Interface bidirectionnelle Prolog - SMT

Nicolas Lhost

Wim Vanhoof

Étienne Payet
Fred Mesnard

PROLOG

Introduction et concepts de base

Prolog: Introduction

- Qu'est-ce que Prolog?
  - Langage de **pro**grammation **log**ique créé dans les années 1970
  - Calcul de prédicats de premier ordre grâce aux faits et relations
  - Très déclaratif
  - Concepts fondamentaux: unification, récursivité et backtracking

- Domaines d'application
  - Intelligence artificielle, systèmes experts, planification, vérification formelle et programmation par contraintes.

Prolog: Syntaxe et sémantique

- Les **FAITS** décrivent des relations:

```
parent(john, alice).        parent(alice, marie).
parent(alice, bob).
```

- Les **REGLES** permettent de déduire de nouvelles informations

```
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```

- Les **REQUÊTES** permettent d'interroger les relations

```
grandparent(X, marie).      X/john
grandparent(john, Y).   ⇨   Y/bob      ;      Y/marie
grandparent(alice, Z).      false.
```

Prolog: Mécanismes d’exécution

Résolution SLD (*Selective Linear Definite clause resolution*):

- Méthode d'inférence pour répondre à des requêtes
- Cherche des substitutions pour les variables qui rendent les clauses vraies
- Utilise l'**unification** pour comparer les termes

```
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```

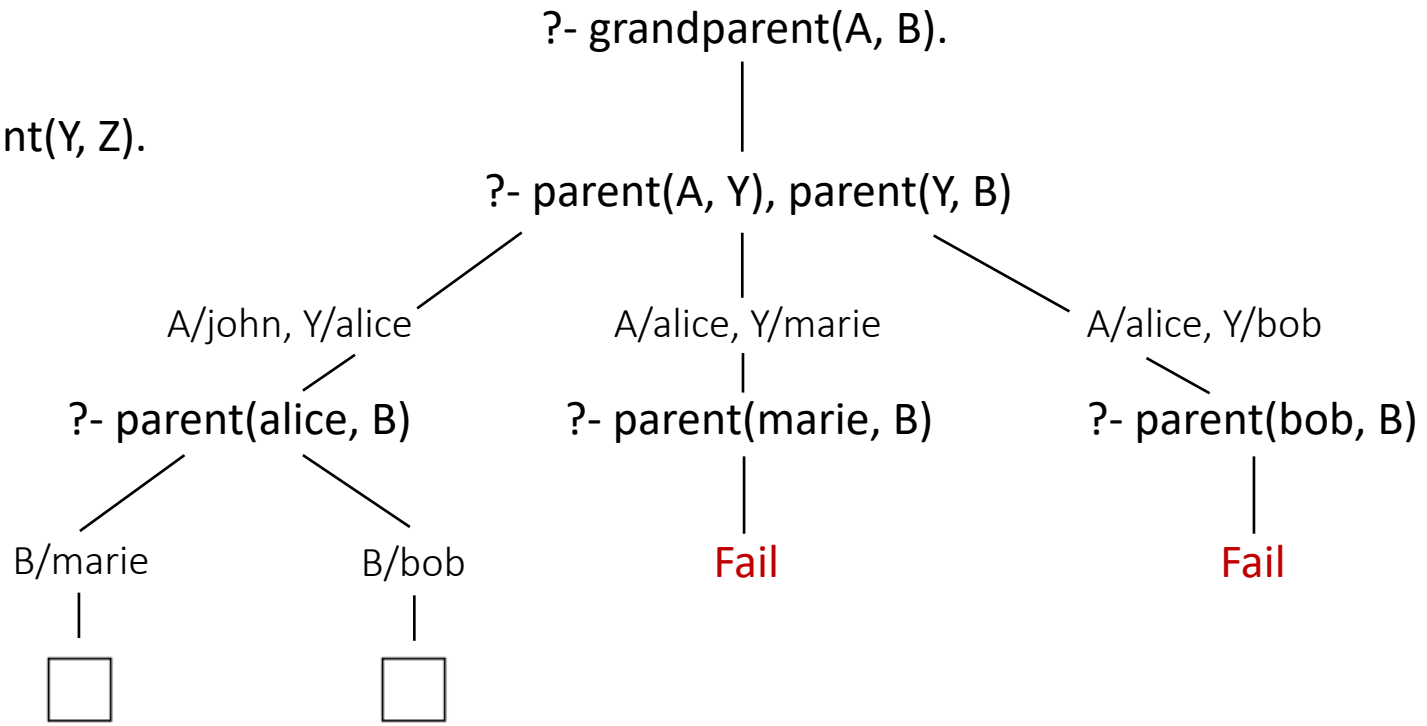
Arbre de recherche ayant la requête de départ comme racine:

- Dans l’ordre de déclaration
- Si plusieurs clauses possibles, il y a un **point de choix**
- L’arbre est créé en profondeur. Si pas de solution, **backtracking** jusqu’au dernier point de choix

Prolog: Mécanismes d'exécution - Exemple

```
parent(john, alice).
parent(alice, marie).
parent(alice, bob).

grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```



Prolog: Récursivité et listes

- La récursion est largement utilisée en Prolog pour résoudre des problèmes complexes.

- Les listes sont des structures de données récursives, représentées par des paires tête-queue [H|T]. Exemple: [1,2,3] = [1|[2|[3|[]]]]

- Les fonctions récursives incluent généralement un cas de base (pour les listes vides) et un cas récursif (pour les listes non vides).

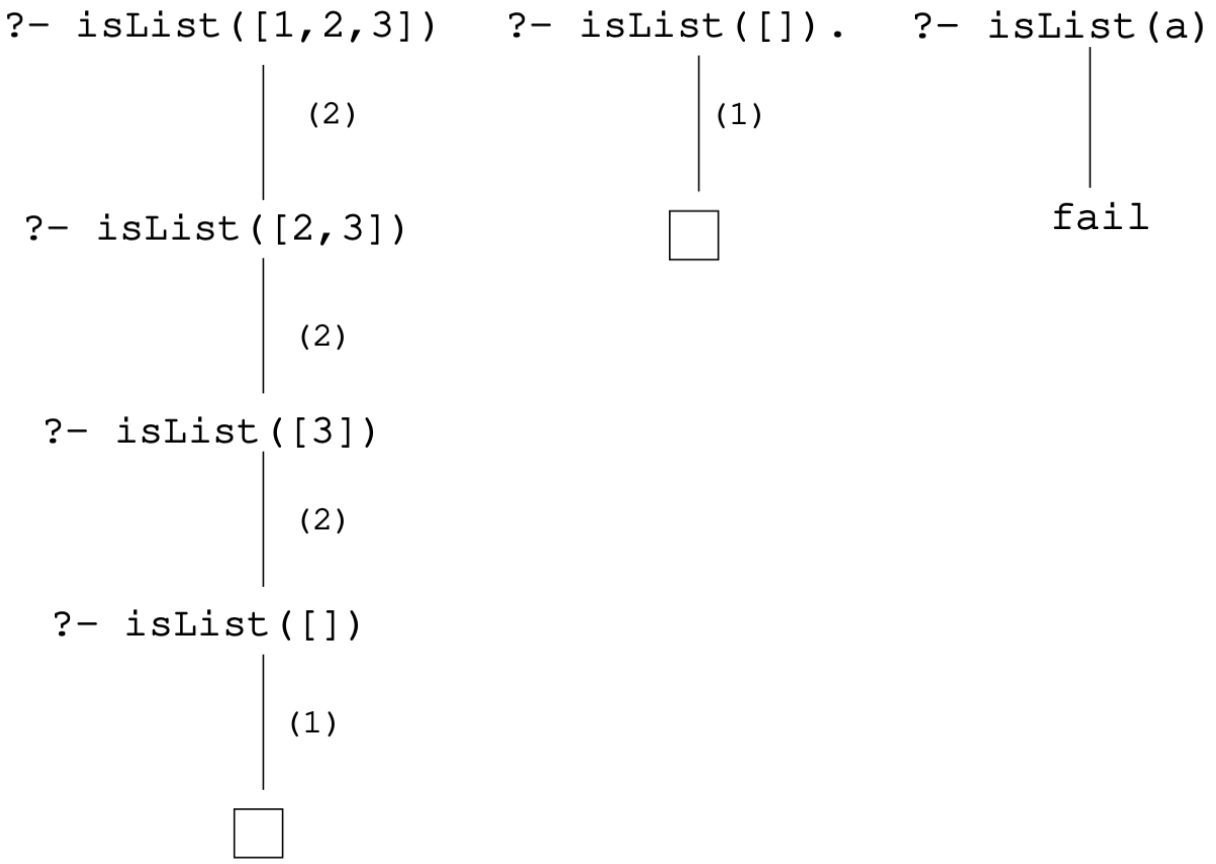
```
isList([]).
isList([X|Xs]):- isList(Xs).
```

Prolog: Récursivité et listes - Exemple

- Relation isList pour déterminer si un terme est une liste

```
isList([]).
isList([X|Xs]):- isList(Xs).


?- isList([1,2,3]).
?- isList([]).
?- isList(a).
```



Prolog: CLP *(Constraint Logic Programming)*

- Description déclarative du problème
- La solution requiert l'exploration d'un espace de recherche
  - Domaines finis CLP(FD) , les booléens CLP(B) , les réels CLP(R) , etc.
- Résolution de problèmes combinatoires et d'optimisation

```
:- use_module(library(clpfd)).

solve(X, Y) :-
   X in 1..9,
   Y in 1..9,
   X + Y #= 10,
   label([X, Y]).
```

```
?- solve(X, Y).
X/1, Y/9 ;
X/2, Y/8 ;
X/3, Y/6 ;
…
X/9, Y/1 ;
false
```

```
?- solve(2, Y).
Y/8
```

```
?- solve(2, 8).
true
```

Prolog: CLP - N-Queens problem

```
n_queens(N, Qs) :-
        length(Qs, N),
        Qs ins 1..N,
        safe_queens(Qs).

safe_queens([]).
safe_queens([Q|Qs]) :-
        safe_queens(Qs, Q, 1),
        safe_queens(Qs).

safe_queens([], _, _).
safe_queens([Q|Qs], Q0, D0) :-
        Q0 #\= Q,
        abs(Q0 - Q) #\= D0,
        D1 #= D0 + 1,
        safe_queens(Qs, Q0, D1).
```

```
?- n_queens(8, Qs), label(Qs).

 Qs = [1, 5, 8, 6, 3, 7, 2, 4] .
```

```
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
```

SMT

Rappels des concepts de base

Problème SAT (satisfiabilité booléenne)

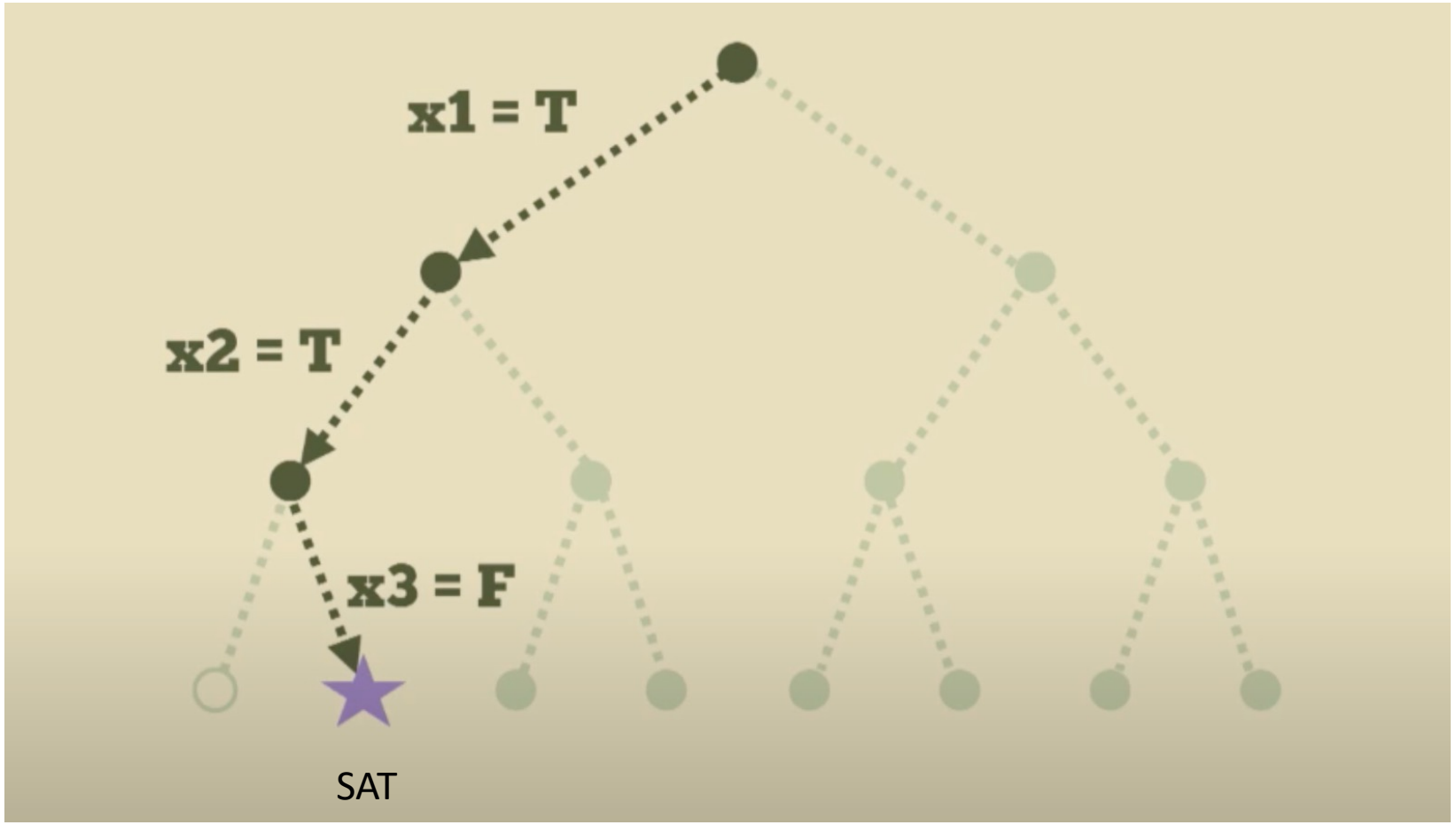
- Permet de déterminer si un problème est satisfiable
  - Input: Formule de logique propositionnelle avec k variables
  - Output: Vrai ou Faux

$$((A \wedge B) \vee C) \wedge \neg(B \vee A)) \wedge (B \vee \neg C)$$

$\Rightarrow$ Calculer la table de vérité de la formule pour voir si on a un 1 pour la globalité (se calcule en temps 2^k)

Par force brute, ça va prendre énormément de temps. On va donc utiliser des solvers SAT pour les problèmes de décision

Algorithme Deepfirst



Algorithmes SAT

- Le brute force ne fonctionnera pas, il faut choisir intelligemment les littéraux et clauses

- Unit propagation

```
;; x1 = false
;; x2 = false
(or x1 x2 x3)

;; x3 => true
```

```
(and (or x4)
     (or (not x4) x5))

;; x4 => true
;; x5 => true
```

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

Chercher des infos faciles et trouver des conflits

- Algorithme DPLL, CDCL, etc.

Solver SMT

« Problème de décision pour des formules de **logique du premier ordre** avec égalité (sans quantificateurs), combinées à des **théories** dans lesquelles sont exprimées certains symboles de prédicat et/ou certaines fonctions » - Wikipedia

$\Rightarrow$ Théories: Arithmétique (réels, integers), Bitvectors, Arrays, etc.

$\Rightarrow$ Logique du premier ordre:

- Variables/constantes: x, y
- Fonctions: foo(x)
- Prédicats (ou relations) sur les éléments: x > y
- Connecteurs logiques: &&, ||, =>, not
- Quantificateurs: ∃, ∀

Fonctionnement de base d'un solveur SMT

Deux cœurs principaux : Solver SAT et une ou plusieurs procédures de décision de la théorie

$$(x < 0 \vee x > 1) \wedge (x = y + 5) \wedge (y > 0)$$

$$(\quad a \quad \vee \quad b \quad) \wedge (\qquad c \qquad) \wedge (\quad d \quad)$$

$\Rightarrow$ Si pas de modèle SAT, la formule n'est pas satisfiable

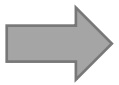
$\Rightarrow$ Si modèle SAT, vérification de la cohérence avec des procédures de décision de la théorie

- $\Rightarrow$Si cohérence, la formule est satisfiable
- $\Rightarrow$Si pas cohérence, on cherche un autre modèle

SMT: Commandes de base

- Affichage de texte dans l'output `(echo "starting Z3...")`
- Déclaration de constantes et de fonctions
```
(declare-const a Int)
(declare-fun f (Int Bool) Int)
```
- Assertion de formules
```
(assert (> a 10))
(assert (< (f a true) 100))
```
- Vérification de satisfiabilité: sat, unsat ou unknown `(check-sat)`
- Instanciation des variables et fonctions qui rendent le model sat `(get-model)`

```
(echo "starting Z3...")
(declare-const a Int)
(declare-fun f (Int Bool) Int)
(assert (> a 10))
(assert (< (f a true) 100))
(check-sat)
(get-model)
```



```
starting Z3...
sat
(model
  (define-fun a () Int
    11)
  (define-fun f ((x!1 Int) (x!2 Bool)) Int
    (ite (and (= x!1 11) (= x!2 true)) 0
      0))
)
```

SMT

Les théories

Théorie: Arithmétique non linéaire

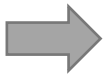
Plus compliqué pour Z3 car indécidable -> peut retourner des unknown

```
(declare-const a Int)
(assert (> (* a a) 3))
(check-sat)
(get-model)

(declare-const b Real)
(declare-const c Real)
(assert (= (+ (* b b b) (* b c)) 3.0))
(check-sat)

(reset)

(declare-const b Real)
(declare-const c Real)
(assert (= (+ (* b b b) (* b c)) 3.0))
(check-sat)
(get-model)
```



```
sat
(model
  (define-fun a () Int
    (- 8))
)

unknown

sat
(model
  (define-fun b () Real
    (/ 1.0 8.0))
  (define-fun c () Real
    (/ 1535.0 64.0))
)
```

Théorie: Bitvectors

- Les CPU et principaux langages de programmation utilisent l'arithmétique sur des vecteurs de bits de taille fixe.
- Bitvectors signés ou non signés
- Supporte les Bitvectors de taille arbitraire:
  - Binaire: `#b0100 ;bitvector de taille 4 représentant 4`
  - Hexadécimal: `#x0a0 ;bitvector de taille 12 représentant 160`
  - Taille fixée : `(_ bv10 32) ;bitvector de taille 32 représentant 10`
- Choisir l'affichage avec option

```
(display (_ bv10 32))
(set-option :pp.bv-literals false)
(display #x0a0)
(display #b0100)
```

```
#x0000000a

(_ bv160 12)
(_ bv4 4)
```

Théorie: Arrays

- Théorie basique des tableaux:
  - Select: `(select a i)` retourne la valeur du tableau a stockée à l'indice i
  - Store: `(store a i v)` retourne un tableau identique à a contenant v à l'indice i
  - Déclaration: `(declare-const a1 (Array Int Int))` peut prendre n'importe quel sort (type)
- Exemple:

```
(declare-const x Int)
(declare-const y Int)
(declare-const a1 (Array Int Int))
(assert (= (select a1 x) x))
(assert (= (store a1 x y) a1))
(check-sat)
(get-model)
```

```
sat
(model
  (define-fun y () Int
    1)
  (define-fun a1 () (Array Int Int)
    (_ as-array k!0))
  (define-fun x () Int
    1)
  (define-fun k!0 ((x!1 Int)) Int
    (ite (= x!1 1) 1
      0))
)
```

Interface Prolog - SMT

Tirer profit des deux langages

Interface: Avantages partagés

- Exploiter les capacités de raisonnement de Prolog pour manipuler et générer des formules SMT
  - Ecrire du code à la volée
  - Réagir à la satisfiabilité et aux modèles

- Utiliser les solveurs SMT pour résoudre des problèmes de contraintes complexes et obtenir des solutions optimisées
  - Théories spécifiques
  - Raisonnement plus symbolique et conceptuel
  - Précision et performance

Interface: Etat de l’art

Interfaces

- pyZ3 : interface Python – Z3
- SWIPrologZ3 : interface SWI Prolog – Z3 (en travail, interface basique)
- Ciao – Yices/Z3

Parsers

- Package smtlib de SWI Prolog
- Ciao SMT

Interface: Types d’interface

Haut niveau	Bas niveau
Pipes et fichiers pour la communication + Facilité de mise en œuvre et de débogage + Flexibilité pour intégrer différents solveurs SMT - Performances inférieures car entrée/sortie disque	Intégration directe des solveurs SMT dans Prolog via un autre langage + Meilleures performances car intégration étroite et appels directs de fonctions - Moins flexible pour changer de solveur SMT - Complexité de mise en œuvre et de maintenance accrue

Interface: Bases de l’interface

- Haut niveau
  - Temps et facilité de mise en place
  - ISO-Prolog <> SMT-LIB2

- Basé sur le package SMTLIB de SWI Prolog
  - Lecture et écriture du langage SMT-LIB2
  - Représentation en liste de commandes manipulables par Prolog
  - Via Streams et Fichiers

Interface: Package SMTLIB

```
(set-logic QF_LIA)
(declare-fun w () Int)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (> x y))
(assert (> y z))
(set-option :print-success false)
(push 1)
(assert (> z x))
(check-sat)
(get-info :all-statistics)
(pop 1)
(push 1)
(check-sat)
(exit)
```

```
X = list([
  [reserved('set-logic'),symbol('QF_LIA')],
  [reserved('declare-fun'),symbol(w),[],symbol('Int')],
  [reserved('declare-fun'),symbol(x),[],symbol('Int')],
  [reserved('declare-fun'),symbol(y),[],symbol('Int')],
  [reserved('declare-fun'),symbol(z),[],symbol('Int')],
  [reserved(assert),[symbol(>),symbol(x),symbol(y)]],
  [reserved(assert),[symbol(>),symbol(y),symbol(z)]],
  [reserved('set-option'),[keyword('print-success'),symbol(false)]],
  [reserved(push),numeral(1)],
  [reserved(assert),[symbol(>),symbol(z),symbol(x)]],
  [reserved('check-sat')],
  [reserved('get-info'),keyword('all-statistics')],
  [reserved(pop),numeral(1)],
  [reserved(push),numeral(1)],
  [reserved('check-sat')],
  [reserved(exit)]
]).
```

Interface: Fonctionnement général

1. Création d'un nouveau stream et choix du nom du fichier

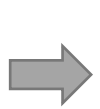
   smt_new_stream('nqueens', Script),

2. Ecriture des commandes de script SMT

   smt_declare_fun(x, [], 'Int', Script),
   smt_declare_fun(y, [], 'Int', Script),
   smt_assert([>, x, y], Script),
   smt_check_sat(Script),

3. Résolution du script via un solver SMT choisi
   - Affichage de la réponse du solver dans la console et création de fichiers pour le script et le résultat

   smt_solve_with_z3(Script),



nqueens.smt2

nqueens.result.smt2

4. Retour au point 2 ou fermeture du stream

   smt_close_stream(Script),

Interface: Commandes de script SMT

(assert ⟨*term*⟩)
(check-sat)
(check-sat-assuming (⟨*prop_literal*⟩*))
(declare-const ⟨*symbol*⟩ ⟨*sort*⟩)
(declare-datatype ⟨*symbol*⟩ ⟨*datatype_dec*⟩)
(declare-datatypes (⟨*sort_dec*⟩$^{n+1}$) (⟨*datatype_dec*⟩$^{n+1}$))
(declare-fun ⟨*symbol*⟩ (⟨*sort*⟩*) ⟨*sort*⟩)
(declare-sort ⟨*symbol*⟩ ⟨*numeral*⟩)
(define-fun ⟨*function_def*⟩)
(define-fun-rec ⟨*function_def*⟩)
(define-funs-rec (⟨*function_dec*⟩$^{n+1}$) (⟨*term*⟩$^{n+1}$))
(define-sort ⟨*symbol*⟩ (⟨*symbol*⟩*) ⟨*sort*⟩)
(echo ⟨*string*⟩)
(exit)
(get-assertions)
(get-assignment)
(get-info ⟨*info_flag*⟩)
(get-model)
(get-option ⟨*keyword*⟩)
(get-proof)
(get-unsat-assumptions)
(get-unsat-core)
(get-value (⟨*term*⟩$^{+}$))
(pop ⟨*numeral*⟩)
(push ⟨*numeral*⟩)
(reset)
(reset-assertions)
(set-info ⟨*attribute*⟩)
(set-logic ⟨*symbol*⟩)
(set-option ⟨*option*⟩)

Interface: Ecriture des commandes SMT

- Toutes les commandes de SMTLIB pour l'écriture de scripts sont disponibles

```
smt_set_option(produce-proofs, true, Script),
smt_assert([<, [f, x, 2], 5], Script),
smt_set_logic('QF_LIA', Script),
smt_declare_const(x, 'Int', Script),
smt_check_sat(Script),
smt_get_model(Script),
```

- Commande spéciale pour écrire directement dans le stream
  - Utiliser des commandes non SMTLIB2
  - Eviter des bugs potentiels

```
smt_parse("(maximize (+ x y))", Script)
```

Interface: Résolution du script via un solver

- Choix du solver via prédicat Prolog

```
smt_solve_with_z3(Script),
smt_solve_with_cvc4(Script),
```

- Différences dans les solvers
  - Valeurs d'options par défaut différentes smt_cvc4_options(Script)
  - Outputs et modèles différents

Z3:
```
sat
hello
(
  (define-fun z () Int
    (- 1))
  (define-fun x () Int
    1)
  (define-fun y () Int
    0)
)
```

CVC4:
```
sat
"hello"
(model
(define-fun x () Int 1)
(define-fun y () Int 0)
(define-fun z () Int (- 1))
)
```

Interface

Nouvelles fonctionnalités

Interface: Fonctionnalités basiques

- Ecriture de Script plus poussée

```
loop(0).
loop(N) :-
    N > 0,
    Double is N*2
    smt_assert([=, [select, array, N], Double],Script),
    N1 is N - 1,
    loop(N1).
```

- Exécution directe de fichier

```
smt_solve_file('nqueens.smt2').
```

- Chargement de fichier dans le stream actuel

```
smt_load_file('nqueens.smt2', Script),
```

Interface: Fonctionnalités avancées

- Vérification de la satisfiabilité ou de l'insatisfiabilité
  - Dans le script, on indique la réponse attendue pour un check-sat

```
smt_check_sat(Script),
```

```
smt_check_sat_continue_if_sat(Script),
smt_check_sat_continue_if_unsat(Script),
```

  - Quand on solve, le prédicat smt_solve_with_z3/cvc4 renvoie **true** si toutes les réponses attendues sont correctes et **false** dans le cas contraire

```
(smt_solve_with_z3(Script) -> actions si true ; actions si false).
```

  - Ecriture d'une «balise» (echo "continue-if-sat") dans le fichier SMT pour indiquer quel check-sat est à vérifier. Dans l'output du solver:

```
continue-if-sat
sat
```

```
continue-if-sat
unsat (ou unknow)
```

Interface: Fonctionnalités avancées

- Transformation de valeurs en contraintes
  - Dans le script, on indique quelles variables vont voir leur valeur être transformée en contrainte

    smt_get_model(Script),  smt_get_model_to_constraint_for([x,y], Script)

  - Quand on solve avec smt_solve_with_z3/cvc4, les contraintes vont être automatiquement écrites à la suite du Script.

  - « Balisage » avec des echo dans le fichier SMT pour indiquer les variables et le modèle dans lequel prendre les valeurs à transformer en contraintes.
    - Création de contraintes avec nom unique
    - Permet de faire plusieurs smt_get_model_to_constraint_for avant un solve

Interface: Contraintes Automatiques - Exemple

Script de base:

```
(declare-fun x () Int )
(declare-fun y () Int )
(declare-fun z () Int )
(assert (> x y ))
(assert (> y z ))
(check-sat )
(echo "model-to-constraint-start-1")
(echo "(x y)")
(get-model )
(echo "model-to-constraint-end-1")
```

Modèle:

```
sat
model-to-constraint-start-1
(x y)
(
  (define-fun z () Int
    (- 1))
  (define-fun x () Int
    1)
  (define-fun y () Int
    0)
)
model-to-constraint-end-1
```

Script avec contraintes rajoutées:

```
(declare-fun x () Int )
(declare-fun y () Int )
(declare-fun z () Int )
(assert (> x y ))
(assert (> y z ))
(check-sat )
(echo "model-to-constraint-start-1")
(echo "(x y)")
(get-model )
(echo "model-to-constraint-end-1")
(define-fun x_from_model_1 ()
   Int 1 )
(define-fun y_from_model_1 ()
   Int 0 )
(assert (or (not (= x x_from_model_1 ))
            (not (= y y_from_model_1 ))
 ))
```

smt_get_model_to_constraint_for([x,y], Script)

Interface: Fonctionnalités avancées

- Récupération de la valeur de constantes du dernier modèle dans Prolog
  - Dans le script, on indique quelle constante va voir sa valeur être récupérée dans une variable de Prolog

    smt_get_last_model_value(x, X, Script)

  - Doit être effectué après smt_solve_with_z3/cvc4, pour avoir un modèle dans lequel aller récupérer la valeur de la constante voulue
    - Récupère dans le dernier modèle
    - Peut être utilisé dans prolog pour d'autres calculs ou pour la vérification de la solution

Interface: N-Queens - Soluce and Verify

```
queens(N):-
   smt_new_stream('Nqueens', Script),
   queens_declaration(N, Script),
   queens_on_board(N, Script),
   queens_not_same_column(N, Script),
   smt_define_fun('diagonal-threat', [['x1','Int'], ['y1','Int'], ['x2','Int'], ['y2','Int']], 'Bool',
         [=, [abs, [-, 'x1', 'x2']], [abs, [-, 'y1', 'y2']]], Script),
   queens_not_same_diagonal(N, 0, Script),
   smt_check_sat(Script),
   smt_get_model(Script),
   smt_solve_with_z3(Script),
   queens_get_solution(Script, N, 0, AllValues),
   valid_solution(AllValues, N),
   print_board(AllValues),
   smt_close_stream(Script).
```

Interface: N-Queens - Soluce and Verify

```
(declare-fun r0 () Int )
(declare-fun r1 () Int )
(declare-fun r2 () Int )
(declare-fun r3 () Int )
(assert (and (>= r0 0 ) (< r0 4 )))
(assert (and (>= r1 0 ) (< r1 4 )))
(assert (and (>= r2 0 ) (< r2 4 )))
(assert (and (>= r3 0 ) (< r3 4 )))
(assert (distinct r0 r1 r2 r3 ))
(define-fun diagonal-threat ((x1 Int )(y1 Int )(x2 Int )(y2 Int ))
   Bool (= (abs (- x1 x2 )) (abs (- y1 y2 )))
)
(assert (not (diagonal-threat r0 0 r1 1 )))
(assert (not (diagonal-threat r0 0 r2 2 )))
(assert (not (diagonal-threat r0 0 r3 3 )))
(assert (not (diagonal-threat r1 1 r2 2 )))
(assert (not (diagonal-threat r1 1 r3 3 )))
(assert (not (diagonal-threat r2 2 r3 3 )))
(check-sat )
(get-model )
```

```
sat
(
  (define-fun r0 () Int 5)
  (define-fun r1 () Int 3)
  (define-fun r2 () Int 6)
  (define-fun r7 () Int 2)
  (define-fun r3 () Int 0)
  (define-fun r6 () Int 4)
  (define-fun r4 () Int 7)
  (define-fun r5 () Int 1)
)
```

```
. . . . . Q . .
. . . Q . . . .
. . . . . . Q .
Q . . . . . . .
. . . . . . . Q
. Q . . . . . .
. . . . Q . . .
. . Q . . . . .
```

Interface: N-Queens - Find all solutions

```
queensAllSolutions(N):-
    smt_new_stream('NqueensAllSolution', Script),
    queens_declaration(N, Script),
    queens_on_board(N, Script),
    queens_not_same_column(N, Script),
    smt_define_fun('diagonal-threat', [['x1','Int'], ['y1','Int'], ['x2','Int'], ['y2','Int']], 'Bool',
          [=, [abs, [-, 'x1', 'x2']], [abs, [-, 'y1', 'y2']]], Script),
    queens_not_same_diagonal(N, 0, Script),
    allrows(N, Rows),
    getAllSolutions(Script, N, Rows, [], Solutions),
    writeAllSolutions(Solutions),
    smt_close_stream(Script).
```

Interface: N-Queens - Find all solutions

```
getAllSolutions(Script, N, Rows, Acc, Solutions) :-
    smt_check_sat_continue_if_sat(Script),
    smt_get_model_to_constraint_for(Rows, Script),
    (smt_solve_with_z3(Script) ->
        (queens_get_solution(Script, N, 0, Solution),
        valid_solution(Solution, N),
        getAllSolutions(Script, N, Rows, [Solution | Acc], Solutions))
    ;
        Solutions = Acc
    ).
```

Interface: N-Queens - Soluce and Verify

```
[…]
(echo "continue-if-sat")
(check-sat )
(echo "model-to-constraint-start-1")
(echo "(r0 r1 r2 r3)")
(get-model )
(echo "model-to-constraint-end-1")
```

solve

```
(define-fun r0_from_model_1 () Int 2 )
(define-fun r1_from_model_1 () Int 0 )
(define-fun r3_from_model_1 () Int 1 )
(define-fun r2_from_model_1 () Int 3 )
(assert (or (not (= r0 r0_from_model_1 ))
      (not (= r1 r1_from_model_1 ))
      (not (= r2 r2_from_model_1 ))
      (not (= r3 r3_from_model_1 )))
)
(echo "continue-if-sat")
(check-sat )
(echo "model-to-constraint-start-2")
(echo "(r0 r1 r2 r3)")
(get-model )
(echo "model-to-constraint-end-2")
```

solve

```
(define-fun r0_from_model_2 () Int 1 )
(define-fun r1_from_model_2 () Int 3 )
(define-fun r3_from_model_2 () Int 2 )
(define-fun r2_from_model_2 () Int 0 )
(assert (or (not (= r0 r0_from_model_2 ))
      (not (= r1 r1_from_model_2 ))
      (not (= r2 r2_from_model_2 ))
      (not (= r3 r3_from_model_2 )))
)
(echo "continue-if-sat")
(check-sat )
(echo "model-to-constraint-start-3")
(echo "(r0 r1 r2 r3)")
(get-model )
(echo "model-to-constraint-end-3")
```

solve

[1,3,0,2]
[2,0,3,1]

Interface: N-Queens - Soluce and Verify

```
continue-if-sat
sat
model-to-constraint-start-1
(r0 r1 r2 r3)
(
  (define-fun r0 () Int 2)
  (define-fun r1 () Int 0)
  (define-fun r3 () Int 1)
  (define-fun r2 () Int 3)
)
model-to-constraint-end-1
```

```
continue-if-sat
sat
model-to-constraint-start-2
(r0 r1 r2 r3)
(
  (define-fun r0 () Int 1)
  (define-fun r1 () Int 3)
  (define-fun r3 () Int 2)
  (define-fun r2 () Int 0)
  (define-fun r1_from_model_1 () Int 0)
  (define-fun r0_from_model_1 () Int 2)
  (define-fun r2_from_model_1 () Int 3)
  (define-fun r3_from_model_1 () Int 1)
)
model-to-constraint-end-2
```

```
continue-if-sat
unsat
model-to-constraint-start-3
(r0 r1 r2 r3)
(error "line 108 column 12: model is not available")
model-to-constraint-end-3
```

Résultats

Avantages de l’interface

Résultat: Apports de Prolog pour SMT

- Centralisation du code et créations de fichiers
- Ecriture automatisée
  - Paramétrisation
  - Facilite l'écriture de code
- Récupération de valeur dans le modèle
  - Manipuler les données trouvées par le solver SMT
  - Vérifier la solution
- Capacité de trouver toutes les solutions possibles
- Vérification de satisfiabilité
  - Capacité d'actions différentes en fonction du résultat

Résultat: Apports de SMT pour Prolog

Expressivité

- Utilisation de théories non présentes dans Prolog
  - BitVectors: Cybersécurité, analyse de programmes, vérification de modèles de systèmes matériels
  - Tableaux: N'importe quel type d'index et de valeur

Performance

- Tableaux
  - Vision théorique et manipulation de contraintes symboliques
- Les réels et l'arithmétique non linéaire

Résultat: Expressivité - Exemples

Tableaux: Index non numérique

SMT

```
(declare-const nameAge (Array String Int))
(assert (= (select nameAge "Alice") 25))
(assert (= (select nameAge "Bob") 30))
(assert (= (select nameAge "Charlie") 35))
(declare-const totalAge Int)
(assert (= totalAge (+  (select nameAge "Alice")
                        (select nameAge "Bob")
                        (select nameAge "Charlie"))))
(check-sat)
(get-value (totalAge))
```

PROLOG

```
name_age("Alice", 25).
name_age("Bob", 30).
name_age("Charlie", 35).

total_age(Total) :-
    findall(Age, name_age(_, Age), Ages),
    sumlist(Ages, Total).
```

Index numérique

[5,10,15,20,25]
array(3,6,9,12,15)

Résultat: Expressivité - Exemples

BitVectors: Inversion de bit 1100 1010 <> 1010 1100

SMT

```
(set-logic QF_BV)
(declare-fun input () (_ BitVec 8))
(declare-fun output () (_ BitVec 8))
(assert (= output (concat ((_ extract 3 0) input)
                          ((_ extract 7 4) input))))
(assert (= input #b11001010)) ; Entrée : 202
(check-sat)
(get-model)
(exit)
```

PROLOG

```
:- use_module(library(clpfd)).

bit_permute(Input, Output) :-
    Input in 0..255,
    Output in 0..255,
    HighBits #= Input // 16,
    LowBits #= Input mod 16,
    Output #= LowBits * 16 + HighBits.

% Test avec 202
permute(Output) :-
    bit_permute(202, Output).
```

Résultat: Expressivité - Exemples

BitVectors: Tableau de BitVectors avec des index en BitVectors

SMT

```
(declare-const A (Array (_ BitVec 3) (_ BitVec 4)))
(declare-const idx1 (_ BitVec 3))
(declare-const idx2 (_ BitVec 3))
(declare-const idx3 (_ BitVec 3))
(declare-const idx4 (_ BitVec 3))
(assert (= idx1 #b000))
(assert (= idx2 #b001))
(assert (= idx3 #b010))
(assert (= idx4 #b011))
(assert (=  (bvadd (select A idx1) (select A idx2))
        (bvadd (select A idx3) (select A idx4))))
(check-sat)
(get-model)
```

PROLOG

```
bit_vector([0, 0, 0], 0).
bit_vector([0, 0, 1], 1).
bit_vector([0, 1, 0], 2).
bit_vector([0, 1, 1], 3).
bit_vector([1, 0, 0], 4).
bit_vector([1, 0, 1], 5).
bit_vector([1, 1, 0], 6).
bit_vector([1, 1, 1], 7).

bv_add(X, Y, Z) :-
  bit_vector(X, X_int),
  bit_vector(Y, Y_int),
  bit_vector(Z, Z_int),
  Z_int is X_int + Y_int.
```

```
array_a(A, Index, Value) :-
  bit_vector(Index, Index_int),
  member((Index_int, Value), A).

solution(A) :-
  array_a(A, [0, 0, 0], V1),
  array_a(A, [0, 0, 1], V2),
  array_a(A, [0, 1, 0], V3),
  array_a(A, [0, 1, 1], V4),
  bv_add(V1, V2, Sum1),
  bv_add(V3, V4, Sum2),
  Sum1 == Sum2.
```

Résultat: Performance - Exemples

Tableaux: Exécution et contraintes symboliques

SMT

```
(declare-const x Int)
(declare-const y Int)
(declare-const a1 (Array Int Int))
(assert (= (select a1 x) x))
(assert (= (store a1 x y) a1))
(assert (not (= x y)))
(check-sat)
```

SMT

```
(declare-fun x () Int)
(declare-fun y () Int)
(assert (forall ((x Int) (y Int)) (= (+ x y) (+ y x))))
(check-sat)
```

Résultat: Performance - Exemples

Réels: Calcul d'intersection de cercles

SMT

```
(declare-fun x1 () Real)
(declare-fun y1 () Real)
(declare-fun r1 () Real)
(declare-fun x2 () Real)
(declare-fun y2 () Real)
(declare-fun r2 () Real)
(declare-fun x () Real)
(declare-fun y () Real)
(assert (= (+ (* (- x x1) (- x x1)) (* (- y y1) (- y y1))) (* r1 r1)))
(assert (= (+ (* (- x x2) (- x x2)) (* (- y y2) (- y y2))) (* r2 r2)))
(assert (= x1 1)) (assert (= y1 1)) (assert (= r1 3))
(assert (= x2 4)) (assert (= y2 1)) (assert (= r2 2))
(check-sat)
(get-value (x y))
```

x 3.3333333333333333333333333333?
y (- 0.8856180831641267317355849656 12?)

PROLOG: clp(BNR)

```
circle1(1, 1, 3).
circle2(4, 1, 2).

intersect(X, Y) :-
   circle1(X1, Y1, R1),
   circle2(X2, Y2, R2),
   { (X - X1)*(X - X1) + (Y - Y1)*(Y - Y1) == R1*R1},
   { (X - X2)*(X - X2) + (Y - Y2)*(Y - Y2) == R2*R2}.
```

```
X::real(3.2360679774997894, 4),
Y::real(-1, 3).
```

Résultat: Performance - Exemples

Réels: Calcul non linéaire de puissances

SMT

```
(declare-fun x () Real)
(declare-fun y () Real)
(declare-fun z () Real)
(assert (> x 0))
(assert (> y 0))
(assert (> z 0))
(assert (= (+ (* x x x) (* y y y)) (* z z z)))
(assert (= (+ (* x x) (* y y) (* z z)) 1000))
(assert (< x y))
(assert (< y z))
(check-sat)
(get-value (x y z))
```

x = 12.007904850906793263346465570481?
y = 20.0
z = 21.349712435805057068764188878998?

PROLOG: clp(BNR)

```
solve :-
   { X >= 0.0, Y >= 0.0, Z >= 0.0,
     X*X*X + Y*Y*Y == Z*Z*Z,
     X * X + Y * Y + Z * Z == 1000,
     X =< Y, Y =< Z },
   solve([X,Y,Z]),
```

X::real(0.0, 16.69590879946623),
Y::real(16.695906735903264, 22.360680379086777),
Z::real(21.035374331129102, 22.360680379086777)

Améliorations

Critiques de l’interface

Améliorations

- Expressivité dans l'interface
  - Reste proche du Lisp smt_assert([=,y,[*,2,x]], Script)
- Intégration plus fine
  - Être plus bas niveau en gardant le côté ISO-Prolog et SMTLIB2
- Concurrence pour négation constructive
  - Prolog utilise la négation par l'échec. Pas toujours juste au niveau logique

```
nationalite(fred, francais).
nationalite(pierre, francais).
nationalite(nicolas, belge).

etranger(X) :- not(nationalite(X, francais)).
```

```
?- etranger(nicolas).
true
```

```
?- etranger(X).
false
```

Conclusion

Interface et stage

Conclusion

- Interface de haut niveau réussie entre Prolog et les solveurs SMT
  - Combinaison de la flexibilité et l'expressivité de Prolog avec la performance et les puissantes capacités de résolution des solveurs SMT
- Amélioration/Offre d’un outil dans divers domaines
  - Vérification formelle, synthèse de programmes, intelligence artificielle, logique de contraintes
- Opportunités d'amélioration et d'extension déjà discutées
- Impact sur les communautés de la programmation logique et des solveurs SMT
  - Encourager la collaboration et l'exploration des synergies entre les deux domaines
  - Participer au développement de solutions innovantes pour résoudre des problèmes complexes et expressifs

Avez-vous des questions?

ANNEXES

Code N-Queens

Interface: N-Queens - Soluce and Verify

```
queens(N):-
   smt_new_stream('Nqueens', Script),
   queens_declaration(N, Script),
   queens_on_board(N, Script),
   queens_not_same_column(N, Script),
   smt_define_fun('diagonal-threat', [['x1','Int'], ['y1','Int'], ['x2','Int'], ['y2','Int']], 'Bool',
         [=, [abs, [-, 'x1', 'x2']], [abs, [-, 'y1', 'y2']]], Script),
   queens_not_same_diagonal(N, 0, Script),
   smt_check_sat(Script),
   smt_get_model(Script),
   smt_solve_with_z3(Script),
   queens_get_solution(Script, N, 0, AllValues),
   valid_solution(AllValues, N),
   print_board(AllValues),
   smt_close_stream(Script).
```

Annexes: N-Queens

```
queens_declaration(N, Script) :-
      succ(PrecN, N),
      forall(between(0, PrecN, I),
         (   atom_concat('r', I, VarName),
            smt_declare_fun(VarName, [], 'Int', Script)
         )
      ).

queens_on_board(N, Script) :-
      succ(PrecN, N),
      forall(between(0, PrecN, I),
         (   atom_concat('r', I, VarName),
            smt_assert([and,[>=, VarName, 0],[<, VarName, N]], Script)
         )
      ).
```

Annexes: N-Queens

```
queens_not_same_column(N, Script) :-
    succ(PrecN, N),
    findall(Queen, (between(0, PrecN, I), atom_concat('r', I, Queen)), Queens),
    smt_assert([distinct | Queens], Script).

queens_not_same_diagonal(N, I, _) :-
    I >= N.
queens_not_same_diagonal(N, I, Script) :-
    I < N,
    JStart is I + 1,
    queens_not_same_diagonal_inner_loop(N, I, JStart, Script),
    I1 is I + 1,
    queens_not_same_diagonal(N, I1, Script).
```

Annexes: N-Queens

```
queens_not_same_diagonal_inner_loop(N, _, J, _) :-
    J >= N.
queens_not_same_diagonal_inner_loop(N, I, J, Script) :-
    J < N,
    atom_concat('r', I, R1),
    atom_concat('r', J, R2),
    smt_assert([not, ['diagonal-threat', R1, I, R2, J]], Script),
    J1 is J + 1,
    queens_not_same_diagonal_inner_loop(N, I, J1, Script).
```

Annexes: N-Queens - Soluce and Verify

```
valid_solution(Queens, N) :-
        length(Queens, N),
        on_board(Queens, N),
        valid_rows(Queens),
        valid_diagonals(Queens).

on_board([], _).
on_board([Q | Rest], N) :-
            Q >= 0,
            Q < N,
            on_board(Rest, N).

valid_rows(Queens) :-
            sort(Queens, Sorted),
            length(Queens, Len),
            length(Sorted, Len).
```

```
valid_diagonals([]).
valid_diagonals([Q | Rest]) :-
            valid_diagonal(Q, Rest, 1),
            valid_diagonals(Rest).

valid_diagonal(_, [], _).
valid_diagonal(Q, [R | Rest], Dist) :-
            Diff is abs(Q - R),
            Diff =\= Dist,
            NewDist is Dist + 1,
            valid_diagonal(Q, Rest, NewDist).
```

Interface: N-Queens - Find all solutions

```
queensAllSolutions(N):-
    smt_new_stream('NqueensAllSolution', Script),
    queens_declaration(N, Script),
    queens_on_board(N, Script),
    queens_not_same_column(N, Script),
    smt_define_fun('diagonal-threat', [['x1','Int'], ['y1','Int'], ['x2','Int'], ['y2','Int']], 'Bool',
          [=, [abs, [-, 'x1', 'x2']], [abs, [-, 'y1', 'y2']]], Script),
    queens_not_same_diagonal(N, 0, Script),
    allrows(N, Rows),
    getAllSolutions(Script, N, Rows, [], Solutions),
    writeAllSolutions(Solutions),
    smt_close_stream(Script).
```

Interface: N-Queens - Find all solutions

```
getAllSolutions(Script, N, Rows, Acc, Solutions) :-
    smt_check_sat_continue_if_sat(Script),
    smt_get_model_to_constraint_for(Rows, Script),
    (smt_solve_with_z3(Script) ->
        (queens_get_solution(Script, N, 0, Solution),
        valid_solution(Solution, N),
        getAllSolutions(Script, N, Rows, [Solution | Acc], Solutions))
    ;
        Solutions = Acc
    ).
```

Annexes: N-Queens - Find all solutions

```
queens_get_solution(_, N, N, []).
queens_get_solution(Script, N, I, [Value | Rest]) :-
    I < N,
    atom_concat('r', I, Var),
    smt_get_last_model_value(Var, Value, Script),
    I1 is I + 1,
    queens_get_solution(Script, N, I1, Rest).
```

Annexes: N-Queens - Find all solutions

```
allrows(N, Rows) :-
        succ(PrecN,N),
        numlist(0, PrecN, NumList),
        maplist(row_var, NumList, Rows).

    row_var(N, RowVar) :-
        atom_concat('r', N, RowVar).

    writeAllSolutions([]).
    writeAllSolutions([Solution | Rest]) :-
        writeln(Solution),
        writeAllSolutions(Rest).
```