Génération automatique de test de conformité de systèmes réactifs

Thierry JÉRON
Pampa/ Irisa Rennes

- Etat de l'art partiel: methodes formelles et génération automatique de tests
- TGV: aspects techniques
  - principes généraux
  - relation de conformité
  - algorithme de génération
  - gestion des temporisateurs
  - traitement de l'asynchronisme
  - génération classique versus “à la volée”

Test de conformité et méthodes formelles

- Il ne suffit pas d'utiliser des méthodes formelles pour générer des tests “corrects”

SDL
Estelle
Lotos

$\xrightarrow[\text{génération automatique}]{\text{manuel}}$

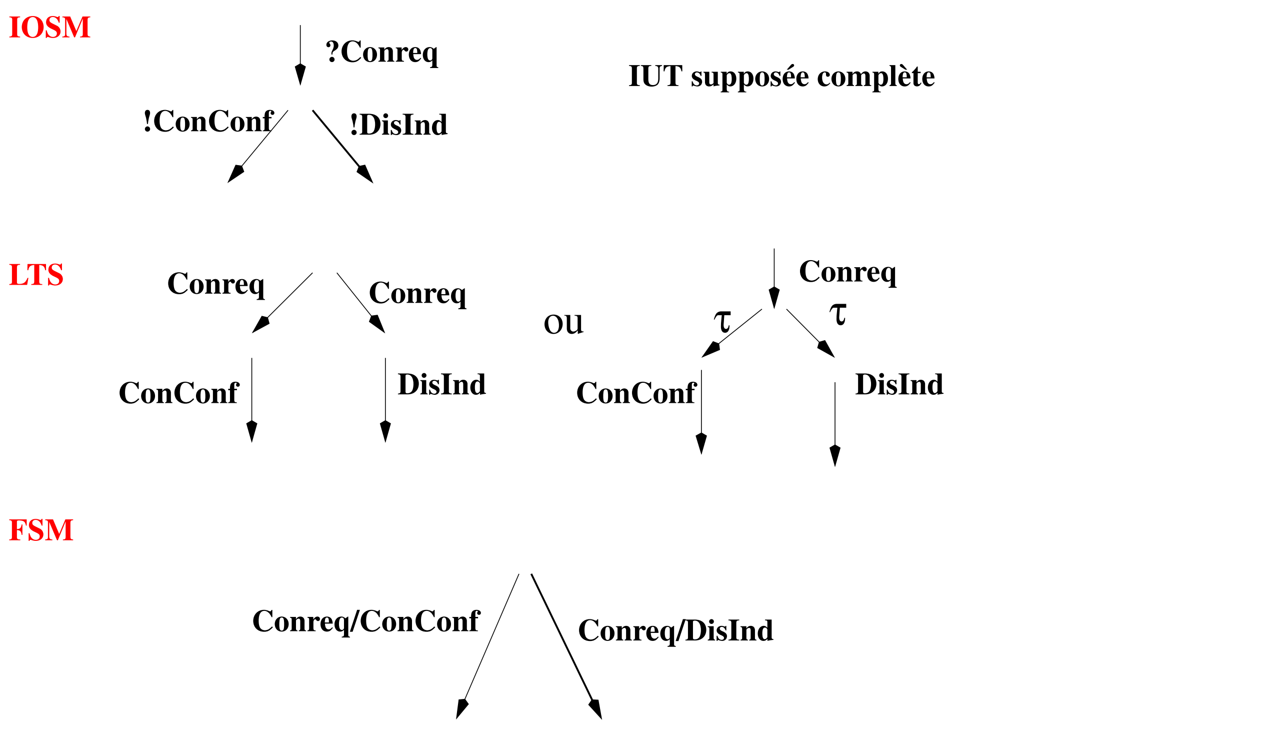
TTCN

- Il faut définir formellement:
  - le modèle de la spécification, IUT, Testeur
  - conformité: implémentations acceptables
  - verdict: interprétation d'une interaction par le testeur
  - communication testeur/IUT

Méthodes étudiées ayant une base formelle

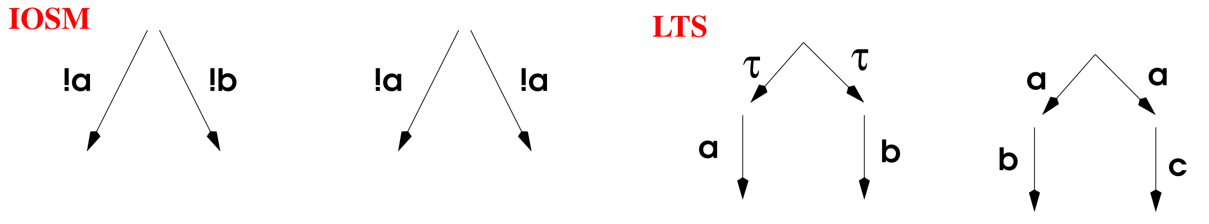
- **FSM: méthodes fondées sur les automates**
  - **- voir exposé de A. Cavalli**
  - - Sidhu, Fujiwara, ...
- **LTS: méthodes fondées sur les systèmes de transition étiquetés**
  - **- l'exemple du testeur canonique de Brinksma**
  - - De Nicolas- Henessy 84, Abramsky 87, Brinksma 88, Brinskma et al. 90, Drira 93, Tretmans 92
- **IOSM: LTS avec distinction entrées/sorties**
  - **- l'exemple de la théorie de TVEDA**
  - - Phalippou, Tretmans, Gaudel, Bernot
- **FMCT: Formal Methods in Conformance Testing**

Modélisation

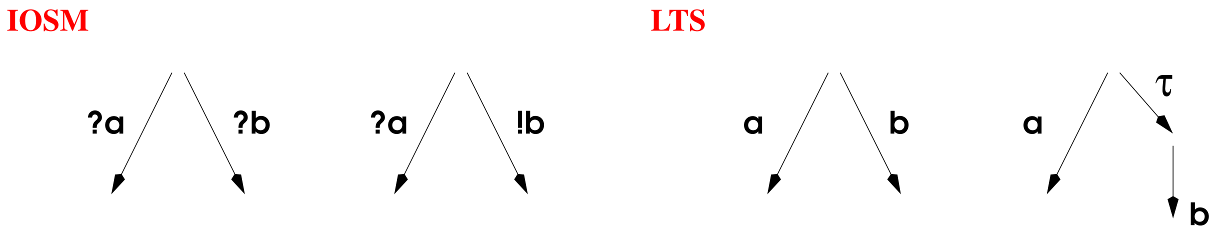


Choix et non-déterminisme

❑ Interne



❑ Externe

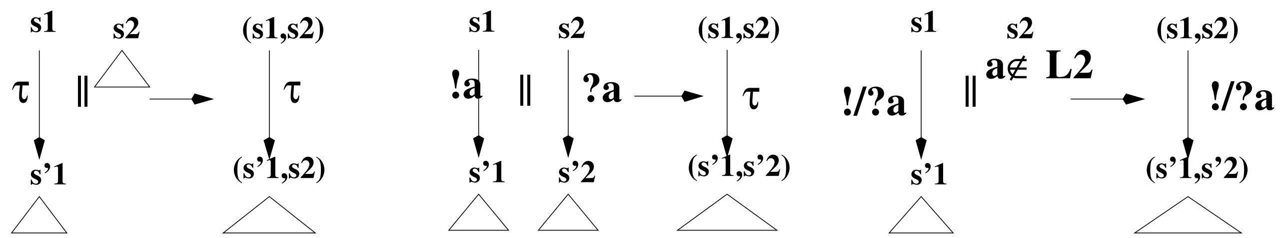


Communication

Composition parallèle pour définir S||T, I||T, (S||E)||T

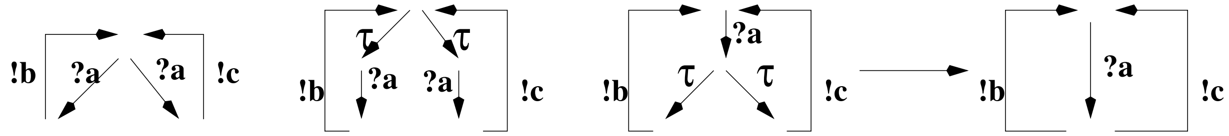
LTS: rendez-vous à la Lotos

IOSM: synchrone à la CCS



Pouvoir d'observation du test

- **IOSM: traces => automate de traces: déterministe et sans t**



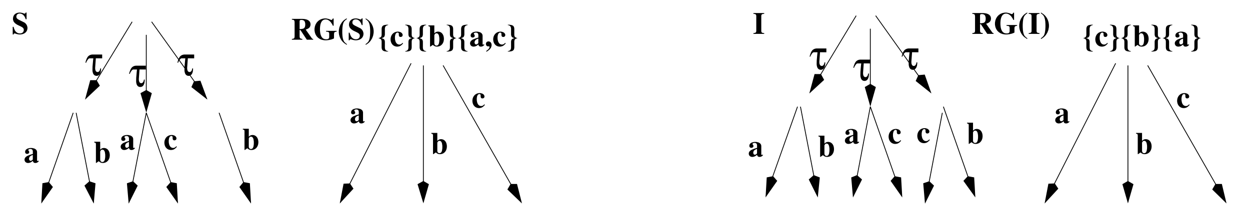
blocages de sortie: modélisation par sortie

- **LTS: traces et refus => graphes de refus: automates de traces + refus**

Relations de conformité

Pas de déf. unique: dépend du pouvoir d'observation, de la modélisation et de ce qu'on veut considérer comme IUT conforme.

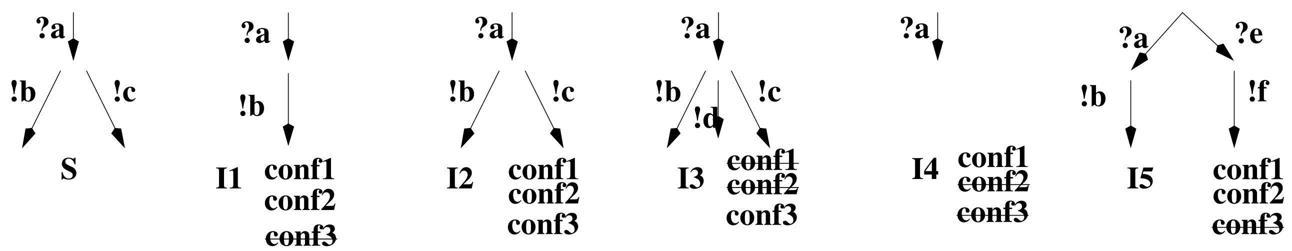
LTS: I conf S ssi $(\forall\ \sigma \in Tr(S)(\sigma \in Tr(I) \Rightarrow Ref(\sigma,I) \subseteq Ref(\sigma,S))$



IOSM: I conf1 S ssi $(\forall\ \sigma \in Tr(S)(\sigma \in Tr(I) \Rightarrow O(\sigma,I) \subseteq O(\sigma,S))$

I conf2 S ssi $(\forall\ \sigma \in Tr(S)(\sigma \in Tr(I) \Rightarrow O(\sigma,I) \subseteq O(\sigma,S))$ et $O(\sigma,I) = \emptyset \Leftrightarrow O(\sigma,S)=\emptyset)$

I conf3 S ssi $(Tr(S) \subseteq Tr(I))$

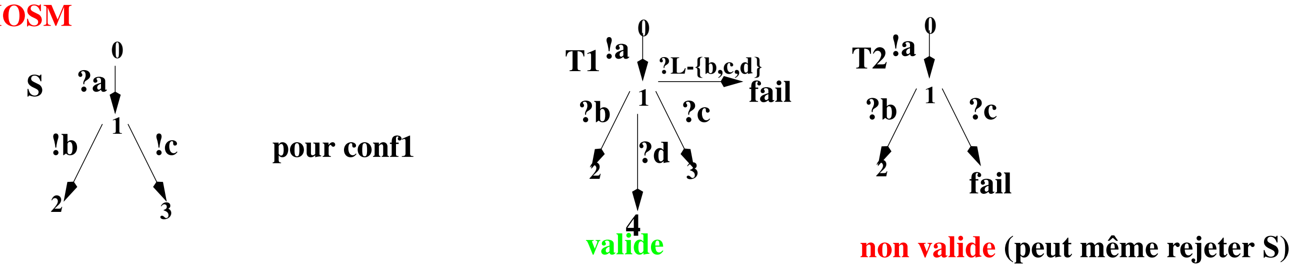


FSM: équivalence des automates

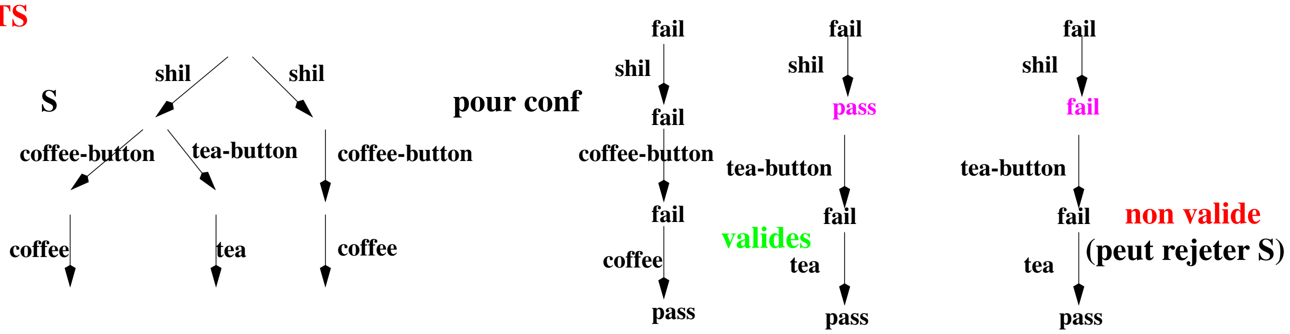
Tests et verdicts

Test valide: ne rend un verdict fail que si l'IUT est non conforrme pourla relation de conformité choisie

IOSM



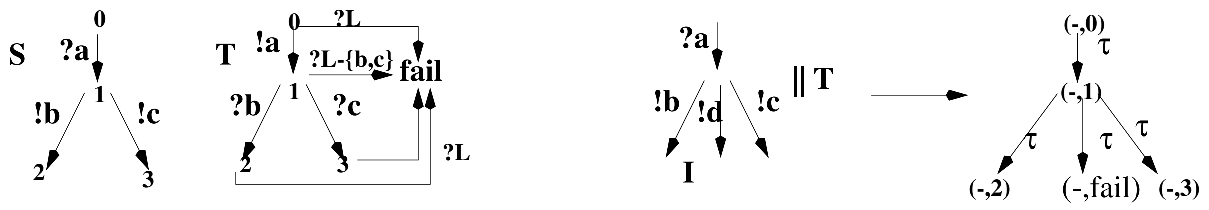
LTS



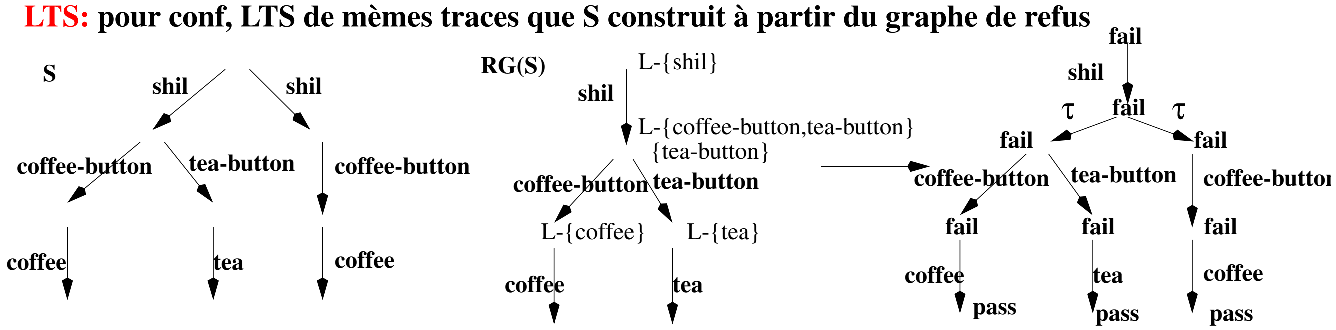
Testeurs canoniques

Processus unique dont la composition avec l'implémentation permet de s'assurer de la conformité (complétude). Il regroupe tous les tests valides.

IOSM: pour conf1, image mirroir de l'automate de trace + état fail sur réceptions non spécifiées



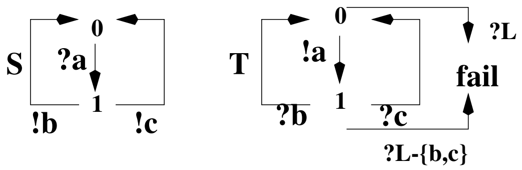
LTS: pour conf, LTS de mèmes traces que S construit à partir du graphe de refus



=> Sélection des tests

Sélection des test

Pour s'assurer de la conformité d'une spec., l'ens. des tests finis à passer souvent très gros voire infini.



→ infinité de tests finis pour conf1

Or l'effort de test est limité en temps, cout.

Il faut donc sélectionner un ens. de tests finis raisonnable.

- utiliser des objectifs de test
- relacher la spécification (admettre + d'IUT conformes)
- affaiblir la relation d'implantation (idem)
- faire des hypothèses de test (restreindre le domaine des IUT)

Hypothèses de test

Hypothèse principale: l'IUT peut être modélisée par un IOSM (LTS, FSM)

❑ Régularité

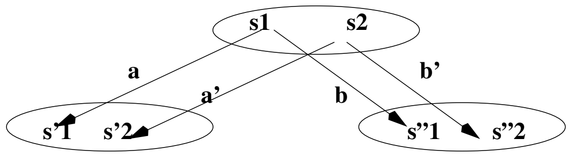
- l'IUT est modélisée par un IOSM ayant au plus N états.
- on montre qu'un nouvelle relation de conformité sur un nombre fini de traces finies est équivalente à la relation d'implantation initiale.
- similaire aux hypothèses faites dans le cas des FSM

❑ Indépendance

- S= S1 x S2 =Pr(S,L1) x Pr(S,L2) et on fait l'hypothèse que I = I1 x I2
- I1 conf S1 et I2 conf S2 et I = I1 x I2 => I conf S
- générer les tests de I1 et I2 avec une hypothèse de régularité.
- |tests(S)|= f(|S|)=f(|S1|x |S2|) >> f(|S1|) + f(|S2|) => effort de test réduit
- exemples: plusieurs instances de la même fonction sans partage de mémoire.
- contre-exemple: interfonctionnement de service
-
-
-
-

❑ Uniformité

- il existe une partition des états et actions de l'IOSM S qui est compatible avec la structure de S. On fait l'hypothèse qu'elle est conservée dans l'IUT.

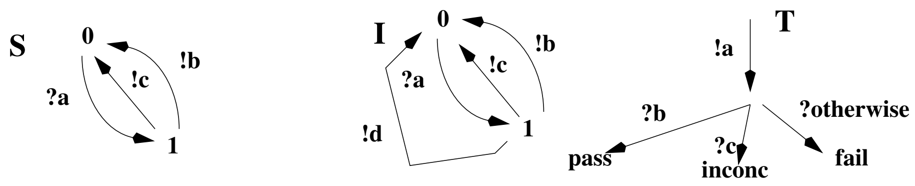


Produire les tests pour l'abstraction (quotient).

Pratique habituelle.

❑ Equité

- provient du non-déterminisme des spécifications et implantations



- hypothèse d'èquité bornée: en essayant le même test un nombre borné de fois, l'implantation montrera tous ses comportements possibles pour ce test.

Formal Methods in Conformance Testing

❑ Draft de document de normalisation

❑ Fédérateur des différentes approches de la génération de tests basées sur les méthodes formelles (LDS, LOTOS, Estelle) et leurs modèles (LTS, IOSM, FSM).

❑ Méta théorie du test: décrit les principes à adopter pour définir une méthode de génération automatique de tests de conformité.

- définitions de la conformité, concepts d'architecture de test, exécution des tests, etc..

❑ But: pouvoir comparer les méthodes automatiques décrites dans un formalisme commun.

Qu'est ce que la conformité ?

- **Pas de définition unique: dépend du degré de similarité voulu entre Spec et IUT, du degré d'observabilité possible. Mais pour une méthode donnée, il faut la définir.**

SPECS: ensemble des spécifications formelles .

Une spec. **S** de ***SPECS*** est un objet mathématique.

IMPS: ensemble des implémentations.

Une impl. **IUT** de ***IMPS*** n'est pas un objet mathématique.

Pour raisonner, il faut supposer qu'elle peut être modélisée par un élément $\mathbf{m_{IUT}}$ d'un formalisme ***MODS*** (LTS, FSM, etc.)

La conformité est caractérisée par une relation d'implantation ***imp*** $\subseteq$ ***MODS*** $\times$ ***SPECS.***

IUT est conforme à **S** par rapport à la relation ***imp*** si $\mathbf{m_{IUT}}$ ***imp*** **S**

On pourra donc raisonner sur l'ensemble des $\mathbf{m_{IUT}}$ conformes à **S** pour ***imp***, pas sur l'ensemble des **IUT** conformes à **S**

Tests et architecture de test

TESTS: ensemble des tests (notation par exemple TTCN)

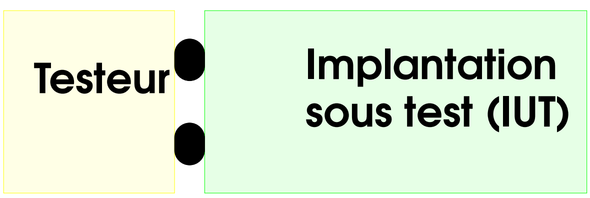
Cas de test: **t** $\in$ ***TESTS***

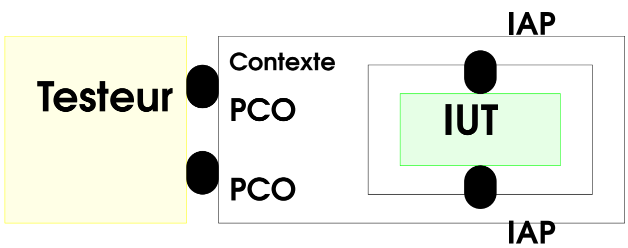
Suite de tests: ensemble de tests $\in$ ***P(TESTS)***

Contexte (archi. de test): transformation de comportements: ***C: MODS*** $\rightarrow$ ***MODS***

$\mathbf{C(m_{IUT})}$: comportement de l'**IUT** observé aux PCOs.

Idéal: pas de contexte i.e. $\mathbf{C(m_{IUT}) = m_{IUT}}$





Exécution des tests

exec: ***TESTS*** $\times$ ***MODS*** $\rightarrow$ ***OBS***
ou ***OBS*** est l'ensemble des observations $\sigma \in$ ***OBS.***
Par exple: ***exec*****(t,m_{IUT}) = t || m_{IUT}**

Verdict de **t** $\in$ ***TESTS*** exécuté sur **IUT** modélisé par **m_{IUT}**:
$verd_t$: ***OBS*** $\rightarrow$ {pass, fail, inconclusive}.

Passage d'un test:
IUT *passe* le test **t** s'il ne produit pas le verdict fail.
IUT *passe* une suite de tests **T** ssi elle passe tous les tests **t** $\in$ **T.**

Instanciation de ces notions pour différentes méthodes de génération de tests et différentes FDT (SDL, Lotos,Estelle)
Bibliographie: D. Hogrefe (IWTPS95, IWTCS96 (Chapman & Hall))

TGV: aspects techniques

- **Principes généraux**
- **Relation de conformité**
- **Algorithme de génération**
- **Gestion des temporisateurs**
- **Traitement de l'asynchronisme**
- **Génération classique versus “à la volée”**
- **Perspectives**

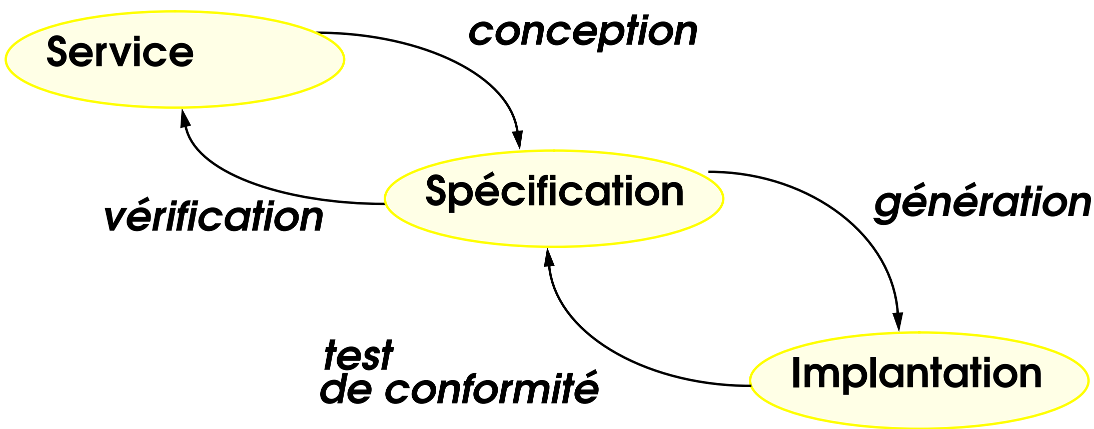
L'approche TGV

Formalisation de l'activité de génération manuelle

- **Modélisation** de la spécification, de l'implantation, des tests, et des objectifs de test par des **systèmes de transitions à entrée et sorties** (distinction entre actions observables et contrôlables)
- **Définition formelle** d'une **relation de conformité** entre la spécification et ses implantations.
- **Sélection** des tests par les objectifs de tests.
- **Correction** des tests vis à vis de la relation de conformité:
  - rejet par le test => non conformité
  - non conformité => on peut produire un test qui peut rejeter l'implantation
- **Efficacité** de la génération basée sur une algorithmique issue du domaine de la vérification.

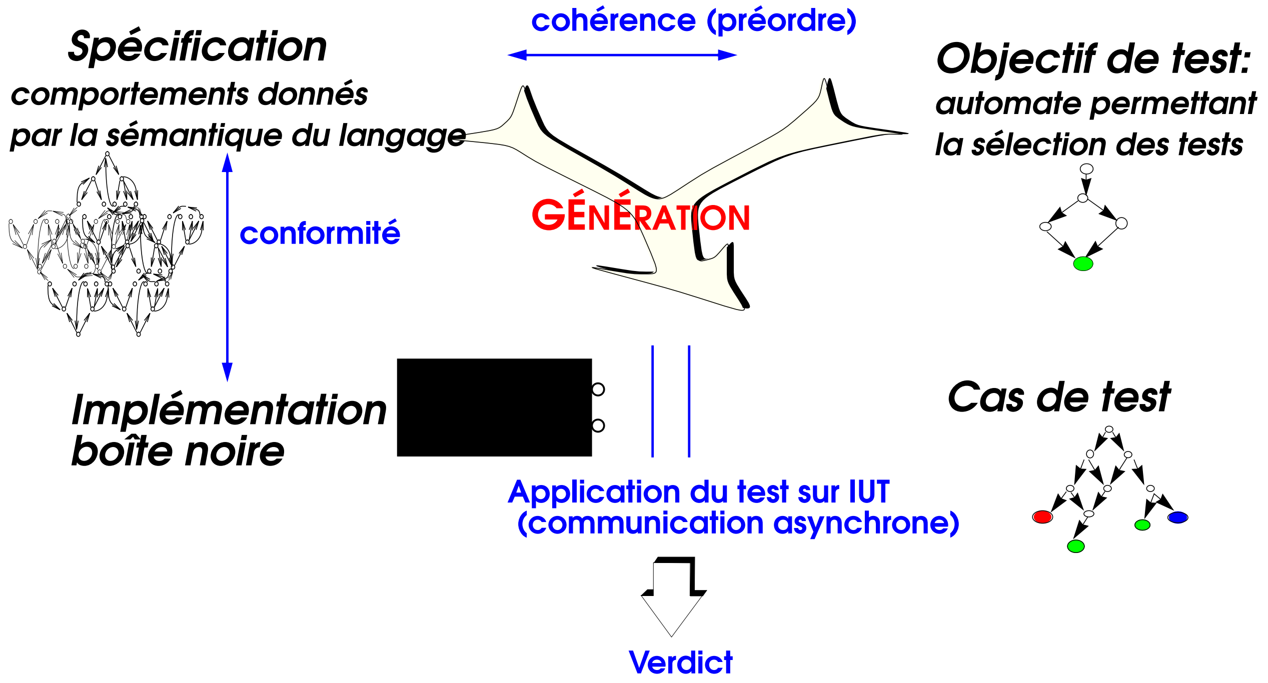
Le test et la vérification dans le processus de conception

- Différences mais aussi similutudes.



- Algorithmique de la vérification fondée sur l'exploration des comportements possibles de la spécification réutilisable dans le cadre de la génération de tests.

Principes généraux



Modélisation: les systèmes de transitions à entrées/sorties

- Un **IOLTS** est un quadruplet **(Q,A_I,A_O,! , q_{init})** où
  - **Q** est un ensemble fini d'états, q_{init} est l'état initial
  - A_I et A_O sont les alphabets d'entrée et de sortie,
  - **A** = {?} x A_I τ {!} x A_O
  - !ϖ **Q** x **A** x **Q** est la relation de transition.

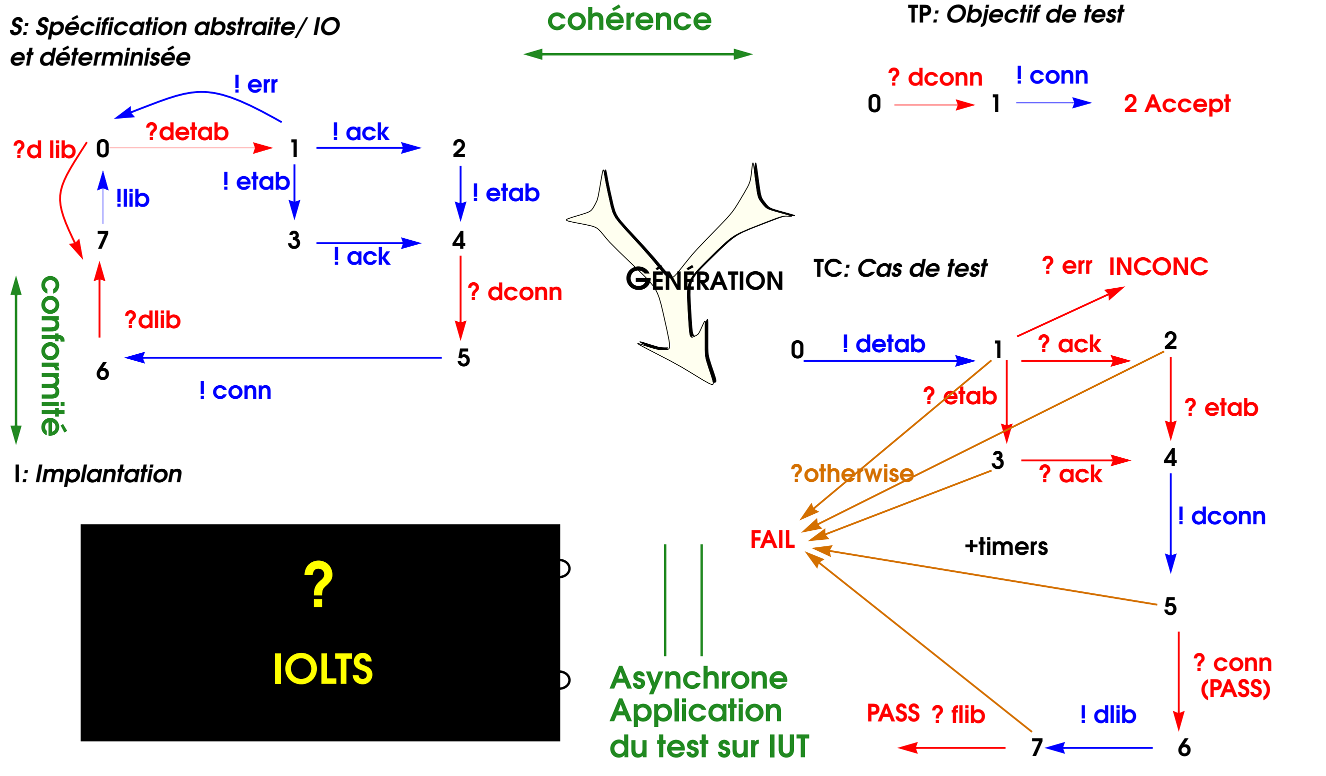
☞ **Spécification S:** comportement visible modélisé par un IOLTS déterministe.

☞ **Implantation I:** comportement visible modélisé par un IOLTS.

☞ **Objectif de test TP:** IOLTS d'alphabet identique à celui de S,
+ ensemble d'états accepteurs Accept (automate).

☞ **Cas de test TC:** IOLTS sans cycle, déterminste, d'alphabets miroirs de S et I,
+ timers: armements (Start), désarmements (Cancel) et échéances (Timeout)
+ verdicts: FAIL, PASS, (PASS), INCONCLUSIVE
contrôlable: pas de choix entre sorties, ni entrée/sortie.

Modélisation: exemple



Cohérence entre objectif et spécification

- Un objectif de test décrit (informellement) une propriété d'enchaînement d'interactions de la spécification que l'on veut particulièrement tester.
- Son intention est de sélectionner des comportements parmi les comportements de la spécification.

  Il ne doit donc pas être satisfait par toutes les séquences.
- Cohérence: notion faible de satisfaction:

> Un objectif de test TP est cohérent avec la spécification S si il existe au moins une sous-séquence de S acceptée par TP

Relation de conformité

- Relation liant une implantation I à une spécification S et décrivant formellement quelles sont les implantations I **conformes** à S.
- Doit tenir compte de la différence entre événements **observables** et **contrôlables**.

I *ioconf* S ssi <u>après toute trace de S</u>, les sorties permises pour I doivent être incluses dans celles permises pour S.

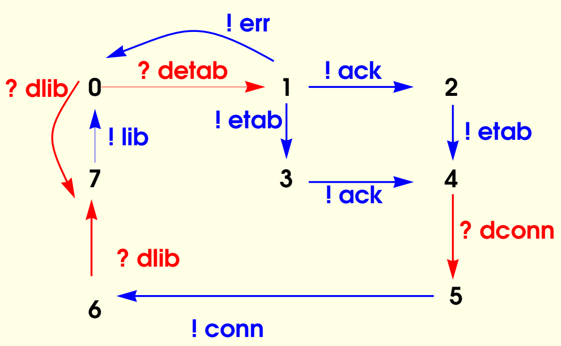
I ioconf S

ssi

$$\forall \sigma \in \mathbf{traces}(S),\ O(I \text{ after } \sigma) \subseteq O(S \text{ after } \sigma)$$

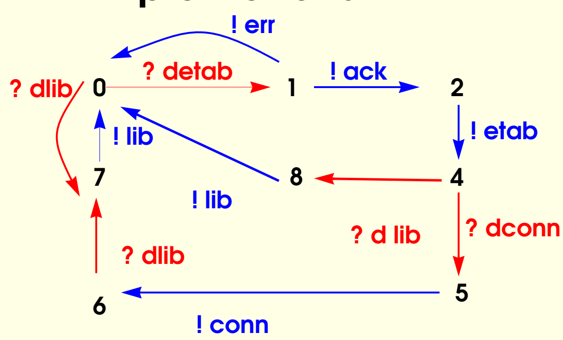
Conformité : exemples

Spécification S

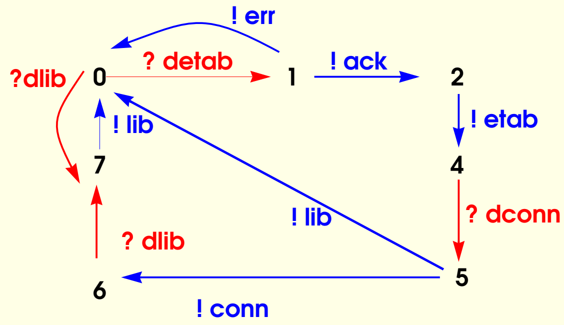


Implantations

I$_1$



I$_2$



I ioconf S

ssi

$$\forall \sigma \in \mathbf{traces}(S),\ O(I \text{ after } \sigma) \subseteq O(S \text{ after } \sigma)$$

I$_1$ *ioconf* S

I$_2$ *not ioconf* S

Forme des cas de tests à produire

❑ **Comportement** : ensemble de séquences d'**interactions** entre l'IUT et le testeur, vu du testeur, voulant vérifier la **conformité** de l'IUT à la spécification, par rapport à l'objectif de test.

✚ **Timers:** observation des blocages et livelocks de l'implantation.

❑ **Verdicts:** associés à certaines interactions avec la signification:

- **FAIL:** non-conformité pour la relation ioconf.
- **(PASS):** l'objectif est atteint. Mais une séquence d'actions appelée Postambule doit ramener l'IUT dans l'état initial. Ceci peut conduire à un verdict FAIL. Le verdict (PASS) est donc provisoire.
- **PASS:** l'objectif a été atteint et l'IUT est dans l'état initial.

Postambule =la séquence entre le verdict (PASS) et le verdict PASS.

- **INCONC:** le comportement observé, soit ne permet pas d'atteindre (PASS), soit n'est pas considéré car un cas de test ne peut pas être exhaustif.

Exemple de test TTCN

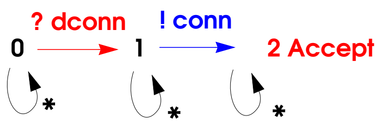
```
+---------------------------------------------------------------------------------------------------------+
|                                        Test Case Dynamic Behaviour                                      |
+---------------------------------------------------------------------------------------------------------+
| Test Case Name      : essai                                                                             |
| Group               :                                                                                   |
| Purpose             :                                                                                   |
| Default             :                                                                                   |
| Comments            :                                                                                   |
+-------+-------+----------------------------------------------+-------------------+----------+-----------+
| Nr    | Label | Behaviour Description                        | Constraints Ref   | Verdict  | Comments  |
+-------+-------+----------------------------------------------+-------------------+----------+-----------+
|     1 |       | dr! retab, St terr, St tack, St tetab        | retab0            |          |           |
|     2 |       |   de? etab, Cl tetab, Cl terr                | etab1             |          |           |
|     3 |       |     dr? ack, Cl tack                         |                   |          |           |
|     4 | L1    |       de! rconn, St tconn                    | rconn2            |          |           |
|     5 |       |         dr? conn, Cl tconn                   | conn3             | (PASS)   |           |
|     6 |       |           dr! rlib, St tlib                  | rlib4             |          |           |
|     7 |       |             dr? lib, Cl tlib                 | lib5              | PASS     |           |
|     8 |       |             ? tlib                           |                   | FAIL     |           |
|     9 |       |         ? tconn                              |                   | FAIL     |           |
|    10 |       |     ? tack                                   |                   | FAIL     |           |
|    11 |       |   dr? ack, Cl tack, Cl terr                  |                   |          |           |
|    12 |       |     de? etab, Cl tetab                       | etab1             |          |           |
|    13 |       |       GOTO L1                                | etab1             |          |           |
|    14 |       |     ? tetab                                  |                   | FAIL     |           |
|    15 |       |   dr? err, Cl tetab, Cl tack, Cl terr        |                   | INCONC   |           |
|    16 |       |   ? tetab                                    |                   | FAIL     |           |
|    17 |       |   ? tack                                     |                   | FAIL     |           |
|    18 |       |   ? terr                                     |                   | FAIL     |           |
+-------+-------+----------------------------------------------+-------------------+----------+-----------+
```

Principe de l’algorithme

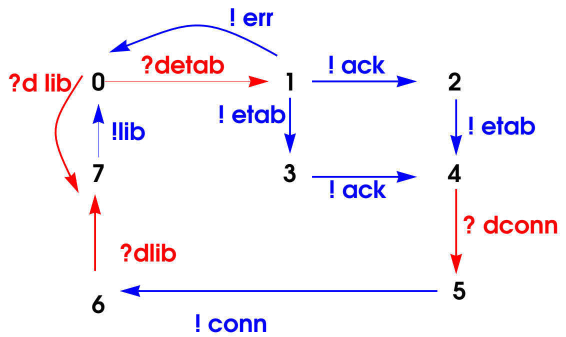
- **Premier parcours du produit entre S et TP**: construction d’un DAG (Graphe Dirigé Acyclique) squelette de test.
  - **à la descente (push):**
    - on vérifie la cohérence jusqu’à atteindre des états accepteurs de TP.
    - calcul du postambule par plus court chemin jusqu’à l’état initial de S,
  - **à la remontée (pop)** à partir des états accepteurs de TP
    - on synthétise un DAG constituant l’ossature du cas de test TC en assurant la contrôlabilité et en ajoutant des verdicts sur certaines transitions.
- **Deuxième parcours en profondeur du DAG résultant** : décoration du DAG par la gestion des timers.
- **Troisième parcours**: traduction au format TTCN.

Produit : objectif de test x spécification

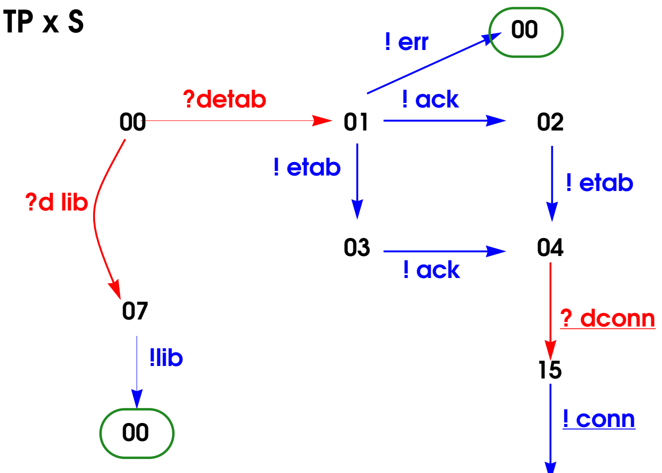
Objectif de test TP

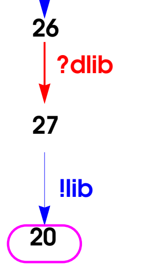


Spécification S



TP x S





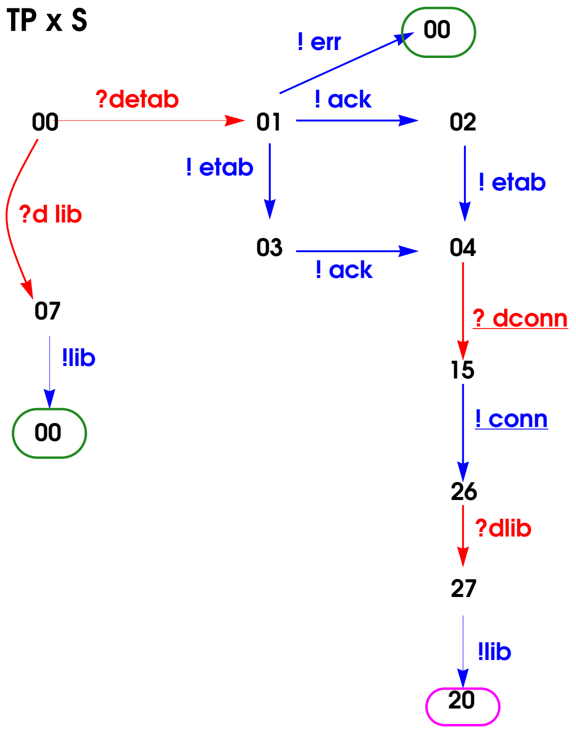
IOLTS acyclique

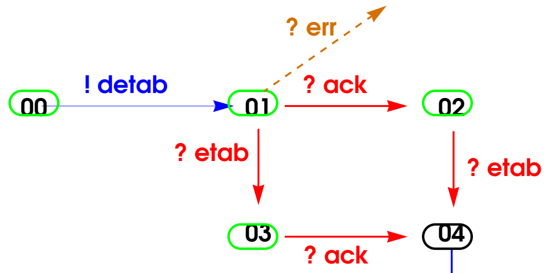
- coupure des boucles
- arrêt sur états Accept

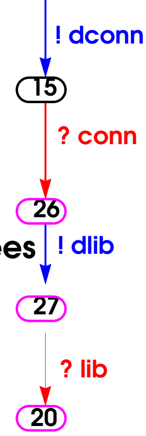
c’est la définition du produit complet
la DFS décide de la partie nécéssaire à construire

Synthèse du DAG par DFS de TP x S

TP x S

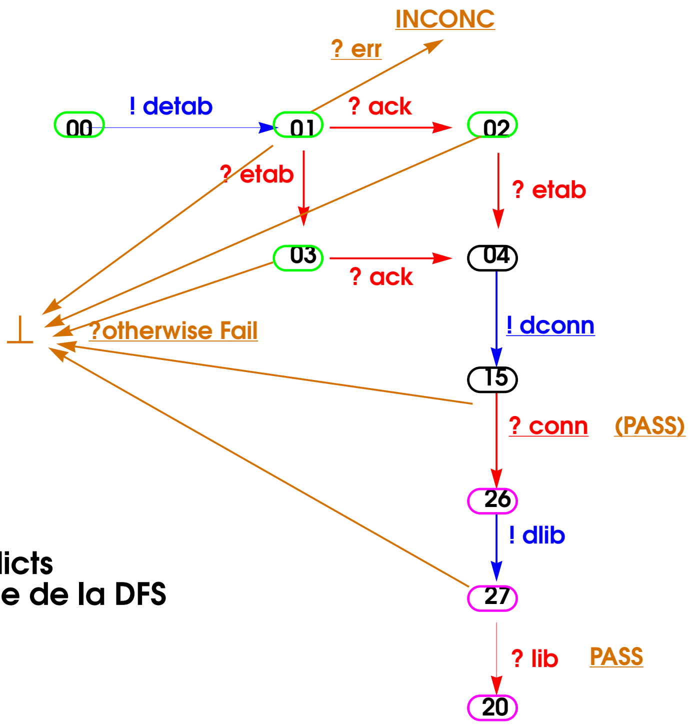






Synthèse du DAG
- en remontant des états (Accept, Initial)
- inversion entrées/sorties
- contrôlabilité sur sorties
- préservation de toutes les entrées
- préambule, corps du test et postambule.

Verdicts



Décoration par verdicts
pendant la remontée de la DFS

Principe de gestion des timers

assurer la finitude de l'application d'un cas de test sur l'IUT

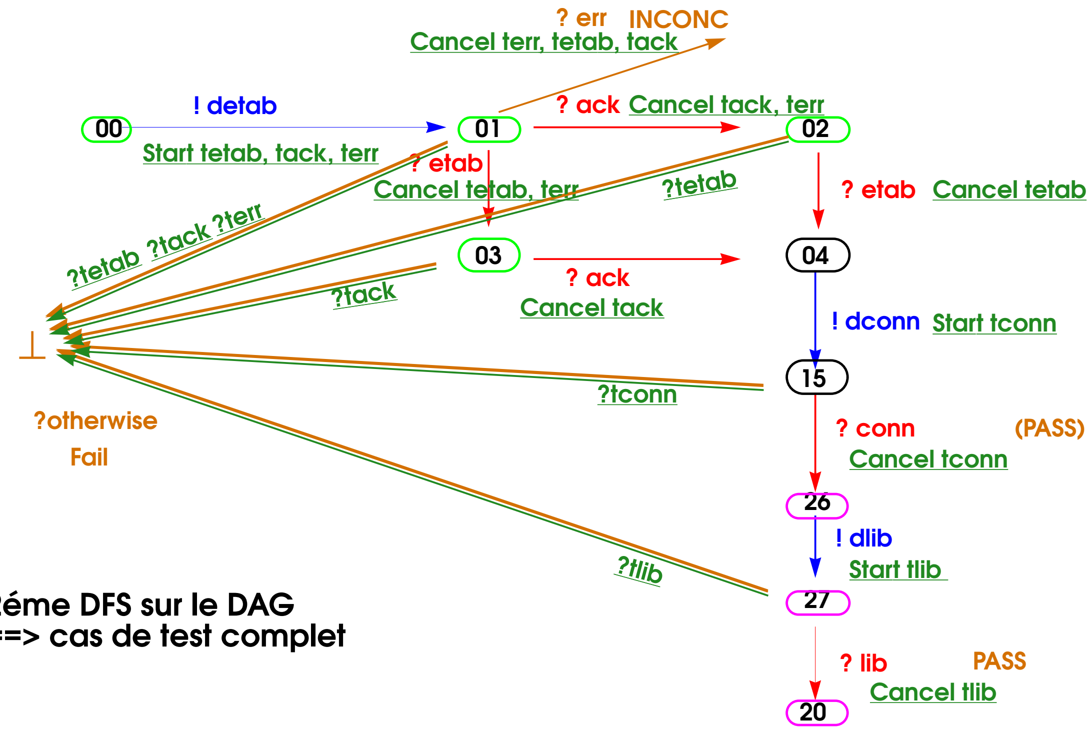
- **Propriétés attendues:**

  blocage de l'IUT) détection par l'expiration d'un timer,
  Inversement, si les délais des timers sont suffisants,
  expiration de timer) blocage de l'IUT.

- **Règles de gestion:** à chaque entrée i attendue dans le cas de test correspond un timer t_i. Trois opérations sur t_i :
  - $\mathrm{Start}\ t_i$: doit être fait dans toute transition atteignant le premier état où l'entrée i est possible.
  - $\mathrm{Cancel}\ t_i$: à la réception de i ou quand celle-ci n'est plus possible.

  Attention à la différence entre concurrence et non-déterminisme.
  - $\mathrm{Timeout}\ t_i$: représente l'observation d'un blocage sur l'attente de i. Doit être possible dans les états où i est possible.

Gestion des timers



2éme DFS sur le DAG
==> cas de test complet

Résultats théoriques

- **Complexité:** linéaire en temps et mémoire par rapport à **TPxS**
- **Vérification de la cohérence** si **TP** cohérent avec **S**, l'algo construit **TC(TP,S)** qui comprend un préambule, un postambule, des verdicts et la gestion des timers.
- **Correction :** si l'application du cas de test **TC(TP,S)** sur l'implantation **I** produit un verdict **FAIL** alors l'implantation **I** est non conforme à la spécification **S** pour la relation **ioconf**.

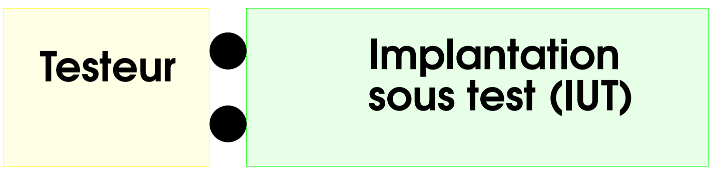
  insuffisant car un test rendant tjrs PASS est correct
- **"Exhaustivité" :** Hypothèse d'équité bornée sur l'implantation**I**.

Pour toute implantation **I** non conforme à sa spécification **S**, il existe un objectif de test **TP** cohérent avec **S** tel qu'un cas de test **TC(TP,S)** généré par l'algorithme et appliqué **n** fois sur **I** produise un verdict **FAIL**.

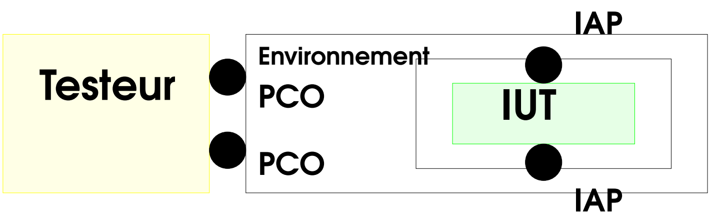
En pratique: si on suspecte un comportement erroné de I, on peut décrire un objectif qui produit le bon test.

Impact de l'architecture

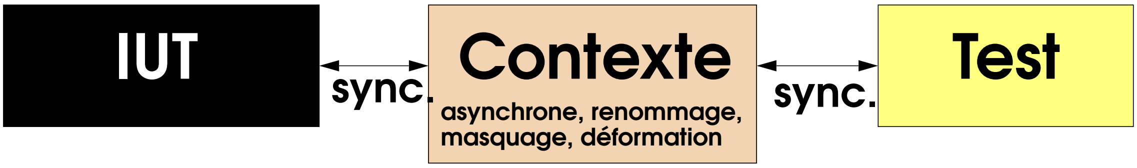
- **Cas idéal: contrôle et observation directs**



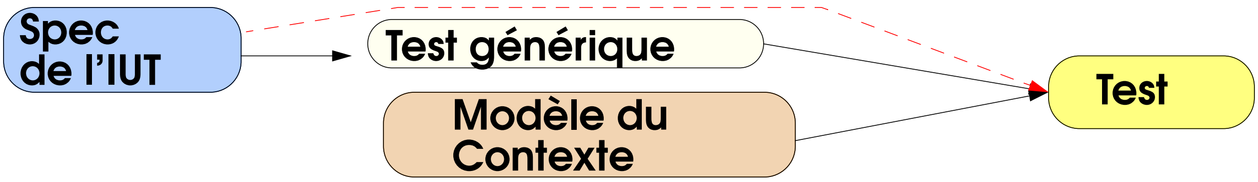
- **Cas général: contrôle et observation indirects**



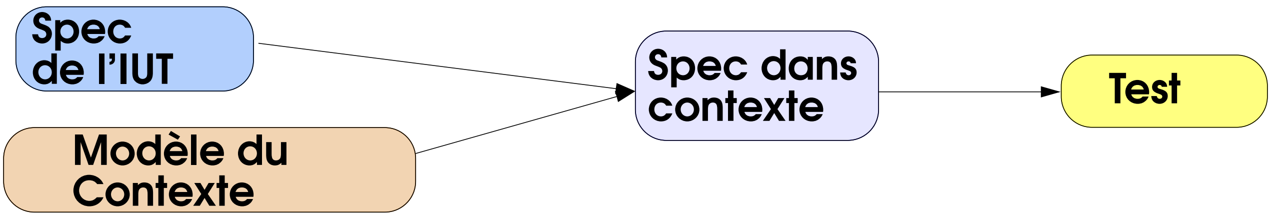
Prise en compte de l'architecture



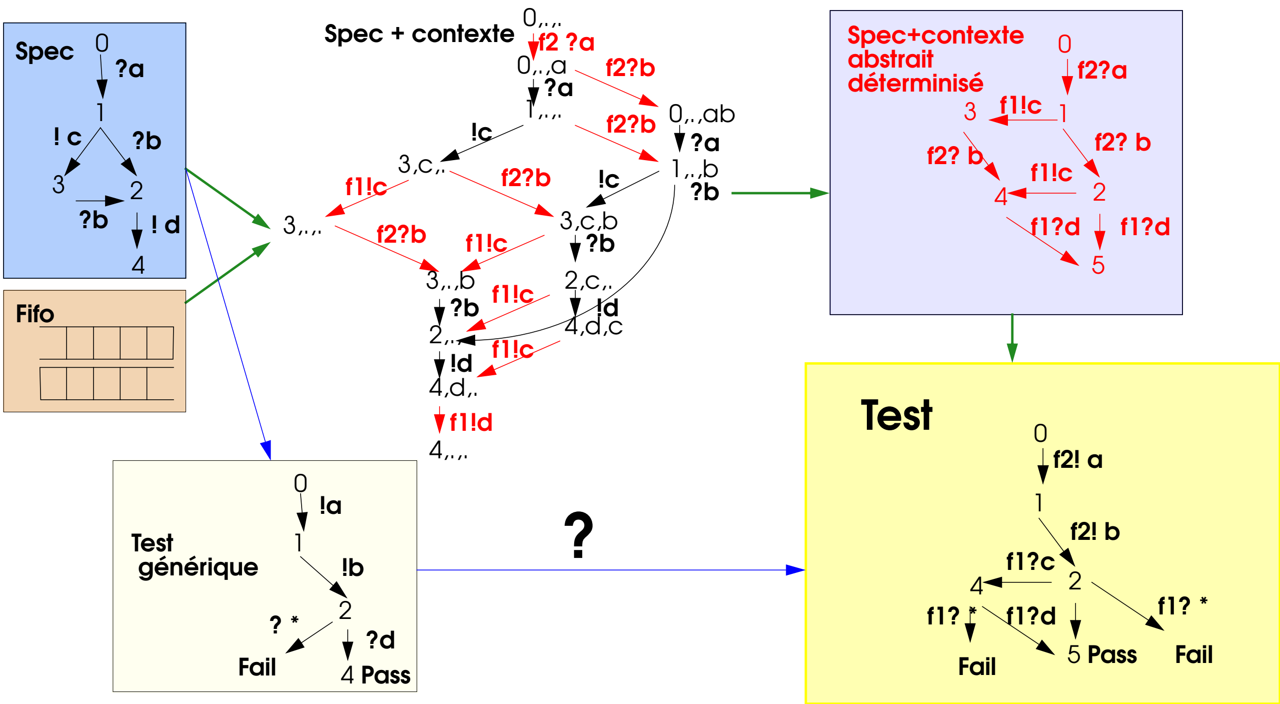
- 1ère vue: transformation d'un test générique par le contexte



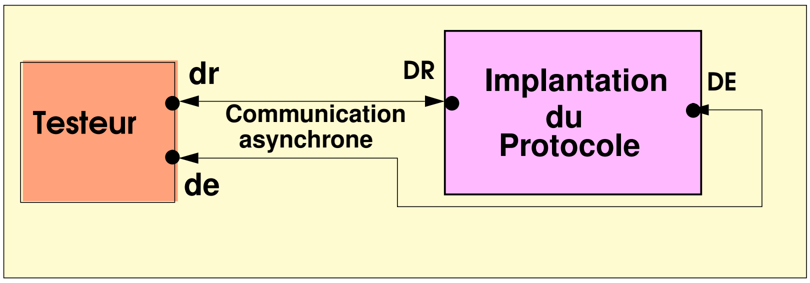
- 2ème vue: conformité de (IUT+contexte) à (Spec+contexte)

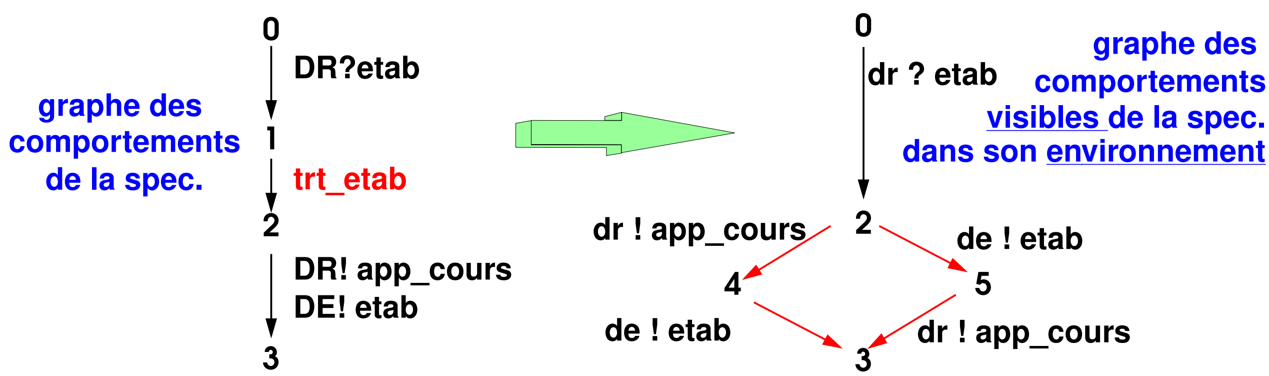


1er exemple du pb de l'asynchronisme



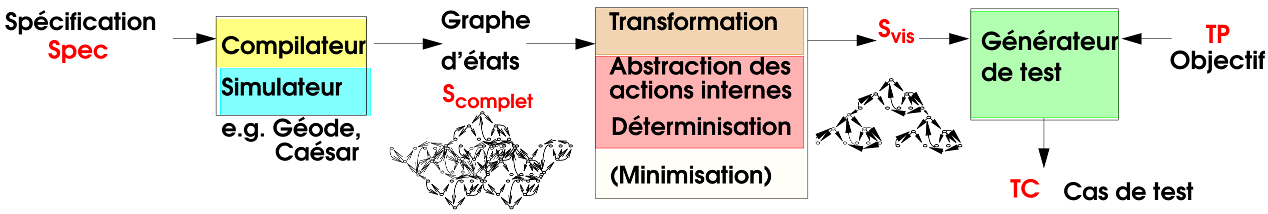
2ème exemple du pb de l'asynchronisme





Génération à la volée

❑ Principe classique de génération



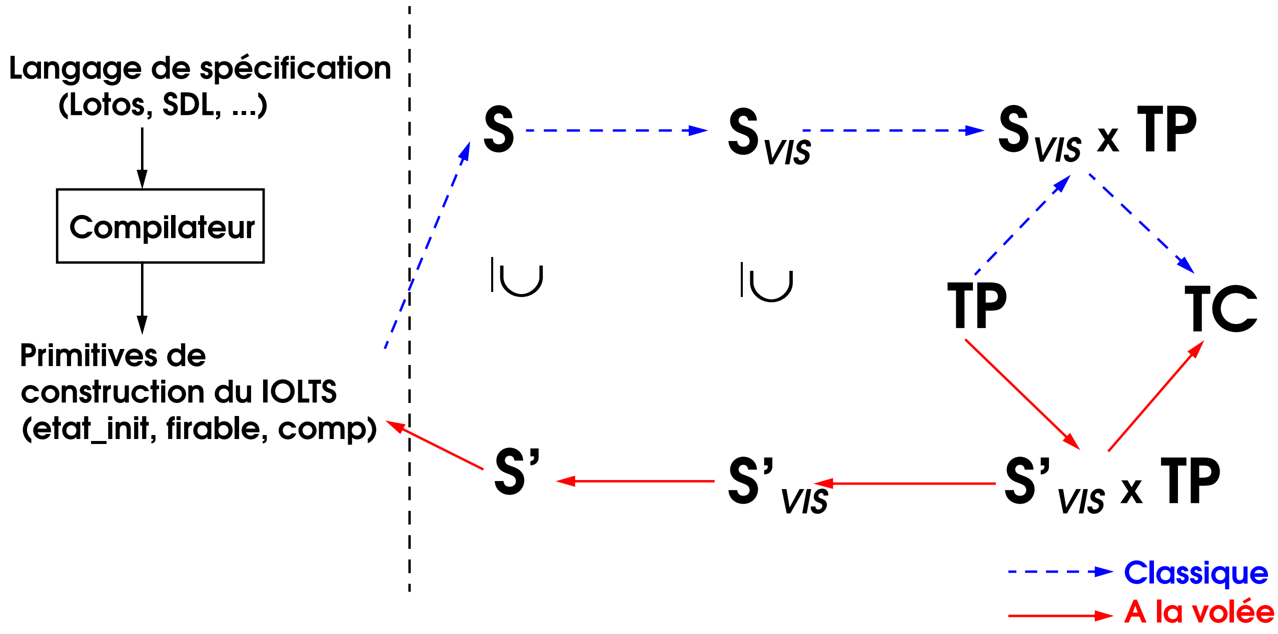
TGV peut s'utiliser ainsi mais peut faire beaucoup mieux :

❑ Génération à la volée:

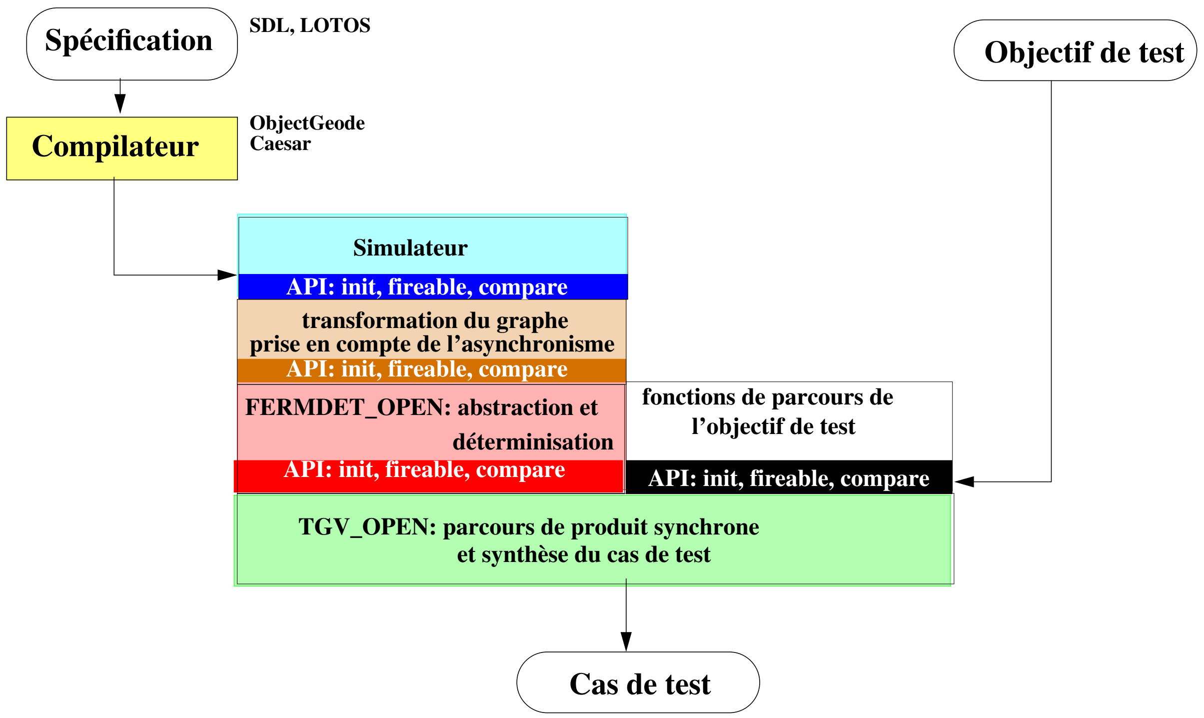
- pour TP donné, TGV ne parcourt qu'une partie de Svis
- idée: générer TC en ne construisant que la partie nécessaire de S_{vis}, et la partie nécéssaire de $S_{complet}$ correspondante.
- comment? en intégrant toutes les phases en une seule par l'utilisation d'API qui pilotent le simulateur.

→ " Construction paresseuse"

Génération classique et à la volée



Principe de TGV à la volée



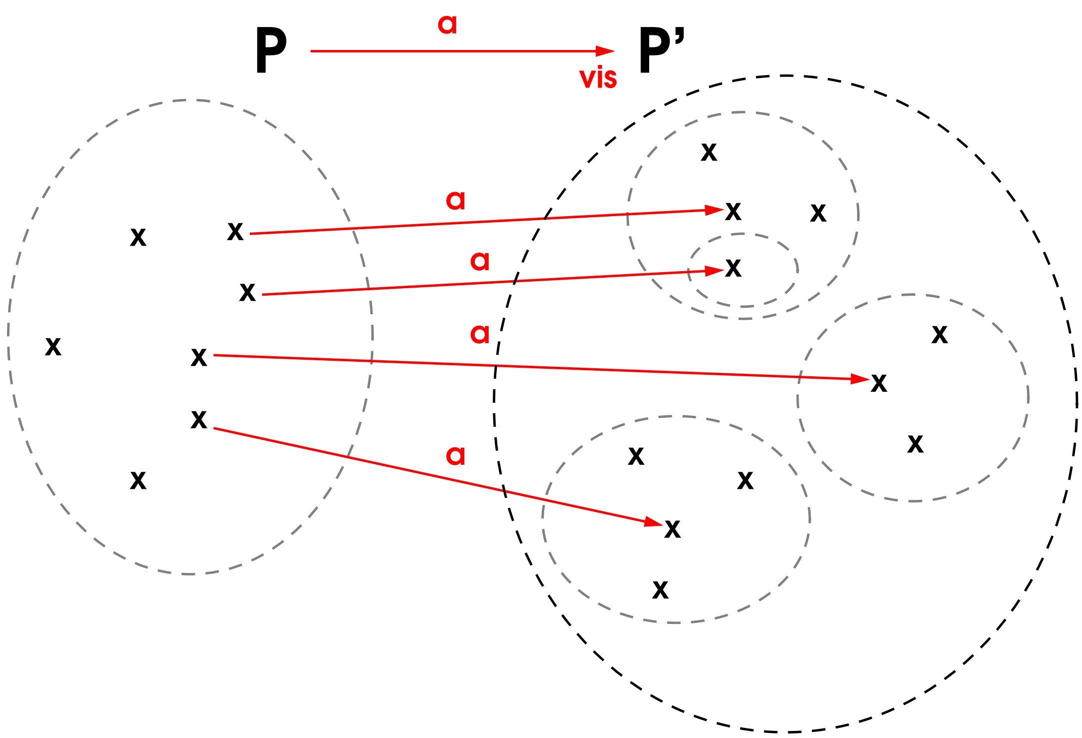
Abstraction et déteminisation à la volée S abstrait et déterminisé : S_{VIS}

- Soit une spécification S = (Q, A, ->, q_{init})
  avec -> $\subseteq$ Q x (A $\cup$ {τ}) x Q
- S_{VIS} (déterministe et abstraite) = (Q_{VIS}, A, $->_{VIS}$, $q_{initVIS}$) :
  - Q_{VIS} est inclus dans 2^Q, l'ensemble des parties de Q, accessibles depuis $q_{initVIS}$ par $->_{VIS}$
  - $q_{initVIS} = \tau^*(q_{init})$
  - $->_{VIS} \subseteq Q_{VIS}$ x A x Q_{VIS} est définie ainsi. Soient P, P' $\in Q_{VIS}$, a $\in$ A, alors :

$$P \text{ -a->}_{VIS} P' \text{ ssi } P' = \bigcup_{\{q' \mid q \in P,\, q \text{ -a-> } q'\}} \tau^*(q')$$

-

Exemple



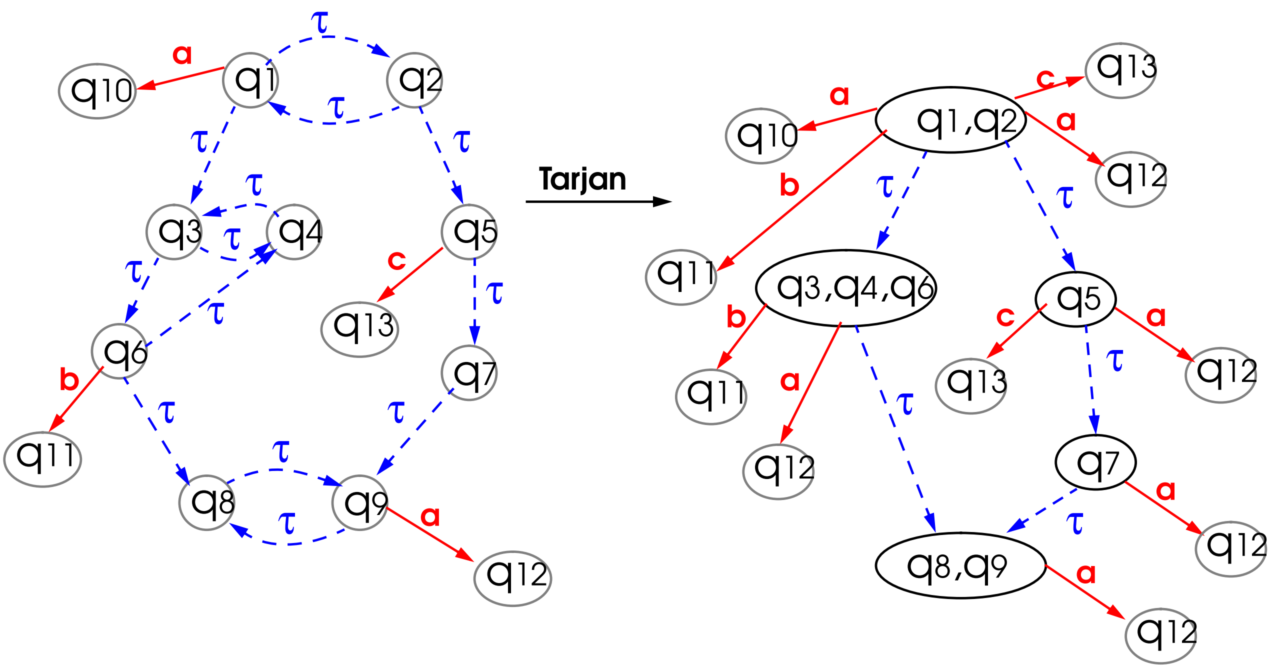
Abstraction : arbre de τ-CFC

- Algorithme de Tarjan adapté
- Calcul de composantes fortement connexes maximales par τ
  - $q \in Q$, τ-CFC (q) = {q' | q=τ=>q' et q'=τ=>q}
- Algorithme de calcul de τ-CFC par parcours en profondeur, construction d'arbres de τ-CFC
  - à la descente, recherche des τ-CFC
  - à la remontée, synthèse des transitions tirables depuis une τ-CFC fille vers sa τ-CFC mère

Pourquoi Tarjan

- Synthèse des transitions tirables sur chaque τ-CFC
- Complexité lineaire sur le nombre d'états de S
- Une τ-CFC est calculee au plus une seule fois

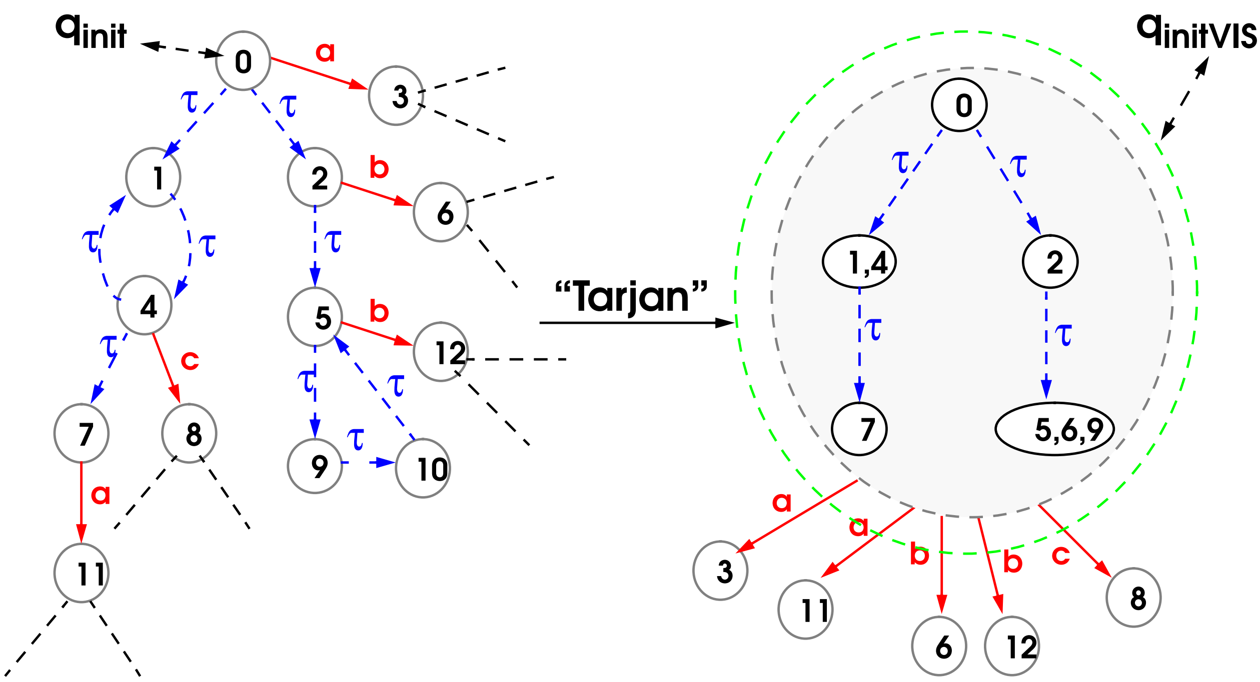
Exemple



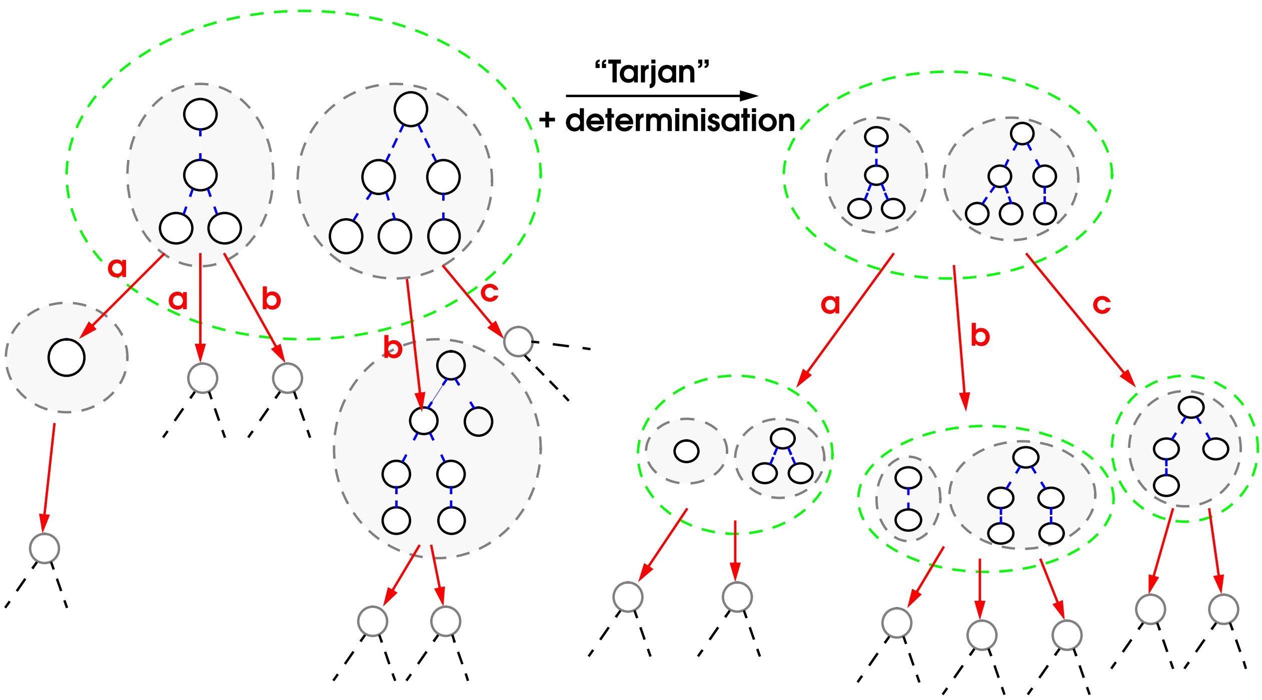
Déterminisation : forêt de τ-CFC

- $q_{initVIS}$ = arbre de τ-CFC de racine la τ-CFC de q_{init}
- Soit P in Q_{VIS}, une forêt de τ-CFC, a in A alors :
  P -a->_{VIS} P' ssi P' est la forêt de τ-CFC que l'on peut atteindre depuis les arbres de τ-CFC de P par l'action visible a

Exemple : calcul de l'état initial



Exemple : calcul des successeurs



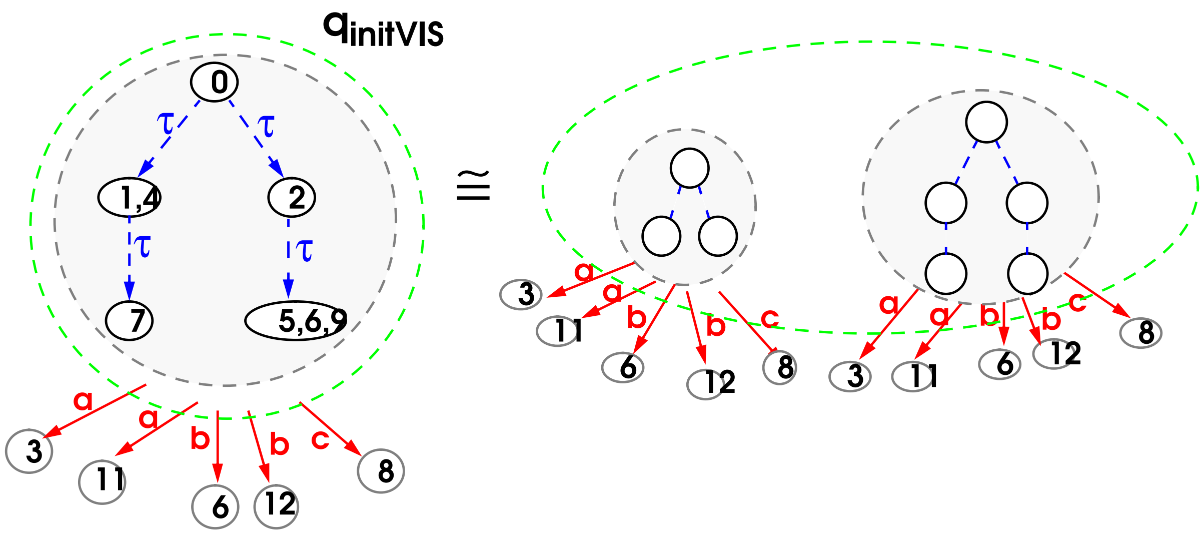
Le problème du postambule

- La génération de postambules d'un cas de test nécessite de pouvoir revenir à l'**état initial** $q_{initVIS}$ de S_{VIS}

- Equivalence pour l'état initial $q_{initVIS}$ **plus faible** :
  - Un état P de Q_{VIS} est jugé équivalent à l'état initial si tout arbre de P possède un ensemble de transitions d'actions et d'états d'arrivée tirables identique à celui de $q_{initVIS}$
  - problème du dépliage

- Equivalence entre deux états : égalité de leur forêt

Exemple : équivalence avec l'état initial



Perspectives scientifiques

- ❑ Prise en compte de l'architecture de test:
  - asynchronisme: pour l'instant trop limité (approximation pour fifo)
  - test dans un contexte
- ❑ Expressivité des objectifs de test
  - rapport entre propriétés de la spec (vérif.) et objectifs
  - quelles sont les propriétés testables?
- ❑ Nouveaux algorithmes pour améliorer la qualité des tests:
  - tests avec cycles (CFC de Tarjan)
  - graphes de refus (synthèse des refusals dans Fermdet)
- ❑ Génération à la volée
  - les nouveaux algorithmes doivent s'intégrer dans l'approche “à la volée” => choix algorithmiques

Perspectives scientifiques (suite)

- ❑ Interopérabilité: architectures de tests avec testeurs répartis
  - littérature très pauvre
  - distribution, synthèse de réseaux, observation répartie
- ❑ Génération automatique de tests exécutables
  - plateforme d'exécution de tests
- ❑ Vérif et test de logiciels objets distribués
  - développement objet (OMT/UML) et validation
- ❑ Test de programmes synchrones
  - utilisation de la synthèse de contrôleurs
  - passage au monde asynchrone

Bibliographie

❑ Normes

OSI-Open Systems Interconnection, Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework - Part 1 : General Concept - part 2 : Abstract Test Suite Specification - part 3 : the Tree and Tabular Combined Notation (TTCN). International Standard ISO/IEC 9646-1/2/3, 1992.

Z.100 CCITT/SGx/WP3-1, Specification and Description Language, SDL, CCITT Recommendation Z.100, 1988

❑ Test des LTS

S. Abramsky. Observational equivalence as a testing equivalence. Theoretical Computer Science, 53(3), 1987.

E. Brinksma, R. Alderden, J. Langerak, R.Van de Lagemaat, and J. Tretmans.A Formal Approach to Conformance Testing. In J. DeMeer, L. Mackert, and W. Effelsberg, editors, Second International Workshop on Protocol Test Systems, p 349-363, North Holland, 1990.

E. Brinskma. A theory for the derivation of tests. In Protocol Specification, Testing and Verification VIII, pages~63--74, North-Holland, 1988.

K. Drira, P. Azéma, and F. Vernadat. Refusal graphs for conformance tester generation and simlification: a computational framework. In Protocol Specification, Testing and Verification XIII, IFIP Transactions, North-Holland, 1993.

R. De Nicola and M. Henessy. Testing equivalences for processes. Theoretical Computer Science, 34:83--133, 1984

J. Tretmans. A formal approach to conformance testing. PhD thesis, University of Twente, Enschede, The Netherlands, 1992.

J. Tretmans. Testing Labelled Transition Systems with Inputs and Outputs. Int. Workshop on Protocols Test Systems, Evry - FRANCE, September 1995.

J. Tretmans. Test Generation with Inputs, Outputs and Quiescence. In T. Margaria and B. Steffen, editors, Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'96, Passau, Germany, March 1996.

L. Verhaard, J. Tretmans, P. Kars, and E. Brinksma. On asychronous testing. In G. Von~Bochman, R. Dssouli, and A. Das, editors, Fifth International Workshop on Protocol Test Systems, IFIP Transactions, North Holland, 1993.

.

❑ Test des FSM

A.R. Cavalli, S.U. Kim, and Maigron P. Automated Protocol Conformance Test Generation Based on Formal Methods for LOTOS Specifications. In Elsevier Science Publishers B.V., editor, Protocol Test Systems V, p212-222, Montréal, 1993.

T. Chow. Testing software design modelled by finite-state machines. IEEE Transactions on Software Engineering, SE-4(3), may 1978.

S. Fujiwara, G. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. IEEE Transactions on Software Engineering, 17(6), june 1991.

G. Gonenc. A method for the design of fault-detection experiments. IEEE Transactions on Computing, C-19, june 1970.

K Sabnani and A. Dahbura. A protocol testing procedure. Computer Network and ISDN Systems, 15(4), 1988.

S. Vuong, W. Chan, and M. Ito. The UIOv method for protocol test sequence generation. Second International Workshop on Protocol Test Systems, Berlin, October 1989.

❑ Test des IOSM

M. Clatin, R. Groz, M. Phalippou, and R. Thummel. Two approaches linking a test generation tool with verification techniques. In 8th Int. Workshop on Protocols Test Systems, Evry - FRANCE, September 1995.

M. Phalippou. Test sequence using Estelle or SDL structure information. In FORTE'94, Berne, October 1994.

M. Phalippou. Relations d'implantations et Hypoth\`eses de Test sur des automates à entrées et sorties. PhD thesis, Université de Bordeaux, France, 1994.

❑ TGV

J.-C. Fernandez, C. Jard, T. J\'eron, and G. Viho. Using on the fly verification techniques for the generation of test suites, Computer Aided Verification, CAV 96, New Brunswick(USA), juil. 96, LNCS 1102.Rapport Inria no 2987.

J.-C. Fernandez, C. Jard, T. J\'eron, and G. Viho. An experiment in automatic generation of test suites for protocols with verification technology. Science of Computer Programming. Special Issue on Industrial Relevant Applications of Formal Analysis Techniques. A paraitre. Rapport Inria 2923, Irisa no 1035

T. Jéron, P. Morel: Abstraction et déterminisation à la volée: application à la génération de test. A paraitre dans CFIP'97 Ingéniérie des Protocoles. Sept 97, Liège, Belgique.

L. Doldi, V.Encontre, J.C.Fernandez, T.J\'eron, S. Le Bricquir, N.Texier and M. Phalippou. Assessment of automatic generation methods of conformance test suites in an industrial contex. In IFIP TC6 9th International Workshop on Testing of Communicating Systems. Baumgarten, B., Burkhardt, H.-J. and Giessler, A. Chapman & Hall. Sep 1996.

❑ Divers

C. Chevrier, R. Castanet, B. Le Saëc, R. Casadessus. Génération de séquences de test par calcul de monoïdes. CFIP'95 Ingéniérie des Protocoles. C. Jard et P. Rollin éditeurs. Hermès. Rennes, Mai 95.

J. Grabowski, D. Hogrefe, and R. Nahm. Test case generation with test purpose specification by MSCs. In Elsevier Science B.V. (North-Holland), editor, 6th SDL Forum, p253-266, Faergemand, O. and Sarma, A., Darmstadt , 1993.

R. Lai and W. Leung. Industrial and academic protocol testing : the gap and the means of convergence. Computer Networks and ISDN Systems, 27:537--547, 1995.