Génération automatique de tests de conformité: l'approche de TGV

Thierry Jéron
Irisa / Inria Rennes
http://www.irisa.fr/pampa

Plan

- **Introduction au test de conformité**
- **TGV: principes, architecture et algorithmes**

Le test de conformité

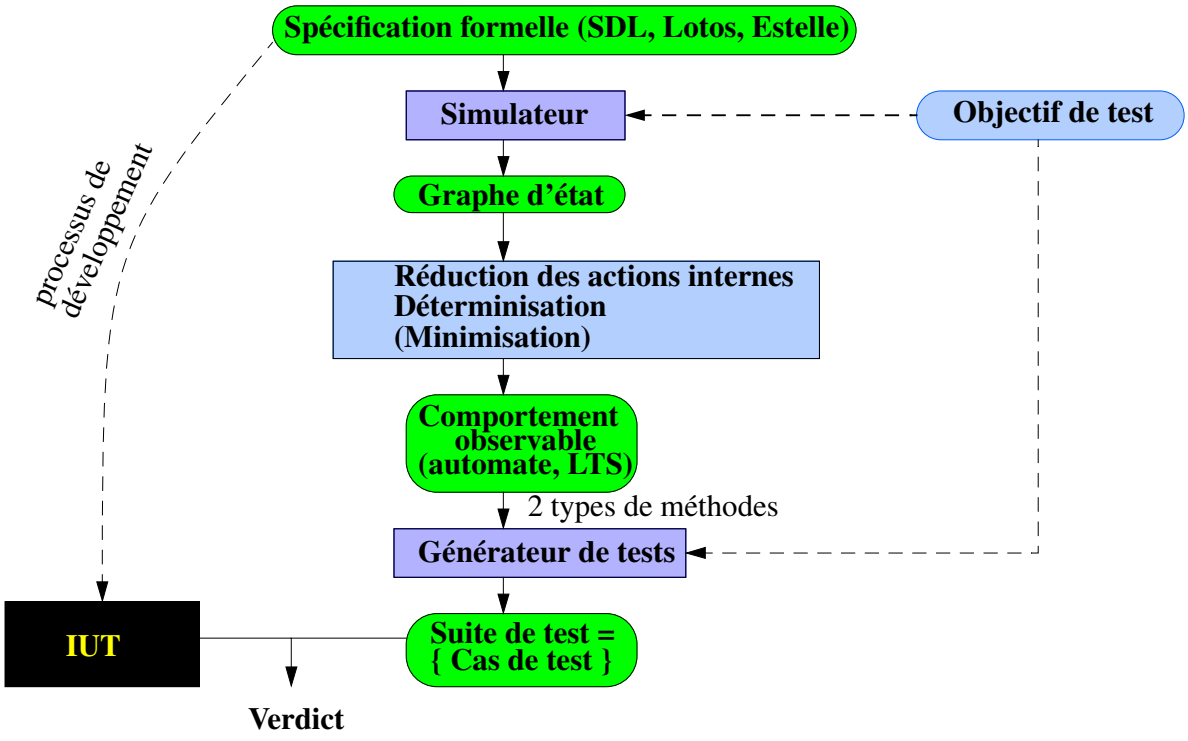
- **Domaine:** programmes réactifs (par ex. protocoles)
- **Problème:** vérifier qu'une implémentation sous test (IUT) est conforme à sa spécification.
- **Test boîte noire:** le code de l'IUT n'est pas connu, seulement ses interactions avec l'environnement => le testeur
- **Pratique du test:** alternance de stimuli et d'observations de l'IUT avec verdicts sur la (non-)conformité de l'IUT.
- **OSI Conformance Testing Methodology and Framework (ISO 9646):** standard définissant une méthodologie pour le test de protocoles ISO: concepts, spécification, exécution des tests
  Langage de description des tests: TTCN (Tree and Tabular Combined Notation).
  S'applique plus largement (ATM, ISDN)
  Informel: spécifications informelles, pas de définition formelle de la conformité, de l'exécution, des verdicts, etc.

Motivations pour la formalisation

- **Etat de l'art industriel:**
  - **la plupart des tests sont écrits à la main:**
    coût élevé, processus long et répétitif, sujet à erreur.
  - **ISO9646 n'apporte pas de solution idéale:**
    pas de définition formelle de la conformité => verdicts biaisés.
- **Apports de l'automatisation:**
  - **gains énormes en temps et en coût**
  - **qualité des tests**
- **Principe:**
  - **utiliser des spécifications formelles (SDL, Lotos, Estelle, ...)**
  - **formaliser le processus de test**
  - **concevoir des algorithmes de génération**

FMCT (Formal Methods in Conformance Testing Z 500) est un projet de standard répondant aux 2 premiers critères.

Méthodologie générale



Méthodes

	fondées sur les automates	fondées sur la théorie du test
modèles de spécification	machines de Mealy s —i/o→ s’	systèmes de transition (LTS ou IOLTS) s —i→ s’ —o→ s”
modèles d’IUT	machines de Mealy	LTS ou IOLTS
hypothéses sur spec.	complète/ i, déterministe, minimale, fortement connexe	aucune
Hypothéses sur l’IUT	complète/ i, déterministe, minimale ou nbre d’etat de la spec + k, fortement connexe	(complète en réception)
conformité	Modèle de faute: sortie et/ou transfert sortie: en s, i produit o’ transfert: en s, i produit o mais va en s”	Relation de conformité: par exple I ioconf S ssi après toute trace de S, les sorties de I sont incluses ds celles de S.
principe de génération	un test (appliquer **i,** vérifier **o**, vérifier **s’**) par transition de la spec enchaîné en une suite de longueur minimale.	un testeur canonique par spec, ou tests aléatoires ou guidés par un objectif
techniques	TT, DS, UIO, W, etc	
algorithmes	théorie des automates et flots ou prog linéaire	parcours de graphes, techniques de verif.
correction / non-biais	asurée pour le modèle de fautes	assurée pour la relation de conformité
exhaustivité	pour un modèle de faute et hypothèses	si équité bornée de l’IUT

Comparaison

- **Automates:**
  - la plupart des méthodes ne permettent pas de traiter de spécifications (et d’IUT) non-déterministes.
  - hypothèses souvent irréalistes sur l’IUT
  - complexité => limité à des spécifications de petites tailles.
- **LTS:**
  - à l’origine peu réalistes (testeur canonique de Brinksma) mais le devenant avec distinction entrée/sortie (Tretmans)
  - moins discriminants: pas de séquence d’identification d’état
  - prise en compte du non-déterminisme par la relation de conformité
  - algorithmes efficaces, en particulier à la volée.

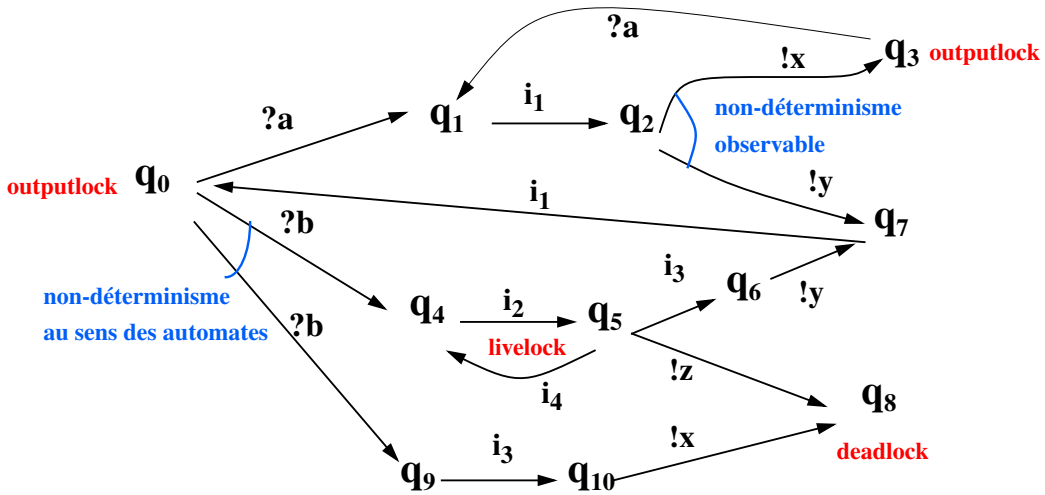
L'approche TGV

Observation de la pratique industrielle (tests manuels) ⇒

- **Théorie du test:** fondée sur les IOLTS.
- **Relation de conformité: ioco** cohérente avec l'intuition des tests manuels.
- **Sélection des tests:** utilisation d'objectifs de tests.
- **Algorithmique:** à la volée couvrant tout le processus de génération depuis la spécification (SDL, Lotos) jusqu'au test.
- **Qualité des tests:** limitation des verdicts inconclusifs.
- **Non-biais:** assuré par construction i.e. seules les IUT non conformes peuvent être rejetées par un test.
- **Exhaustivité:** théorique i.e. pour toute IUT non conforme, on peut générer un test qui peut, sous l'hypothèse d'équité bornée, rejeter l'IUT.
- **Prototypage:** TGV développé conjointement par IRISA et VERIMAG.
- **Transfert:** en cours dans l'environnement ObjectGéode (Vérilog).

Systèmes de transitions à entrées/sorties: IOLTS

$M = (Q^M, A^M, \rightarrow_M, q_0^M)$ avec $A^M = A_I^M \cup A_O^M \cup I^M$



Notations

Let $q, q', q_{(i)} \in Q^M$, $Q \subseteq Q^M$, $a_{(i)} \in A_I^M \cup A_O^M$, $\tau_{(i)} \in I^M$, and $\sigma \in (A_I^M \cup A_O^M)^*$.

- $q \stackrel{\epsilon}{\Longrightarrow}_M q' \equiv (q = q' \vee q \stackrel{\tau_1 \ldots \tau_n}{\rightarrow}_M q')$ and $q \stackrel{a}{\Longrightarrow}_M q' \equiv \exists q_1, q_2 : q \stackrel{\epsilon}{\Longrightarrow}_M q_1 \stackrel{a}{\rightarrow}_M q_2 \stackrel{\epsilon}{\Longrightarrow}_M q'$
- $q \stackrel{a_1 \ldots a_n}{\Longrightarrow}_M q' \equiv \exists q_0, \ldots q_n : q = q_0 \stackrel{a_1}{\Longrightarrow}_M q_1 \ldots \stackrel{a_n}{\Longrightarrow}_M q_n = q'$.
- $Trace_M(q) = \{\sigma | q \stackrel{\sigma}{\Longrightarrow}_M\}$ and $Trace_M(M) = Trace_M(q_0^M)$.
- q **after**$_M$ $\sigma = \{q' | q \stackrel{\sigma}{\Longrightarrow}_M q'\}$ and Q **after**$_M$ $\sigma = \cup_{q \in Q} q$ **after**$_M$ σ.
- $Out_M(q) = \{a \in A_O^M | q \stackrel{a}{\Longrightarrow}_M\}$ and $Out_M(Q) = \{Out_M(q) | q \in Q\}$.

Relation de conformité

- **Spécification: S= (Q^S, A^S, $\rightarrow_S$, q_0^S) avec $A^S = A_I^S \cup A_O^S \cup I^S$**
- **Implémentation: I= (Q^I, A^I, $\rightarrow_I$, q_0^I) avec $A^I = A_I^I \cup A_O^I \cup I^S$**
  et $A_I^I \supseteq A_I^S$, $A_O^I \supseteq A_O^S$

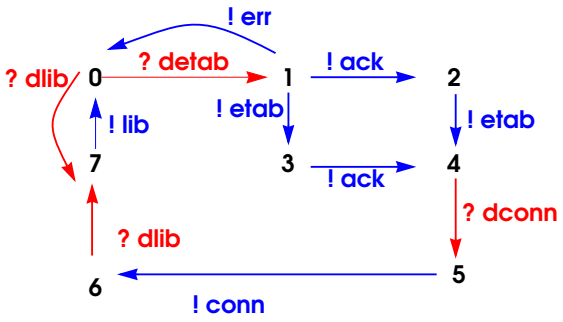
  On suppose que I ne peut refuser aucune entrée
- **Relation de conformité:**

  Evénements observables: interactions avec l'environnement et blocages.
  Blocages supposés détectés par des temporisateurs.
  Blocages modélisés dans **S** par une action δ considérée comme une sortie.
  S^δ est obtenu en ajoutant une boucle delta à chaque état de blocage de **S**.
  Informellement: I est conforme à **S** si après toute trace de S^δ,
  les sorties de **I** sont incluses dans celles de S^δ
  en particulier **I** ne peut se bloquer que si **S** l'autorise.

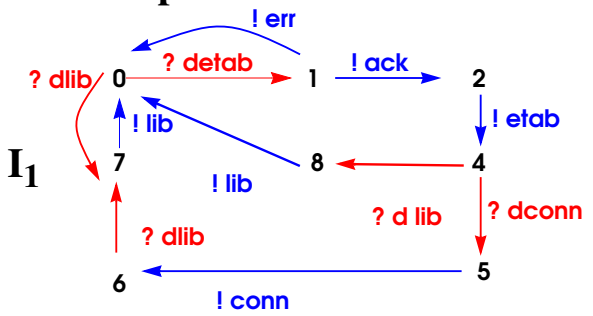
I ioco S $\Leftrightarrow \forall \sigma \in$ Traces(S^δ), Out(I after σ) $\subseteq$ Out(S^δ after σ)

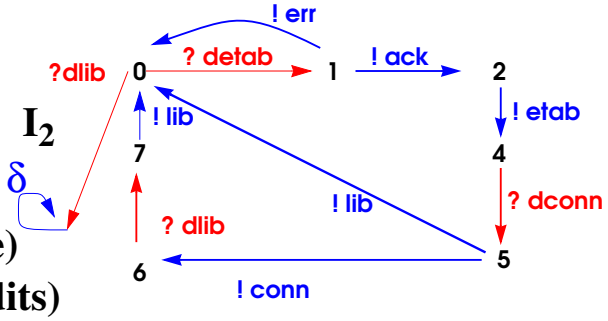
Conformité: exemple

Spécification S



Implantations





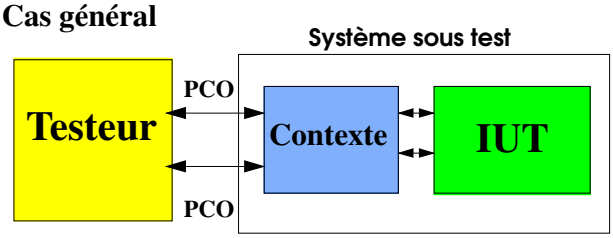
I ioco S
ssi
$\forall \sigma \in traces(S^{\delta}), Out(I \text{ after } \sigma) \subseteq Out(S^{\delta} \text{ after } \sigma)$

I_1 *ioco* S (choix d'implantation, spécification partielle)

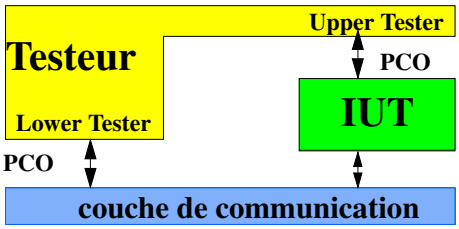
I_2 *not ioco* S (sortie et blocage interdits)

Test dans un contexte

Cas général



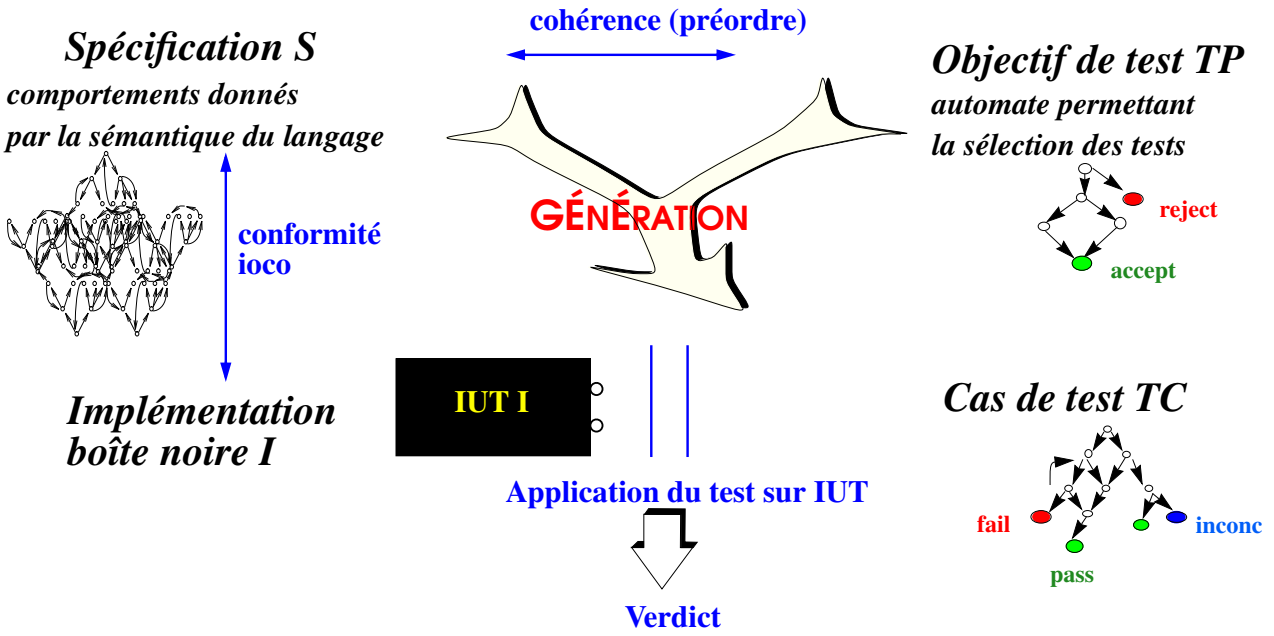
Test distant



- **Le contexte C transforme la perception de l'IUT I en C || I en modifiant l'observabilité et la controlabilité.**
- **Nécéssité d'une nouvelle définition de la conformité:**
  I ioco S in context C ⇔ C || I ioco C || S
- **Attention:**
  I ioco S in context C ≠> I ioco S (perte d'information)
  I ioco S ≠> I ioco S in context C (le test ds un contexte est biaisé)

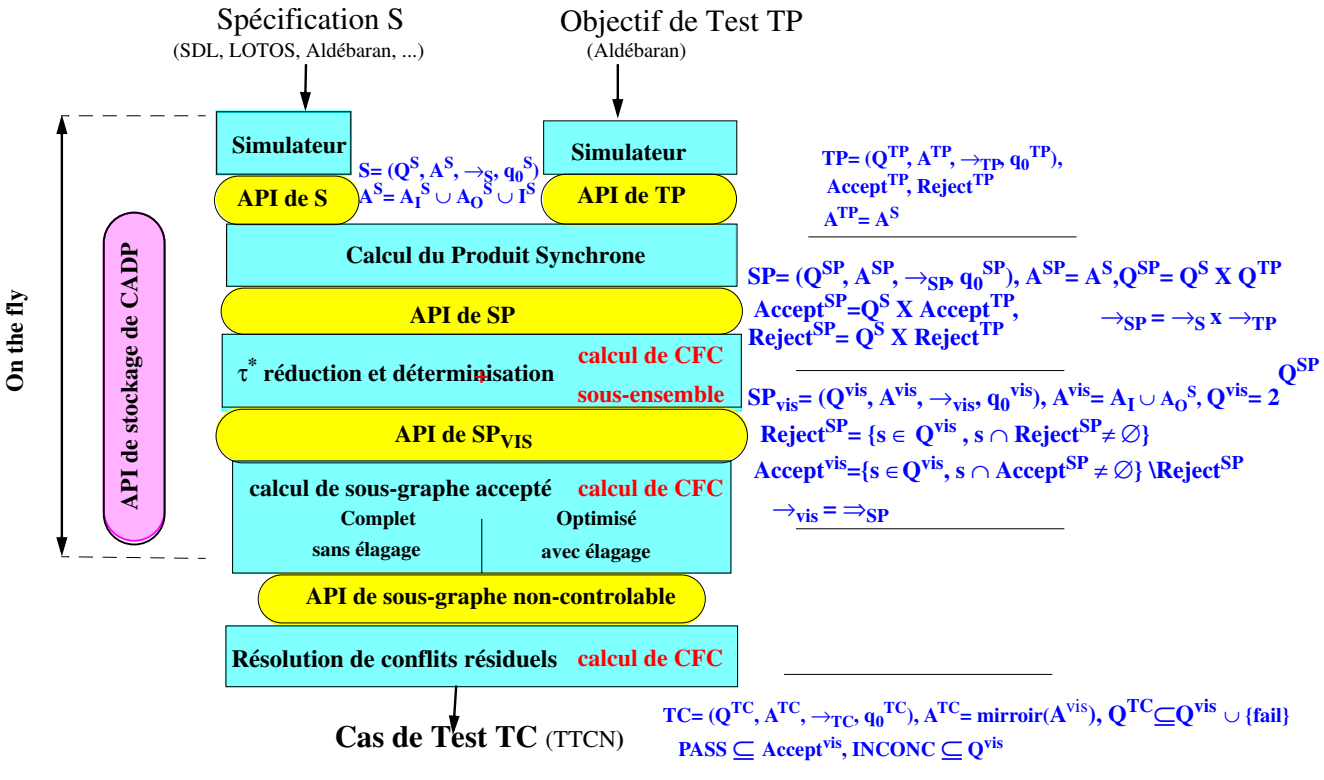
Pas très satisfaisant mais peut on faire mieux ?

Principes généraux de la génération



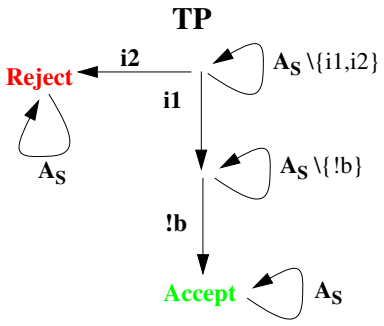
Propriétés : non-biais: I || TC -> fail => not (I ioco S)
exhaustivité: si équité de I, not(I ioco S) => exists TP tq TC(S,TP) || I -> fail

Architecture fonctionnelle de TGV



Objectifs de test

Ils permettent la **sélection** de comportements de S pour cibler des cas de test
Ils sont **complets** et **déterministes**.
Les états Reject et Accept sont des états **puits**.
Sémantique habituelle des **automates**.



Intérêt des actions internes dans les objectifs:
mieux cibler les tests dans le cas de test dans un contexte

Algorithmique

- **Un algorithme de calcul de composantes fortement connexes comme "framework".**
- **Instanciations:**
  - **pour la τ*-réduction et déterminisation**
  - **calcul de sous-graphe complet accepté par un objectif de test**
  - **élagage pour extraction d'un cas de test**
  - **calcul de sous-graphe avec élagage**

Composantes Fortement Connexes

- Un graphe dirigé est **fortement connexe** ssi pour toute paire (u,v) de sommets, il existe un chemin de u à v et un chemin de v à u.
- Une **Composante Fortement Connexe (CFC)** d'un graphe dirigé est un sous-graphe maximal fortement connexe.
- Les CFC induisent une **relation d'équivalence** sur les sommets.
  Le quotient du graphe par cette relation d'équivalence est un graphe dirigé acyclique: le **graphe réduit des CFC**.
- Un algorithme de Tarjan fondé sur un parcours en profondeur (DFS) permet de calculer les CFC d'un graphe dirigé (et aussi le graphe réduit des CFC). Sa complexité est linéaire en temps et en mémoire.

  On utilise cet algorithme comme "framework" à 3 niveaux dans TGV:
  - τ^*-réduction (et déterminisation)
  - calcul de sous-graphe accepté par un objectif de test
  - élagage pour résolution de conflits de controlabilité

Algorithme de Tarjan

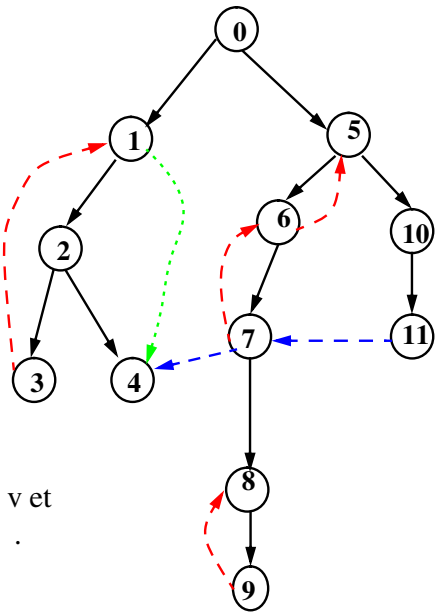
Durant une DFS, les arcs menant d'un sommet vers un nouveau sommet, appelés **tree-arcs** ⟶ , forment un **arbre couvrant**.
Les autres arcs forment une partition en 3 classes:
fronds - - → d'un descendant vers un ancètre
cross-links - - → joignant 2 sous-arbres
short-cuts ⋯→ d'un ancètre vers un descendant.

Number(v): numéro dans l'ordre de parcours de la DFS.

Racine de CFC: premier sommet visité dans la CFC.
C'est la racine du sous-arbre couvrant de la CFC

Lowlink(v)=plus petit numéro d'un sommet de même CFC que v et atteignable par des ⟶ et au plus un - - → ou un - - → .

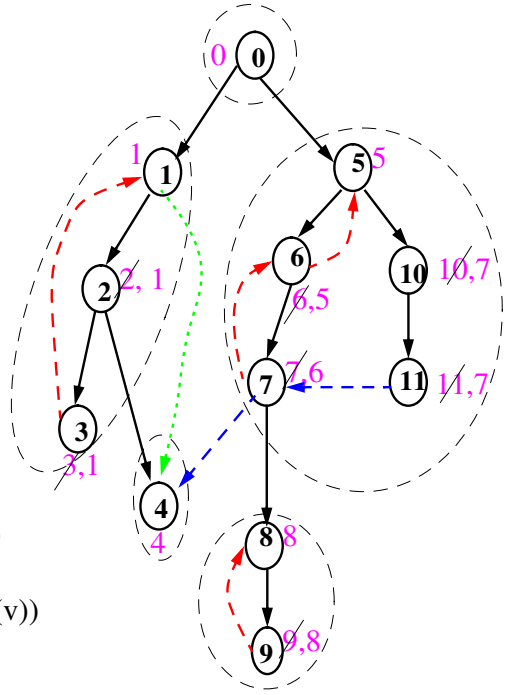
v racine de CFC ⟺ Lowlink(v) = Number(v)



Tarjan (suite)

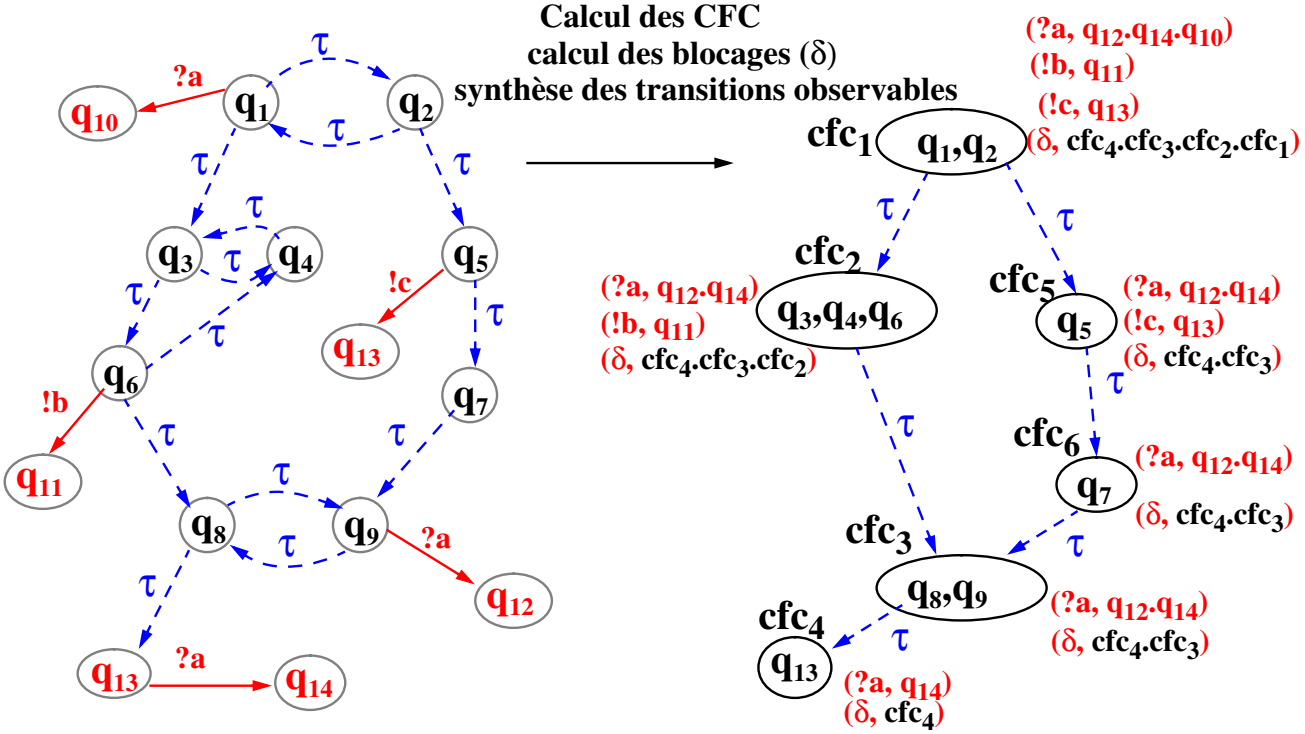
Deux piles:
- DFS-stack pour la DFS
- CFC-stack pour les états dont la CFC n'est pas déterminée

Lowlink est synthétisé le long des ⟶, ⇢ et ⇢
- initialement i:=0;
- v ⟶ w : lowlink(w) := number(w) := i := i+1;
  push(w) dans DFS-stack et CFC-stack
- v ⇢ w ou v ⇢ w et w in CFC-stack:
  lowlink(v):=min(lowlink(v), number(w))
- pop(v) de DFS-stack: si lowlink(v) = number(v) alors
  (* v est racine de CFC *),
  tant que number(w) >= number(v)
  pop(w) de CFC-stack et w -> CFC(v)
  soit u tq u ⟶ v :
  lowlink(u):=min(lowlink(u), lowlink(v))

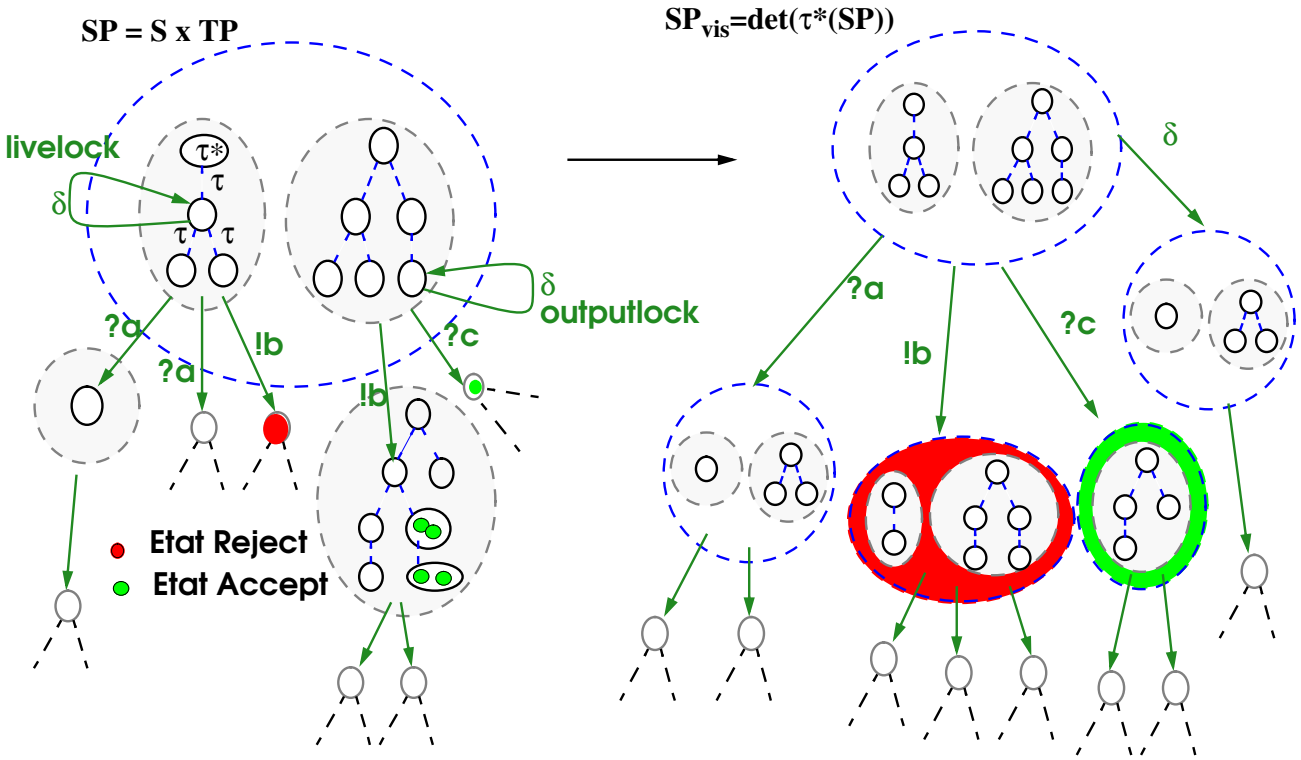


τ*-réduction (vue locale)

Calcul des CFC
calcul des blocages (δ)
synthèse des transitions observables

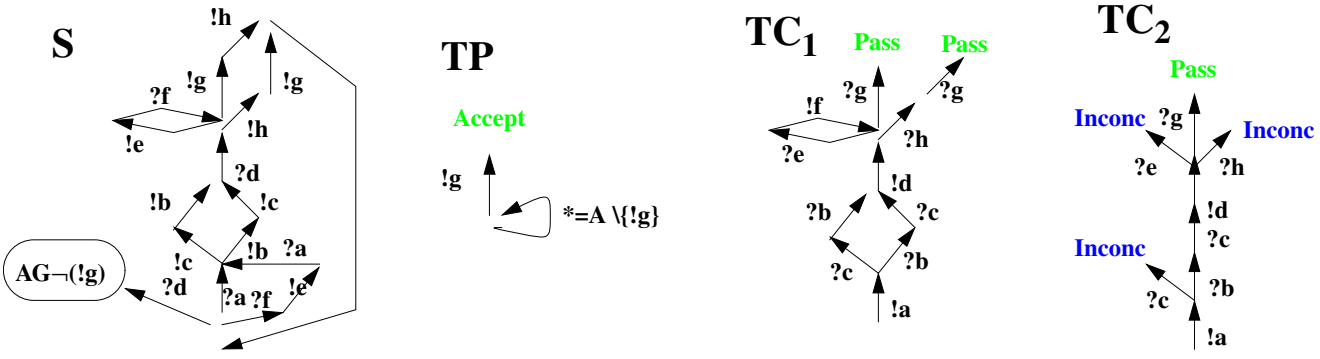


τ*-réduction et déterminisation (vue globale)



Motivations pour le calcul de cas de tests

- **La qualité des tests:** réduire autant que faire se peut les verdicts Inconclusifs qui ne donnent aucune information.
  Verdict Inconclusif à l'éxécution: réexécution jusqu'à obtenir Fail ou Pass.



Intuitivement TC1 est meilleur que TC2.
Sa probabilité de produire un cas de test Inconclusif est plus faible.

Nous proposons un algorithme qui limite les Inconclusifs en rendant les tests plus adaptatifs.

Calcul de sous-graphe de PS_{vis} accepté

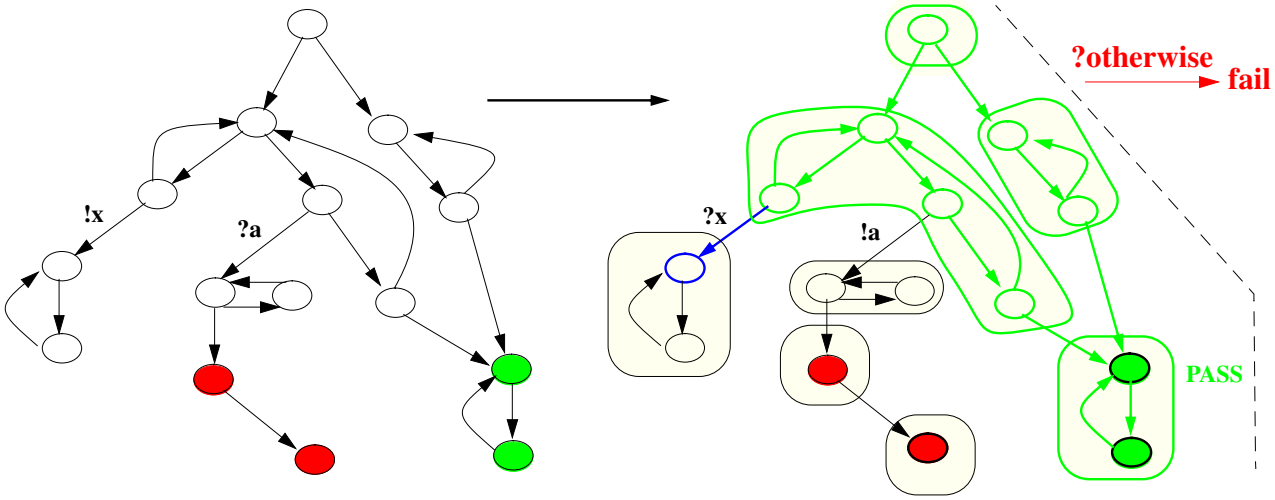
PS_{vis}=det(τ*(SxTP))
+ Accept + Reject

CTG = (L2A ∪ Inconc ∪ {fail} ,$\overline{A}_{vis}$, s_0, →)

L2A = { s | ∃ σ, s $\xrightarrow{\sigma}_{vis}$ Accept} , PASS = Accept

Inconc = {s' ∉ L2A| ∃ s ∈ L2A, ∃ x∈ A_o^{vis}, s $\xrightarrow{x}_{vis}$ s'}

= { s $\xrightarrow{\bar{a}}$ s' | s $\xrightarrow{a}_{vis}$ s', s ∈ L2A et s' ∈ L2A∪ Inconc}
∪ {s $\xrightarrow{?x}$ fail | s ∈ L2A, s $\not\xrightarrow{!x}_{vis}$ }



Synthèse de L2A superposé au calcul de CFC

Rafinement de partition des arcs:

même CFC	entre CFC
$Tree\text{-}arc_{in}$ →	$Tree\text{-}arc_{out}$ →
$cross\text{-}link_{in}$ - - →	$cross\text{-}link_{out}$ - - →
$short\text{-}cut_{in}$ - - →	$short\text{-}cut_{out}$ - - →
frond - - →	

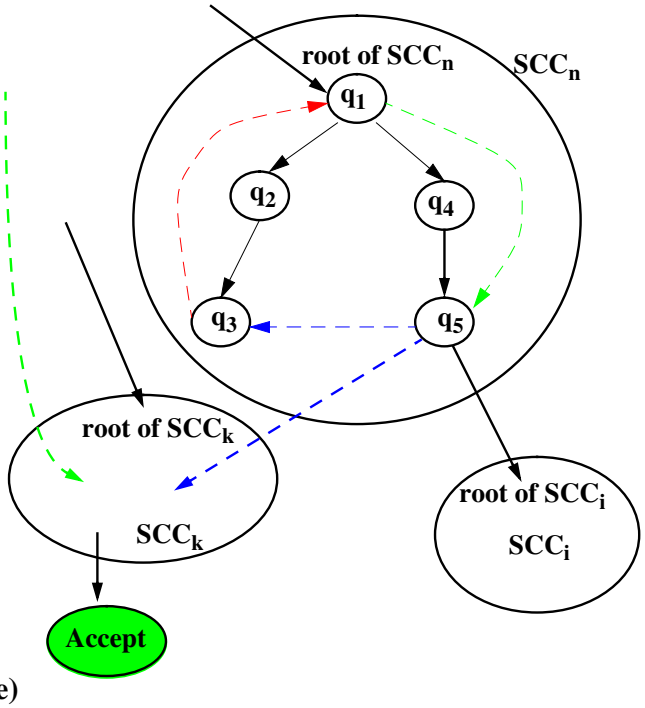
Pendant le calcul des CFC, L2A est synthétisé vers les racines de CFC le long des:

$Tree\text{-}arc_{in}$ → $Tree\text{-}arc_{out}$ →

($cross\text{-}link_{in}$ - - →) $cross\text{-}link_{out}$ - - →

(frond - - →)

L2A de tous les états d'une CFC sont mis à jour lorsque la CFC est complétée (on dépile la racine)



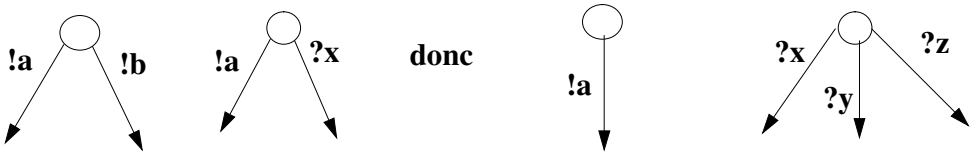
Comparaison avec le model-checking

- **Abstraction faite des Inconclusifs et Fail, l'algorithme de calcul de CTG peut être vu comme un algorithme de model-checking local de la formule de CTL AG ¬ Accept fournissant un diagnostic complet. (ou explication complète de EF Accept)**

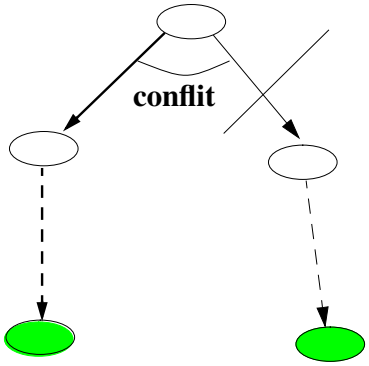
Résolution de conflits

- **En général CTG n'est pas un cas de test: il n'est pas contrôlable**

Configurations interdites car le testeur contrôle ses sorties



- **Résolution des conflits par un parcours en arrière de CTG**



L'élagage modifie l'accessibilité à l'état initial => problème d'accessibilité

Plusieurs parcours possibles:
- largeur en partant de l'ens. des états Pass
-> plus courts chemins
- profondeur

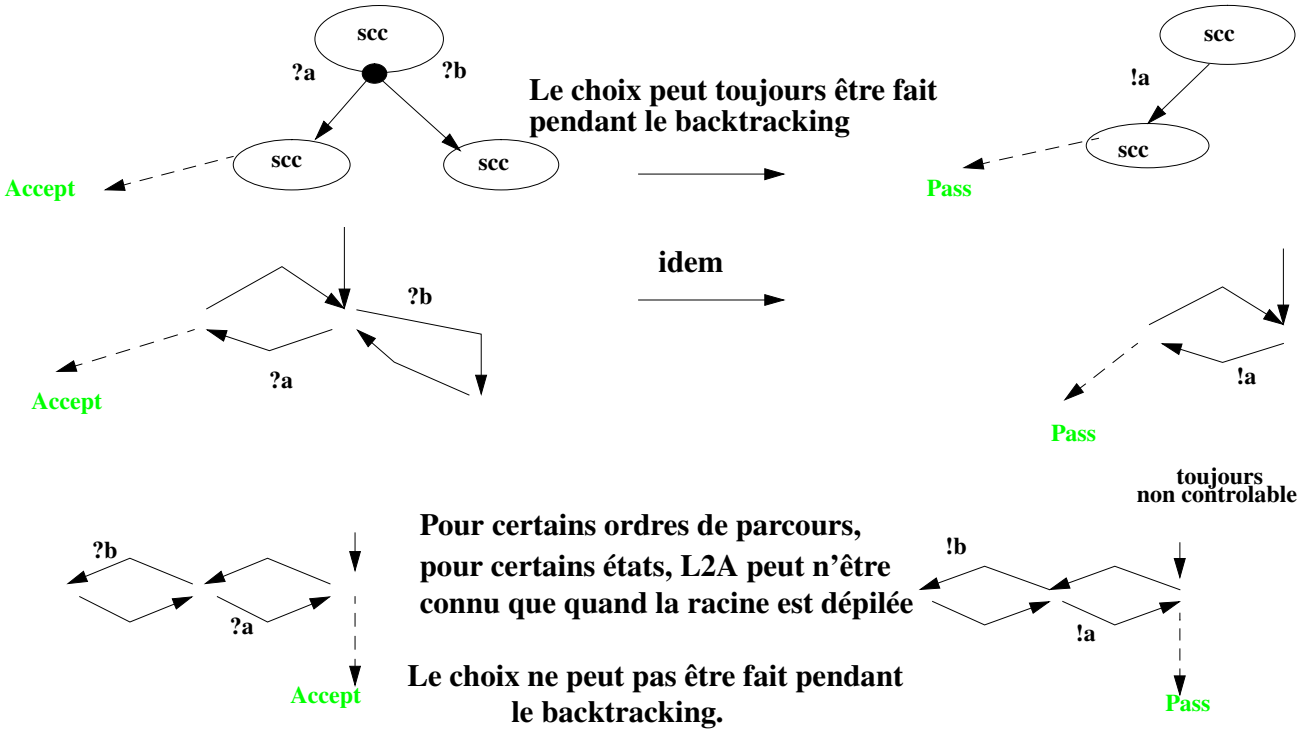
- profondeur avec CFC (Tarjan) :
synthèse de l'information L2init ds $\rightarrow_{CTG}^{-1}$
-> garbage collection

En résumé

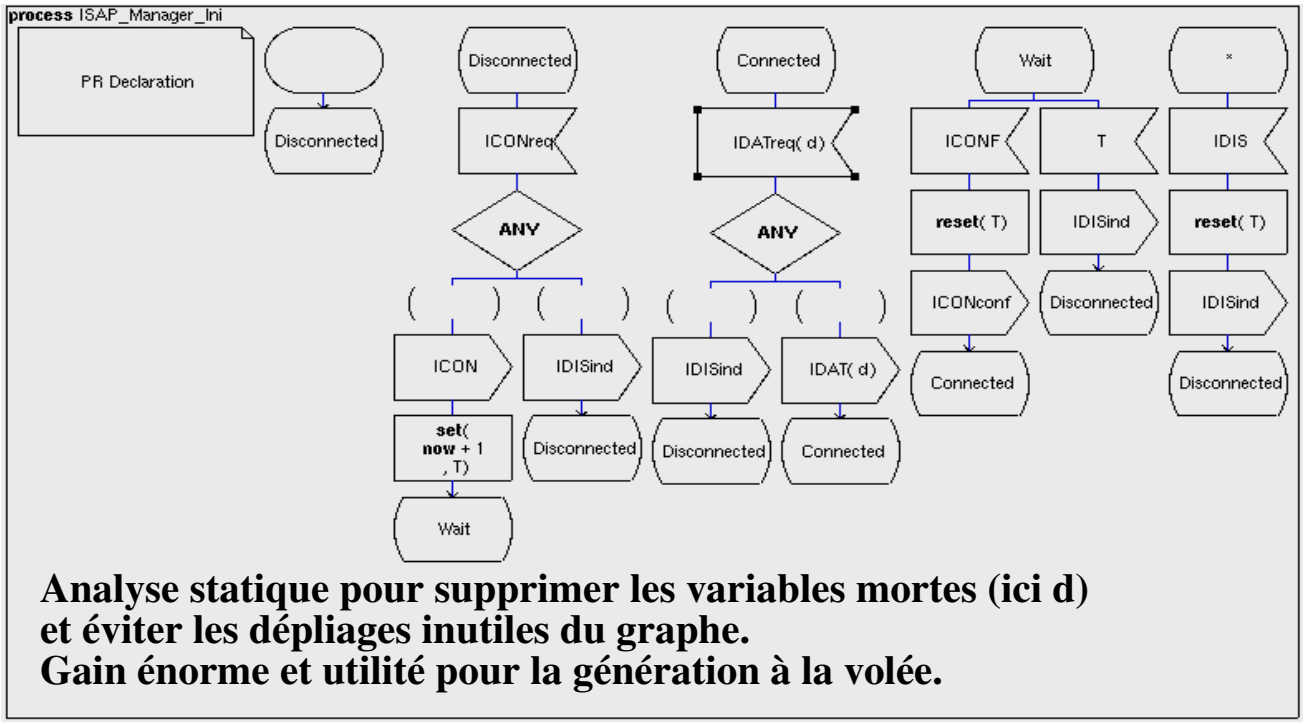
- CTG n'est pas controlable.
- Le calcul de CTG fournit un graphe contenant tous les tests pour une spécification S et un objectif de test TP donnés.
- CTG peut ensuite être élagué pour fournir un cas de test contrôlable.
- Mais le calcul de CTG nécessite de parcourir tout PS_{vis} donc tout S (au moins jusqu'aux états Accept et Reject)
  => peu adapté à la génération à la volée

- Idée:
  - effectuer l'élagage possible pendant la DFS en synthétisant L2A aussitôt que possible (en utilisant les cross-$link_{in}$ et frond)
  - ne permet pas de régler tous les conflits de controlabilité.
    => élagage sur le graphe résultant pour supprimer les conflits résiduels.

Différents problèmes de controlabilité

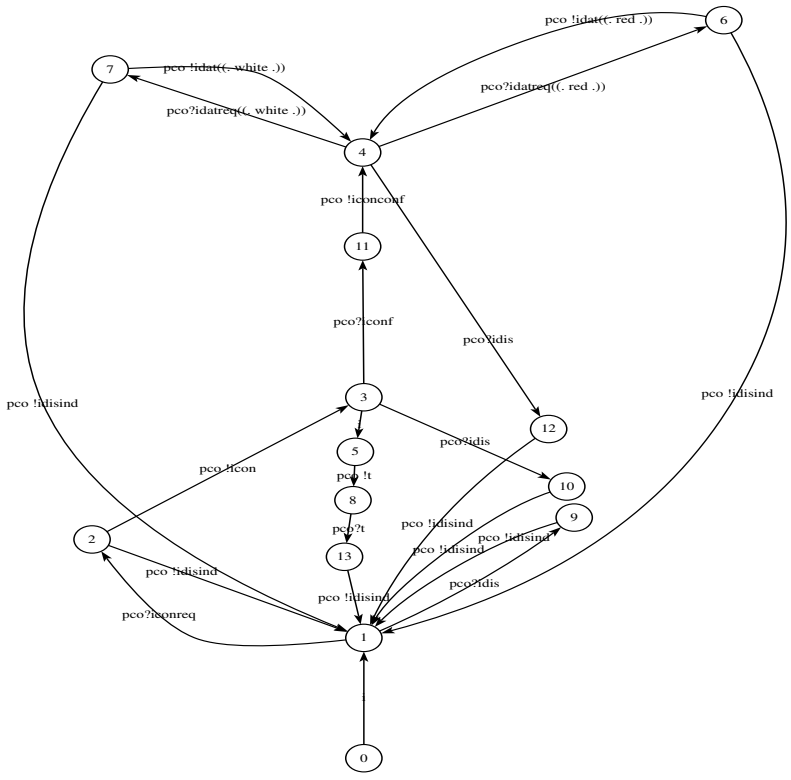


Exemple: processus initiateur d'Inres en SDL



Analyse statique pour supprimer les variables mortes (ici d) et éviter les dépliages inutiles du graphe.
Gain énorme et utilité pour la génération à la volée.

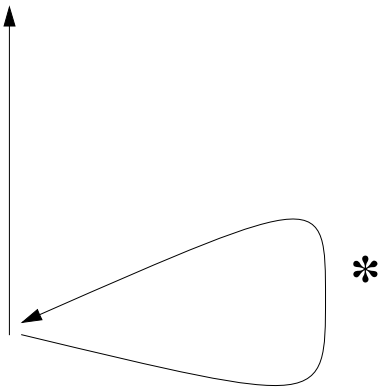
Graphe d'état (non minimal)



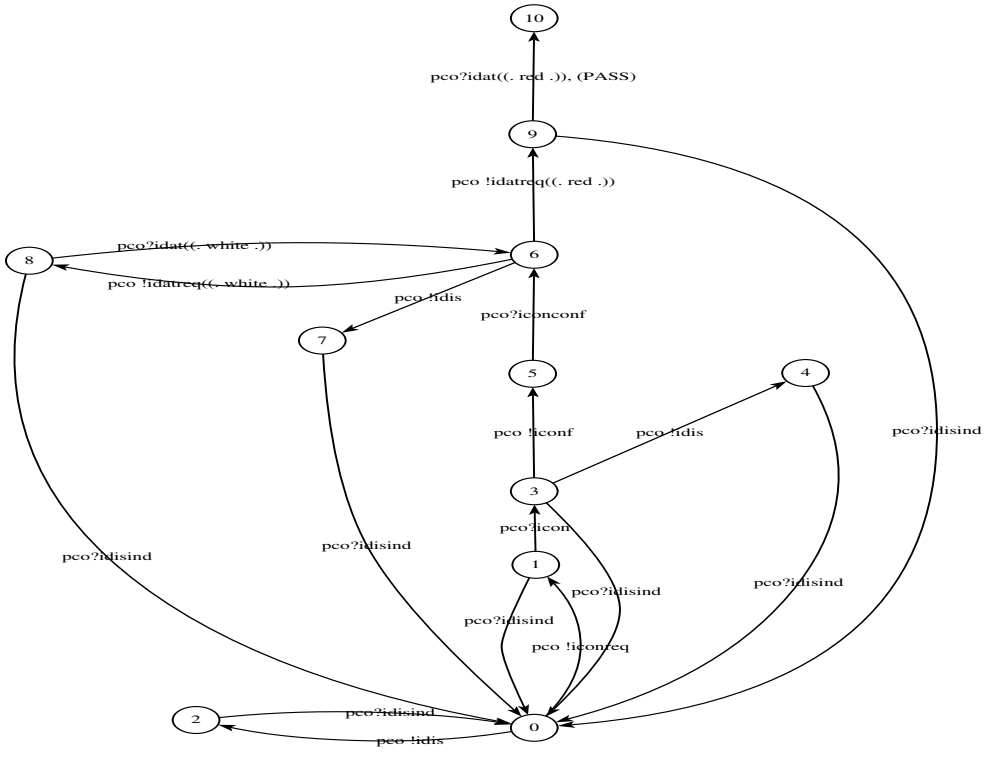
Objectif de test

Accept

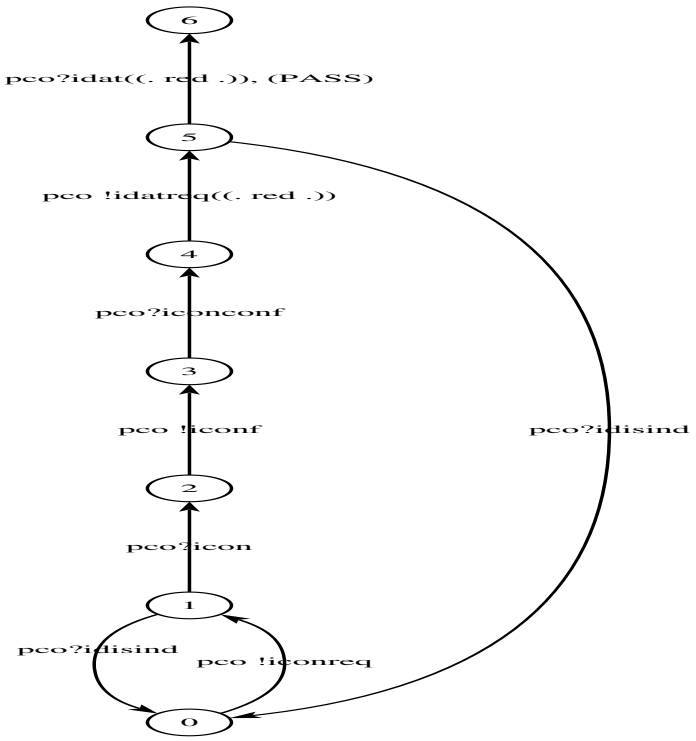
idat(0)



CTG généré à la volée par TGV



Cas de test produit à la volée



Cas de test TTCN

Test Case Dynamic Behaviour					
Test Case Name : graphes/t01_fly_obs_2					
Group :					
Purpose :					
Default :					
Comments :					
Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1	L1	pco! iconreq, St tidisind, St ticon	iconreq0		
2		pco? icon, Cl ticon, Cl tidisind	icon2		
3		pco! iconf, St ticonconf	iconf3		
4		pco? iconconf, Cl ticonconf	iconconf4		
5		pco! idatreq, St tidisind, St tidat	idatreq5		
6		pco? idat, Cl tidat, Cl tidisind	idat6	(PASS)	
7		pco? idisind, Cl tidat, Cl tidisind	idisind1		
8		GOTO L1	idisind1		
9		? tidat		FAIL	
10		? tidisind		FAIL	
11		? ticonconf		FAIL	
12		pco? idisind, Cl ticon, Cl tidisind	idisind1		
13		GOTO L1	idisind1		
14		? ticon		FAIL	
15		? tidisind		FAIL	

Conclusion

- Architecture de TGV décrite par des API bien adapté à la génération à la volée
- En découle une algorithmique efficace fondée sur un seul algorithme.
- Complexité:
  - t*-réduction :linéaire
  - déterminisation: exponentielle
    mais sur un sous-graphe partiel
  - calcul de CTG et élagage: linéaire

Perspectives

- Transfert de TGV dans ObjectGéode en cours
- Test distribué:
  - schéma de distribution de testeur centralisé
  - utilisation d'horloges logiques pour affiner le test.
- Test de logiciel objet distribué: connexion à UML.
- Génération de tests symboliques: connexion de TGV à un solveur de contraintes.
- Test et théorie des jeux: améliorer le contrôle sur l'IUT.