Satisfiability Modulo Theory
SMT

SAT : rappels

- **Le problème de SATisfaction :**
  - **Soient N variables Bool**
    - *a,b,c*
  - **Soit f:B^k->B, exprimé comme T contraintes :**
    - *Contrainte = OR sur un sous ensemble des variables positive ou niée*
    - *Combinées entre elles avec AND*
      - a OR b
      - b OR c
      - !b
  - **Existe-t-il une valuation v des k variables telle que f(v) vrai ?**
    - *Si oui : SAT et on peut l' exhiber*
      - SAT: a !b c
    - *Si non (plus dur) : UNSAT et on peut exhiber un UNSAT-core*
      - Explique l'incohérence entre les contraintes contradictoires

SAT : Impact

- **Intérêt :**
  - **Exemple NP-complet par excellence => très expressif**
    - *Tout problème NP peut s'exprimer comme un problème SAT*
  - **Enorme progrès technologique**
    - *Solvers SAT efficaces sur des milliers de variables ! (2^2000 = 10^200)*
  - **Standardisation des formats et Compétitions**
    - *Nombreux solvers SAT, domaine actif*

- **Défauts/Limites de SAT:**
  - **Encodage des entiers, de l'arithmétique ? ~Compilation !**
  - **N constante prédéfinie => structures dynamiques ?**
  - **Problème très bas niveau : pas de structure**
  - **Performances codage dépendantes pour un même problème**

SAT Modulo Theory : SMT

- **Le problème SMT :**
  - **Soient N variables Typées**
    - *Entier i, Boolean b, Rationel q, Real x, BitVector bv*
    - *Array : ~Map ou fonction : indice arbitraire sur type arbitraire*
  - **Soit f exprimée comme T contraintes /assert (AND):**
    - *contrainte sur un sous ensemble des variables*
      - q1 < q2 AND i > 0
      - Tab [j] = 3 AND a OR !b
  - **Existe-t-il une valuation (fonctionnelle) v des N variables telle que f(v) ?**
- **Modulo Theory**
  - **Théorie = typage de variables, opérations sur ces variables, contraintes entre variables**
  - **Tout est signature :**
    - *une variable = une fonction arité 0*
    - *Fonction : arguments typés, retour typé*

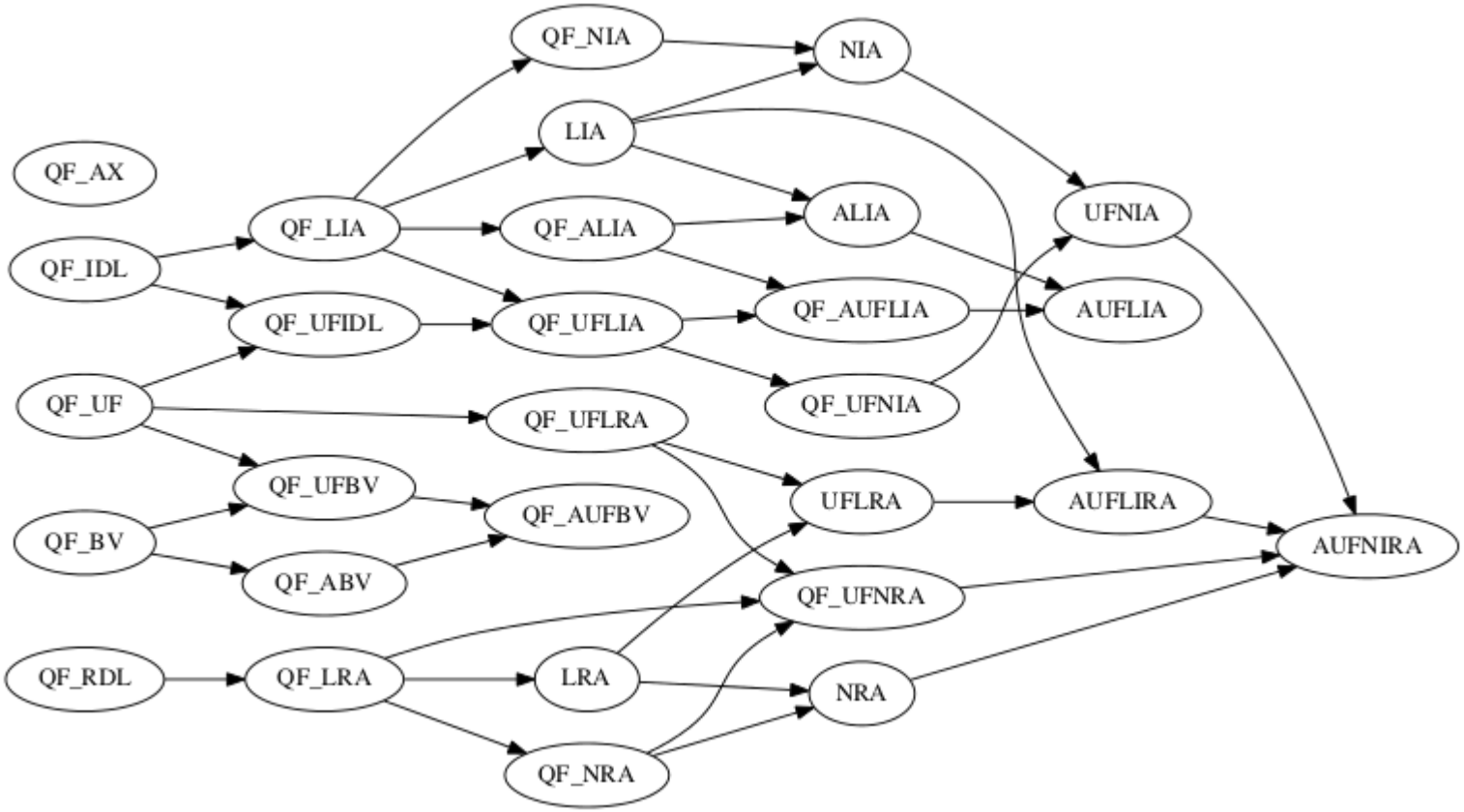
Les théories

- Linear Integer Arithmetic
  - Type Int
  - Arithmétique linéaire : +, -
  - Comparaisons : <, =
- Arrays
  - Type array <I, T>, I et T arbitraire
  - Opérations :
    - *select (array, indice) -> valeur/cellule*
    - *store(array, indice, valeur) -> array homogène*
    - *comparaison*
- Real, Rationals, …

- Deux types de logique pour les contraintes
  - QF : quantifier free (simple)
  - Non QF : Exists, Forall sont autorisés (/!\ Indécidabilité)

Logiques entre elles

- Structure modulaire des théories et de leur combinaison
- Max : Quantifiers, Arrays, Uninterpreted Functions, Non-linear Integer and Real (mixed) Arithmetic : très très expressif !



SMT Exemple

```
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (>= (* 2 x) (+ y z)))
(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)
(push)
(assert (= x y))
(check-sat)
(pop)
(exit)
```

- Très facile d'emploi (notation préfixe)
- Définir ses propres fonctions
  - Spec structurée
- Grande généralité
- Solver interactif
- Codage vers SAT transparent
  - coder les faits connus en Bool
    - *a > 3 -> variable bool*
  - coder les axiomes en contraintes
    - *x < y && y < z => x < z*
    - *x < y XOR x=y XOR y < x*
  - Chainer les requêtes, combiner les théories
- Autres solutions possibles
  - QF_LIA => Simplexe/ILP !

SMT impact

- Plusieurs solvers SMT utilisant SMTlib
  - Un standard pour les problèmes SMT
- Z3 Microsoft
  - Le plus abouti, auto-configuré, multi-solvers, optimisation, interpolants…
- Yices
  - Très performant sur certains exemples
- Autres : CVC4, … souvent spécialisés sur une théorie (e.g. BitVector)

- Très facile à intégrer :
  - solver naturellement interactif
    - *Push/pop*
  - API /drivers dédiés
    - *JSMTlib, drivers Z3 multi-langages*

Z3

Z3

- Z3 de Microsoft
  - Un solveur SMT généraliste puissant
  - Des extensions du standard SMT sont supportées
  - Des extensions pour optimiser un objectif
- Utilisation via
  - Outil ligne de commande
  - Interaction depuis des programmes
    - *Nombreux langages supportés, API C++ et Python recommandées*
- Solver en ligne :
  - https://compsys-tools.ens-lyon.fr/z3/
  - Aussi le tuto : https://jfmc.github.io/z3-play/

Encoder un problème en SMT : types simples

- Définir les variables du problème
  - Bool : Booléens
  - Int : entiers
  - Real : réels
  - Domaine fini (énumération)

- Les « variables » sont des fonctions d'arité 0 munies d'un domaine

```
(declare-fun a () Int)
(declare-const a Int)

(declare-datatypes () ((Color Red Green Blue)))
(declare-const D Color)
```

- NB: les variables n'ont donc qu'une seule valeur dans la solution

Encoder les contraintes du problème

- La notation est préfixée et parenthésée

Ajouter a et b :

(+ a b)

Egalité de deux variables a b

(= a b)

On imbrique : a supérieur stricte à 2 fois b

(> a (* 2 b))

- On énonce un ensemble d'assertions, i.e. des expressions Booléenes qui doivent toutes être vraies dans la solution

Pythagore a-t-il des solutions entières ? C^2 = A^2+B^2

(assert (= (* c c) (+ (* a a) (* b b)))

Poser la question : check-sat, get-model

- Un premier exemple complet :

```
(declare-const a  Int) (declare-const b  Int) (declare-const c Int)
(assert (= (+ (* a a) (* b b)) (* c c)))
(check-sat)
sat
(get-model)
(model
  (define-fun c () Int
    3)
  (define-fun b () Int
    (- 3))
  (define-fun a () Int
    0)
)
```

Ajout de contraintes

```
(assert (> 0 a))
(assert (> 0 b))
(assert (> 0 c))
(check-sat)
sat
(get-model)
(model
  (define-fun c () Int
    15)
  (define-fun b () Int
    9)
  (define-fun a () Int
    12)
)
```

Ajout de contraintes

```
(assert (> 0 a))
(assert (> 0 b))
(assert (> 0 c))
(check-sat)
sat
(get-model)
(model
  (define-fun c () Int
    15)
  (define-fun b () Int
    9)
  (define-fun a () Int
    12)
)
```

Ajout de contraintes


```
(assert (= 3 a))
(check-sat)
sat
(get-model)
(model
  (define-fun c () Int
    5)
  (define-fun b () Int
    4)
  (define-fun a () Int
    3)
)
```

Jusqu'à UNSAT

```
(assert (= c 6))
unsat
(error "line 16 column 10: model is not available")
```

Opérateurs Booléens

```
(declare-const a Bool)
(declare-const b Bool)
(define-fun demorgan () Bool
    (= (and a b) (not (or (not a) (not b)))))
(assert (not demorgan))
(check-sat)
```

Unsat

On aussi =>, xor, …

Théories Int et Réels


```
(declare-const a Int)
(declare-const b Int)
(declare-const c Int)
(declare-const d Real)
(declare-const e Real)
(assert (> a (+ b 2)))
(assert (= a (+ (* 2 c) 10)))
(assert (<= (+ c b) 1000))
(assert (>= d e))
(check-sat)
(get-model)
```

```
sat
(
 (define-fun d () Real
  0.0)
 (define-fun c () Int
  0)
 (define-fun b () Int
  0)
 (define-fun e () Real
  0.0)
 (define-fun a () Int
  10)
)
```

Sur les entiers : opérateurs


```
(assert (= a 10))
(assert (= r1 (div a 4))) ; integer division
(assert (= r2 (mod a 4))) ; mod
(assert (= r3 (rem a 4))) ; remainder
(assert (= r4 (div a (- 4)))) ; integer division
(assert (= r5 (mod a (- 4)))) ; mod
(assert (= r6 (rem a (- 4)))) ; remainder
```

Quelques éléments pratiques

- La contrainte « tous deux à deux distincts », on cite simplement les expressions

(distinct a b c)

- On peut assez facilement changer un problème des reels aux entiers et réciproquement

(< a b) (+ a b) (* a b) etc… sont compatibles

- Ne pas hésiter à mettre des (check-sat) frequents

(check-sat)

- Les opérateurs + et * ainsi que or et and ont des arités arbitraires

(+ x 1 x 2) => 2*x + 3

Exemple : Vache canard

- On compte les pieds et les têtes de canard et de vaches qui passent une porte; combien y a-t-il de chaque espèce ?
- Avec 20 pieds, 6 têtes ?

- Demander au solver d'exhiber des valeurs qui fonctionnent

- Essayer des valeurs qui ne fonctionnent pas

Les tableaux

- Les tableaux sont en fait plus proches d'un Map : type de clé et valeur arbitraire

```
(declare-const a1 (Array Int Int))

(declare-datatypes () ((Color Red Green Blue)))
(declare-const arr (Array Int Color))
```

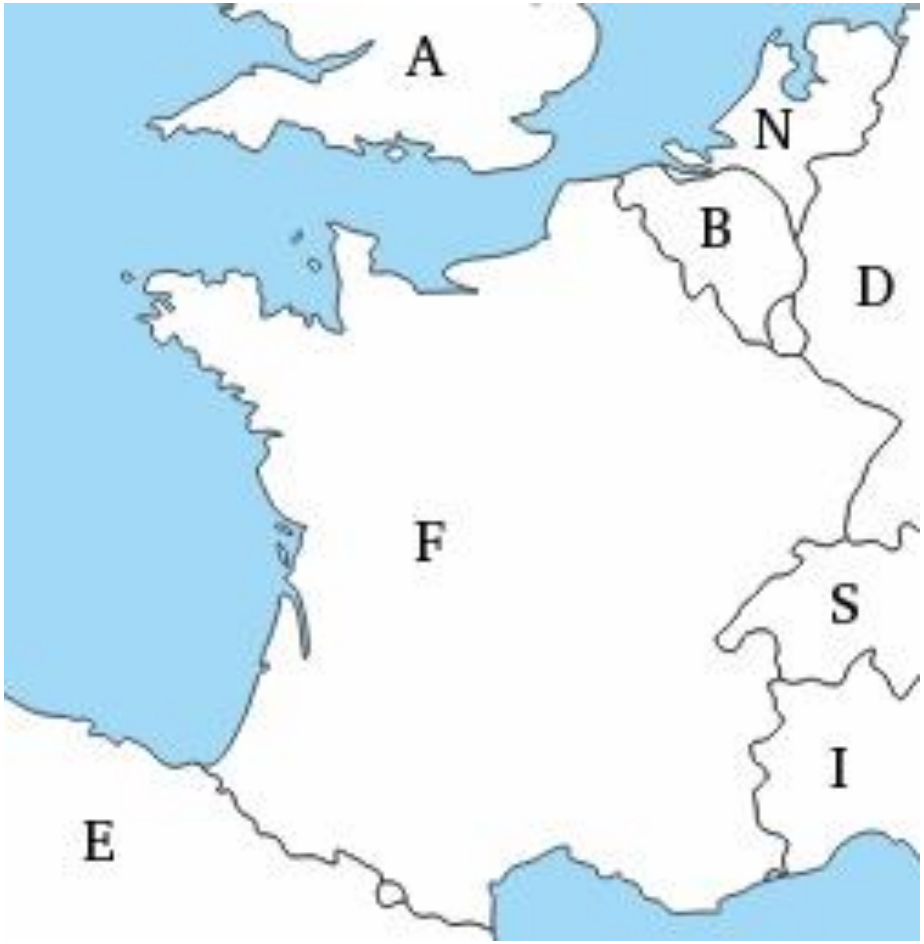
- On dispose de deux operations sur les Array :

```
;Comparaison d'égalité
(declare-const a1 (Array Int Int))
(declare-const a2 (Array Int Int))
(assert (= a1 a2))
;Selection d'une case :
(assert(= (select a1 3) (select a2 2)))
(assert(not (= (select a1 3) (select a2 3))))
```

Fonctions

```
; if then else
(assert (= y (ite (> x 0) x 0)))
;define the max function
(define-fun max ((x Int) (y Int)) Int (ite (< x y) y x))
; use it !
(assert (< z (max x y))
```

Exercice : Les couleurs



- On considère la carte suivante, que l'on souhaite colorier à l'aide de seulement trois couleurs (Rouge, Vert, ou Bleu), mais de manière à ce que deux pays qui partagent une frontière terrestre n'aient jamais la même couleur.
- Et si on ajoute le Luxembourg (entre B,D et F) ?
- (Bonus Dessinez une carte UNSAT avec 4 couleurs)

Exercice

On considère le problème suivant.

Chuck est un pilote d'avion qui doit faire 6 trajets dans 6 villes différentes (**A**lbany, **B**irmingham, **C**incinatti, **D**etroit, **E**l Paso, **F**argo) pour 6 clients différents (**U**nderwood, **V**anderkook, **W**ood, **X**ing, **Y**oung, **Z**ellman) qui se rendent en ville pour chacun une raison différente (**H**alloween, **I**dylle, **J**eu, **K**art, **L**oisir, **M**arriage).

Chuck a oublié qui est allé où, et pour quelle raison. On souhaite reconstruire cette information. On donne les indices suivants :

1. **E**l Paso (où Chuck a amené soit le voyageur "**I**dylle" soit le voyageur "**M**arriage") n'est pas l'endroit où **V**anderkook est allé.
2. **B**irmingham est là où Chuck a amené soit la personne fêtant "**I**dylle" soit **W**ood, mais pas les deux.
3. Chuck a amené à **A**lbany la personne qui allait aux "**L**oisirs" ou alors **V**anderkook, mais pas les deux.
4. **Y**oung est allé faire du **K**arting.
5. **U**nderdown est allé soit à **C**incinatti soit à **A**lbany (mais pas aux deux)
6. Ni **X**ing ni **V**anderkook ne sont allés à **F**argo. **F**argo c'était pour une "**I**dylle" ou un "**M**arriage".
7. A **D**etroit il y avait soit "**H**alloween" soit des "**L**oisirs".

Exercice : Les 8 reines

- On souhaite poser 8 reines sans vis-à-vis sur un échiquier 8x8
- Les reines attaquent les cases en diagonale, la rangée et la colonne

Exercice : ordonnancement

•In this problem, we need to schedule tasks on different work stations, with some constraints. The tasks are part of a job - say building a bike.
- each task in a job must start only after the previous task has been completed.
- a task cannot be paused - the time it takes to complete cannot be divided. - the work stations can only work on one task.

•3 job shop

We have three different jobs, each one consisting of three different tasks. Each task is assigned to a machine and has a predefined duration.
We have three jobs `jo0, job1, job2` containing task. We define the problem parameters in the following lists, where each element of the list is a pair `(m,d)` where `m` represents the machine where the task has to be executed and `d` is the duration of the task:

```
job0 = [(0,1), (1,2), (2,2)];
job1 = [(0,2), (2,1), (1,3)];
job2 = [(1,3),(2,3)];
```

•*The jobs are a list of tasks, and each task is a pair where the first element represents the machine number that can execute the task and the second is the duration of the task.*

Enigme « d'Einstein »

Albert Einstein's enigma

Five men live in five houses of five different color.
They smoke five different brands of cigar,drink five different beverages,and keep five different pets.

We know that:
*** The Norwegian lives in the first house.**
*** The brit lives in the red house.**
*** The Swede keeps dogs as pets.**
*** The Dane drinks tea.**
*** The green house is just on the left of the white house.**
*** The green house owner drinks coffee.**
*** The man who smokes Blend lives next to the one who keeps cats.**
*** The person who smokes Pall Mall rears birds.**
*** The owner of the yellow house smokes Dunhill.**
*** The man living in the house right in the center drinks milk.**
*** The German smokes Prince.**
*** The man who smokes Blend has a neighbor who drinks water.**
*** The Norwegian lives next to the blue house.**
*** The man who keeps horses lives next to the man who smokes Dunhill.**
*** The owner who smokes Blue Master drinks beer.**

The question is ... Who keeps a fish?
For Einstein 98% of the people in the world cannot solve this problem.