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

J.-C. Fernandez (Spectre/Vérimag/INRIA),
C. Jard (Pampa/Irisa/CNRS),
T. Jéron (Pampa/Irisa/INRIA),
G. Viho (Pampa/Irisa/IFSIC)

- Projet EGT/STSIE/DGA
  - Etude sur la génération (automatique) de tests de conformité pour les protocoles.
    Evaluation de techniques formelles.
  - Vérilog - Cap-Sesa-Régions - CNET - INRIA - Celar

Le test de conformité

- Pourquoi ?

Interopérabilité
⇑
Protocoles normalisés
⇑
Implantations conformes
⇑
Test de conformité

- Comment ?

  - la vérification par test que l'implantation (de protocole) possède un comportement conforme à 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

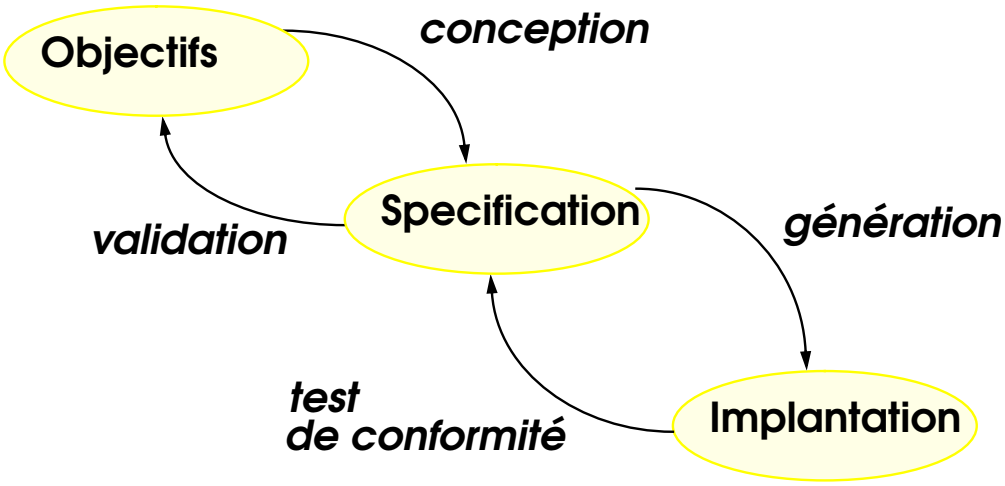
En pratique ...

- 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.

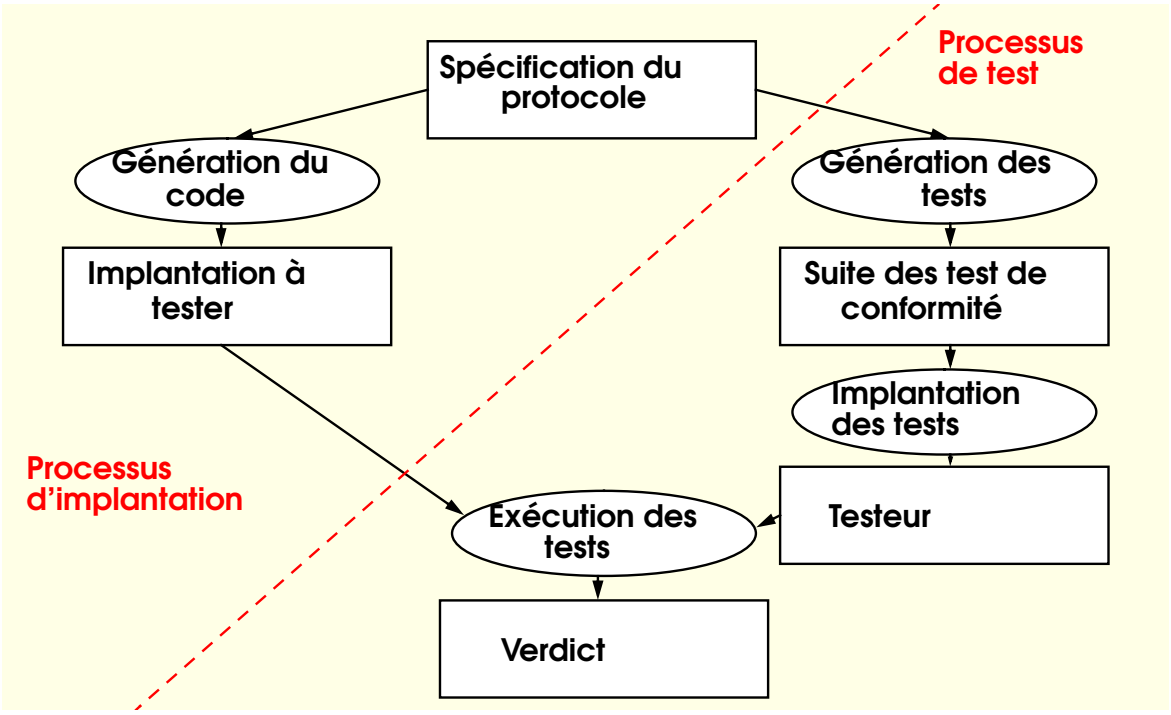
⇨ *Donner un sens précis aux verdicts des tests*

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

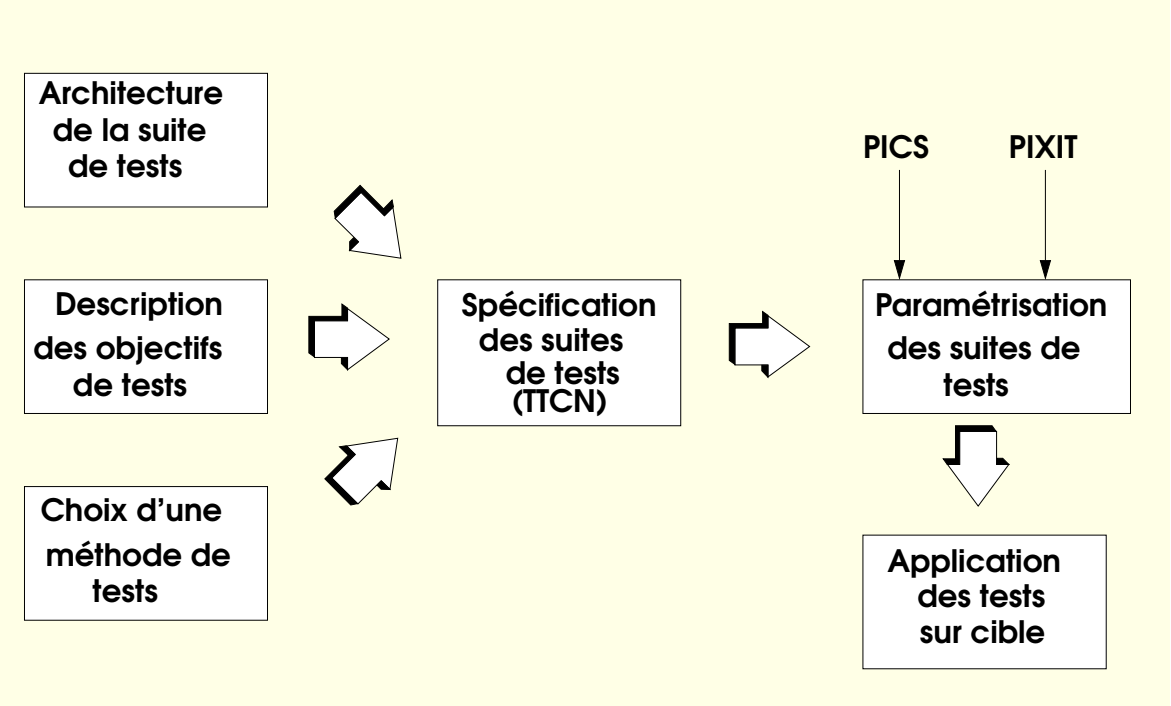
La méthodologie IS-9646 : processus de conception



IS-9646 : le processus de test



IS-9646 : écriture des suites de test



Exemple de test TTCN (I)

Exemple de test TTCN (II)

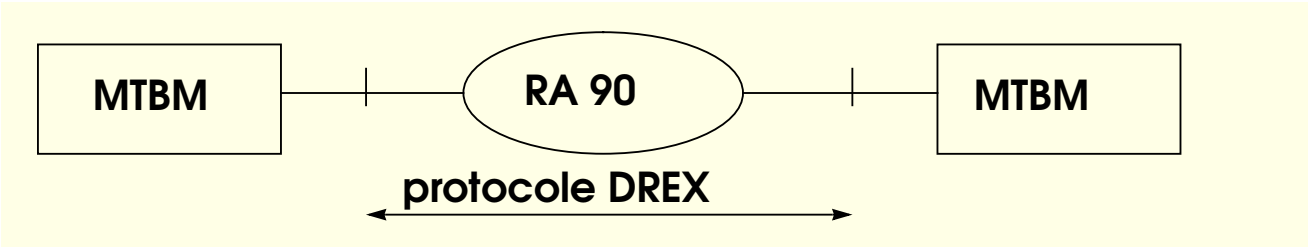
Constat sur la génération manuelle

- 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**

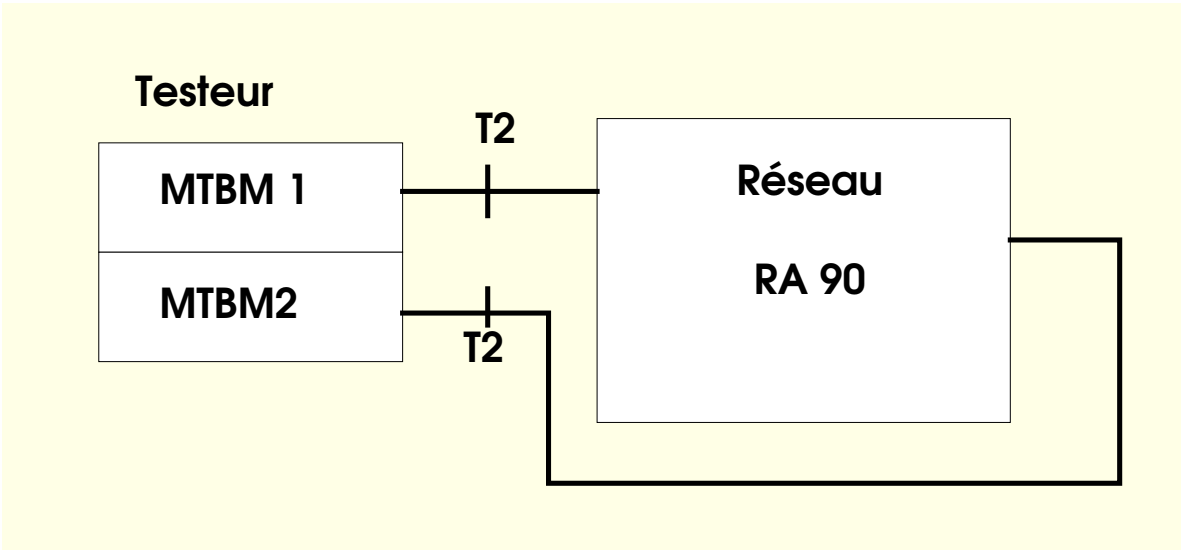
Le protocole DREX

- Protocole de signalisation des futurs réseaux militaires à intégration de services (Priorité, cloisonnement, groupes fermés reconfiguration automatique, sécurité, taux d'erreur, tps de transit, renvoi, abonné individuel déplaçable, identité de l'appelant, signalisation d'usager à usager).



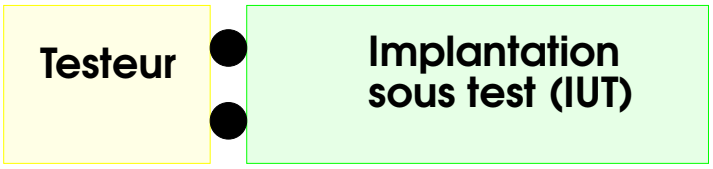
- Description des services en un seul point de référence (T)
  - réseau vu comme une boite noire
  - description de terminal à terminal.
- Spécifications en termes de protocole.

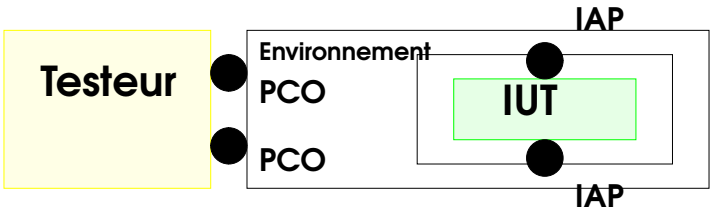
Architecture de test



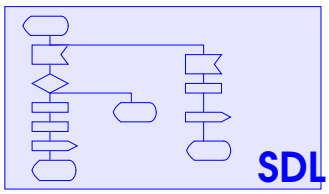
Impact de l'architecture

- Contrôle et observation indirects





L'étude EGT



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

Méthodes dynamiques

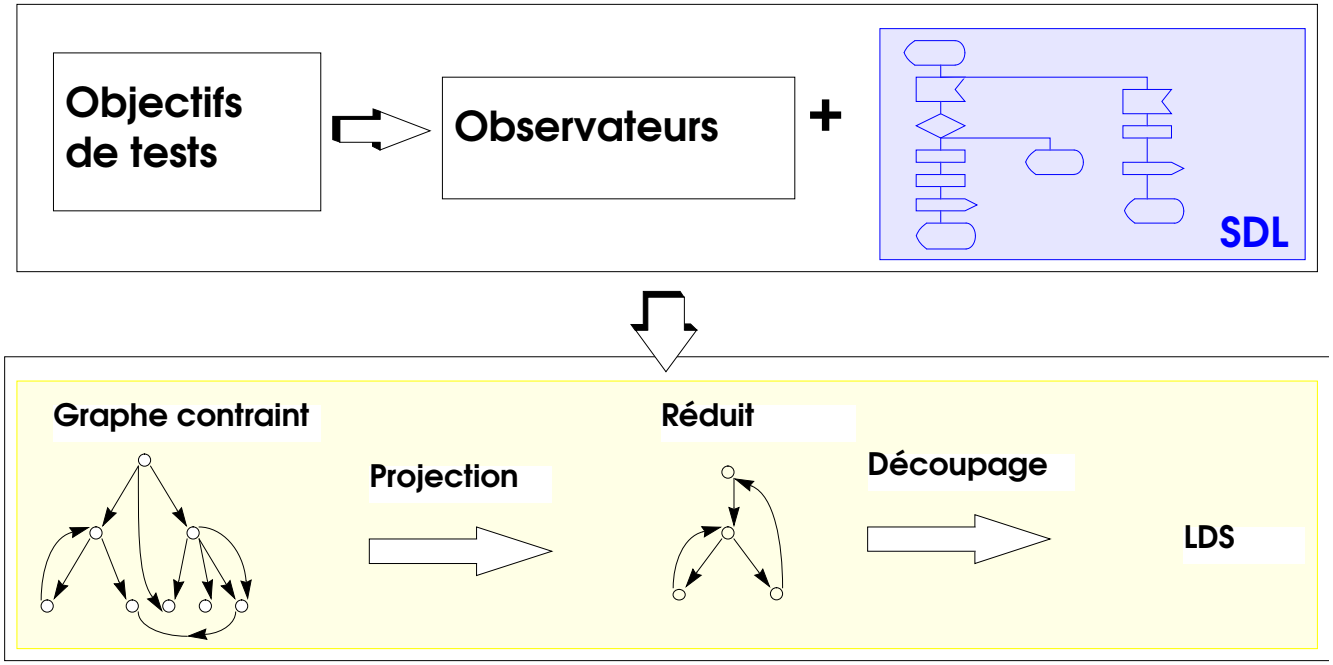
- Topic
- Outils de vérification

Méthodes statiques

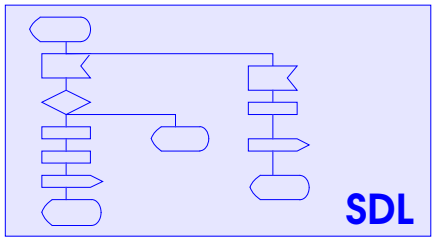
- TVeda

Rapport d'analyse comparée des méthodes

TOPIC(Verilog)



TVEDA (CNET)



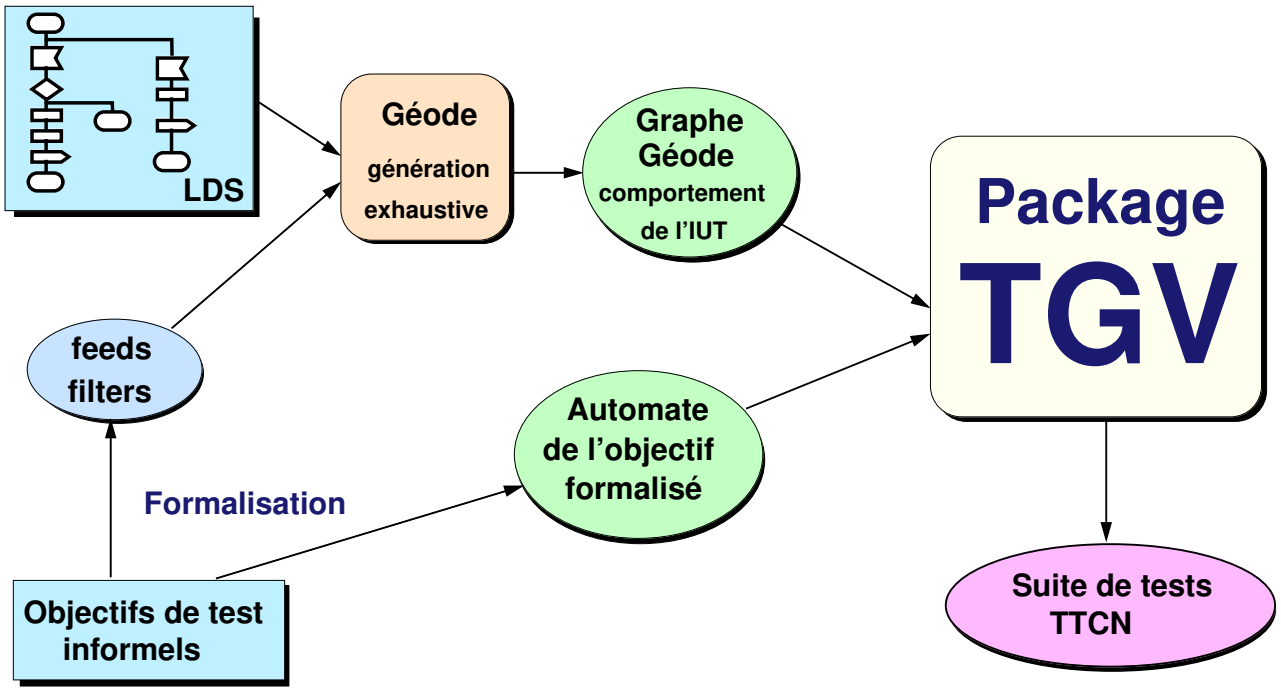
Méthodes statiques

- Tour des transitions
- Test à un pas

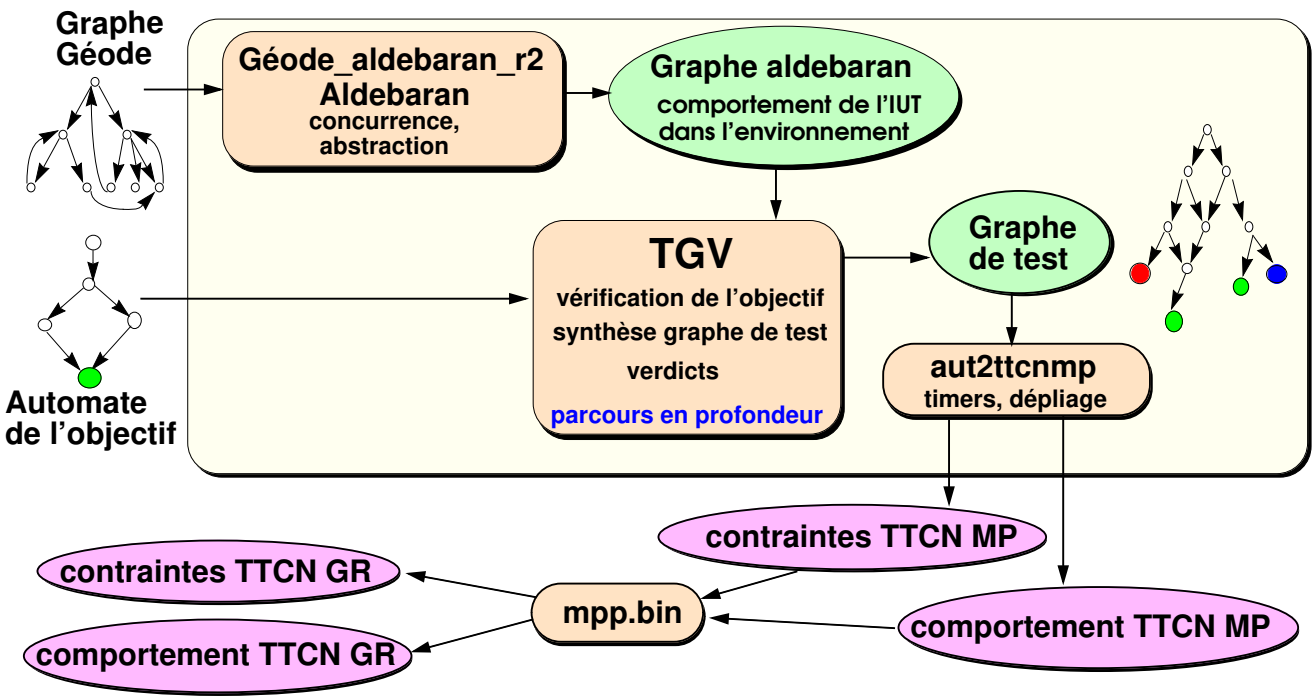
  90% des tests manuels

- Manipulations symboliques
- Un seul processus

TGV (INRIA)



Le package TGV



Algorithme de base

Fondé sur un parcours en profondeur simultané de l'automate de l'objectif et du graphe des comportements possibles du système (testeurs+spec) vu depuis les testeurs.

- **Validité de l'objectif: inclusion de l'automate de l'objectif dans le graphe de la spécification.**
- **Calcul du graphe de test: sous graphe des comportements compatibles avec l'objectif.**

Pourquoi un graphe plutôt qu'un arbre?

- plus d'information qu'un arbre: losanges.
- taille plus petite: pas de duplication de transition.

Graphe décoré (timers) puis déplié dans aut2ttcnmp**.**

Forme des tests générés

- **Contraintes : TTCN obtenu à partir des paramètres des messages de l'arbre de test.**
- **Comportement :** ensemble de séquences d'actions (un arbre ou un DAG) décrivant les interactions entre l'IUT et le testeur, vu du testeur, voulant vérifier la conformité de l'IUT à la specification, par rapport à l'objectif de test. Certaines actions sont décorées par des verdicts:
  - **FAIL:** non-conformité.
  - **(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 est provisoire.
  - **PASS:** l'objectif a été atteint et l'IUT est dans l'état initial. Un **Postambule** est identifié par une séquence entre un verdict (PASS) et un 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 d'objectif : priorité t131010202

Automate aldébaran:

```
des (0,3,4)
(0,"dr!etab(pri = (if_present = true, value = (pri_dr = pri_dr,
pri_de = (if_present = false, value = pri_de), mact = ma_1)))",1)
(1,"de!conn(pri = (if_present = true, value = (pri_dr = pri_dr,
pri_de = (if_present = true, value = pri_de), mact = ma_1)))",2)
(2,"dr?conn(pri = (if_present = true, value = (pri_dr = pri_dr,
pri_de = (if_present = true, value = pri_de), mact = ma_1)))",3)
PASS 3
```


Principe de l'algorithme

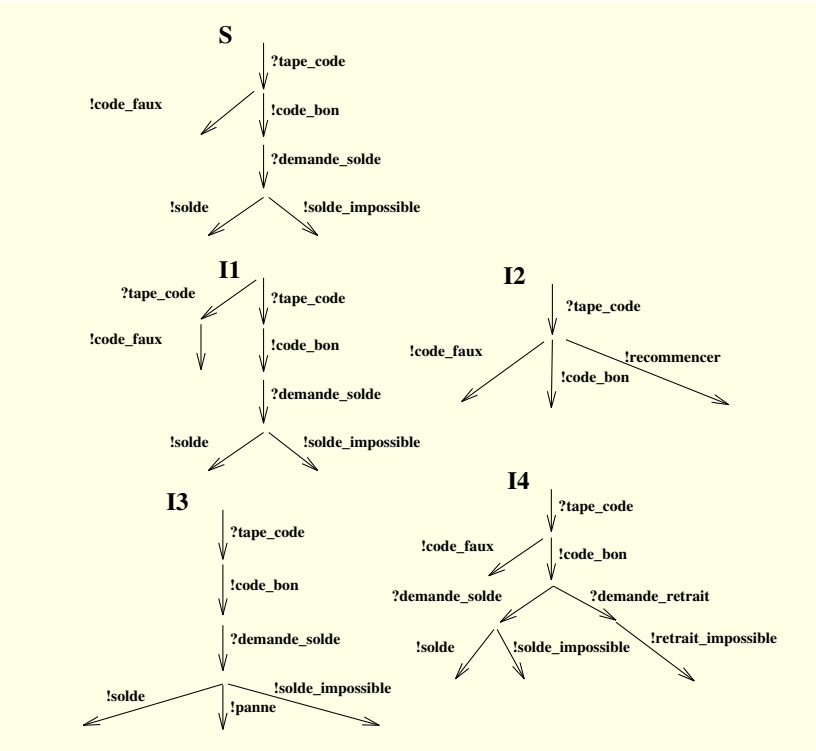
- **Construction d'un DAG (Graphe Dirigé Acyclique) par parcours en profondeur du produit synchrone de la spécification abstraite et de l'objectif :**
  - à la descente on vérifie la cohérence jusqu'à atteindre des états accepteurs de l'objectif et revenir à l'état initial de la spécification,
  - en remontant à partir de ces états, on synthétise un DAG constituant l'ossature du cas de test.
  - pendant la synthèse, à certaines transitions sont associées des verdicts
- **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.**

Relation de conformité

- Une *relation de conformité* est une relation liant une implantation I à une spécification S et décrivant formellement sous quelle condition I peut être déclarée conforme 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



I1, I4 ioconf S
I2, I3 not ioconf S

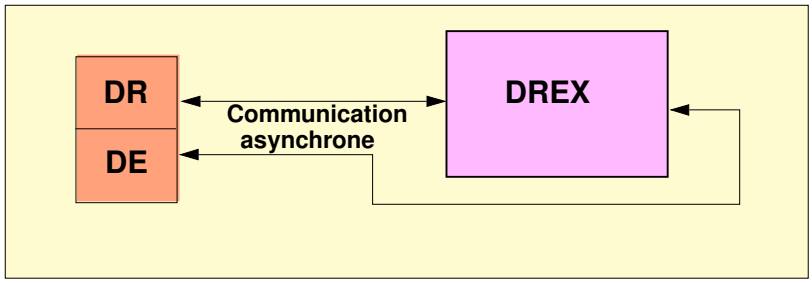
Propriétés de l'algorithme

- **Cohérence :** un objectif de test TP est dit cohérent avec la spécification S si au moins une séquence de S satisfait l'objectif et permet le retour à l'état initial.

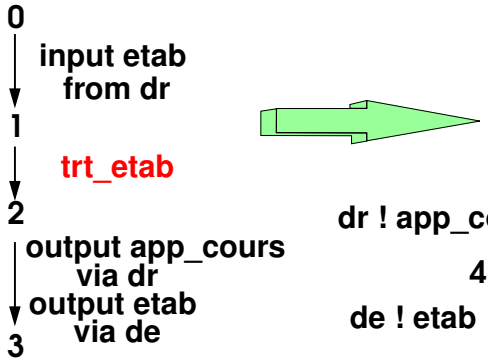
  L'algorithme appliqué à TP et S, rend vrai et construit un cas de test TC(TP,S) et seulement si TP est cohérent avec S.

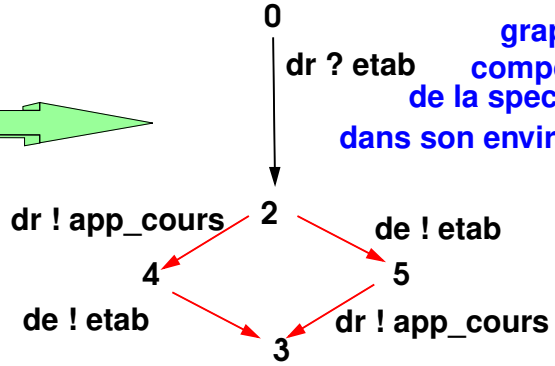
  TC(TP,S) comprend un préambule, un postambule, des verdicts et la gestion des timers.
- **Correction :** si l'application d'un cas de test TC(TP,S) généré par l'algorithme sur une implantation I produit un verdict FAIL alors l'implantation I est non conforme à la spécification S.
- **Exhaustivité :**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

Prise en compte de l'asynchronisme



graphe des comportements de la spec.



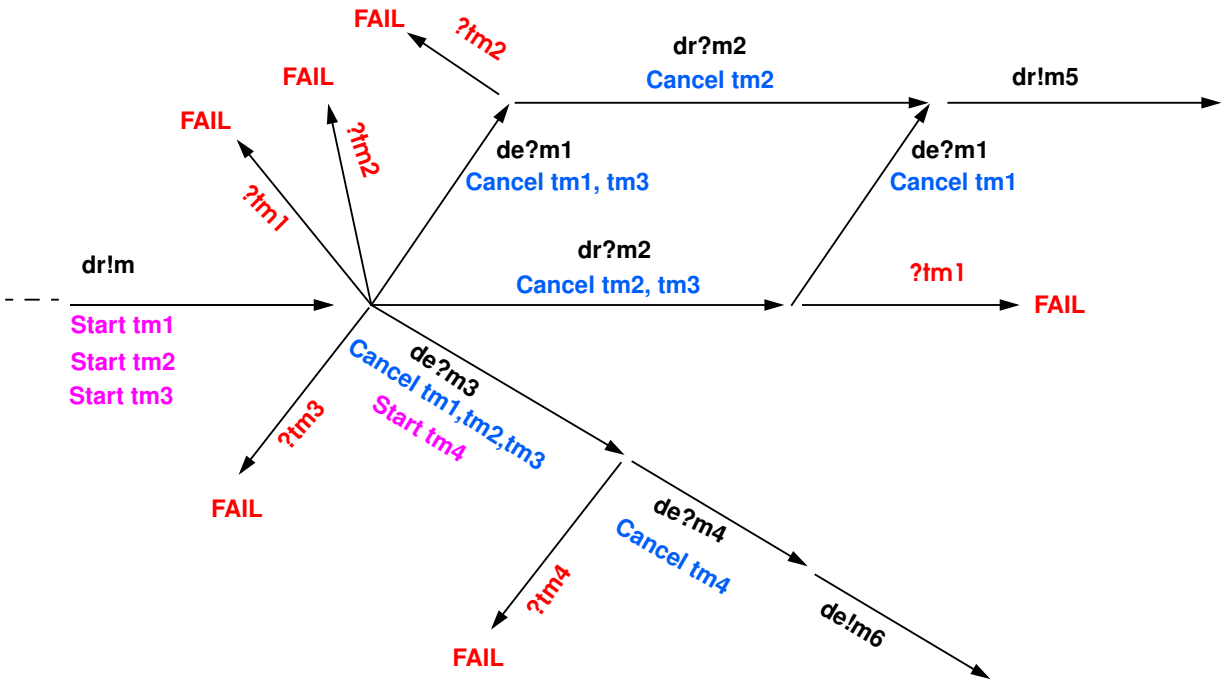


graphe des comportements de la spec. dans son environnement

Principe de gestion des timers

- Les timers doivent assurer la finitude de l'application d'un cas de test sur une implantation:
  - tout blocage de l'implantation doit être détecté par l'expiration d'un timer,
  - si les délais des timers sont suffisants, chaque expiration de timer doit correspondre à un blocage.
- A 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.
  - **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



Différences avec les tests manuels

- **Verdicts FAIL:** dans les tests manuels, parfois une réception de message impossible dans la spec mène à FAIL.
- **Timers:** nommage différent. Exprirations de timers manquant dans les tests manuels.
- **Préambules:** pour nous jamais de références à un préambule.
- **Postambules:**
  - pour nous, pas de postambule après FAIL.
  - différences dues aux feed.
  - dans les tests manuels, il manque parfois une transition.
- **Problèmes de modélisation:** impossible de se placer dans le contexte des tests SUU 105 et 106 avec les filtres.

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 l'IUT.
  - calcul des *losanges* modélisant la concurrence.
  - parcours de produit synchrone de graphes.
  - synthèse à la volée du graphe de test.

Résultats obtenus

❑ Le prototype TGV

- réalisé rapidement dans l'environnement CADP.
- met l'accent sur l'aspect comportemental des suites de test.
- montre la maturité de la méthode automatique utilisée.

❑ Analyse des suites de test produites par TGV

- rapidité de la génération : quelques secondes.
- syntaxiquement proches des suites manuelles : préambule, corps, postambule, timers, contraintes.
- évitent certaines erreurs intrinsèques aux méthodes manuelles.
- quelques différences : problèmes de modélisation des objectifs.

Bilan de l'étude

❑ Avantages des outils automatiques/ écriture manuelle:

- spécification LDS disponible pour autres méthodes de validation (simulation, vérification),
- gain en temps même si la spec doit être écrite (divisé par 2 au moins),
- gain en qualité du TTCN
- maintenance évolutive des suites de test

❑ Complémentarité des trois prototypes: TGV, TOPIC, TVEDA.
Etude d'un outil intégrant les avantages de chacun:

- 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