La Vérification Formelle

Basics

Stephane Maag

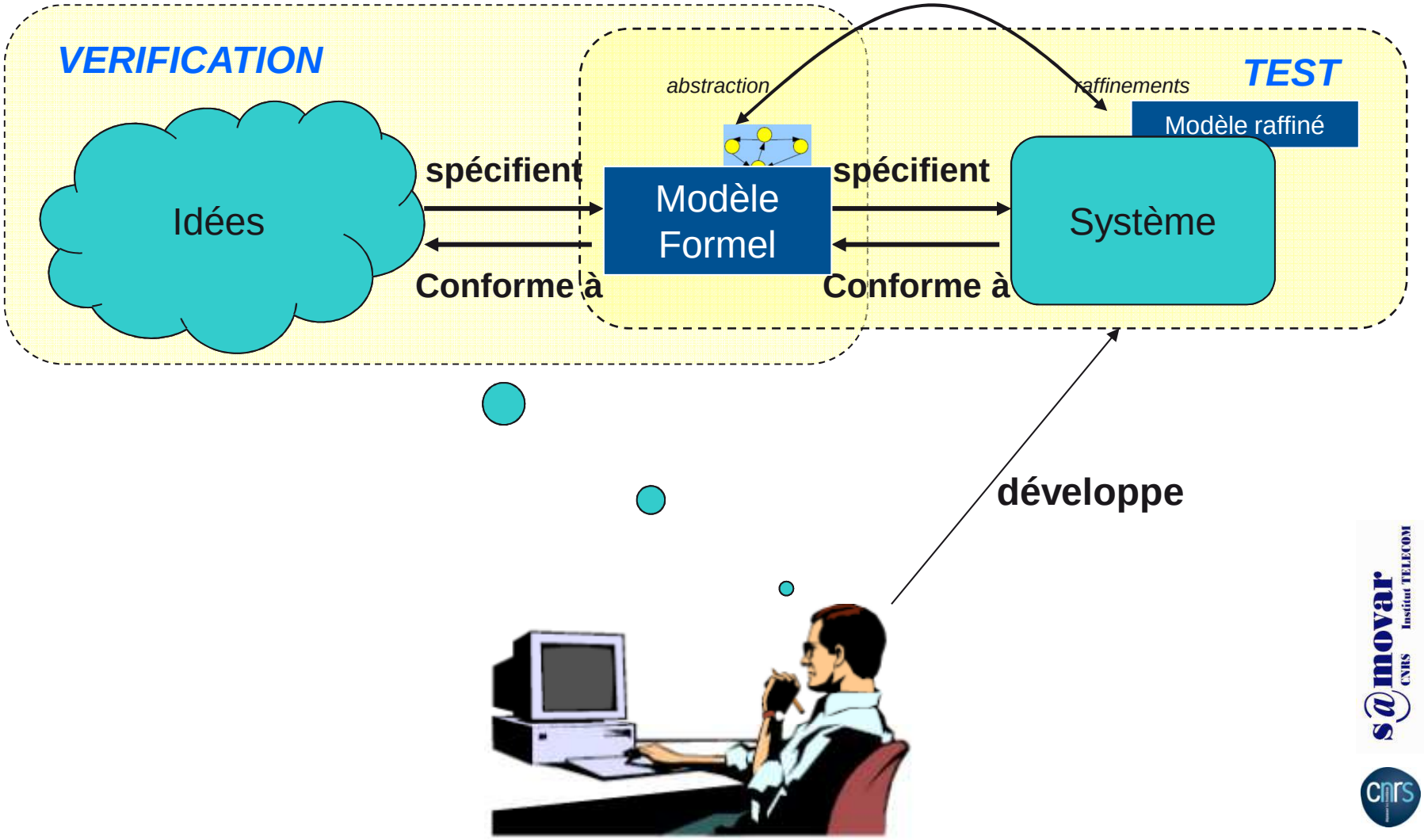
CNRS Samovar

Stephane.Maag@it-sudparis.eu

Content

- 3 Techniques de Verification
  - Equivalence de modèles
  - Preuves
  - Model-checking
- CTL
- *Sat* / BDD

Rappel


VERIFICATION
TEST
abstraction
raffinements
Modèle raffiné
Idées
spécifient
Modèle Formel
spécifient
Système
Conforme à
Conforme à
développe

Techniques de Vérification formelle

- **3 techniques majeures**

  - **Vérification du code**
    - Analyse statique – pas de modèles formels
    - Reverse engineering
    - BLAST, SLAM : pour les prog. C
    - Bandera: JAVA
    - Verisoft: C++

  - 3 types de Vérification des modèles
    - Equivalences de modèles
    - Méthodes déductives (preuve)
    - Model checking

Equivalence de modèles

- **Equivalence de modèles** (Equivalence checking)
  - **Comparaison de 2 spécifications : comparaison de modèles**
  - **Prouver que le comportement d'un système est équivalent à un comportement donné**
  - **Vérification complète, mais en pratique non réalisable sur des gros modèles**

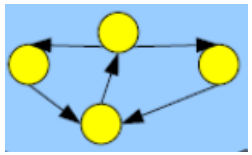
Techniques de « preuve »

- **Méthodes déductives** (Theorem proving)
  - **Prouver mathématiquement qu'une propriété extraite du cahier des charges est vérifiée dans le modèle**
    - Exple: timer dans un datagramme TCP respecté dans la spécification
    - Outil COQ (INRIA) – « theorem prover » - inférence
    - HOL (Higher Order Logic – *Austrialian National University*) – Meta-Language (ML)
    - Essentiellement pour les propriétés qualitatives/fonctionnelles
    - Vérification sur des systèmes à états infinis

Techniques de model-checking

- **Model checking**
  - **Idée: *trouver, dans un modèle formel, le contre exemple d'une propriété définie à l'aide d'un langage logique*.**
  - **Modèles Markoviens ou quantative-bound LTS**
  - **Beaucoup plus répandu – utilisé dans de nombreux domaines**
  - **De nombreux outils**
  - **Prop. qualitative/quantitative – adapté à la QoS**
  - **Outils: SPIN, PRISM, UPPAAL, etc.**
  - **Intérêt industriel certain car aide à la modélisation et rapide obtention des erreurs**

Principes de base du model-checking

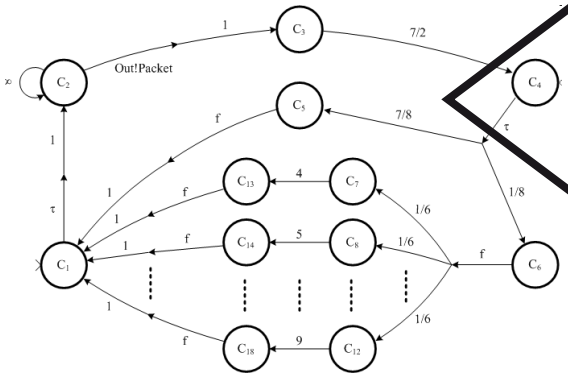


Modèle formel

Un Model Checker est un outil qui vérifie si un ***Modèle*** satisfait une ***Propriété***

Compilateur, simulateur

Propriété



Système de transitions étiquetés

model-checker

Diagnostic

(Vrai / Faux + contre exemple)

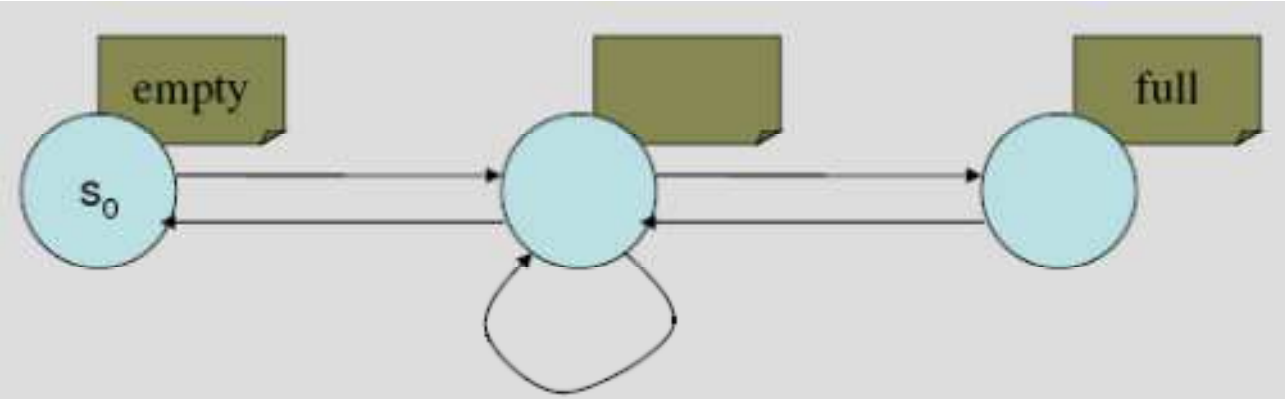
Modèle: un terme plein de bon sens !

- Here models – as they are used for model-checking are just *annotated graphs:*
  - A finite set of states, S
  - Some initial state s_0
  - A transition relation between states, $T \subseteq S \times S$
  - A finite set of atomic propositions, AP
  - A labelling function $L : S \rightarrow P(AP)$

- known as a Kripke structure:
  - Labelled Transition systems, LTS
  - Finite State machines, FSM
  - State charts, …

** For a physicist a "model" is a differential equation; For a biologist, it may be … mice or frogs*

An Example



AP = {empty, full}

Some LTL formula that are valid for this model:

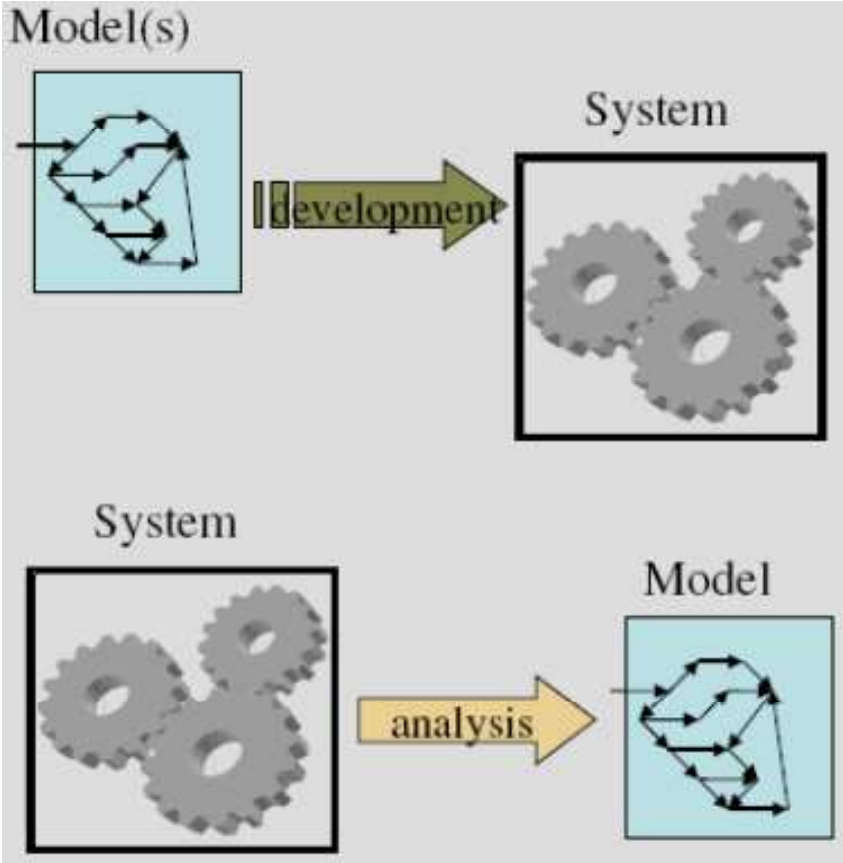
empty $\Rightarrow$ (X ¬empty)

full $\Rightarrow$ (X ¬full)

(X is for neXt)

What are models good for?

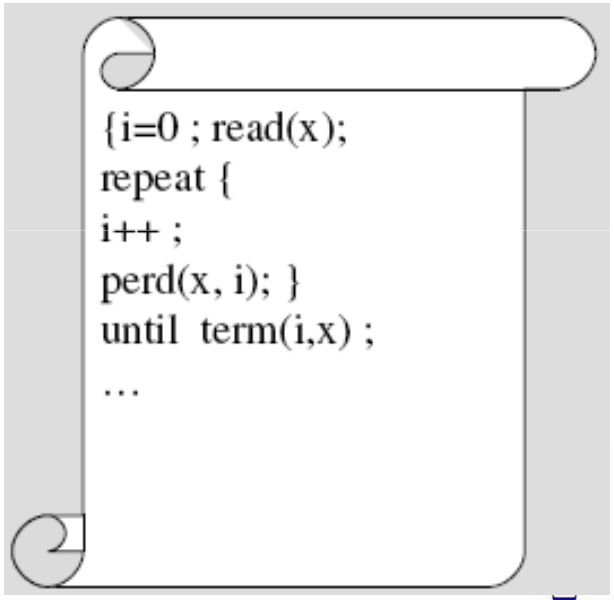
- System description and design:
  - The future system must conform to the model(s)
  - The model(s) may be used as a starting point for (automatic) development
- System analysis
  - Observing the existing system, one extracts a model and studies it
- …
- Essential role in V and V and quality assurance



Programs

- Everybody knows what it is ☺

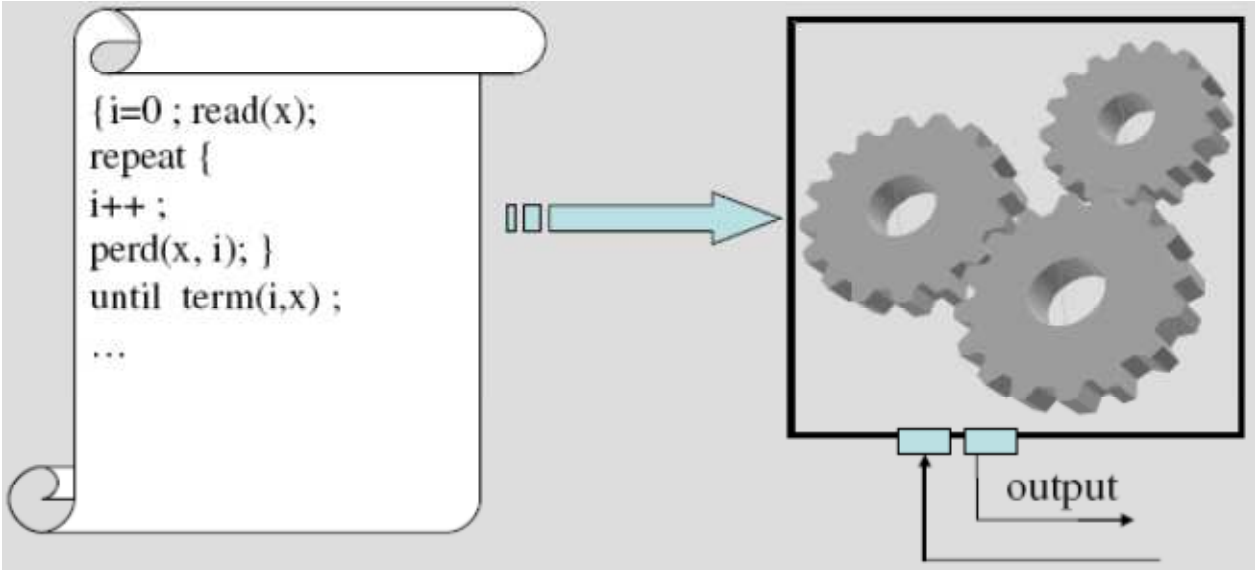
- Here:
  - A program is a piece of text in a (hopefully) well defined language
  - There is a syntax, some semantics, and compilers



- *"A program is a very detailed solution to a much more abstract problem"* [Ball2005]

Why are programs useful?

- They can be compiled and embedded into some systems.

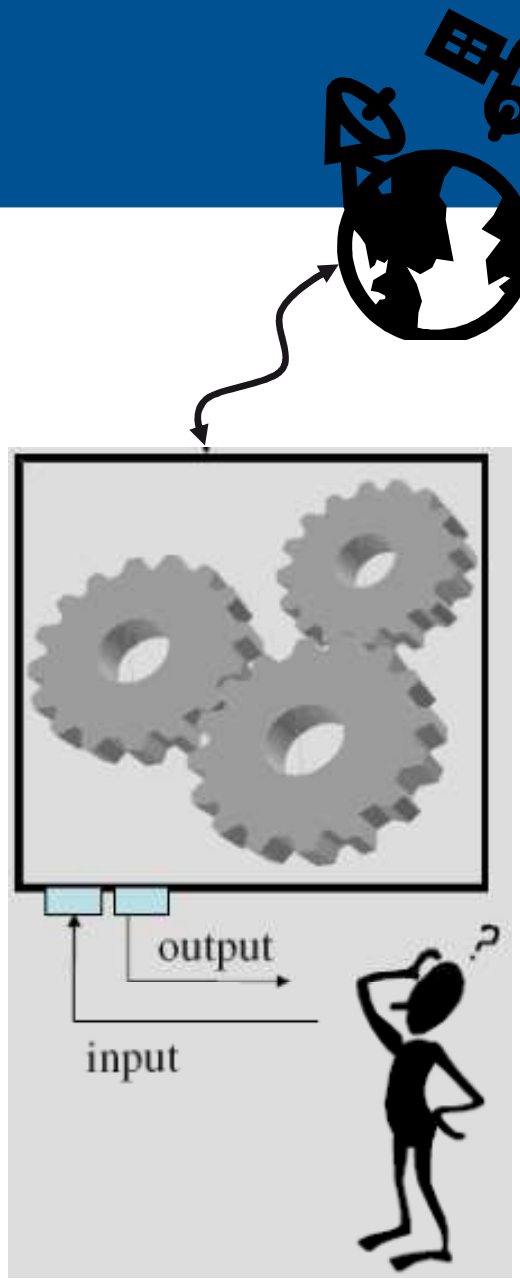


Interlude

- **A program text or a specification text is NOT the system !**

Systems

- A system is a dynamic entity, embedded in the physical world
- It is observable via some limited interface/procedure
- It is not always controllable
- Quite different from a piece of text (formula, program) or a diagram



Systems are the actual objects of interest

- **How to ensure that a system satisfies certain properties?**

- Properties?

1. Texts in natural languages…

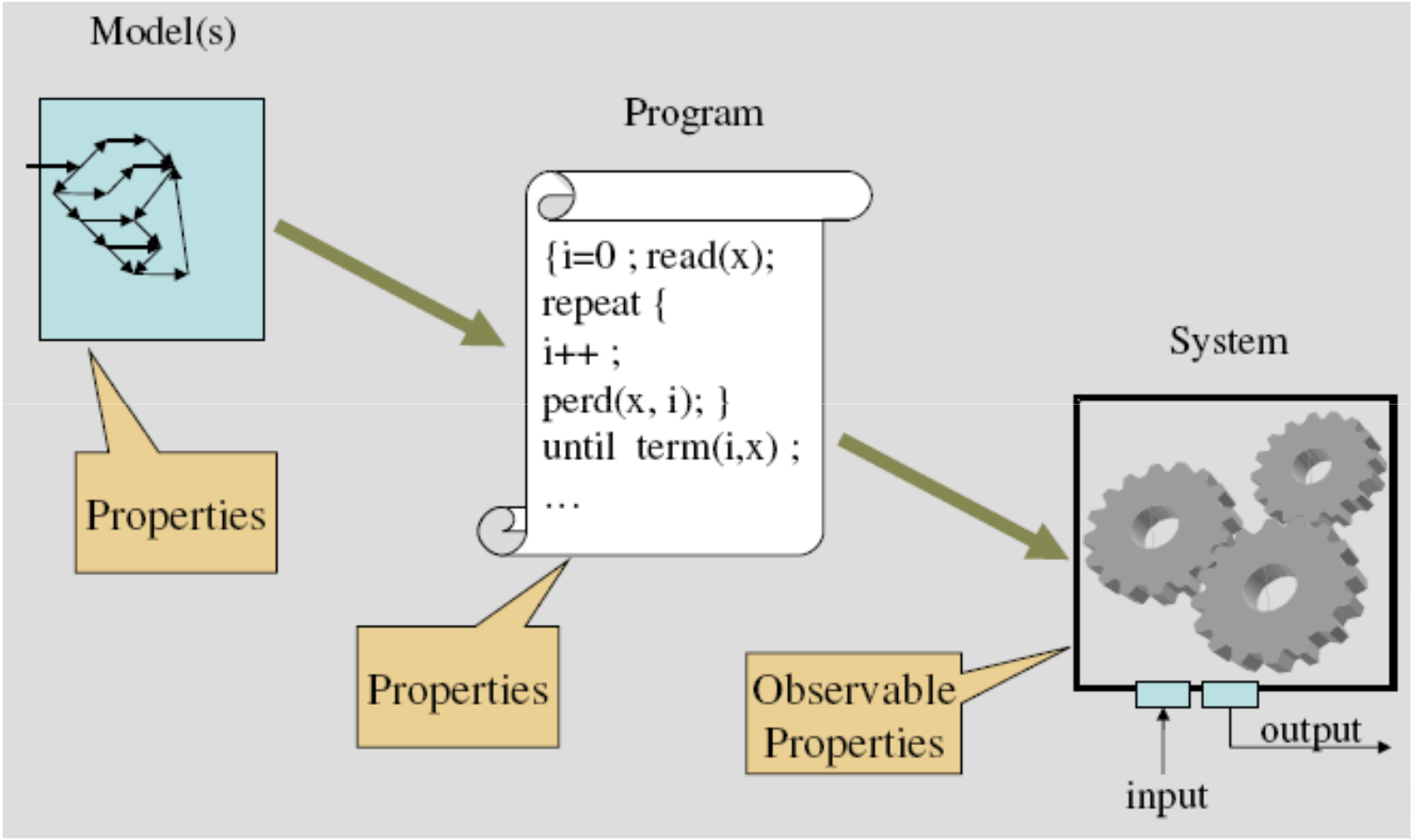
   “Calls to lock **and** unlock **must** alternate**.”**

2. Formulas in a given specification logic

   (locked ⇒ X unlocked) ∧ (unlocked ⇒ X locked)

3. Sets of mandatory or forbidden behaviours

The classical process



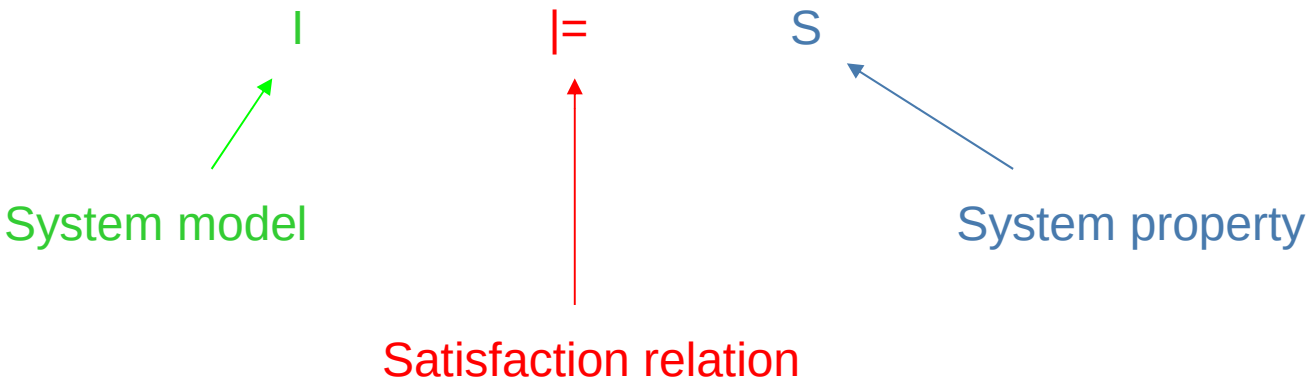
Properties…, specification languages…

- Logic-based specification languages
  - VDM, Z, CASL, HOL, B, JML, …
  - Temporal Logics: LTL, CTL, …
- Behaviour-based specification languages
  - Lotos, Promela, CSP, State charts, Petri Nets, Timed automata…
- Usages
  - Global requirement on the system as a whole, or of some subsystems
  - Assertions in programs and models: pre-conditions, post-conditions, invariants.

Types de propriétés fonctionnelles

Atteignabilité	**Un état du système peut être atteint**
	Le train peut traverser le passage à niveau
Vivacité	**Sous certaines conditions, un évènement finira par se produire**
	Lorsque le train a annoncé son arrivée, la barrière finira par s'ouvrir
Sûreté	**Un évènement indésirable ne se produira jamais**
	Il est impossible que la barrière soit ouverte et le train soit au niveau de la barrière.
Absence de blocage	**Le système ne se trouvera jamais dans une situation où il ne peut plus évoluer**
	Lorsque la barrière est fermée, elle peut toujours se réouvrir
Équité	**Un évènement se produira infiniment souvent**
	La barrière sera ouverte infiniment souvent

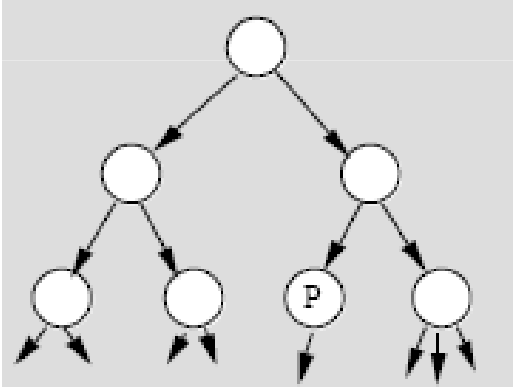
Model-checking problem



The CTL Logic
Computation Tree Logic

- CTL allows to reason on computation tree

Examples



There exists a path with a state in which P holds

EF P

Temporal operators on an execution : X, F, G, U

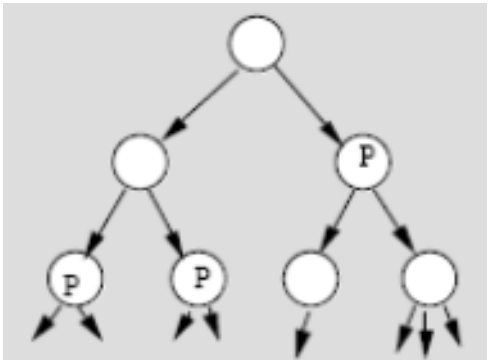
- X φ : the next state satisfies φ (neXt)
- F φ : there exists a state in the future which satisfies φ (Future)
- G φ : all the states satisfy φ (Global)

  G φ (= ¬F ¬φ)

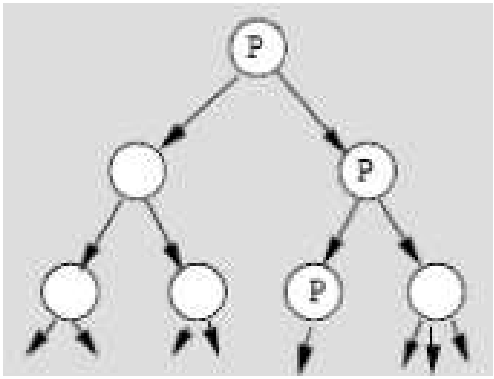
- φ U Ψ : a state in which Ψ holds and up to this state φ holds true (Until)

  F Ψ ⇔ true U Ψ

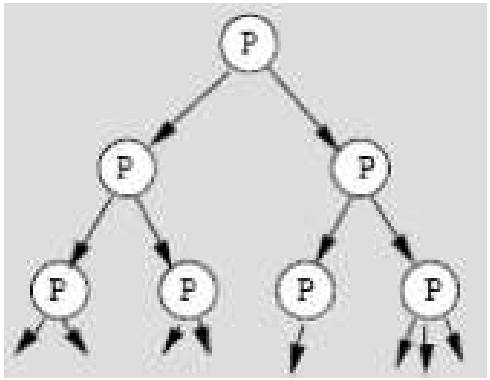
EXAMPLES



On each path there exists a state in which P holds true
AF P (= ¬E¬F P)

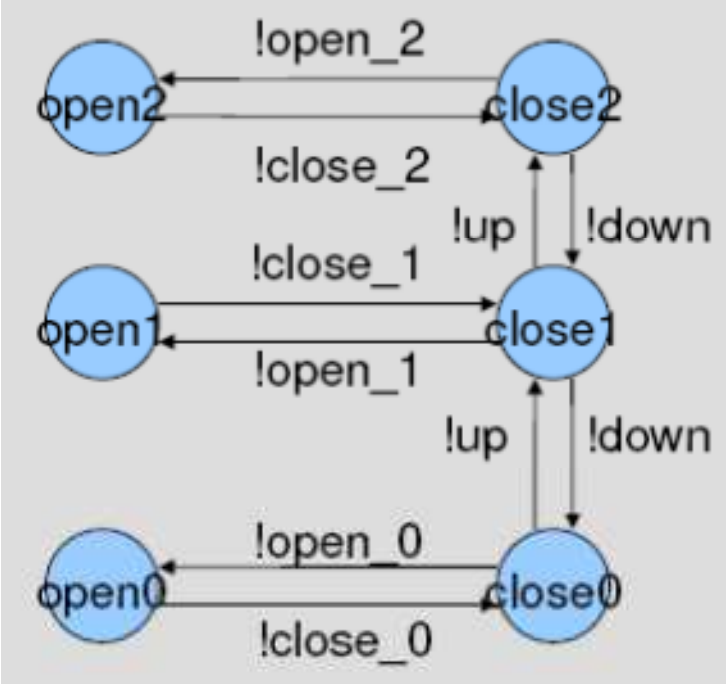


There exists an infinite path on which P holds in each state
EG P (= E ¬F ¬P)



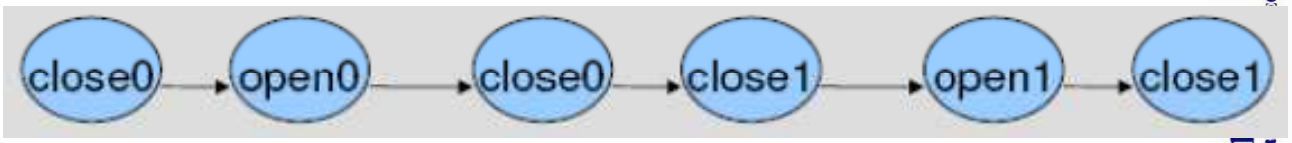
In all reachable states, P holds true
AG P (= ¬EF ¬P)

The temporal operators are of two types:
- on an execution (a path) – E
- on all executions (all paths) – A



Formulas associated to the states of the automaton
L(openi) = {open, level = i}, i=0,1,2
L(closei) = {¬open , level = i } i=0,1,2

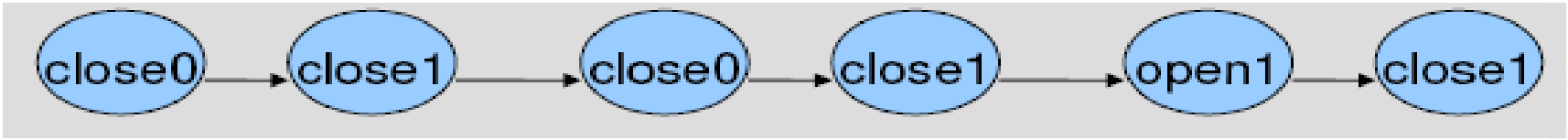
an execution of the automaton



s,0 |= X open s,0 |= F ¬close
s,2 |= X ¬open ∧ X level = 1
s,i |= G F ¬open i = 0,…, 5

Notation: s|= P ⇔ s,0 |= P

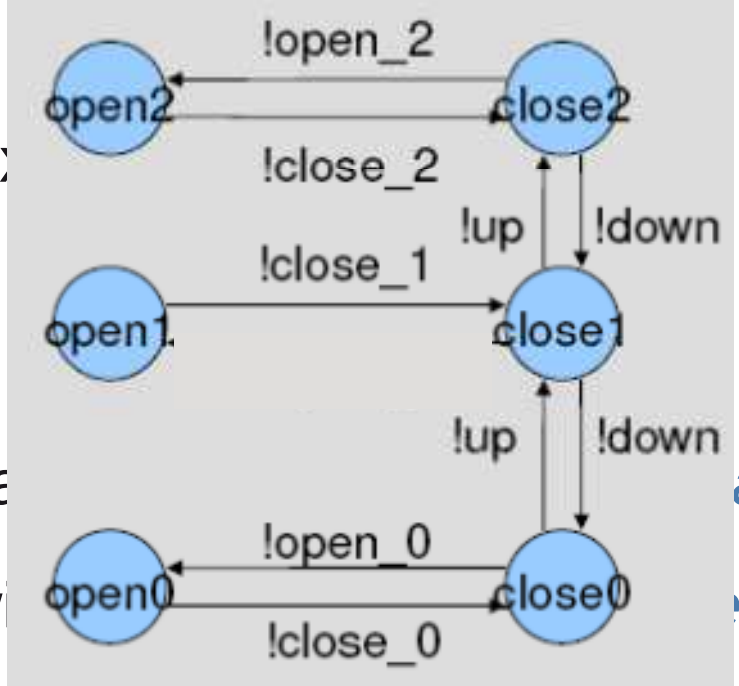
s |= G ¬open



s |= ¬open U level = 1

Temporal operators on all executions : A, E

- A φ : all the executions starting from the current state satisfy φ
- E φ : there ex… from the current state φ
- E F φ : we ca… *afety property*
- A F φ : we wi… *eness property*



s,3 |= A X ¬open

Notation : A |= φ iff s,0 |= φ where φ contains A or E

Example

Asc the controller of the lift :

$$Asc \models E\ G\ \neg open$$

$$Asc \models AG\ (open \Rightarrow AX\ \neg open)$$

$$Asc \models AG\ (\neg open \Rightarrow EX\ open)$$

Precise definition of CTL

- Syntactical restrictions:
  - Each temporal operator X, F, G, U have to be on immediate scope of a A or E , the combinations are:
    - **AX,AF, AG, AU, EX, EF, EG, EU**

- Syntax: atomic propositions are CTL formulas
  - if f and g are CTL formulas, then

    ¬f, f ∧ g, AX f, EX f, A(fUg), E(fUg) are also CTL formulas

- Extensions :
  - $f \vee g = \neg(\neg f \wedge \neg g)$
  - AF g = A(true U g) EF g = E(true U g)
  - AG f = ¬E(true U ¬f) EG f = ¬A(true U ¬f)

Semantic of CTL

- s |= f (f atomic) **iff** f ∈ L(s)
- s |= ¬f **iff** s |≠ f
- s |= f ∧ g **iff** s|=f and s|=g
- s,0 |= AX f **iff** for all s such that s_0 = s,0, s,1|= f
- s,0 |= EX f **iff** it exists a s such that s_0 = s,0 and s,1|= f
- s,0 |= A (f U g) **iff** for all s s.t. s_0 = s,0, it exists i≥0 s.t. s,i |= g and
  for all j<i, s,j |= f
- s,0 |= E (f U g) **iff**
  it exists a s s.t. s_0 = s,0 and
  it exists i≥0 s.t. s,i |= g and
  for all j<i, s,j |= f

Algorithme CTL

- **Principe:**
  - **On dénote A (structure Kripke) et φ une formule CTL**
  - **On marque chaque état q de A et chaque sous formule Ψ de φ si q |= Ψ**
    - On construit q. φ à partir de q. Ψ
    - A |= φ iff q0. φ = vrai
  - **Le nombre d'états doit être fini**

Algorithme CTL (1)

procedure marking(phi, A)

- **cas 1** : φ = f *(*atomic*)*
  for all q **in** A.Q **do**
  if f ∈ L(q) **then** q. φ := true
  else q. φ := false

- **cas 2** : φ = ¬ Ψ
  marking (Ψ, A);
  for all q **in** A.Q **do** q. φ := not(q. Ψ)

- **cas 3** : φ = Ψ1 ∧ Ψ2
  marking (Ψ1, A); marking (Ψ2, A);
  for all q **in** A.Q **do**
  q. φ := and (q.Ψ1, q.Ψ2)

- **cas 4** : φ = EX Ψ
  marking(Ψ, A);
  for all q **in** A.Q **do** q. φ := false;
  for all (q, q') **in** A.T **do**
  if q'. Ψ **then** q. φ := true

- **cas 5**: φ = AX Ψ *(* as ¬EX¬ Ψ *)*

- **cas 6** : φ = E Ψ1 U Ψ2
  marking (Ψ1, A); marking (Ψ2, A);
  *(*initialisations : *)*
  for all q **in** A.Q **do**
  q. φ := false; q.dejavu := false;
  (at the beginning LL = {q | q |= Ψ2} : *)*
  LL := { } ;
  for all q **in** A.Q **do**
  if q. Ψ2 **then** LL := LL + {q} ;
  for all q **in** LL **do**
  LL := LL {q} ; q. φ := true;
  for all (q', q) **in** A.T **do**
  if q'.dejavu = false **then**
  q'.dejavu := true;
  if q'. Ψ1 **then** LL := LL + {q'} ;

- **cas 7** : phi = A Ψ1 U Ψ2

(+complex, same principle*)*

Cons and pro of CTL

☺ Model checking of linear complexity

☹ difficulties or unwillingness to express some kinds of properties (but they are advanced techniques resolving that issue!)

Other temporal logics:

CTL*, PLTL (PSPACE complet), FCTL (*Fairness*), TCTL (*Timers),* Logiques avec *passé:* pas de model-checkers.

Problème !!

- **Le nombre d'états d'un système est exponentiel dans son nombre de variables**

⇒ Les algorithmes naïfs des model-checkers ne suffisent plus

⇒ Comment éviter, ou du moins restreindre l'effet négatif de l'explosion combinatoire?

Model-checking Symbolique: ens. d'états, BDD

- **Il existe plusieurs autres techniques pour gérer un nombre colossal d'états: les explorations on-the-fly, abstractions;**
- **En combinant toutes ces techniques, nous savons gérer de gros automates:**
  - **En 1992, 10^{20} états ont été analysés, en 2012: 10^{50}**

    A = < Q, T, L >
  - **Notations:**

    Sat (Φ) = ensemble d'états satisfaisant Φ

    $S \subseteq Q$, Pre (S) = ens. des prédécesseurs immédiats de S

Obtention de Sat(Φ), Φ∈CTL (1)

$$Sat(\neg\Psi) = Q \backslash Sat(\Psi)$$

$$Sat(\Psi \wedge \Psi') = Sat(\Psi) \cap Sat(\Psi')$$

$$Sat(EX\ \Psi) = Pre(Sat(\Psi))$$

$$Sat(AX\ \Psi) = Q \backslash Pre(Q \backslash Sat(\Psi))$$

$$Sat(EF\ \Psi) = Pre^*(Sat(\Psi))$$

Obtention itératives d'états

Calcul de Pre*(S)

```
X := S ; Y := { } ;
while Y ≠ X do (*computation of the fix point*)
Y := X ; X := X + Pre(X) ;
(* + is the union*)
return(X) ;
```

Obtention de Sat(Φ), Φ∈ CTL (2)

Cas de A Ψ1 U Ψ2 - définition récursive:

Ψ2 ∨ (Ψ1 ∧ EX true ∧ AX(A Ψ1 U Ψ2)

P1 := Sat [Ψ1] ; P2 := Sat [Ψ2] ;

X := P2 ; Y := { } ;

while Y ≠ X **do**

Y := X ; X := X + (P1 /\ pre(Q) /\ (Q\Pre(Q\X)))

return(X) ;

Ce qui est recherché

- **Une implémentation efficace**
  - **Pour représenter l'ens. Sat(f) avec f atomic,**
  - **Pour calculer Pre(S) à partir de la représentation de S,**
  - **Pour calculer le complémentaire, l'union et l'intersection,**
  - **Pour l'égaliter de 2 ensembles.**

BDD - Binary Decision Diagrams

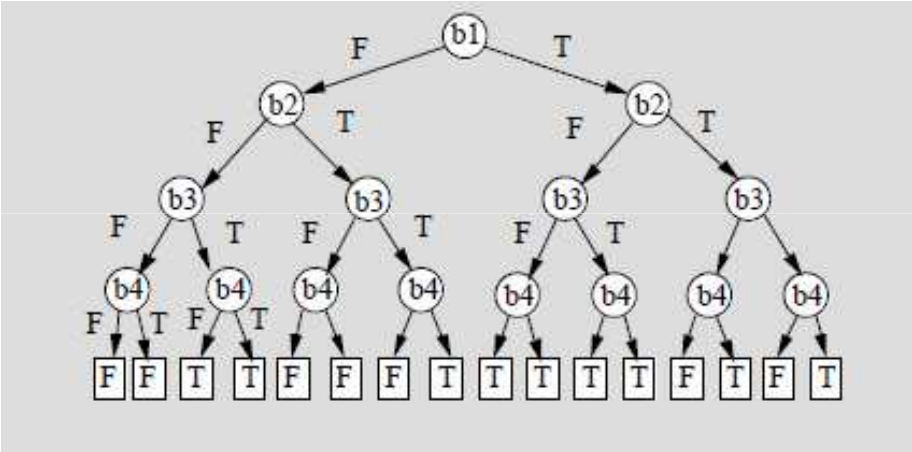
- **x1, … , xn: variables booléennes**
- **<b1, … , bn>: vecteur de booléens**
- ***Comment représenter l'ens. de vecteurs tq***

  Φ (x1,…,xn) soit vraie ?

Solution classique: arbre de décision

BDD

- **Exemple:** (x1∨ x3) ∧ (x2 ⇒ x4)



Avantage: Test en *n* comparaisons

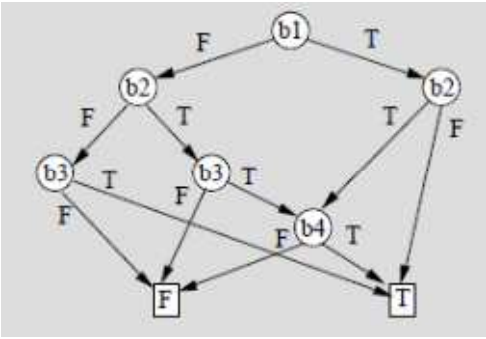
Inconvénient: taille exponentielle

BDD

- **BDD = arbre de décision réduit**

1. **Les sous arbres étant les mêmes sont partagés**
2. **Le choix inutiles sont omis**

Ex.: si b1 et b2 sont vrais, b3 est inutile



Opérations sur les BDD

- **Ensemble vide, seulement une feuille F**
- **Comparaison de 2 ens.: même BDD**
- **Complémentaire: on remplace les feuilles T par les feuilles F et réciproquement**
- **Union et Intersection: ……., non … complexité quadratique …. stop !**

BDD pour représenter un automate

- **Principe: coder les états et transitions par des n-tuples de booléens.**

Ex.:

Les états:

6 états q0, … , q5, une variable booléen *open*, une variable *level* qui peut prendre les valeurs 0, 1, 2 et ND.

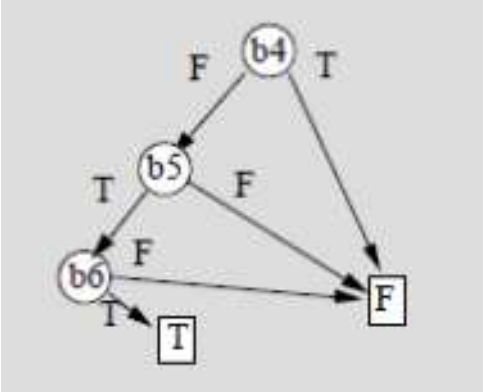
3 bits pour q, 1 pour *open* et 2 pour *level*.

FFT T FF

« en q1, *open* est vraie et *level* vaut 0 »

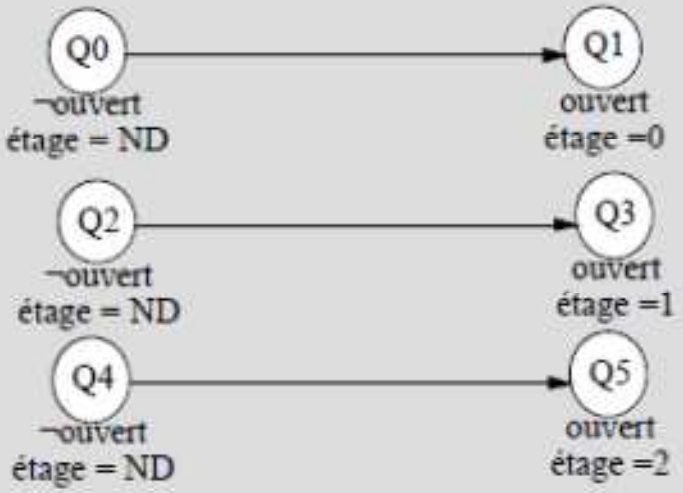
$\Rightarrow \neg$ *open* $\wedge$ *level* = ND

- **Les transitions:**



- **Il peut y avoir des ens. de couples d'états**

- **Si les états sont codés par *n* bits**
  - **Vecteur de *n* *n* bits où on « priorise » le dernier *n***



$<b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ \ b_1\ b_2\ b_3\ b_4\ b_5\ b_6>$

<FFF F TT FFT T FF>

<FTF F TT, FTT T FT>

<FTT F TT, TFT T TF>

Calcul de Pre(S) (1)

- Soit des BDD_T ou BDD_S
- On construit BDD_S où chaque bi des BDD_S devient b'i (complexité: O(n))
- On construit $BDD'_S \cap BDD_S$
  - ens. de couples <s,s'> de T tq s'$\in$S (complexité $O(n^2)$)

Calcul de Pre(S) (2)

- **On « abstrait » / b'i (i.e. on l'oublie)**

(complexité $O(n^2)$)

Nous avons maintenant tous les éléments pour implémenter le model-checker symbolique de CTL.

Problème: la complexité en mémoire au pire des cas est exponentielle

$\Rightarrow$ les performances sont dépendantes de l'ordre des variables

Few Model-Checkers

- SPIN (Promela, *LTL*)
- NuSMV 2 (*CTL*) combines BDD-based model checking with SAT-based model checking.
- FDR (CSP, *refinements*)
- *Timed automata*: UPPAAL, KRONOS
- *Stochastic models*: PRISM, APMC

REFERENCES

- **Deux livres:**

  ***A Roadmap for Formal Property Verification,* Pallab Dasgupta, Springer-Verlag New York Inc., 2006**

  ***Applied Formal Verification,* Douglas L. Perry et Harry Foster, McGraw-Hill Professional, 2005**