$

- **S'il existe une solution S à CE_k : S représente un contre-exemple de φ.**
- **Comment sait-on qu'il n'y a pas de solution à $CE_{k,}$ pour tout k=0,1… ? Quand arrêter la procédure ?**
- **Quand on atteint le « diamètre de récurrence (DR) » du graphe : le plus long chemin sans cycle entre deux états du graphe de transitions.**

NoLoop : $S^{k+1} \rightarrow \mathbb{B}$, $\mathbf{Noloop_{0..k} = T_{0..k} \wedge \bigwedge_{j,i \in [0..k]} S_j \neq S_i}$

- **Solution très coûteuse !**

➔ **Induction.**

➔ **Sur-approximation du calcul des successeurs (Interpolation)**

Induction : définition

- Une propriété **Prop** est un Invariant Inductif (II) du système :

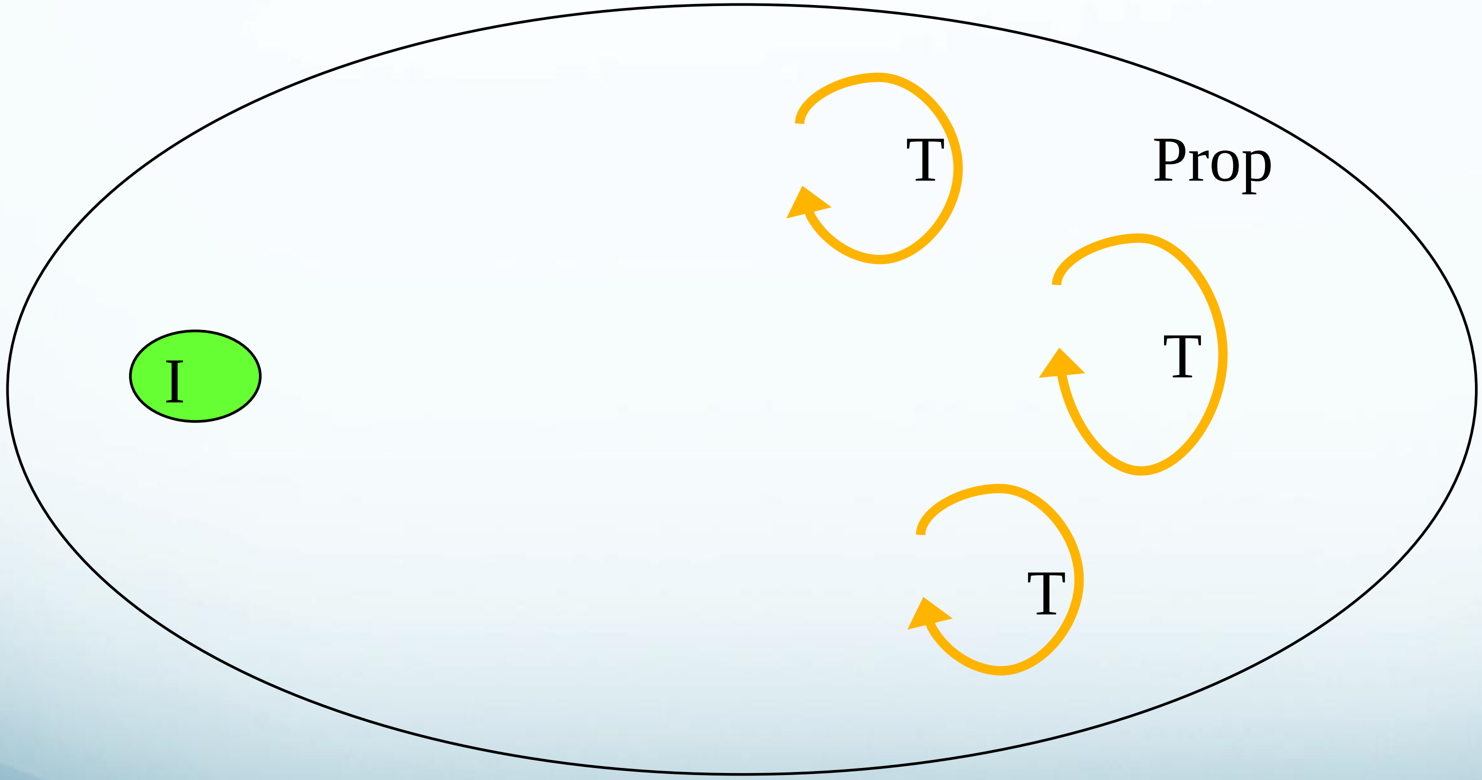
    - **Cas de base (*initiation*) :**

      Init => Prop

    - **Cas Inductif (*consecution*) :**

      Prop $\wedge$ T => Prop'

Induction: schématisation



Induction : exemple

```
x=1;
while (1) {
    x = x+1;
}
```

- **P : $x \geq 1$**
- **Init : $x = 1$**
- **T : $x' = x+1$**

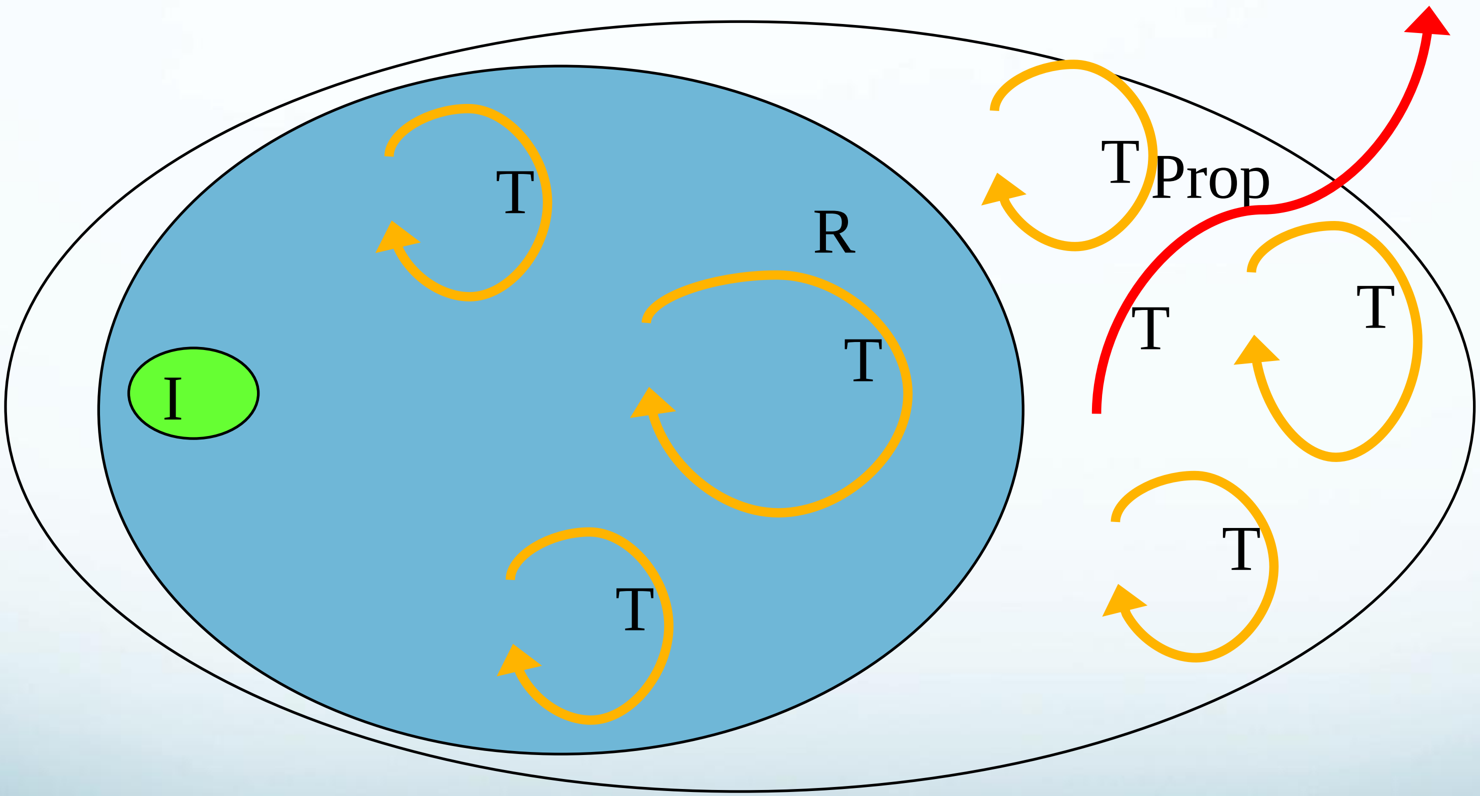
- Init => P ?
  - ✓ ($x=1 \Rightarrow x \geq 1$)
- P ∧ T => P' ?
  - ✓ $x \geq 1 \land x' = x+1 \Rightarrow x' \geq 2$
  - ✓ $x' \geq 2 \Rightarrow x' \geq 1$

Induction renforcée : définition

- Généralement, **Prop** n'est pas un Invariant Inductif !

- Mais, un II R **renforcé** peut exister **(strengthening)…**
  - **Initiatiton : I => R**
  - **Consecution : R $\wedge$ T => R' et R => Prop**

Induction renforcée : schématisation



Induction renforcée : exemple

```
x=1; y=1;
while (1) {
    x = x+y;
    y = y+x;
}
```

- Init => P ?
  - ✓ $(x = 1 \wedge y=1 \Rightarrow y \geq 1)$
- P $\wedge$ T => P' ?
  - ✓ $y \geq 1 \wedge x' = x+y \wedge y' = y+x \overset{?}{\Rightarrow} y' \geq 1$

On pose, **R : $x \geq 0 \wedge y \geq 1$ (=> P)**

- Init => R ?
  - ✓ $(x = 1 \wedge y=1 \Rightarrow x \geq 0 \wedge y \geq 1)$
- R $\wedge$ T => R' ?
  - ✓ $x \geq 0 \wedge y \geq 1 \wedge x' = x+y \wedge y' = y+x$
    $\Rightarrow x \geq 0 \wedge y' \geq 1$

- **P : $y \geq 1$**
- **Init : $x = 1 \wedge y = 1$**
- **T : $x' = x+y \wedge y' = y+x$**

K-induction : définition

M. Sheeran et al. (FMCAD'00)

- Si la propriété Prop est vrai pour tout chemin de taille k-1,partant d'un état initial.
- et que tout chemin simple (accessible ou non), de taille k-1, et qui vérifie Prop, **ne peut être prolongé que par un état vérifiant Prop,**
- Alors Prop est vraie pour tout état accessible.

$$\text{Initiation : } \mathbf{Init}_0 \wedge \mathbf{T}_{0..k} \wedge \mathbf{G}_{0..k} \Rightarrow \mathbf{Prop}_{0..k}$$

$$\text{Consecution : } \mathbf{NoLoop}_{0..k+1} \wedge \mathbf{Prop}_{0..k} \Rightarrow \mathbf{Prop}_{k+1}$$

Unbounded MC par k-induction

```
UMC_k_induction(M=<Init,T,G,I>,Prop) :

    j ← 0 // peut aussi être un nombre positif arbitraire

    Tant que (vrai)
      si SAT(¬(Init_0 ∧ T_0..j ∧ G_0..j ⇒ Prop_0..j) )
           retourner FAILS  // Contre-exemple trouvé

      si TAUT (∧_{1 ≤k≤ j+1} ¬Init_k ∧ NoLoop_0..j+1 ⇒ ¬Init_0   )
           retourner HOLDS  // diamètre de récurrence atteint

      si TAUT  (NoLoop_0..j+1 ∧ Prop_0..j ⇒ Prop_j+1 )
           retourner HOLDS  // cas inductif

      j ← j+1
```

Pour le cas où Prop est vraie, la convergence peut s'avérer très rapide (k << DR)

On a vu

Les SAT solveurs

- **sont capables de résoudre des problèmes de taille importantes**
- **sont très efficaces pour ignorer les faits non importants**
- **peuvent produire des réfutations**
- **peuvent être exploiter dans plusieurs directions:**
  - **BMC**
  - **…**

On a pas vu

- UMC, Interpolants de Craig
- IC3 :
  - Incremental Construction of Inductive Clauses for Indubitable Correctness
  - Le « renforcement » est fait de façon incrémentale
- SMT :
  - SAT Modulo Theory
  - Boolean SAT + Decision procédures pour d’autres domaines (array, integer).

Référence Absolue