Traduction logique et résolution de problèmes

Application à la planification

Maël Valais

8 avril 2019

IRIT – Université Toulouse III (France)

Contexte

Section

On s'intéresse aux **problèmes combinatoires**

Problèmes intéressants pour l'industrie et la recherche

Exemple : stockage de produits dangereux dans des salles séparées selon leur compatibilité

Les **solveurs SAT** très bons pour résoudre les problèmes combinatoires

Contenu

I - L'outil TouIST

II - Utilisation de TouIST avec différents solveurs

III - Application à la planification

I - L’outil TouIST

Motivations pour un nouvel outil

Constat 1 : langage des solveurs peu intuitifs

- ⊕ solveurs SAT très performants
- ⊖ mais langage d'entrée « bas niveau »

Formule	Entrée solveur (DIMACS)

$a \to b \lor \neg(a \to c) \lor (c \to \neg a)$	`p cnf 5 4`
	`-1 5 3 0`
	`-1 -5 4 1 0`
	`-1 -5 4 -2 0`
	`-1 -3 -2 -1 0`

Motivations pour un nouvel outil

Constat 2 : langage existant peu compact

- ⊕ SMT-LIB plus expressif que DIMACS
- ⊖ mais manque de compacité

Formule	Exemple SMT-LIB
$\bigwedge_{i \in [1..1000]} x_i$	`(assert (and x1 (and x2 (and x3 (and x4 (and x5 (and x6 ... x1000 )))))))`

```
(assert
 (and x1
  (and x2
   (and x3
    (and x4
     (and x5
      (and x6
          ... x1000
)))))))
```

Motivations pour un nouvel outil

But technique : développer un outil doté :

- d'un langage intuitif pour la modélisation, bien documenté
- d'une interface graphique conviviale

But académique : apporter une aide dans le domaine de

- l'enseignement de la logique
- la recherche (comparaison de solveurs, codages)

Motivations pour un nouvel outil

Historique :

Date	Version	Fonctionnalités
2010	SAToulouse	SAT intégré
mi-2015	TouIST 1.0	SAT
fin 2016	TouIST 2.0	SAT, SMT
mi-2017	TouIST 3.0	SAT, SMT, QBF

Principe d'utilisation sur un exemple

Exemple du politicien

Si la croissance augmente, alors le chômage diminue ;

Or, la croissance diminue.

Donc le chômage augmente !

Problème

La conclusion est-elle justifiée ?

Étape 1 : modélisation en logique propositionnelle

Si croissance alors pas de chômage ! $\dashrightarrow$ $\texttt{croissance} \rightarrow \neg\,\texttt{chomage}$

Et pas de croissance, $\dashrightarrow$ $\neg\,\texttt{croissance}$

Donc chômage. $\dashrightarrow$ $\texttt{chomage}$

Étape 2 : poser le problème

Théorème

$$\{H_1, \ldots, H_n\} \models C \text{ ssi } \{H_1, \ldots, H_n\} \cup \{\neg C\} \text{ insatisfiable}$$

Hypothèses	H_1:	croissance $\rightarrow$ chomage
	H_2:	$\neg$ croissance
Conclusion	C:	chomage

négation de la conclusion

Ensemble propositionnel correspondant :

$$\Big\{ \underbrace{\text{croissance} \rightarrow \neg\,\text{chomage}}_{H_1}, \quad \underbrace{\neg\,\text{croissance}}_{H_2}, \quad \underbrace{\neg\,\text{chomage}}_{\neg C} \Big\}$$

Étape 3 : résoudre (principe)

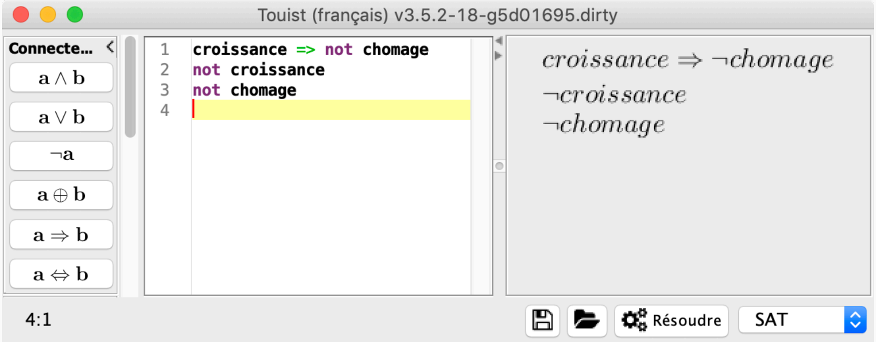
Définition d'un problème SAT

Existe t-il une *valuation* de l'ensemble de formules qui soit *modèle* de celui-ci ?

Dans notre cas : le raisonnement du politicien est **valide**, ssi TouIST ne trouve **aucun modèle** (i.e., *insatisfiable*).

Principe d'utilisation sur un exemple

Étape 3 : résoudre (programme TouIST)



Principe d’utilisation sur un exemple

Conclusion : un modèle trouvé, donc le raisonnement **n’est pas valide !**

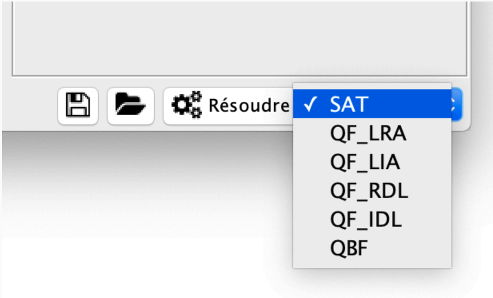
TouIST (Toulouse Integrated Satisfiability Tool)

Interface graphique d'écriture de formules logiques dans un **langage compact** et faisant appel à différents **solveurs externes**.

- sous licence MIT, sources : `github.com/touist/touist`
- interface graphique (**Java**)
- traduction vers solveurs gérée par un compilateur (**OCaml**)
- compatible **SAT, SMT et QBF**

Choix du solveur à la volée

TouIST permet de choisir le solveur à cibler. Le langage d’entrée s’adapte à chaque solveur.



Solveurs compatibles :

Solveur	Formule	Équivalent TouIST
SAT	$a \wedge b \rightarrow c$	`a and b => c`
SMT (QF-LIA)	$(x + y - z \geq 4) \vee b$	`(x + y - z >= 4) or a`
SMT (QF-LRA)	$(x + y - z > 4.0) \rightarrow c$	`(x + y - z > 4.0) => c`
SMT (QF-IDL)	$(x - y > 2) \wedge c$	`(x - y > 2) and c`
SMT (QF-RDL)	$(x - y > 1.0) \wedge c$	`(x - y > 1.0) and c`
QBF	$\forall b.a \wedge b$	`forall b: a and b`

QF-LIA = Quantifier-Free Linear Integer Arithmetic
QF-LRA = Quantifier-Free Linear Real Arithmetic
QF-IDL = Quantifier-Free Integer Difference Logic
QF-RDL = Quantifier-Free Real Difference Logic

Aspects d'implémentation

Langage d'entrée de TouIST :

- variables et ensembles

```
$N = 5
$Lignes = [1..$N]
$Pays = [France, Italie, Espagne]
```

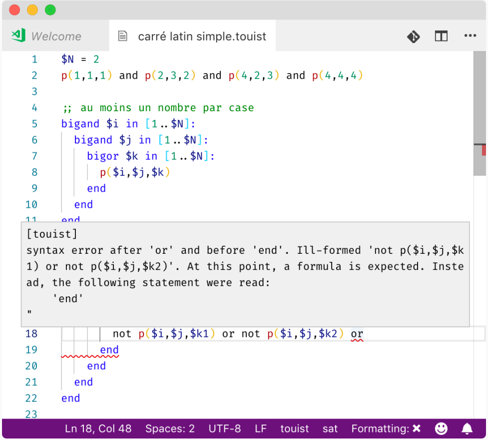
- connecteurs généralisés

```
en(France) and en(Italie) and en(Espagne)

bigand $p in $Pays:
  en($p)
end
```

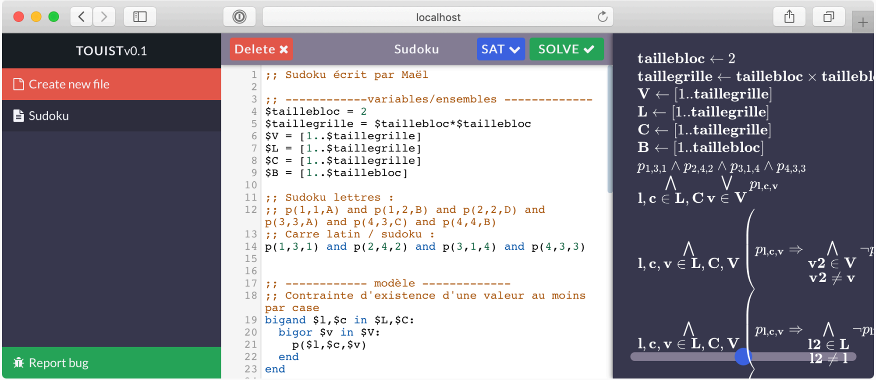
Aspects d’implémentation

- Extension Visual Studio Code publiée : **301 utilisateurs !**



- Et même un plugin Vim !

- Une application web
- plus besoin d'installer Java !



II - Utilisation de TouIST avec différents solveurs

Exemple 1 : le Sudoku (SAT)

	2					4	8	
1			8		2	3		7
				9	3	2	1	
	4	8						1
			6	1	8			
7						9	3	
	1	5	9	4				
4		2	3		6			9
	3	7					5	

Exemple 1 : le Sudoku (SAT)

Solution 1 : écrire un **algorithme *spécifique***

- ⊖ traduction non triviale, développement long
- ⊖ maintenance difficile
- ⊕ algorithme optimal mais spécifique

Solution 2 : utiliser une méthode **déclarative** (SAT)

- ⊕ traduction immédiate des règles
- ⊕ meilleure garantie de correction
- ⊕ solveurs SAT performants...
- ⊖ ...mais moins qu'un algorithme spécifique

Exemple 1 : le Sudoku (SAT). Étape 1 : modélisation

Étape 1 : modélisation

colonne

ligne

valeur (1 à 9)

$$p(\color{blue}{i}, \color{red}{j}, v)$$

	1	2	3	4	
1				9	
2					

$$p(\color{blue}{1}, \color{red}{4}, v)$$

$$p(1,4,\mathbf{1}) \quad p(1,4,\mathbf{2}) \quad p(1,4,\mathbf{3}) \quad \ldots \quad p(1,4,\mathbf{9})$$

Et maintenant...

Comment être certain qu'il y a exactement une valeur par case ?

Étape 2 : règles (exemple de la case (1,4))

Règle 1

La case (1,4) contient au moins une valeur.

$$\bigvee_{k \in \text{Valeurs}} \texttt{p(1,4,k)}$$

S'écrit en TouIST :

```
bigor $k in $Valeurs:
  p(1,4,$k)
end
```

Exemple 1 : le Sudoku (SAT). Étape 2 : règles

Règle 2

Si la case (1,4) a pour valeur 3, alors elle n'a aucune autre valeur différente.

$$\mathtt{p(1,4,3)} \rightarrow \bigwedge_{\substack{k \in [1..9] \\ k \neq 3}} \neg \mathtt{p(1,4,k)}$$

S'écrit en TouIST :

```
p(1,4,3) => bigand $k in [1..9] when $k != 3:
              not p(1,4,$k)
            end
```

Touist (français) v3.5.2-18-g5d01695.dirty - /Users/mvalais/code/touist/test/sat/sudoku.touist

```
;; Sudoku écrit par Maël

;; ------------variables/ensembles -------------
$taillegrille = 9
$taillebloc = int(sqrt(float($taillegrille)))
$V = [1..$taillegrille]
$L = [1..$taillegrille]
$C = [1..$taillegrille]
$B = [1..$taillebloc]

p(1,3,1) and p(2,4,2) and p(3,1,4) and p(4,3,3)


;; ------------ modèle -------------
;; Contrainte d'existence d'une valeur au moins par case
bigand $l,$c in $L,$C:
  bigor $v in $V:
    p($l,$c,$v)
  end
end

;; Contrainte d'une unique valeur par case
bigand $l,$c,$v in $L,$C,$V:
  p($l,$c,$v) =>
  bigand $v2 in $V when $v2 != $v:
    not p($l,$c,$v2)
  end
end

;; Contrainte d'une unique valeur par ligne
bigand $l,$c,$v in $L,$C,$V:
  p($l,$c,$v) =>
  bigand $l2 in $L when $l2 != $l:
    not p($l2,$c,$v)
  end
end

;; Contrainte d'une unique valeur par colonne
bigand $l,$c,$v in $L,$C,$V:
```

$$\mathbf{taillegrille} \leftarrow 9$$

$$\mathbf{taillebloc} \leftarrow \mathrm{int}(\sqrt{\mathrm{float}(\mathbf{taillegrille})})$$

$$\mathbf{V} \leftarrow [1..\mathbf{taillegrille}]$$

$$\mathbf{L} \leftarrow [1..\mathbf{taillegrille}]$$

$$\mathbf{C} \leftarrow [1..\mathbf{taillegrille}]$$

$$\mathbf{B} \leftarrow [1..\mathbf{taillebloc}]$$

$$p_{1,3,1} \wedge p_{2,4,2} \wedge p_{3,1,4} \wedge p_{4,3,3}$$

$$\bigwedge_{\mathbf{l,c} \in \mathbf{L,C}} \bigvee_{\mathbf{v} \in \mathbf{V}} p_{\mathbf{l,c,v}}$$

$$\bigwedge_{\mathbf{l,c,v} \in \mathbf{L,C,V}} \left(p_{\mathbf{l,c,v}} \Rightarrow \bigwedge_{\substack{\mathbf{v2} \in \mathbf{V} \\ \mathbf{v2} \neq \mathbf{v}}} \neg p_{\mathbf{l,c,v2}} \right)$$

$$\bigwedge_{\mathbf{l,c,v} \in \mathbf{L,C,V}} \left(p_{\mathbf{l,c,v}} \Rightarrow \bigwedge_{\substack{\mathbf{l2} \in \mathbf{L} \\ \mathbf{l2} \neq \mathbf{l}}} \neg p_{\mathbf{l2,c,v}} \right)$$

$$\bigwedge_{\mathbf{l,c,v} \in \mathbf{L,C,V}} \left(p_{\mathbf{l,c,v}} \Rightarrow \bigwedge_{\substack{\mathbf{c2} \in \mathbf{C} \\ \mathbf{c2} \neq \mathbf{c}}} \neg p_{\mathbf{l,c2,v}} \right)$$

4:18 Résoudre SAT

Touist (français) v3.5.2-18-g5d01695.dirty - /Users/mvalais/c...

Résultats ☑ Vrai ☐ Faux ☑ Autres Retour à l'édition

Nom	Valeur
p(1,1,2)	Vrai
p(1,2,4)	Vrai
p(1,3,1)	Vrai
p(1,4,3)	Vrai
p(2,1,3)	Vrai
p(2,2,1)	Vrai
p(2,3,4)	Vrai
p(2,4,2)	Vrai
p(3,1,4)	Vrai
p(3,2,3)	Vrai
p(3,3,2)	Vrai
p(3,4,1)	Vrai
p(4,1,1)	Vrai
p(4,2,2)	Vrai
p(4,3,3)	Vrai
p(4,4,4)	Vrai

Précédent Suivant Exporter

Exemple 2 : le Takuzu (SMT)

Jeu du Takuzu

	1		0
		0	
	0		
1	1		0

0	1	1	0
1	0	0	1
0	0	1	1
1	1	0	0

Exemple 2 : le Takuzu (SMT)

Règle 1

La somme d'une ligne/colonne doit être égale à 2.

Possible en logique propositionnelle, mais des nécessite **contraintes de cardinalité** en $O(n^3)$. En SMT :

```
bigand $l in $Lignes:
  p($l,1) + p($l,2) + p($l,3) + p($l,4) == 2
end
```

Exemple 3 : un jeu de Nim (QBF)

Règles du jeu

- 2 joueurs, vert et rouge.
- Chacun son tour ($t \in T$), un joueur prend 1 à 2 allumettes ($n \in A$).
- Si un joueur ne peut plus jouer, il a perdu.

$\longrightarrow$ Et si on gagnait à **tous les coups** ?

Exemple 3 : un jeu de Nim (QBF)

Règle 1 : « gagner »

Le joueur vert gagne ssi il n'existe aucun tour tel que c'est à lui de jouer et il ne reste aucune allumette.

$$vert\_gagne \leftrightarrow \neg \bigvee_{\substack{t \in T \\ t > 0}} tour\_de\_vert(t) \wedge reste(t, 0)$$

```
vert_gagne <=>
not bigor $t in $T when $t>0:
 tour_de_vert($t) and reste($t,0)
end
```

Exemple 3 : un jeu de Nim (QBF)

Règle 2 : « jouer »

Selon son choix, le joueur laisse une ou deux allumettes en moins.

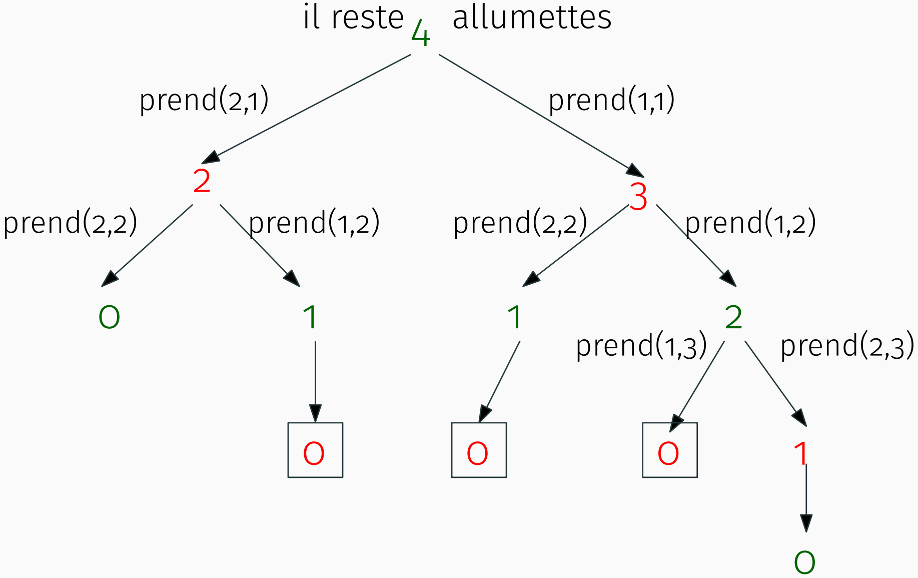
$$\bigwedge_{\substack{t\in T\\ n\in A\\ n\geq 2}} \begin{pmatrix} (reste(t,n) \land prend(t,2) \rightarrow reste(t+1,n-2)) \\ \land (reste(t,n) \land prend(t,1) \rightarrow reste(t+1,n-1)) \end{pmatrix}$$

```
bigand $t,$n in $T,$M when $n>=2:
  ((reste($t,$n) and prend($t,2)) => reste($t+1,$n-2))
 and
  ((reste($t,$n) and prend($t,1)) => reste($t+1,$n-1))
end
```

Exemple 3 : un jeu de Nim (QBF)

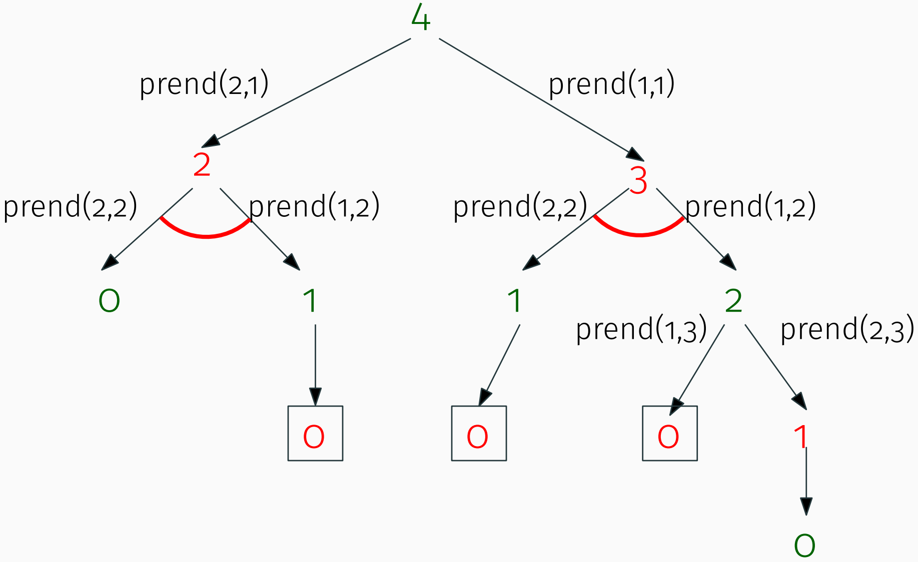
$$\text{QBF} = \underbrace{\boxed{\neg \quad \vee \quad \wedge \quad \rightarrow}}_{\text{Logique propositionnelle}} + \underbrace{\boxed{\forall \quad \exists}}_{\text{quantificateurs sur les propositions}}$$

Arbre de jeu décrivant toutes les parties possibles



Exemple 3 : un jeu de Nim (QBF)

Arbre ET/OU correspondant



Exemple 3 : un jeu de Nim (QBF)

Trouver la stratégie gagnante pour le joueur vert (s'il y en a une)

- SAT = taille formule exponentielle
- QBF = taille formule linéaire

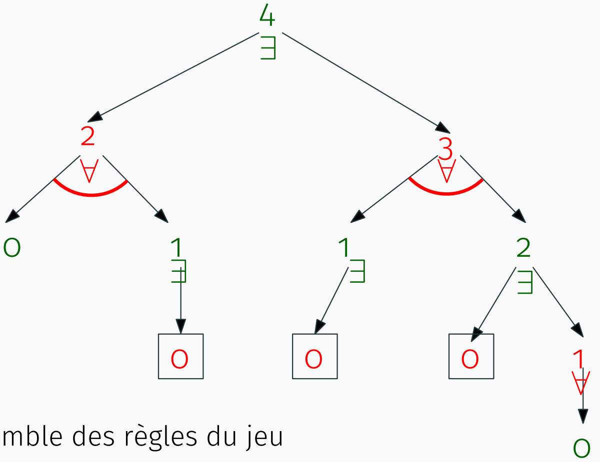
Exemple 3 : un jeu de Nim (QBF)

$\exists$ prend(2,1),prend(1,1).

$\forall$ prend(2,2),prend(1,2).

$\exists$ prend(2,3),prend(1,3).

vert_gagne $\wedge$ Φ



Φ = l'ensemble des règles du jeu

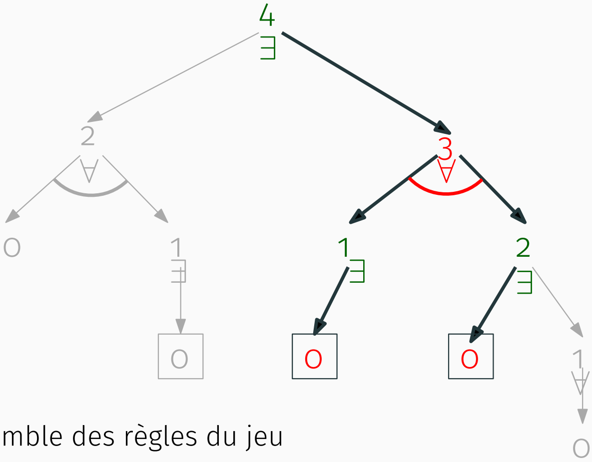
Exemple 3 : un jeu de Nim (QBF)

$\exists$ prend(2,1),prend(1,1).

$\forall$ prend(2,2),prend(1,2).

$\exists$ prend(2,3),prend(1,3).

vert_gagne $\wedge$ Φ

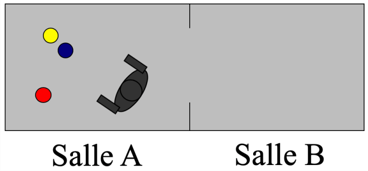


Φ = l'ensemble des règles du jeu

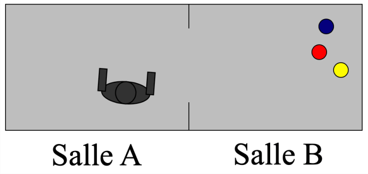
III - Application à la planification

Planification classique (SAT)

État initial



But

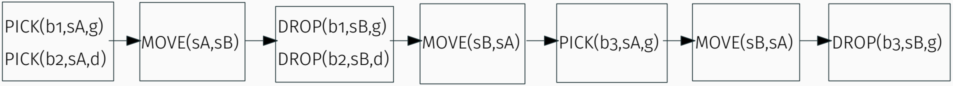


- Partir de l'**état initial** I
- Arriver à un état qui satisfait le **but** G
- À partir d'**actions** A (plusieurs actions possibles par étape)

PICK(balle1, salleA, gauche)	ramasser balle dans main gauche
PICK(balle1, salleA, droit)	ramasser balle dans main gauche
DROP(balle1, salleA, gauche)	lâcher main gauche
DROP(balle1, salleA, droit)	lâcher main droite
MOVE(salleA, salleB)	aller dans l'autre pièce

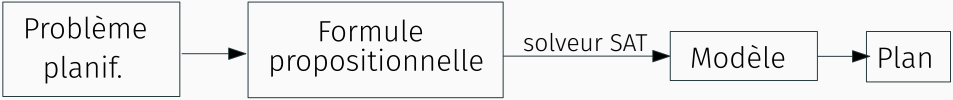
Définition

Plan = séquence d'**étapes** constituées d'**actions**.



Planification classique (SAT)

Étape 1 : traduire le problème de planification en problème SAT



exprime **tous** les plans possibles de longueur $\leq k$

Étape 1 : traduire le problème de planification en problème SAT

1. État initial $\bigwedge_{f\in I} f_0 \wedge \bigwedge_{f\in F\setminus I} \neg f_0$ $\boxed{X_0}$

2. États finaux $\bigwedge_{f\in G} f_k$ $\boxed{X_k}$

3. Transitions

— Pré- et Post-conditions

— Mutex

— Frame-axioms

EFA No-ops

$$\bigwedge_{i=1}^{k} \bigwedge_{f\in F} \left(\neg f_{i-1} \wedge f_i \rightarrow \bigwedge_{\substack{a\in A \\ f\in \mathrm{Add}_a}} a_i \right)^{\text{(EFA)}}$$

$\boxed{X_{i-1}}$

$\boxed{X_i}$

Planification classique (SAT)

Étape 1 : traduire le problème de planification en problème SAT

1. État initial $$\bigwedge_{f \in I} f_0 \wedge \bigwedge_{f \in F \setminus I} \neg f_0$$

2. États finaux $$\bigwedge_{f \in G} f_k$$

3. Transitions

— Pré- et Post-conditions

— Mutex

— Frame-axioms

EFA No-ops

$$\bigwedge_{i=1}^{k} \bigwedge_{f \in F} \left(\neg f_{i-1} \wedge f_i \rightarrow \bigvee_{\substack{a \in A \\ f \in \mathsf{Add}_a}} a_i \right)$$

Planification classique (SAT)

Étape 1 : traduire le problème de planification en problème SAT

1. État initial $\bigwedge_{f \in I} f_0 \wedge \bigwedge_{f \in F \setminus I} \neg f_0$ $\boxed{X_0}$

2. États finaux $\bigwedge_{f \in G} f_k$ $\boxed{X_k}$

3. Transitions

— Pré- et Post-conditions

— Mutex

— Frame-axioms

EFA No-ops

$$\bigwedge_{i=1}^{k} \bigwedge_{f \in F} \left(\neg f_{i-1} \wedge f_i \rightarrow \bigvee_{\substack{a \in A \\ f \in \mathrm{Add}_a}} a_i \right)$$

$\boxed{X_{i-1}} \rightarrow \boxed{X_i}$

Planification classique (SAT)

On représente l'ensemble des plans de longueur < 7 par :

$$\boxed{X_0} \rightarrow \boxed{X_1} \rightarrow \boxed{X_2} \rightarrow \boxed{X_3} \rightarrow \boxed{X_4} \rightarrow \boxed{X_5} \rightarrow \boxed{X_6}$$

Étape 2 : passage en QBF

Pourquoi exploiter QBF pour la planification ?

- les compétitions des solveurs QBF attirent **de plus en plus l'attention**
- Satplan gagne les compétitions IPC'04 et IPC'06 grâce aux amélioration des solveurs SAT **provoquées par les compétitions**

$\longrightarrow$ les solveurs QBF deviennent de plus en plus performants

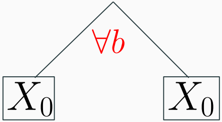
Étape 2 : passage en QBF

Logique propositionnelle | quantificateurs sur les propositions

QBF = $\neg \quad \vee \quad \wedge \quad \rightarrow$ + $\forall \quad \exists$

$\boxed{X_0}$ $\boxed{X_1}$

Logique propositionnelle
deux état =
deux ensembles de variables différents

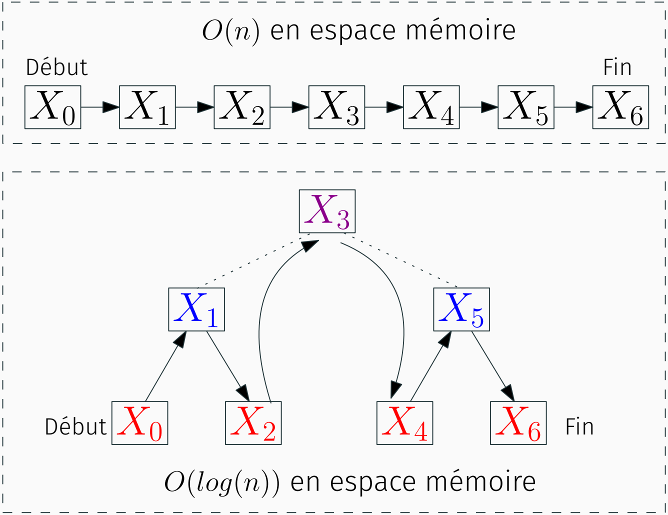


QBF
deux contextes différents en
utilisant le même ensemble de variables
$\longrightarrow$ moins de mémoire

Planification classique (QBF)

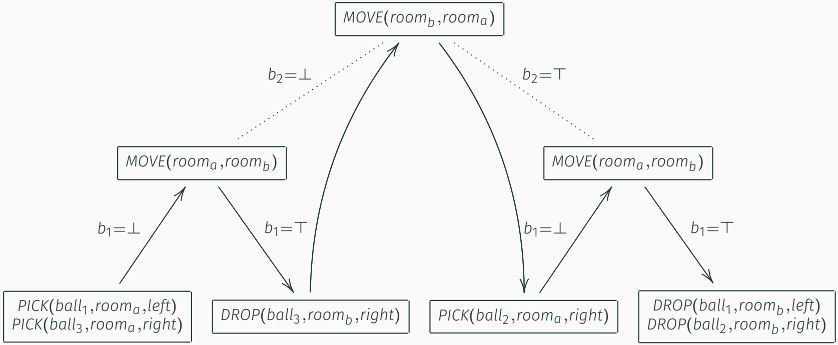
Planif. SAT

Planif. QBF (CTE)



$\longrightarrow$ Théoriquement, beaucoup **moins de mémoire** utilisée.

Étape 3 : exemple de plan retourné et lecture de l’arbre

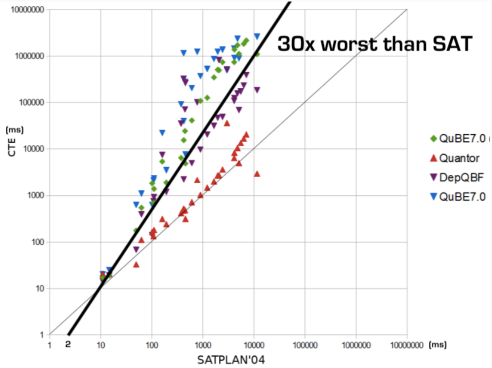


Transitions du plan en 7 étapes retourné par `TouISTPlan` (codage d’arbre compact QBF (CTE) de profondeur 2 dans les espaces d’états avec frame-axiomes explicatifs)

Contribution : nouveaux codages pour résolution rapide

Travail de référence de [Cashmore et al., 2012] :

- En SAT, No-ops est la meilleure transition disponible



- En QBF, No-ops utilise **5 × moins de mémoire** mais **30 × plus lent**

Contribution : nouveaux codages pour résolution rapide

Objectif

Peut-on trouver de meilleurs codages basés sur le CTE de Cashmore ?

Note

Travail au niveau de l'encodage, pas au niveau solveur

Contribution : nouveaux codages pour résolution rapide

Codages
SAT

NO-OPS
(SATPLAN)

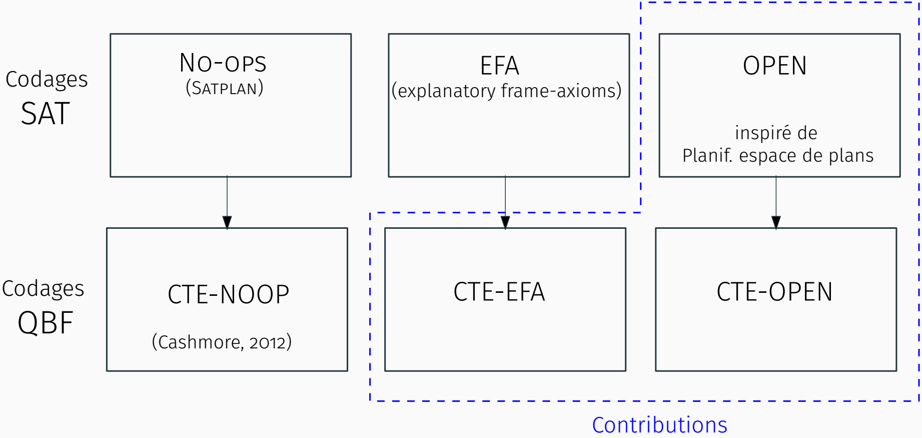
EFA
(explanatory frame-axioms)

OPEN

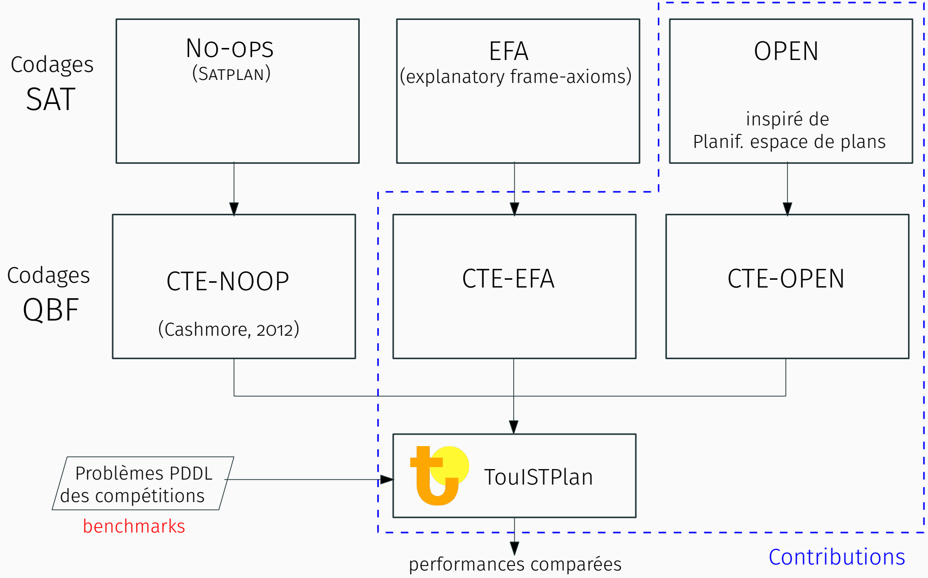
inspiré de
Planif. espace de plans

Contribution

Contribution : nouveaux codages pour résolution rapide



Contribution : nouveaux codages pour résolution rapide



Contribution : nouveaux codages pour résolution rapide

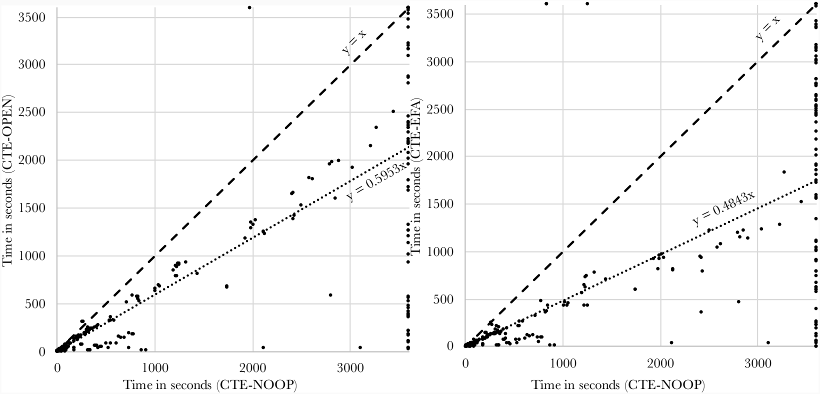
- Utilise les domaines 1 à 8 des IPC (International Planning Competition)
- Solveurs QBF testés : RaREQS, DepQBF, Qute (Quantor trop lent)
- 6000 heures de calcul (2112 problèmes à travers 3 solveurs)
- Timeout de 1 heure pour le problème *”y a-t-il un plan ?”*
- Problèmes QBF générés en utilisant **TouIST** et **TouISTPlan**
  - `https://github.com/touist/touistplan`

TouIST

A friendly language for propositional logic and more

Contribution : nouveaux codages pour résolution rapide

Temps entre codages CTE pour le problème *"y a-t-il un plan ?"* :



CTE-EFA est $2\times$ **plus rapide** CTE-OPEN est $1.7\times$ **plus rapide**

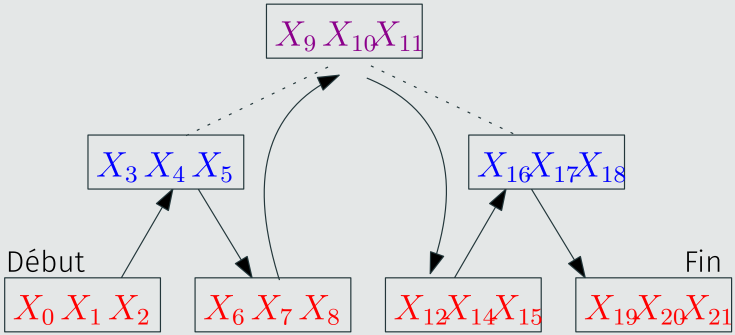
(comparé à CTE-NOOP)

De $30\times$ à **seulement** $15\times$ **moins rapide** que SATPLAN !

Contribution : nouveaux codages pour résolution rapide

Limites : plan de longueur 15 au plus

Solution implémentée



Des métriques pour **expliquer** cette amélioration $2\times$?

- Nombre de littéraux et nombre de clauses
- Ratio entre les types de transitions

Contribution : nouveaux codages pour résolution rapide

Codage	Nb résolus	Temps	Littéraux	Clauses	Ratio transitions
CTE-NOOP	412 (20%)	0%	0%	0%	30%
CTE-EFA	463 (22%)	-55%	-26%	+15%	47%
CTE-OPEN	445 (21%)	-41%	-2%	-28%	17%

Pas vraiment utile :

- Nombre de littéraux et de clauses : pas de corrélation
- Types de transitions : pas de corrélation

Conclusion

Contributions 1

Contributions liées à TouIST :

- Un outil et un langage, TouIST, améliorant l'expressivité pour utiliser les solveurs SAT, SMT et QBF
- aide pour l'enseignement de la logique à l'université
  - ⊕ Étudiants de licence de l'Université Paul Sabatier
  - ⊕ Étudiants de licence de l'**Université de Seville**
- aide pour la recherche
  - ⊕ Codages pour la planification (Toulouse)
  - ⊕ Modélisation pour raisonnement épistémique (Rennes)

Contributions liées à la planification QBF :

- Une traduction systématique de codages depuis SAT vers QBF
- Deux nouveaux codages CTE-EFA et CTE-OPEN **plus performants que l'existant** (CTE-NOOP)
- Contribution d'un **large ensemble de benchmarks** générés à partir des problèmes IPC

Perpectives liées à **TouIST** :

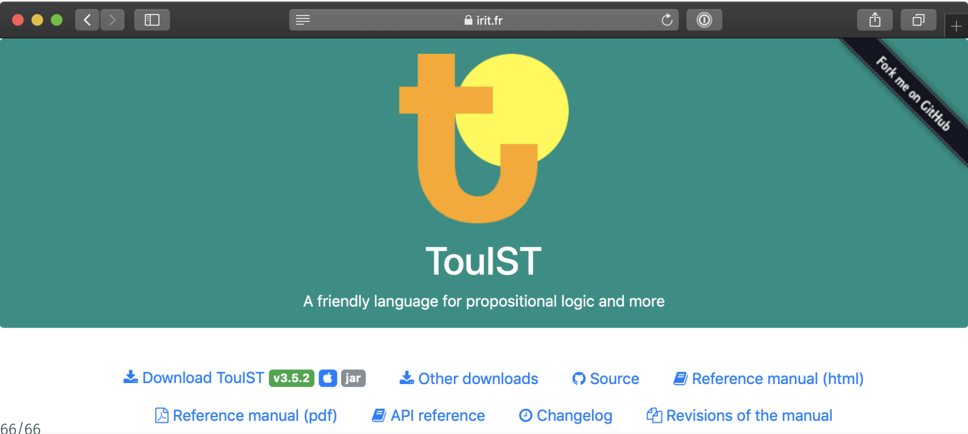
- Finaliser TouIST « pour le web » pour favoriser sa diffusion
- utiliser TouIST pour la **planification épistémique** (DL-PA)
- utiliser TouIST pour améliorer les performance de **LoTREC** (solveur de logiques modales)
- interfacer TouIST à **SESAME** (domaine de l'argumentation)

Perspectives

Perpectives liées à la planification :

- **Publication** de nos nouveaux benchmarks (à QBFEVAL) pour pousser les solveurs QBF à s'améliorer/se spécialiser dans ce domaine
- Trouver **de meilleures métriques** permettant d'expliquer les améliorations et pouvoir les exploiter

Merci


irit.fr
Fork me on GitHub
TouIST
A friendly language for propositional logic and more
Download TouIST v3.5.2 jar
Other downloads
Source
Reference manual (html)
Reference manual (pdf)
API reference
Changelog
Revisions of the manual

Cashmore, M., Fox, M., and Giunchiglia, E. (2012).
Planning as quantified boolean formula.
In Raedt, L. D., Bessière, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., and Lucas, P. J. F., editors, *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 217–222. IOS Press.

Kautz, H. (2004).
Satplan04 : Planning as satisfiability.
In *Abstracts of the 4th International Planning Competition, IPC-04.*

Kautz, H., Selman, B., and Hoffmann, J. (2006).
SatplanÕ04 : Planning as satisfiability.

In *Abstracts of the 5th International Planning Competition, IPC-06*.