La génération de tests (pour les protocoles) à l'aide de la technologie de la vérification

J.-C. Fernandez (Spectre/Vérimag/INRIA-CNRS),

C. Jard (Pampa/Irisa/CNRS),

T. Jéron (Pampa/Irisa/INRIA),

G. Viho (Pampa/Irisa/IFSIC)

http://www.irisa.fr/pampa/

Le test de conformité

- **Problème:** vérifier qu'une implantation (de protocole) est conforme à sa spécification.
- **Test boîte noire:** implantation connue uniquement par ses interactions avec l'environnement.
- **Pourquoi ?**

Interopérabilité
$\Uparrow$
Protocoles normalisés
$\Uparrow$
Implantations conformes
$\Uparrow$
Test de conformité

- **Comment ?** stimuler l'implantation, observer son comportement, en déduire la conformité vis à vis de sa spécification

Méthodes formelles et test de conformité

- **Questions**
  - qu'est ce que veut dire conforme ?
  - comment générer des tests ?
  - comment discuter la qualité des tests ?

- **Avantages d'une approche formalisée**
  - définition précise de la conformité et des différents concepts associés au test
  - algorithmes et méthodes automatiques pour générer des tests
  - validation des tests générés
  - comparaison des méthodes de génération
  - normalisation des algorithmes plutôt que des tests

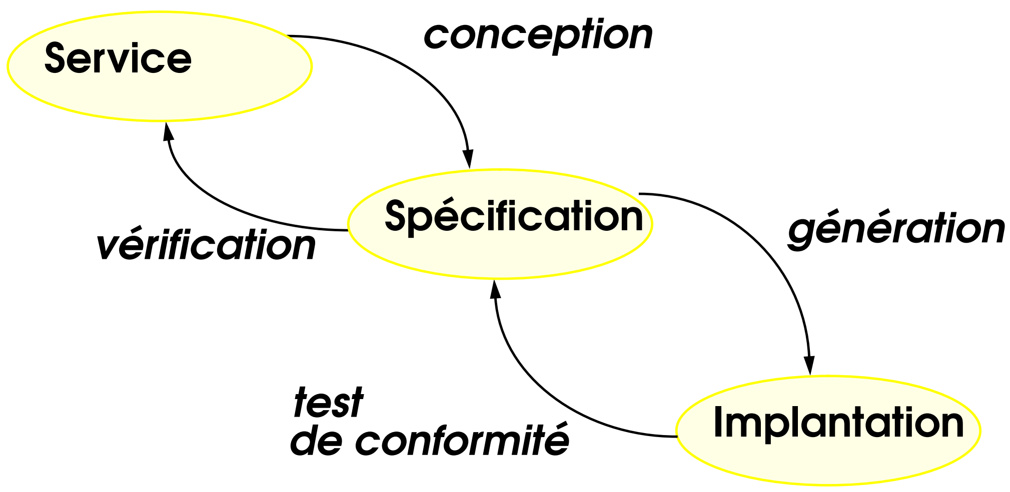
En pratique...

- On teste l'implantation vis à vis de sa spécification formelle (donnée dans un langage de description formelle), et par rapport à une définition formelle de la conformité.

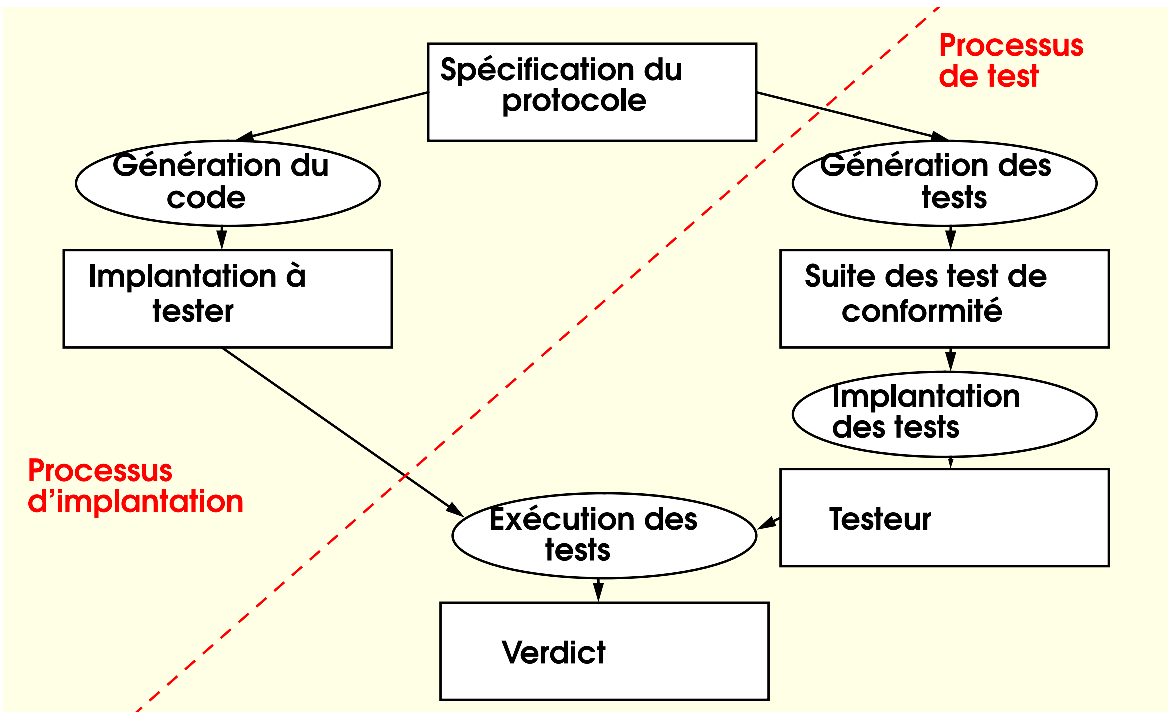
⇨ *Donner un sens précis aux verdicts des tests*

- Fournit un complément à l'approche informelle ISO-9646 (Conformance Testing Methodology and Framework)
- Fait l'hypothèse de la correction de la spécification formelle

Le test et la vérification dans le processus de conception



IS-9646 : le processus de test



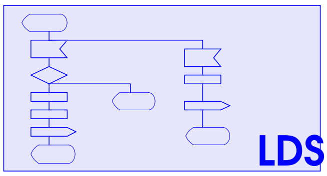
Constat sur la génération manuelle

- La génération manuelle de tests est un processus répétitif
- Jusqu'à 20% des tests générés manuellement sont erronés
- La couverture des tests est incertaine
- Difficulté de la maintenance évolutive

⇨ **AUTOMATISER LA GÉNÉRATION DE TESTS À PARTIR DES SPÉCIFICATIONS LDS**

L'étude EGT/STEI

Verilog, Cap Sesa, Cnet, Inria Grenoble et Rennes



Modèle LDS du DREX (2000 lignes)
+
Tests de référence TTCN (50 cas de test, 1-2 p. chaque)

Méthodes dynamiques

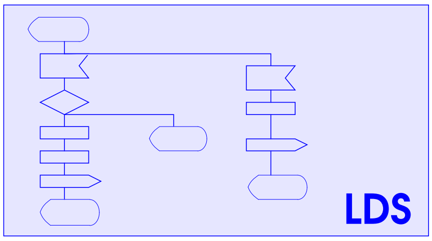
- Topic (Verilog)
- TGV (Inria)

Méthodes statiques

- TVeda (Cnet)

Tests TTCN
Rapport d'analyse comparée des méthodes

TVEDA (CNET)



Méthodes statiques

- ❑ Tour des transitions de l'automate étendu
- ❑ Test à un pas

90% des tests manuels

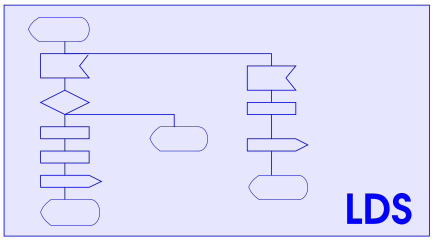
- ❑ Manipulations symboliques
- ❑ Un seul processus

TOPIC(Verilog)

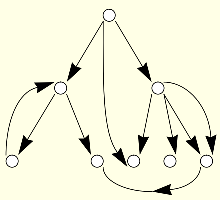
Objectifs de tests

Observateurs ou MSCs

+

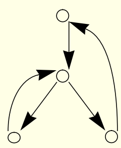


Graphe contraint



Projection

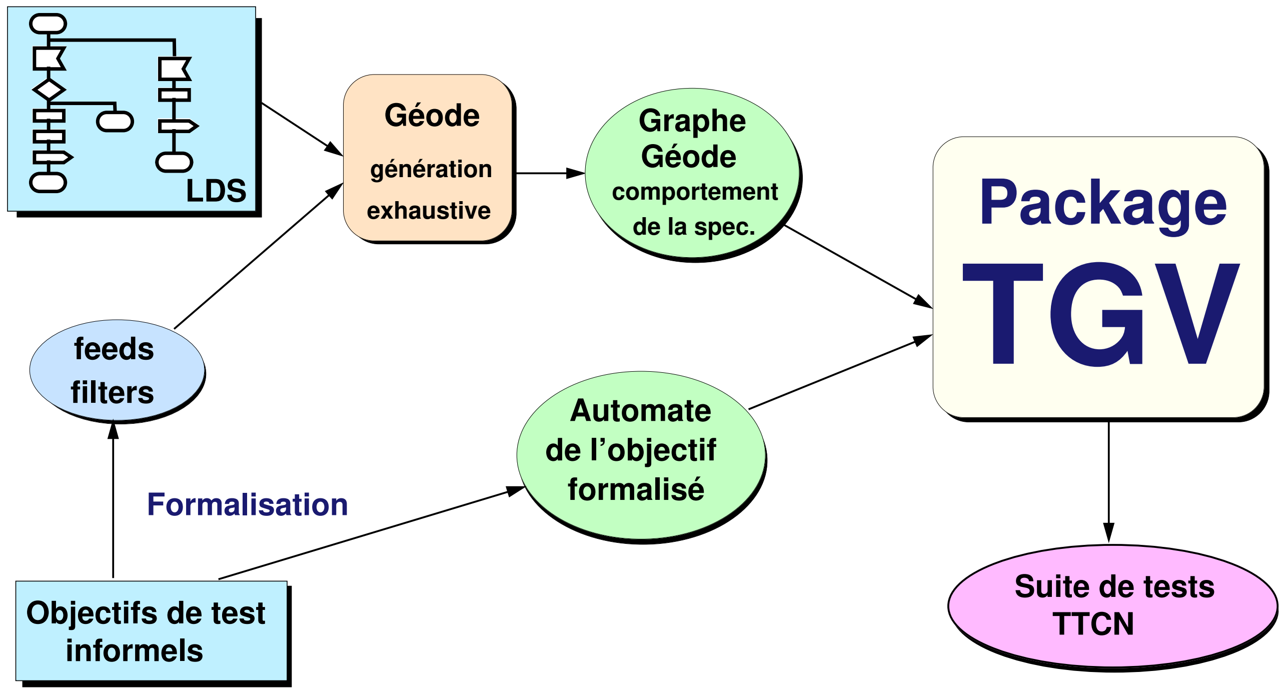
Réduit



Découpage

TTCN

TGV (INRIA)

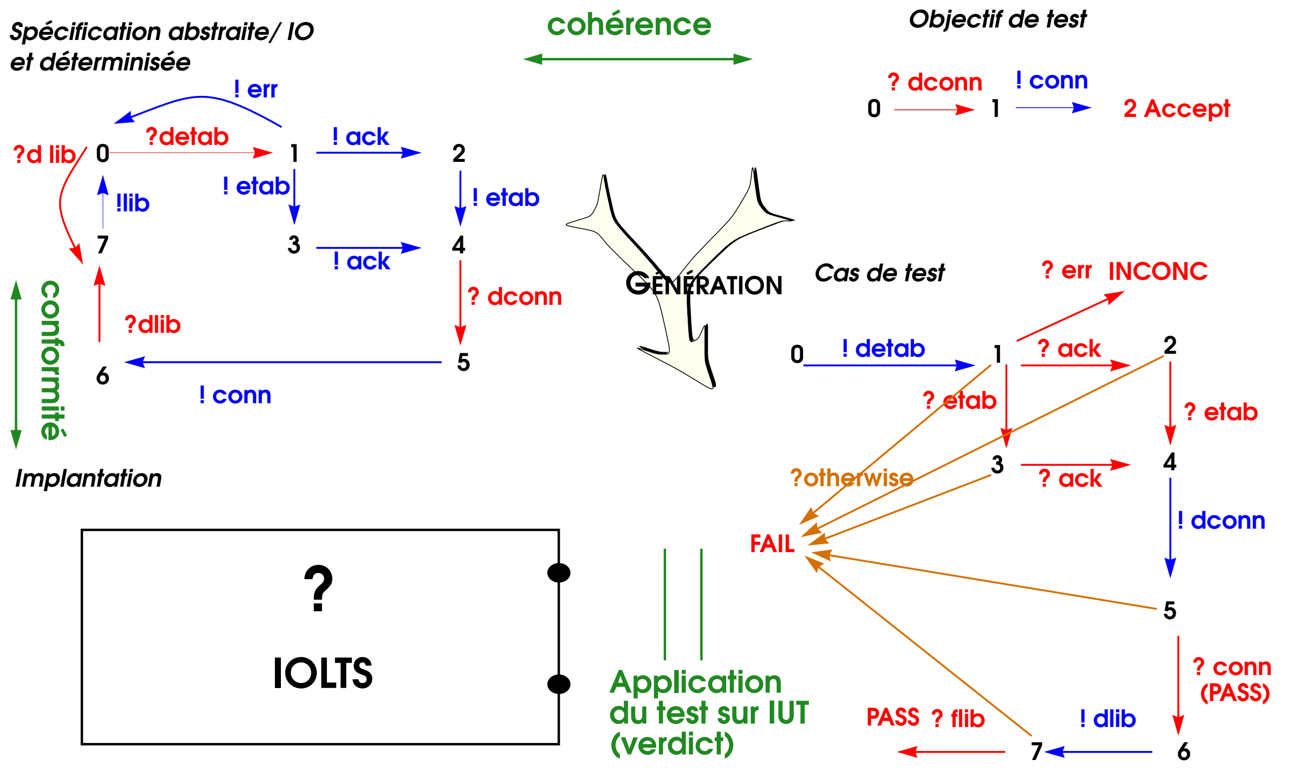


Principes de la génération de test dans TGV

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
- **Algorithmique** fondé sur la vérification

Modélisation



Système 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 sans cycle, d'alphabets identiques à 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.

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

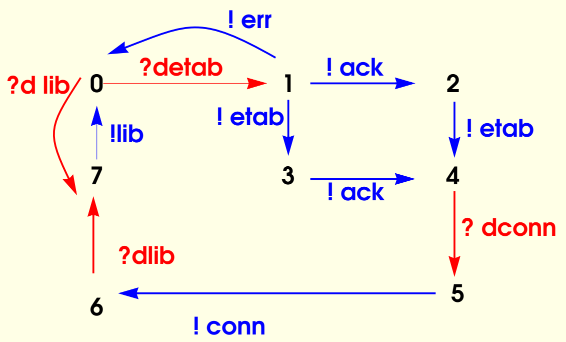
Relation de conformité

- Une relation de conformité est une relation liant une implantation I à une spécification S et décrivant formellement quelles sont les implantations I conformes à S.
- Plusieurs choix sont possibles. La relation *ioconf* choisie tient compte de la différence entre événements observables et contrôlables.

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

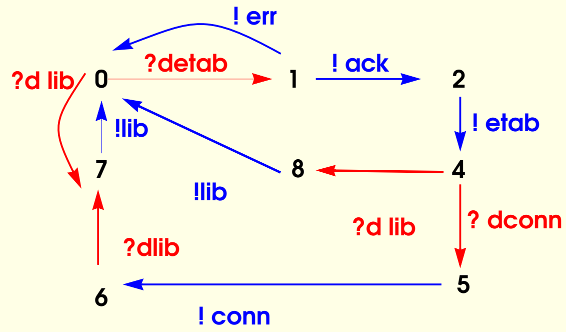
Conformité : exemples

Spécification S

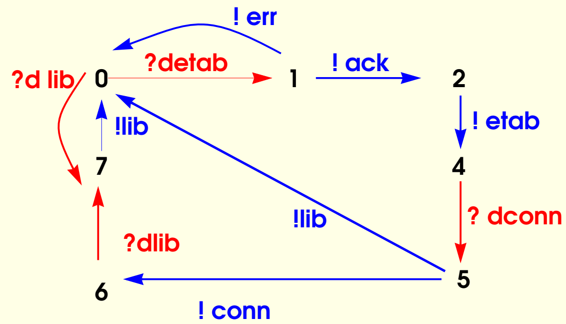


Implantations

I_1



I_2



I_1 *ioconf* S

I_2 *not ioconf* S

Forme des cas de tests générés

- Contraintes : paramètres des messages du cas de test.
- 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.
- Verdicts: associés à certaines interactions avec la signification:
  - FAIL: non-conformité.
  - (PASS): un état accepteur de 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 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 (I)

Nr	Behaviour Description	Constraints Ref	Verdict
1	dr! detab, Start terr, tack, tetab	detab0	
2	dr? etab, Cancel tetab, terr	etab1	
3	dr? ack, Cancel tack		
4	dr! dconn, Start tconn	dconn2	
5	dr? conn, Cancel tconn	conn3	(PASS)
6	dr! dlib, Start tlib	dlib4	
7	? lib, Cancel tlib	lib5	PASS
8	? tlib		FAIL
9	? tconn		FAIL
10	? tack		FAIL
11	dr? ack, Cancel tack, Cancel terr		
12	dr? etab, Cancel tetab	etab1	
13	dr! dconn, Start tconn	dconn2	
14	dr? conn, Cancell tconn	conn3	(PASS)
15	dr! dlib, Start tlib	dlib4	
16	dr? lib, Cancel tlib	lib5	PASS
17	? tlib		FAIL
18	? tconn		FAIL
19	? tetab		FAIL
20	dr? err, Cancel tetab, tack, terr		INCONC
21	? tetab		FAIL
22	? tack		FAIL
23	? terr		FAIL

Principe de l'algorithme

- Algorithme décrit complètement par des règles opérationnelles (à la Plotkin).
- Construction d'un DAG (Graphe Dirigé Acyclique) par parcours en profondeur d'un produit entre S et TP:

  à la descente:

  - 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

  - on synthétise un DAG constituant l'ossature du cas de test TC en assurant la contrôlabilité et en ajoutant des verdicts sur certaines transitions.
- Un deuxième parcours en profondeur du DAG résultant permet de décorer le DAG par la gestion des timers.
- Le DAG est ensuite déplié en arbre pour être traduit au format TTCN.

Principe de gestion des timers

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

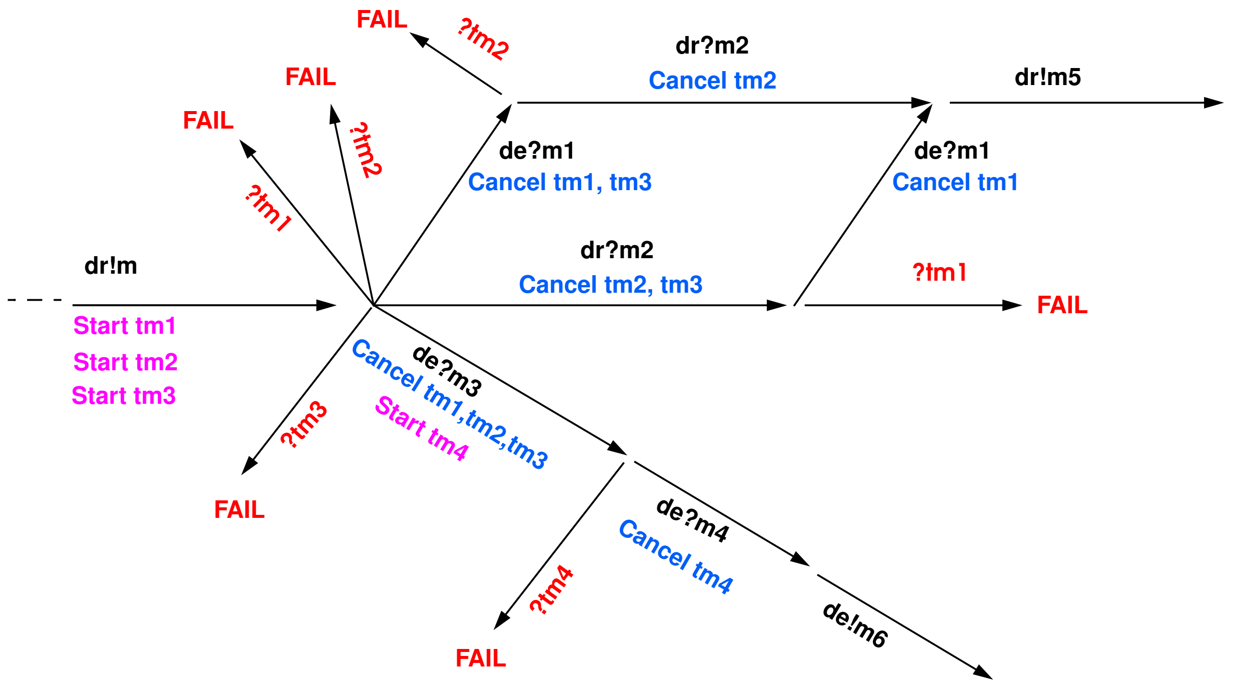
- **Propriétés attendues:**

  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 :
  - Start t_i : doit être fait dans toute transition atteignant le premier état où l'entrée i est possible.
  - 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.
  - 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 : exemple



Résultats théoriques

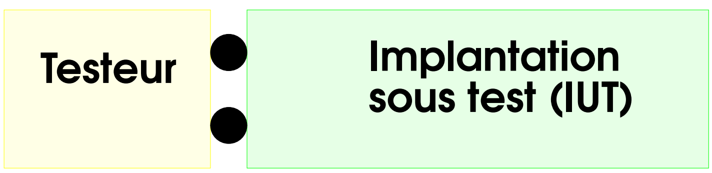
- **Complexité:** linéaire en temps et mémoire par rapport à **TPxS**
- **Vérification de la cohérence :** L'algorithme appliqué à **TP** et **S**, rend vrai et construit un cas de test **TC(TP,S)** et seulement si **TP** est cohérent avec **S**.

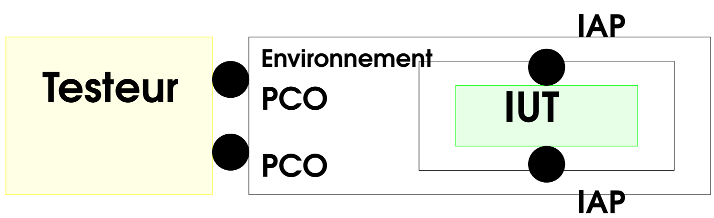
  TC(TP,S) comprend un préambule, un postambule, des verdicts et la gestion des timers.
- **Correction :** si l'application d'un cas de test **TC(TP,S)** généré par l'algorithme sur une implantation **I** produit un verdict **FAIL** alors l'implantation **I** est non conforme à la spécification **S** pour la relation **ioconf**.
- **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 que le cas de test **TC(TP,S)** généré par l'algorithme et appliqué **n** fois sur **I** produise un verdict **FAIL**.

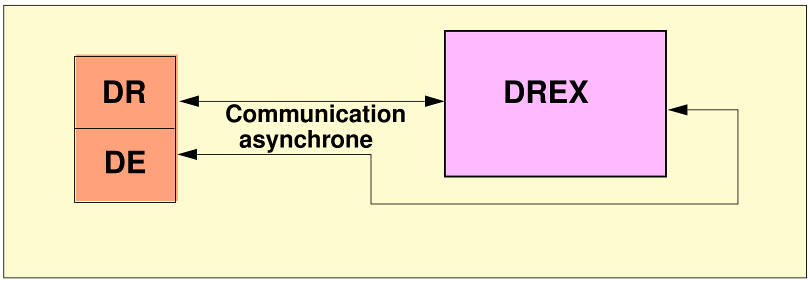
Impact de l'architecture

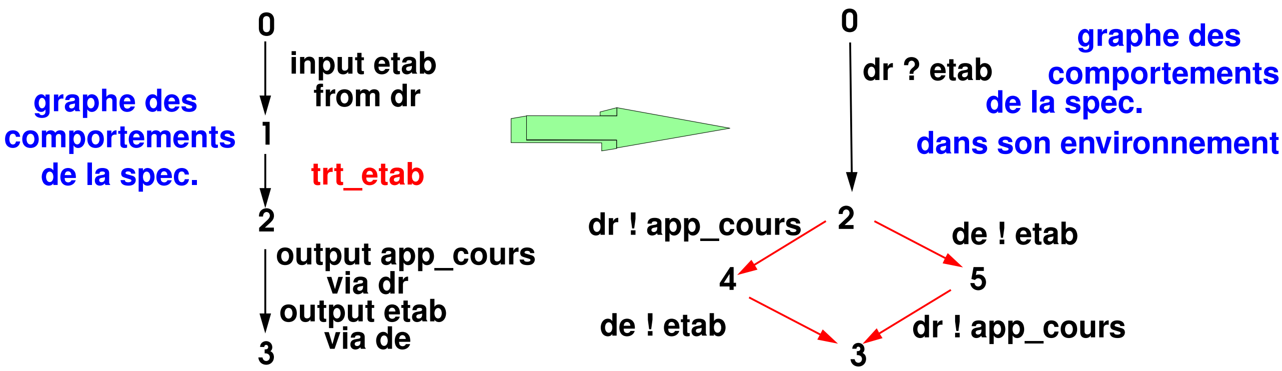
- **Contrôle et observation indirects**



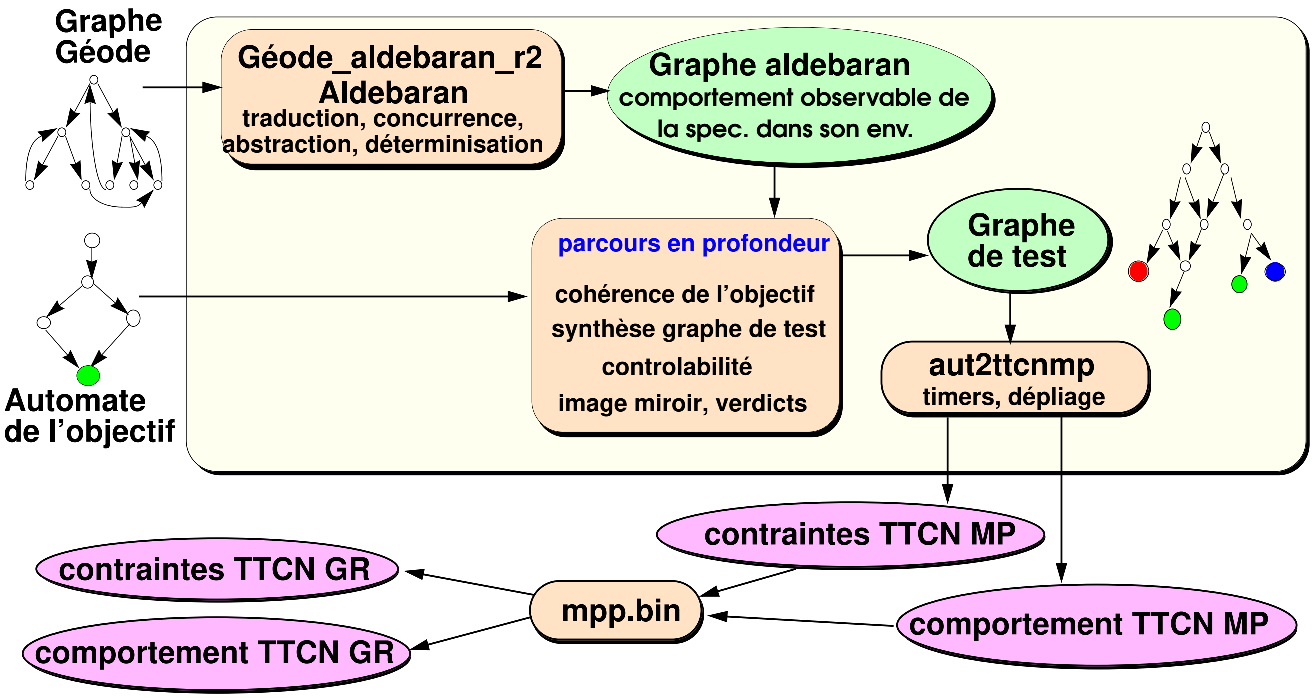


Prise en compte de l'asynchronisme





Le package TGV



Comparaison avec les tests manuels:

meilleure confiance dans les tests générés

- Verdicts: fiables car correspondant à la relation de conformité
- Timers: gestion automatique évoluée
- Préambules: pas toujours les plus courts
- Postambules:
  - pour nous, pas de postambule après FAIL
  - ni INCONC (à rajouter)
  - plus court chemin.
- Problèmes de modélisation des objectifs: inévitable du fait de l'ambiguïté du langage naturel.

Apports de TGV

- Formalisation du développement automatique de suites de test
  - utilisation d'une spécification LDS.
  - formalisation des objectifs de tests.
  - formalisation des verdicts PASS, FAIL, INCONCLUSIVE
  - prise en compte de la concurrence due à l'asynchronisme.
  - modélisation de la gestion des timers.
- Utilisation de l'algorithmique de la vérification
  - abstraction préservant les entrées/sorties de la spec.
  - calcul des *losanges* modélisant la concurrence.
  - parcours de produit synchrone de graphes.
  - synthèse à la volée du graphe de test.

Génération à la volée

- **Problème:** taille du graphe des comportements de la spécification.
- **But:** produire les tests pendant la construction du graphe.
- **Comment?**

$\Rightarrow$ nécessité d'intégrer l'abstraction, la déterminisation dans la construction du graphe.

$\Rightarrow$ arrêt à profondeur donnée.

Expérience en cours pour LOTOS dans CADP et à venir pour LDS dans ObjectGEODE.

Résultats obtenus

- Le prototype TGV
  - réalisé rapidement dans l'environnement CADP.
  - met l'accent sur l'aspect comportemental des suites de test (aspect donnée insuffisant).
  - montre la maturité de la méthode automatique utilisée.
- Analyse des suites de test produites par TGV pour le DREX
  - rapidité de la génération : quelques secondes.
  - syntaxiquement proches des suites manuelles : préambule, corps, postambule, timers, contraintes.
  - évitent certaines erreurs intrinsèques aux méthodes manuelles.

Perspectives

- Etude de la commercialisation d'un outil de test autour du simulateur ObjectGEODE (Verilog), intégrant les avantages de TVEDA3 (Cnet) et TGV:

génération à la volée, prise en compte de l'environnement de test, gestion des timers, objectifs de tests MSC, ou générés automatiquement par calcul symbolique.

- Améliorations de TGV:
  - extension pour LOTOS en cours
  - génération à la volée
- Etudes de cas:
  - Fast Reservation Protocol (ATM)
  - protocole de cache mémoire (Bull)
- Test d'interopérabilité: distribution d'un cas de test sur plusieurs testeurs.