Génération automatique de tests de conformité pour les protocoles

Thierry Jéron

Thierry.Jeron@irisa.fr
Irisa / Inria Rennes
Projet Pampa
http://www.irisa.fr/pampa

Plan

1.Introduction au test de conformité
- principes de base, norme ISO 9646
- architectures de test
- TTCN

2.Techniques basées sur les automates
- machines de Mealy, modèles de fautes,
- aperçu des différentes techniques: TT, DS, UIO, etc.
- aspects algorithmiques et complexité

3.Techniques basées sur les systèmes de transition
- systèmes de transition (à entrée/sortie), relations de conformité, etc.
- algorithmes de génération

4.Test asynchrone et réparti

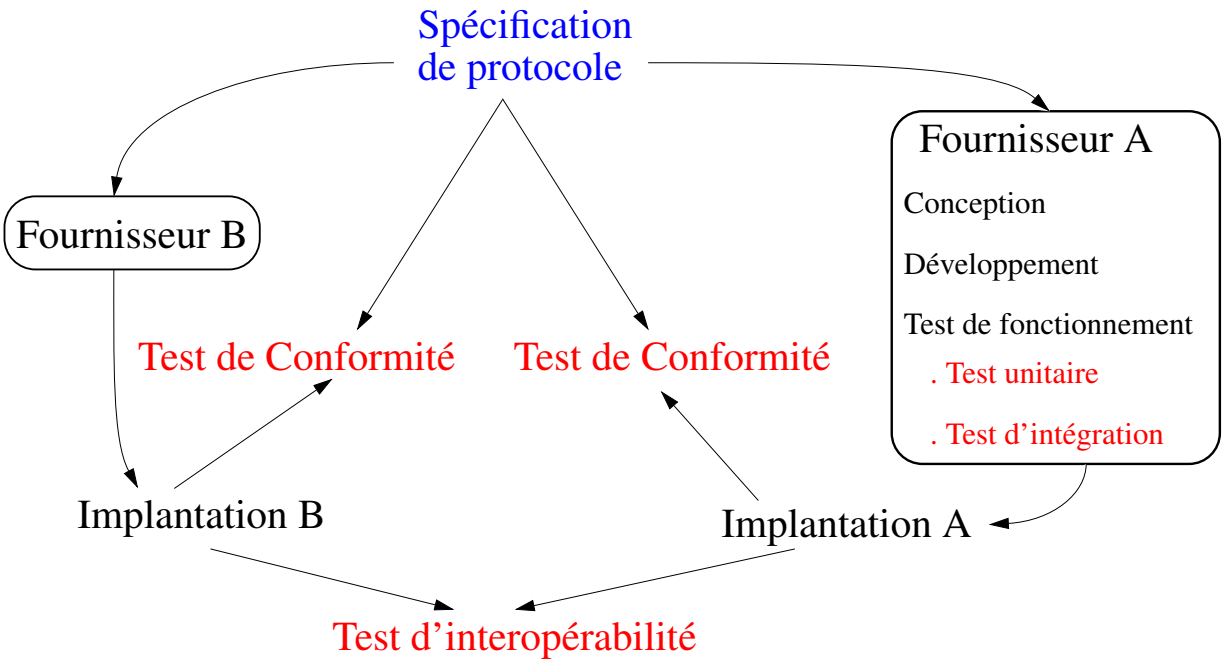
5. Perspectives

1.Introduction au test de conformité

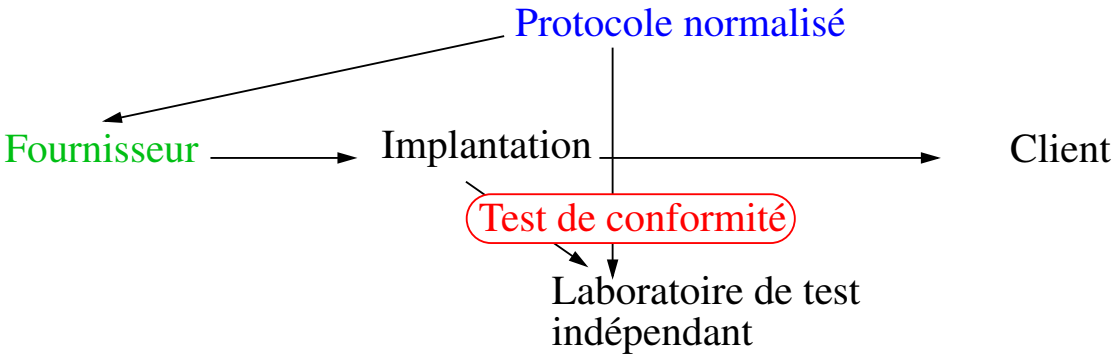
Les principes de base du test de conformité
Aperçu de la norme ISO 9646
“Conformance Testing Methodology and Framework”
+ TTCN
Motivations pour l’automatisation

voir aussi la norme “Formal Methods in Conformance Testing”

Test des systèmes ouverts



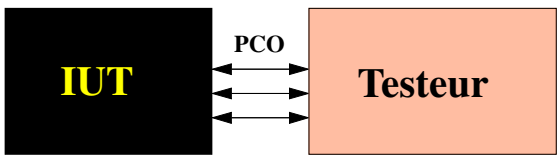
Les acteurs



- **Contexte d'utilisation:**
  - OSI (X25, couches 4-6, ...)
  - Téléphonie (RNIS, GSM, ...)
  -

Le test de conformité

- **Vérifier qu'une implantation sous test (IUT) se comporte correctement vis à vis de sa spécification. En fait rechercher des erreurs dans l'IUT.**
- **Test boîte noire (fonctionnel): le code source de l'IUT n'est pas connu.**
  Par opposition au test boîte blanche (structurel)
  Système ouvert: le comportement externe peut être observé par l'environnement à travers ses interfaces
  PCO = Point of Control and Observation.
- **Pratique du test:** un **testeur** interagit avec l'IUT à travers les PCOs par des **cas de test** (processus prédéfini composé d'interactions), émet des **verdicts** en fonction du comportement attendu décrit par la spécification.



La norme ISO 9646

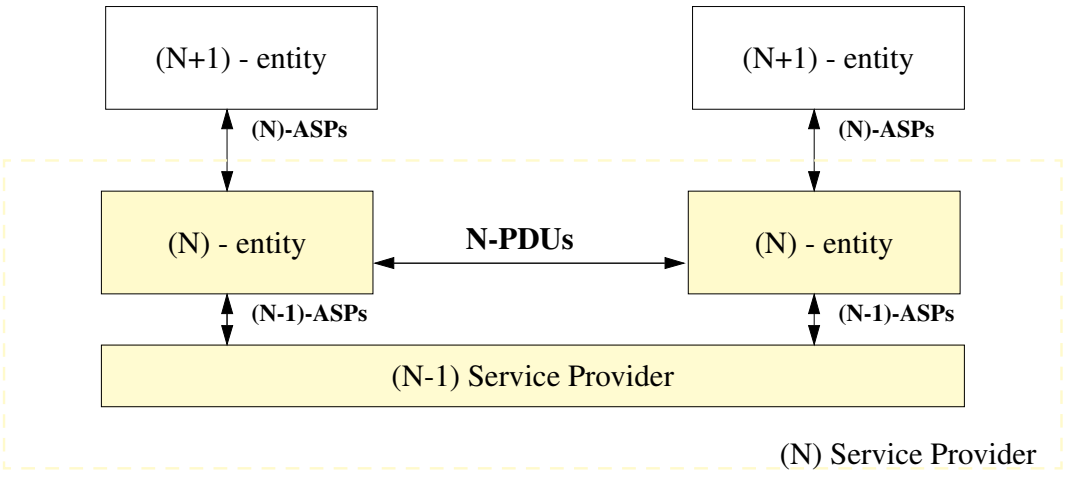
OSI COnformance testing methodology and framework

- Partie 1 . Concepts
- Partie 2. Création de suites de test
- Partie 3. TTCN
- Partie 4. Mise en oeuvre du test
- Partie 5. Relations laboratoire-client
- Partie 6. Test de profils
- Partie 7. PICS

Architectures de test

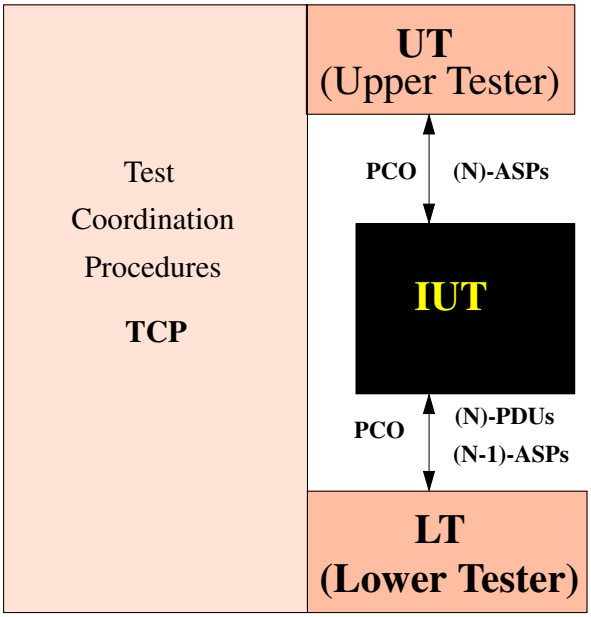
Méthodes de test abstraites

- Défini dans le cadre du modèle de référence OSI

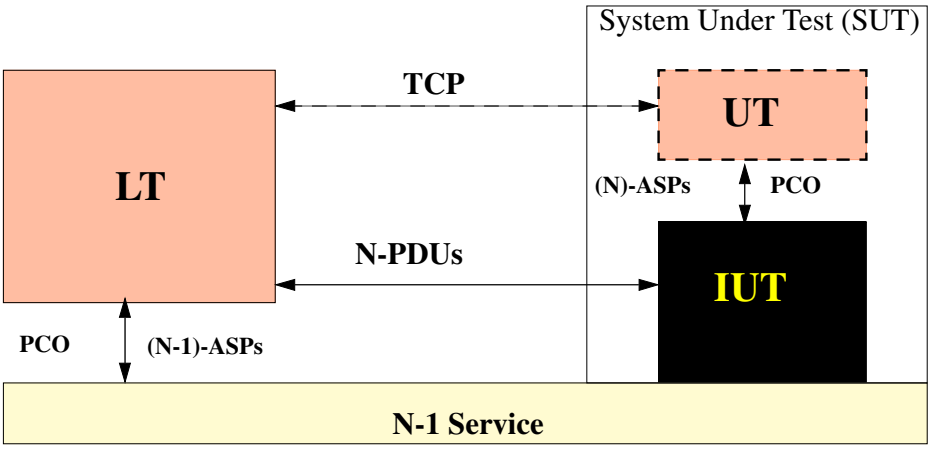


Terminologie: ASP: Abstract Service Primitive
PDU: Protocol Data Unit

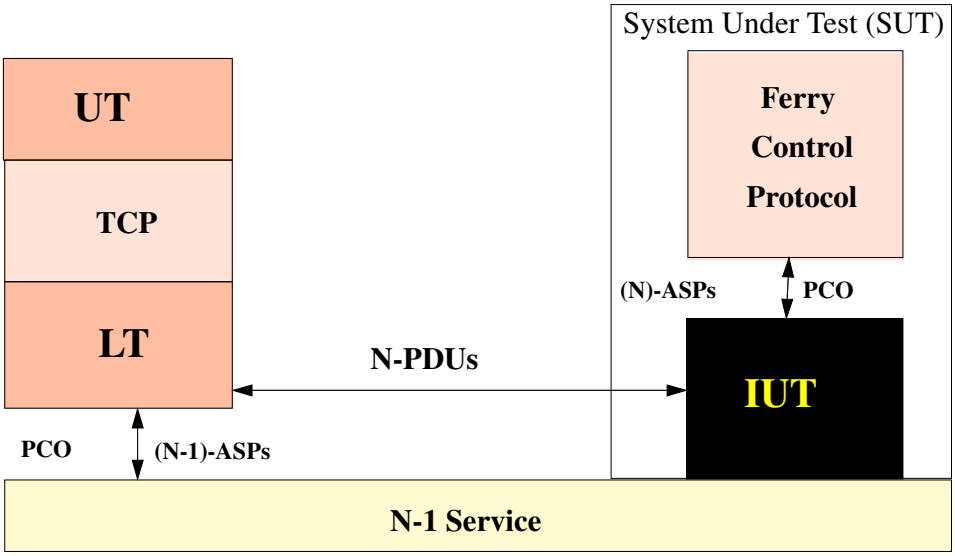
Local single layer test method



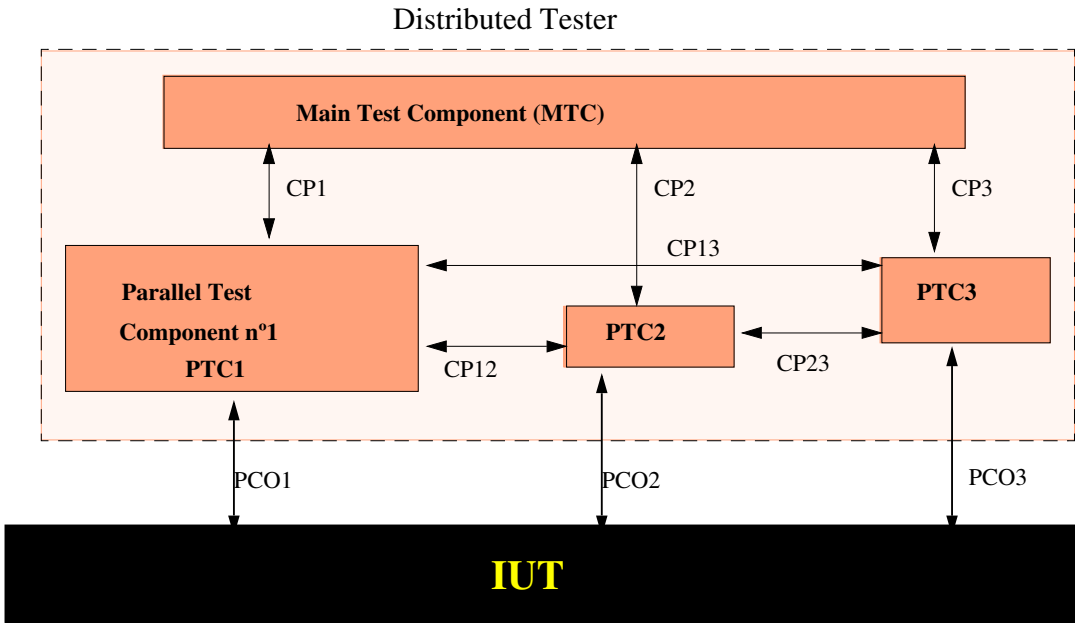
Remote (distributed) single layer test method



Ferry single layer test method



Multi-party testing



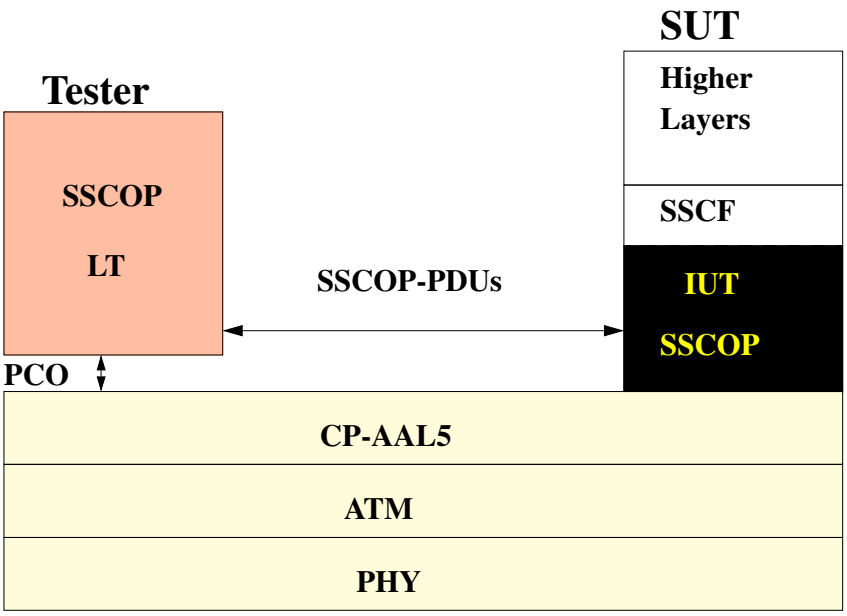
TTCN: Tree and Tabular Combined Notation

- **Notation devenue langage**
- **Deux formes:** TTCN.GR (tableaux), TTCN.MP (ASCCII)
- **Données décrites en ASN.1**
- **Composition d'une suite de test TTCN:**
  - **- Sommaire: structure, index, test purposes**
  - **- Déclarations: types, variables, PCOs, timers**
  - **- Contraintes: valeurs de PDUs**
  - **- Partie dynamique**

TTCN par l'exemple: SSCOP

- **Protocole de transfert de données de l'ATM**
  Normalisé
  Norme contient une spécification SDL: 7000 lignes de SDL PR,
  ~ 100 pages de SDL GR, 175 transitions SDL
  Utilisé dans plusieurs études de cas
- **Suite de test de l'ATM Forum:**
  Normalisée
  ~ 350 pages de TTCN

Test method: remote single layer



Structure de la suite de test

```
- SSCOP/
  - Protocol Capabilities (PC)
      - state_1
            - valid
            - invalid
            - inopportune
      - state_2
            - valid
            - invalid
            - inopportune
      :
      :
      - state_10
            - valid
            - invalid
            - inopportune
- System Parameters (SP)
      - Timer
      - Parm
```

Test Suite Structure

Suite Name : ATMF_SSCOP
Standards Ref : ITU-T Recommendation Q.2110, Service Specific Connection Oriented Protocol(SSCOP)
PICS Ref : ITU-T Recommendation Q.2110, Annex B
PIXIT Ref : ATM Forum/af-test-0067.000, Conformance ATS for the SSCOP for UNI 3.1, Annex A
Test Method(s) : Remote Single Layer Embedded Test Method
Comments :

Test Group Reference	Selection Ref	Test Group Objective	Page Nr
PC/		Protocol Capabilities	91
PC/STATE_1/			91
PC/STATE_1/VAL/			91
PC/STATE_1/INV/			95
PC/STATE_1/INOP/			122
PC/STATE_2/			127
PC/STATE_2/VAL/			127
PC/STATE_2/INV/			140
PC/STATE_4/			167
PC/STATE_4/VAL/			167
PC/STATE_4/INV/			180
PC/STATE_5/			207
PC/STATE_5/VAL/			207
PC/STATE_5/INV/			219
PC/STATE_5/INOP/			246
PC/STATE_7/			248
PC/STATE_7/VAL/			248
PC/STATE_7/INV/			257
PC/STATE_7/INOP/			284
PC/STATE_10/			288
PC/STATE_10/VAL/			288
PC/STATE_10/INV/			324
PC/STATE_10/INOP/			351
SP/		System Parameters	353
SP/TIMER/			353
SP/PARAM/			361

Detailed Comments :

Index et objectifs de test

Test Case Index				
Test Group Reference	**Test Case Id**	**Selection Ref**	**Description**	**Page Nr**
PC/STATE_1/VAL/	S1_V_A1		Verify that the IUT generates the BGN PDU on demand at state 1.	91
PC/STATE_1/VAL/	S1_V_P1		Verify that the IUT sends a BGREJ PDU on reception of retransmitted BGN PDU.	91
PC/STATE_1/VAL/	S1_V_P16		Verify that the IUT accpets a UD PDU at state 1.	94
PC/STATE_1/VAL/	S1_V_P17		Verify that the IUT accpets a MD PDU at state 1.	94
PC/STATE_1/INV/	S1_IV_I1		Verify that the IUT ignores an invalid BGN PDU which is not 32-bit aligned at state 1.	95
PC/STATE_1/INV/	S1_IV_I2		Verify that the IUT ignores an invalid BGAK PDU which is not 32-bit aligned at state 1.	96
PC/STATE_1/INOP/	S1_IO_P3		Verify that the IUT sends a END PDU on reception of a BGAK PDU at state 1.	122
PC/STATE_1/INOP/	S1_IO_P4		Verify that the IUT ignores a BGREJ PDU and remains at state 1.	123

Déclarations

Test Case Variable Declarations			
Variable Name	**Type**	**Value**	**Comments**
VT_MS	INTEGER		Maximum Send state variable
VR_MR	INTEGER		Maximum Receive state variable
VT_S	INTEGER	0	Send state variable
VT_PS	INTEGER	0	Poll Send state variable
VT_A	INTEGER	0	Acknowledge state variable
VT_PA	INTEGER	0	Poll Acknowledge state variable
VT_PD	INTEGER	0	Poll Data state variable
VR_R	INTEGER	0	Receive state variable
VR_H	INTEGER	0	Highest expected state variable
count	INTEGER		general purpose counter
TCV_OCT	OCTETSTRING		general purpose
TCV_N_SQ	INTEGER		to handle N(SQ)
TCV_N_MR	INTEGER		to handle N(MR)
TCV_N_PS	INTEGER		to handle N(PS)
TCV_LIST	LIST_TYPE		to handle LIST field of STAT PDU
TCV_LIST1	LIST_TYPE		to handle LIST field of STAT PDU
Detailed Comments :			

Déclarations: PCO et timers

PCO Declarations			
PCO Name	**PCO Type**	**Role**	**Comments**
LT_PCO	L_SSCOP	LT	Lower boundary of SSCOP
Detailed Comments :			

Timer Declarations			
Timer Name	**Duration**	**Unit**	**Comments**
T_Opr	TESTtime	s	This timer is used to allow test operator intervention
Timer_CC	TimerCCtime	s	The time between transmission of BGN, END, ER, or RS PDU
Timer_POLL	TimerPOLLtime	s	The time between transmission of POLL PDU at active phase
Timer_KEEP_ALIVE	TimerKEEP_ALIVEtime	s	The time between transmission of POLL PDU at transient phase
Timer_IDLE	TimerIDLEtime	s	may be considerably greater than Timer_KEEP_ALIVE
Timer_NO_RESPONSE	TimerNO_RESPONSEtime	s	The maximum time interval during which at least one STAT PDU needs to be received.
T_Wait	WAITtime	s	This timer is used when no response is expected from IUT
Detailed Comments :			

Déclarations: PDUs

TTCN PDU Type Definition		
PDU Name : BGN **PCO Type :** L_SSCOP **Comments :**		
Field Name	**Field Type**	**Comments**
UU	OCTETSTRING[0..UU_Max_Len]	
PAD	OCTETSTRING[0..3]	
RESERVED	OCTETSTRING[3]	
N_SQ	BITSTRING[8]	
PL	BITSTRING[2]	
RSVD	BITSTRING[2]	
PDU_Type	BITSTRING[4]	
N_MR	BITSTRING[24]	
Detailed Comments :		

Contraintes

TTCN PDU Constraint Declaration		
Constraint Name : BGN_R_GEN		
PDU Type : BGN		
Derivation Path :		
Comments : General RECEIVE constraint		
Field Name	**Field Value**	**Comments**
UU	*	
PAD	*	
RESERVED	'000000'O	
N_SQ	COMPLEMENT(INT_TO_BIT(VR_SQ,8))	retransmitted BGN PDU check
PL	?	
RSVD	'00'B	
PDU_Type	'0001'B	
N_MR	?	
Detailed Comments :		

Structure d'un cas de test

- **Objectif de test:** but du cas de test
- **Comportement**: graphe de séquences d'interactions entre Testeur et IUT pour un Objectif de test particulier
  + timers: observation du blocage de l'IUT.
  Préambule: mène à l'état initial de l'objectif de test
  Corps du cas de test: vérifie l'objectif
  Séquence d'identification: vérifie l'état d'arrivée
  Postambule: retour à l'état initial.
- **Verdicts:**

  FAIL: non conformité (blocage non autorisé ou réception non spécifiée)
  (PASS): Objectif de test atteint
  PASS: (PASS) et retour à l'état initial après un postamble
  INCONCLUSIVE: réception spécifiée mais Objectif non atteignable

  En pratique ré-exécution jusqu'à obtention d'un verdict Fail ou Pass

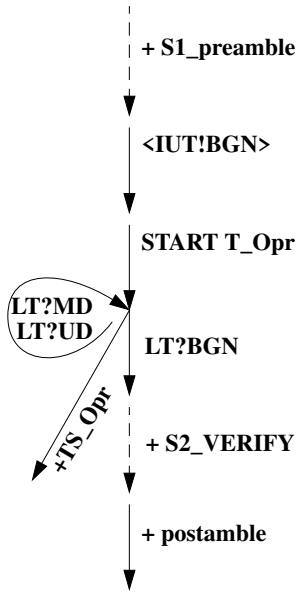
Comportement: cas de test

Test Case Dynamic Behaviour

Test Case Name : S1_V_A1
Group : PC/STATE_1/VAL/
Purpose : Verify that the IUT generates the BGN PDU on demand at state 1.
Configuration :
Default :
Comments : Ref. 5.0 g, Fig. 20(5 of 51)/PICS PC11

Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+S1_PREAMBLE			
2		<IUT!BGN>	BGN_R_GEN		
3		START T_Opr			
4	LB1	LT_PCO?BGN(VR_SQ:=BIT_TO_INT(BGN.N_SQ), VT_MS:=BIT_TO_INT(BGN.N_MR))	BGN_R_GEN		
5		+S2_VERIFY			
6		+postamble			return the IUT in STATE 1
7		LT_PCO?MD	MD_R_GEN		
8		GOTO LB1			
9		LT_PCO?UD	UD_R_GEN		
10		GOTO LB1			
11		+TS_Opr			

Detailed Comments :



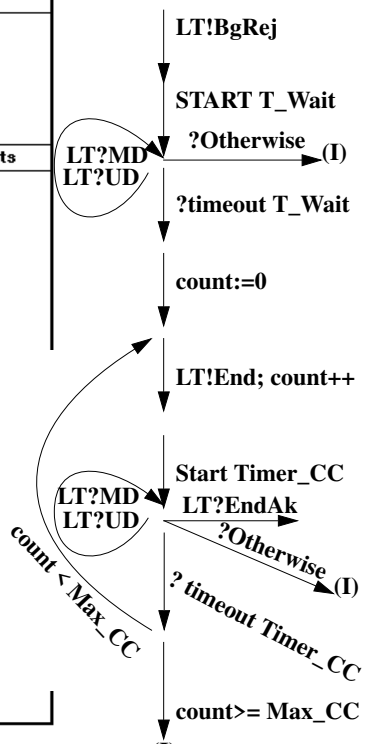
Preambule

Test Step Dynamic Behaviour

Test Step Name : S1_PREAMBLE
Group : PREAMBLE/
Objective : Procedure used to place the IUT at state 1(IDLE) from any state.
Default :
Comments :

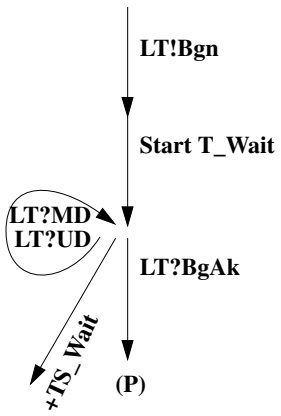
Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		LT_PCO!BGREJ	BGREJ_S_GEN		
2		START T_Wait			
3	LB1	?TIMEOUT T_Wait			
4		(count:=0)			
5	LB2	LT_PCO!END(count:=count+1)	END_S_USER		
6		START Timer_CC			
7	LB3	LT_PCO?ENDAK	ENDAK_R_GEN		
8		LT_PCO?MD	MD_R_GEN		
9		GOTO LB3			
10		LT_PCO?UD	UD_R_GEN		
11		GOTO LB3			
12		?TIMEOUT Timer_CC			
13		[count < Max_CC]			
14		GOTO LB2			
15		[count >= Max_CC]		(I)	
16		LT_PCO?OTHERWISE		(I)	
17		LT_PCO?MD	MD_R_GEN		
18		GOTO LB1			
19		LT_PCO?UD	UD_R_GEN		
20		GOTO LB1			
21		LT_PCO?OTHERWISE		(I)	

Detailed Comments :



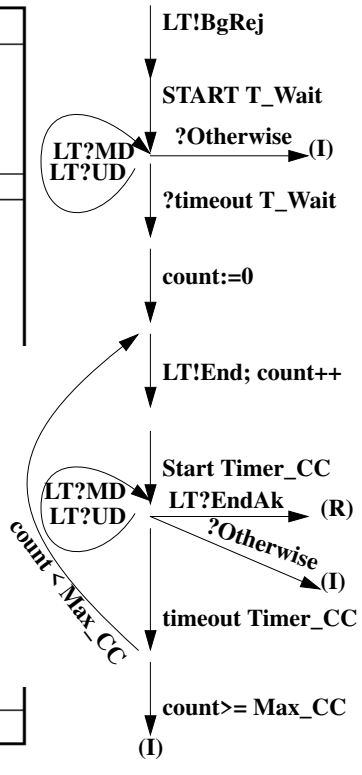
Identification d'état

Test Step Dynamic Behaviour					
Test Step Name : S2_VERIFY					
Group : VERIFY/					
Objective : Procedure used to verify that the IUT is at state 2.					
Default :					
Comments :					
Nr	**Label**	**Behaviour Description**	**Constraints Ref**	**Verdict**	**Comments**
1		LT_PCO!BGN(VT_SQ:=INC_MOD_8(VT_SQ, 1), VR_MR:= GET_VR_MR())	BGN_S_GEN		VR(SQ)x > N(SQ)
2		START T_Wait			
3	LB1	LT_PCO?BGAK(VT_MS:=BIT_TO_INT(BGAK.N_MR))	BGAK_R_GEN	(P)	IUT was at state 2
4		LT_PCO?UD	UD_R_GEN		
5		GOTO LB1			
6		LT_PCO?MD	MD_R_GEN		
7		GOTO LB1			
8		+TS_Wait			
Detailed Comments :					



Postambule

Test Step Dynamic Behaviour					
Test Step Name : postamble					
Group :					
Objective : Procedure used to place the IUT at state 1(IDLE) from any state.					
Default :					
Comments :					
Nr	**Label**	**Behaviour Description**	**Constraints Ref**	**Verdict**	**Comments**
1		LT_PCO!BGREJ	BGREJ_S_GEN		
2		START T_Wait			
3	LB1	?TIMEOUT T_Wait			
4		(count:=0)			
5	LB2	LT_PCO!END(count:=count+1)	END_S_USER		
6		START Timer_CC			
7	LB3	LT_PCO?ENDAK	ENDAK_R_GEN	R	
8		LT_PCO?MD	MD_R_GEN		
9		GOTO LB3			
10		LT_PCO?UD	UD_R_GEN		
11		GOTO LB3			
12		?TIMEOUT Timer_CC			
13		[count < Max_CC]			
14		GOTO LB2			
15		[count >= Max_CC]		I	
16		LT_PCO?OTHERWISE		I	
17		LT_PCO?MD	MD_R_GEN		
18		GOTO LB1			
19		LT_PCO?UD	UD_R_GEN		
20		GOTO LB1			
21		LT_PCO?OTHERWISE		I	
Detailed Comments :					



Des tests abstraits aux tests exécutables: PICS et PIXIT

Informations fournies par le constructeur

- **PICS:** Protocol Implementation Conformance Statement
  Fonctions effectivement réalisées et options retenues
- **PIXIT:** Protocol Implementation eXtra Information for Testing
  Données spécifiques de l'IUT que le testeur doit connaître pour mettre en oeuvre les tests (adresses, valeurs particulières, observabilité, controlabilité).

Permettent de:

- sélectionner les tests applicables,
- paramétrer les tests exécutables (ETS)

Pratique industrielle

- **Ecriture manuelle des suites de tests fondé sur ISO 9646.**
  - processus long et répétitif: de 30% à 60% du coût de développement.
  - jusqu'à 20% des tests manuels erronés (synthaxe, sémantique).
  - couverture incertaine.
  - difficulté de maintenance.
  - difficile de justifier les tests utilisés
    car pas de définition précise de la conformité.

Automatisation

❑ **Principe:**

- utiliser des spécifications formelles (SDL, Lotos, Estelle, ...)
- formaliser le processus de test
- concevoir des algorithmes de génération

❑ **Apports de l'automatisation:**

- gains énormes en temps et en coût
- qualité des tests
- spécification formelle utilisable pour simulation, vérification.

Spécificités Telecom

❑ **Normes**

- Techniques de Description Formelles normalisées:
  Lotos, Estelle, SDL
- Protocoles normalisés (de plus en plus en SDL).
- ISO9646: Conformance Testing Methodology and Framework.
- Langage de description des tests normalisé:
  TTCN (Tree and Tabular Combined Notation).
- Suites de tests normalisées (TTCN).

❑ **Difficultés pour l'automatisation:**

- asynchronisme et non-déterminisme.
- prédominance contrôle (couches basses) ou données (couches hautes).
- spécifications souvent de taille conséquente et très détaillées.
- contrainte de produire des tests comparables aux tests manuels.

Génération automatique de tests

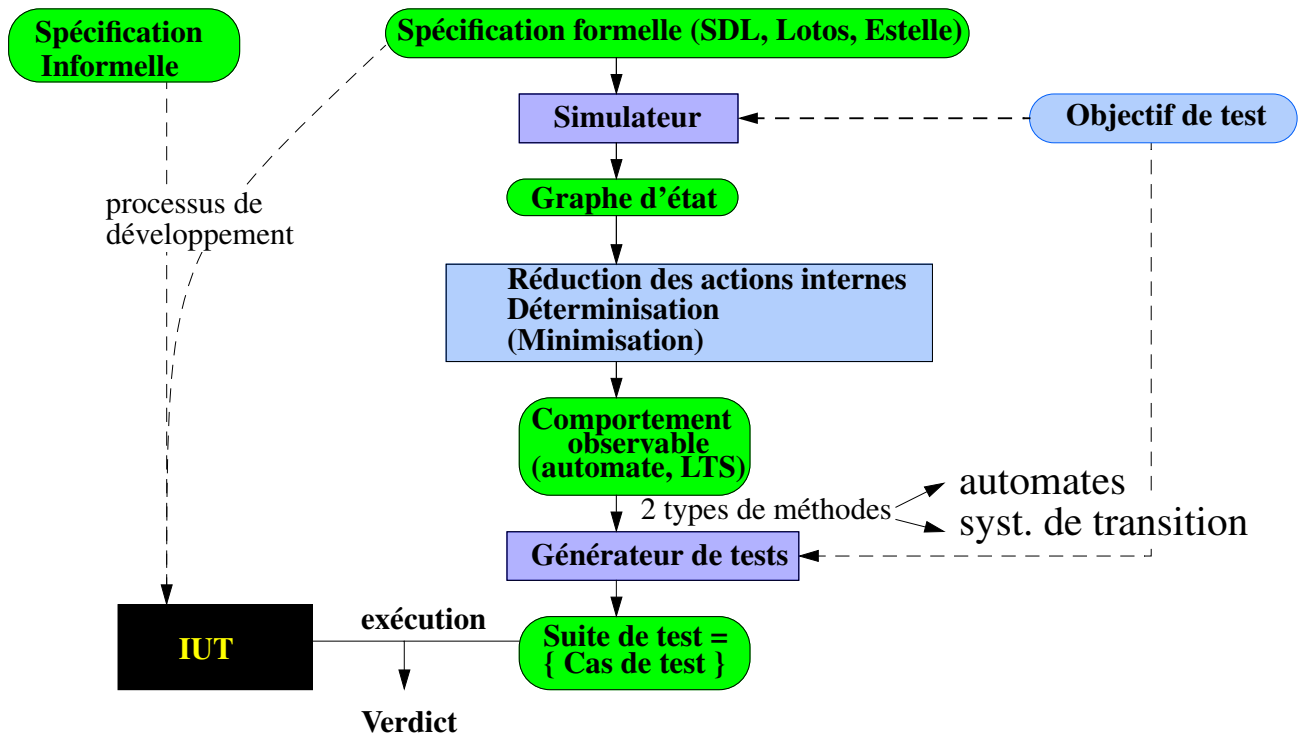
- **Méthodes syntaxiques:** basées sur le texte de la spécification.
  - en général pas de définition de la conformité
- **Méthodes sémantiques:** basées sur le comportement de la spécification.
  Comportement donné par la sémantique opérationelle des langages de spécification.
  Modèle de comportement : automates ou systèmes de transition
  Définition de la conformité: relation de conformité ou modèle de faute.
  - **Méthodes fondées sur les automates**
  - **Méthodes fondées sur les systèmes de transition**

FMCT: Formal Methods in Conformance Testing

Projet de norme Z 500

Formalisation des notions de base pour la génération de test basée sur des méthodes formelles (Estelle, Lotos, SDL)

Méthodologie générale



Méthodes de génération fondées sur les automates

- Origine: utilisées en test de circuits, issues de travaux sur l'identification de machines.
- Transportées dans le domaine du test de protocoles
- Fondements: théorie des automates et des graphes
- Algorithmique: adaptée de l'algorithmique classique des automates (minimisation, etc), des graphes (voyageurs de commerce, circuits eulériens, flots), de la programmation linéaire.

Machines de Mealy

- Modèle pour les spécification, les IUTs
- **Définition:** M = (S, I, O, δ, ω)
  - S: ens. fini d'états
  - I: alphabet d'entrée,
  - O: alphabet de sortie
  - h: S x I → 2^S x 2^O avec
    h(s,i)=(δ(s,i), ω(s,i)) définie par les fonctions
  - δ: S x I → 2^S: fonction de transfert,
    étendue en δ^*: S x I^* → 2^{S*}
  - ω: S x I → 2^O: fonction de sortie,
    étendue en ω^*: S x I^* → 2^{O*}.

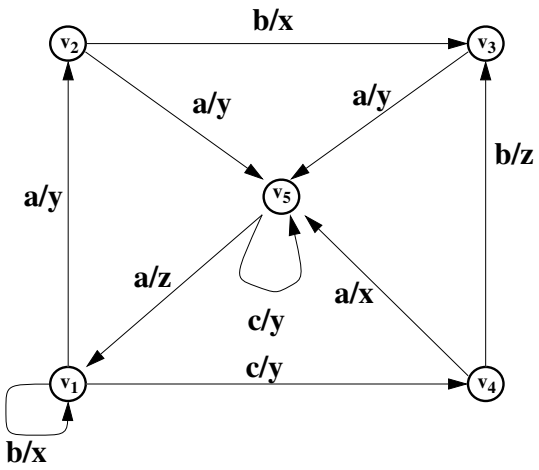
Transitions: s —i/o→ s'

Notions fondamentales

- **Déterminisme:**
  M est déterministe ssi pour tout état s_i et toute entrée a,
  $h(s_i,a)= (\delta(s_i, a), \omega(s_i, a))$, s'il existe, est unique
- **Complétude:** M est complète ssi pour tout état s_i et toute entrée a,
  $h(s_i, a)$ existe.
- **Equivalence**
  s_i et s_j sont equivalents ssi pour tout σ dans I*, $\omega^*(s_i, \sigma) = \omega^*(s_j, \sigma)$
- **Minimalité**
  M est minimale si elle n'a aucun couple d'états équivalents
  Minimisation par algorithme classique de rafinement de partition suivant les sorties possibles

On supposera en général les machines de Mealy sont déterministes, complètes et minimales

Exemple



S =(v1, v2, v3, v4, v5),

I ={a, b, c}, O ={x,y,z}

	δ			ω		
	a	b	c	a	b	c
v_1	v_2	v_1	v_4	y	x	y
v_2	v_5	v_3	v_2	y	x	null
v_3	v_5	v_3	v_3	y	null	null
v_4	v_5	v_3	v_4	x	z	null
v_5	v_1	v_5	v_5	z	null	y

Principe du test

- **Modèle de faute:** pour une transition $s \xrightarrow{i/o} s'$
  - faute de sortie: $s \xrightarrow{i/o'} s'$
  - faute de transfert $s \xrightarrow{i/o} s''$
- **Hypothèse principale:** le comportement de l'IUT (boîte noire) est modélisable par une machine de Mealy.
- **Principe:** tester chacune des transitions de l'IUT pour ses fautes de sortie et transfert en fonction des transitions attendues de la spécification.

Test d'une transition $s \xrightarrow{i/o} s'$

- aller en s
- appliquer i
- vérifier o (fautes de sortie)
- identifier s' (faute de transfert)

Enchainer tous les tests en une séquence de taille minimale.

Variantes

- **Tour des transitions:** pas d'identification d'état d'arrivée (donc pas de faute de transfert) ou existence d'une fonction status $s' \xrightarrow{status/s'} s'$
- **DS (séquence distinguante):** suppose l'existence d'une séquence DS globale qui permet de distinguer tuos les états.

$$\exists\, DS \in I^*, \forall\, s_i, s_j \in S, \omega^*(s_i, DS) \neq \omega^*(s_j, DS)$$

DS existe rarement, très long.

- **UIO (Unique Input Output):** pour chaque état s_i, il existe une séquence UIO_i qui distingue s_i de tout autre état s_j:

$$\forall\, s_i \in S, \exists\, UIO_i \in I^*, \forall\, s_j \in S, \omega^*(s_i, UIO_i) \neq \omega^*(s_j, UIO_i)$$

existence des UIO non assurée mais fréquente

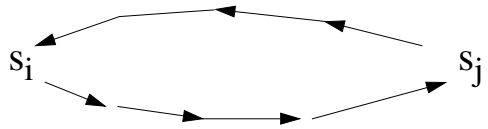
- **W (séquences caractéristiques):** pour paire d'état s_i, s_j de S il existe une séquence cs_{ij} qui distingue distingue s_i de s_j:

$$\forall\, s_i, s_j, \in S, \exists\, cs_{ij} \in I^*, \omega^*(s_i, cs_{ij}) \neq \omega^*(s_j, cs_{ij})$$

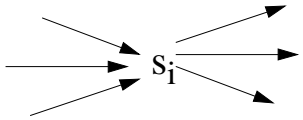
existence des cs_{ij} aussitôt que M minimale

Rappels sur les graphes

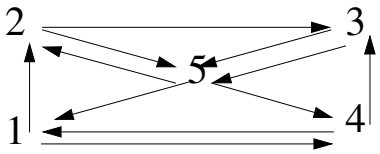
- **Forte connexité:** G=(V,E) est fortement connexe ssi pour toute paire de sommets s_i, s_j, il existe un chemin de s_i à s_j et un chemin de s_j à s_i.



- **Symétrie:** G=(V,E) est symétrique ssi tout sommet a un degré entrant égal à son degré sortant.



- **Circuit eulérien:** circuit passant une et une seule fois par chaque transition.



Exemple:

1 → 2 → 3 → 5 → 2 → 5 → 1 →
4 → 3 → 5 → 4 → 1

NB: ne pas confondre avec circuit hamiltonien (.... chaque sommet)

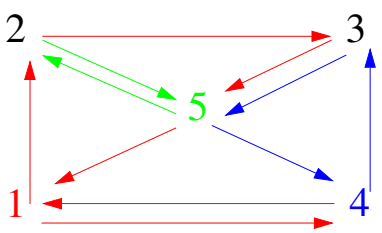
CNS d'existence de circuit eulérien

G=(V,E) possède un circuit eulérien ssi
il est fortement connexe et symétrique.

=>: circuit eulérien => - de tout sommet on peut atteindre tout autre sommet
- dans chaque noeud on entre autant de fois qu'on sort

<=: donné par l'algorithme suivant de complexité linéaire:

$C=s_0$;
tant que G=(V,E) non vide
choisir s dans C ayant des successeurs
calculer un circuit maximal C(s) dans G revenant en s
en enlevant de E les transitions prises
rabouter C(s) à C à partir de s



i/ 1 → 2 → 3 → 5 → 1 → 4 → 1
ii/ 4 → 3 → 5 → 4
iii/ 5 → 2 → 5

1 → 2 → 3 → 5 → 2 → 5 → 1 →
4 → 3 → 5 → 4 → 1

Tour des transition

- **Hypothèses:** spécification et IUT sont des Machines de Mealy fortement connexes et minimale.
- **Pb de la génération de test (Postier Chinois) :** trouver un circuit de longueur minimale passant au moins une fois par toutes les transitions de M.
- **Se ramène au problème du circuit eulérien par la propriété:**

circuit de longueur min. passant au moins une fois par ttes les transitions

=

circuit eulérien de l'augmentation symétrique de coût minimal $G^{\wedge}$ de G

Augmentation symétrique de coût minimal

Augmentation symétrique: graphe symétrique $G^{\wedge}$ obtenu en dupliquant certaines transitions de G.

Coût minimal: trouver un ensemble minimal de transitions à dupliquer.

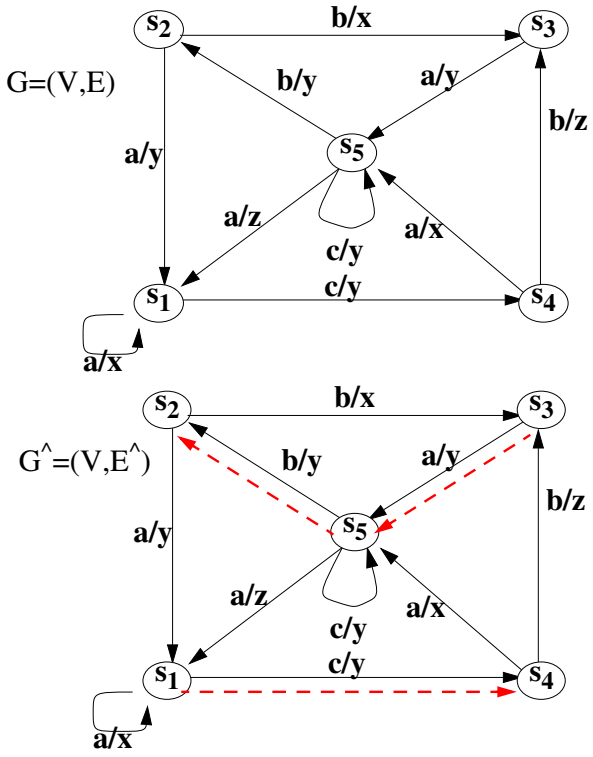
Soit X_{ij} le nombre de duplications de $s_i \xrightarrow{i/o} s_j$ dans $G^{\wedge}$

$$\text{Minimiser } \Sigma_{i,j} X_{ij}$$

$$\text{tel que } \forall s_i, \quad \Sigma_{s_j \to s_i} (X_{ji}+1) = \Sigma_{s_i \to s_k} (X_{ik}+1) \text{ (symétrie)}$$

Obtenu par programmation linéaire ou algorithme de flot.

Exemple



Minimiser $\Sigma_{i,j} X_{ij}$ tq

S_1: $X_{51} + X_{21} + X_{11} - X_{14} - X_{11} = -1$

S2: $X_{52} - X_{21} - X_{23} = 1$

S3: $X_{23} + X_{43} - X_{35} = -1$

S4: $X_{14} - X_{45} - X_{43} = 1$

S5: $X_{45} + X_{35} + X_{55} - X_{52} - X_{51} - X_{55} = -1$

Solution: $X_{14} = 1$, $X_{52} = 1$, $X_{35} = 1$

Calcul d'un circuit eulérien sur $G^{\wedge}$:

$s_1 \xrightarrow{a/x} s_1 \xrightarrow{c/y} s_4 \xrightarrow{b/z} s_3 \xrightarrow{a/y} s_5 \xrightarrow{c/y} s_5$

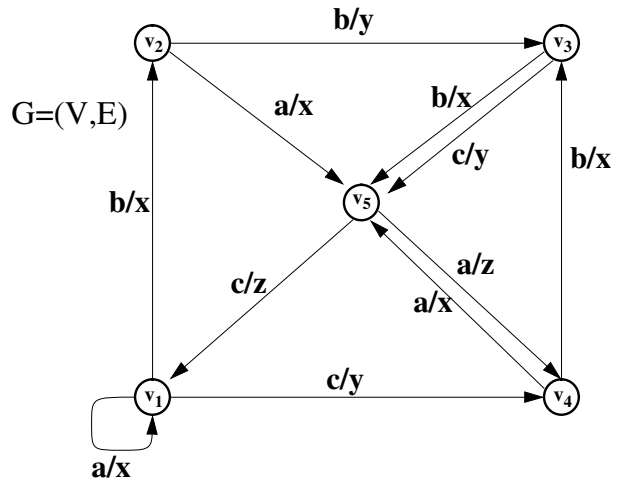
$\xrightarrow{a/z} s_1 \xrightarrow{c/y} s_4 \xrightarrow{a/x} s_5 \xrightarrow{b/y} s_2 \xrightarrow{b/x} s_3$

$\xrightarrow{a/y} s_5 \xrightarrow{b/y} s_2 \xrightarrow{a/y} s_1$

UIO (Unique Input Output)

❑ **Hypothèses:** spécification: minimal, fortement connexe, reset (ri/null)
implémentation: même nombre d'états

$\forall s_i \in S, \exists UIO_i \in I^*, \forall s_j \in S, \omega^*(s_i, UIO_i) \neq \omega^*(s_j, UIO_i)$



1ère étape: calcul des UIO

UIO
v_1: b/x.a/x
v_2: b/y
v_3: b/x.c/z
v_4: b/x.c/y
v_5: c/z

Test(v_1,v_2,b/x) = b/x. UIO(v_2) = b/x.b/y
Test(v_2,v_3,b/y) = b/y.UIO(v_3) = b/y.b/x.c/z
etc...

Problème du test

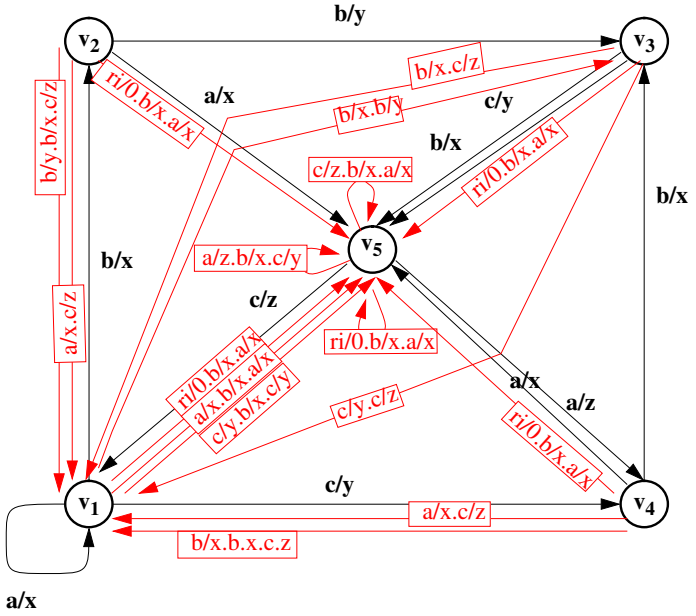
$G' = (V, E \cup E_C)$

$E_C = \{(v_i, v_k, L_l.UIO_j) \mid (v_i, v_j, L_l) \in E \text{ et } tail(UIO_j) = v_k\}$

$Cost(v_i, v_k, L_l.UIO_j) = 1 + Cost(UIO_j)$

Test: tester chaque transition et enchainer les tests en une séquence de longueur minimale.

Pb du Postier Chinois Rural:
trouver un circuit
de coût minimal de G'
contenant tous les arcs de E_C
(passant si nécéssaire par des arcs de E)



Augmentation rurale symétrique

❑ **Problème:** augmenter $G' = (V, E \cup E_C)$ en un graphe symétrique $G^{\wedge} = (V, E^{\wedge})$ en dupliquant seulement les arcs de E.

Théorème: si le sous-graphe $G'[E_C]$ (restreint à E_C) est connexe et couvre les sommets de G', toute augmentation symétrique rurale $G^{\wedge *}$ de G' est fortement connexe et possède donc un circuit Eulérien.

Cond. Suffisante: existence d'un reset ou forte connexité et self loop

Le circuit eulérien de $G^{\wedge *}$ est un circuit de coût minmal de G' contenant les arcs de E_C

Augmentation rurale symétrique et flot maximum

Graphe de flot

$\xi(s_i) = |(s_j,s_i) \in E_C| - |(s_i, s_j) \in E_C|$ (différence de degré entrant / sortant ds E_C)

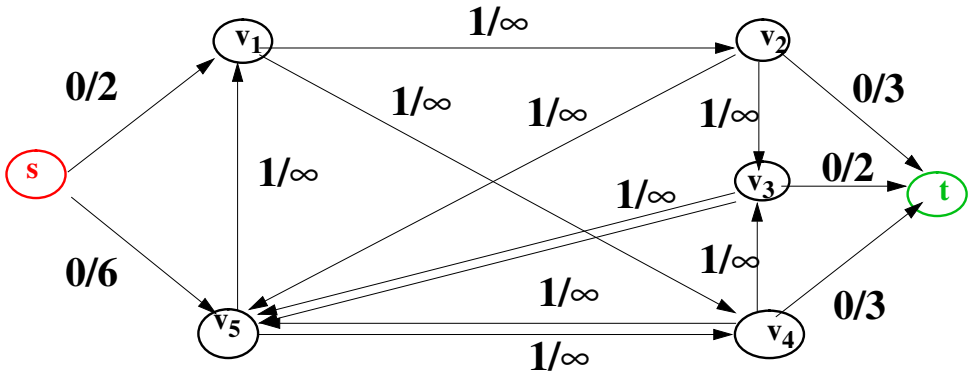
On définit un graphe de flot $G_F = (V_F , E_F)$ avec

$V_F = V \cup \{s, t\}$, $E_F = E \cup \{(s, s_i) \mid s_i \in C\} \cup \{(s_j, t) \mid s_j \in D\}$

avec $C = \{ s_i \mid \xi(s_i) \quad 0 \}$ et $D = \{ s_j \mid \xi(s_j) < 0 \}$

Couts: $C(s, s_i) = 0$ $C(s_i, t) = 0$ $C(s_i, s_j) = 1$

Capacités: $\gamma(s, s_i) = \xi(s_i) \quad 0$, $\gamma(s_i,t) = - \xi(s_i) > 0$ $\gamma(s_i,s_j) = + \infty$



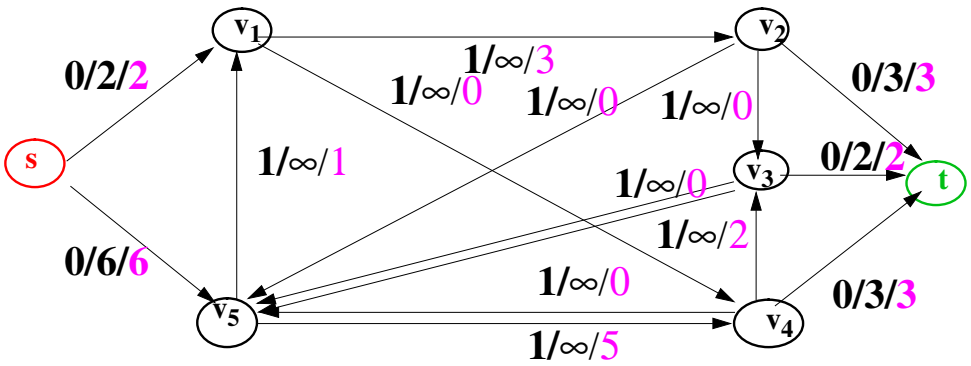
Flot sur G_F

Flot sur G_F : fonction $F : E_F \to Z^+$ tq

$\forall s_i \in V_F - \{s,t\}$, $\Sigma_j F(s_i , s_j) = \Sigma_j F(s_j , s_i)$ flot entrant = flot sortant

$\forall (s_i , s_j) \in E_F$, $F(s_i , s_j) \leq \gamma(s_i,s_j)$ le flot n'exéde pas la capacité

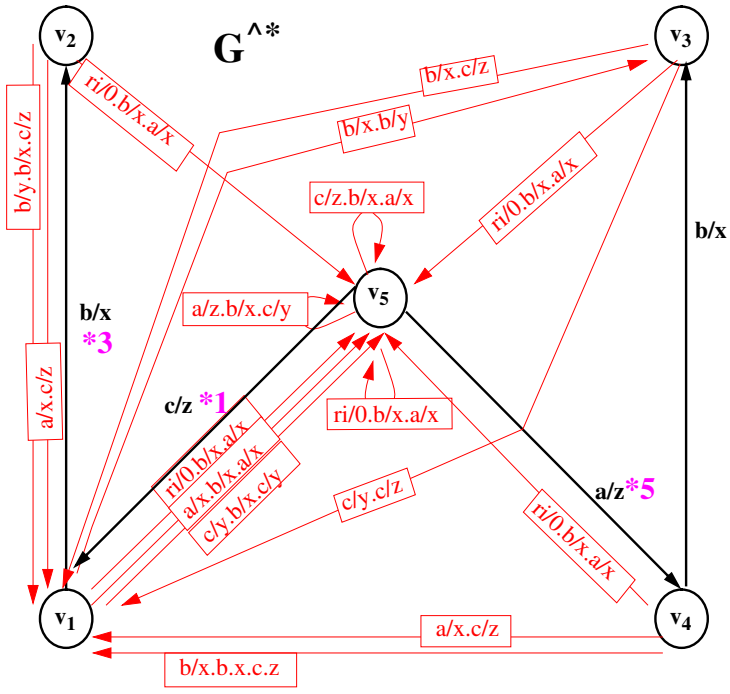
Saturation de tous les arcs incidents à s et t (coût nul)



Coût du flot: $C(F) = \Sigma_{i,j} C(s_i , s_j) . F(s_i , s_j)$

sur l'exemple: $C(F) = 11$

Augmentation rurale symétrique



Circuit eulérien de G^*

Séquence de test optimale:

ri/0 b/x.b/y c/y.c/z a/x.b/x.a/x
a/z.b/x.c/y c/z.b/x.c/y ri/0.b/x.a/x a/z
a/x.c/z c/y.b/x.c/y a/z b/x
b/x.c/z b/x b/y.b/x.c/z b/x
a/x.c/z b/x ri/0.b/x.a/x a/z
b/x ri/0.b/x.a/x a/z b/x.b/x.c/z
ri/0.b/x.a/x a/z ri/0.b/x.a/x c/z.

Complexité

- **Calcul des UIO:** exponentiel et taille des UIO exponentielle
  En pratique, s'ils existent, ils sont courts

- **Flot minimal:** polynomial: $O(n(m+ n \log(n)).\log(m_0/n)$
  avec $m = |E|$, $n = |V|$, $m_0 = |E_C|$.

  En général le Pb du Postier Chinois Rural est NP-complet

- **Circuit eulérien:** linéaire.

Analyse

+ couverture: complète pour un modèle de faute (si hypothèses)

+ théorie bien fondée

- hypothèses sur la spécification: en particulier déterminisme

- hypothèses sur l'IUT: déterminisme, minimalité (ou < |spec| + k)

- complexité: limité à des spécifications de petite ou moyenne taille

Méthodes fondées sur les systémes de transition

- Origine: théorie du test des algèbres de processus.
  Testeur canonique de Brinskma.
  Travaux théoriques mais peu applicables.
  Devenus applicables par adaptation des modèles (Tretmans, Phalippou) et algorithmique efficace (TGV).

- Fondements: algèbres de processus, systèmes de transitions.

- Algorithmique: adaptée de l'algorithmique des automates (minimisation, déterminisation), des graphes, du model-checking

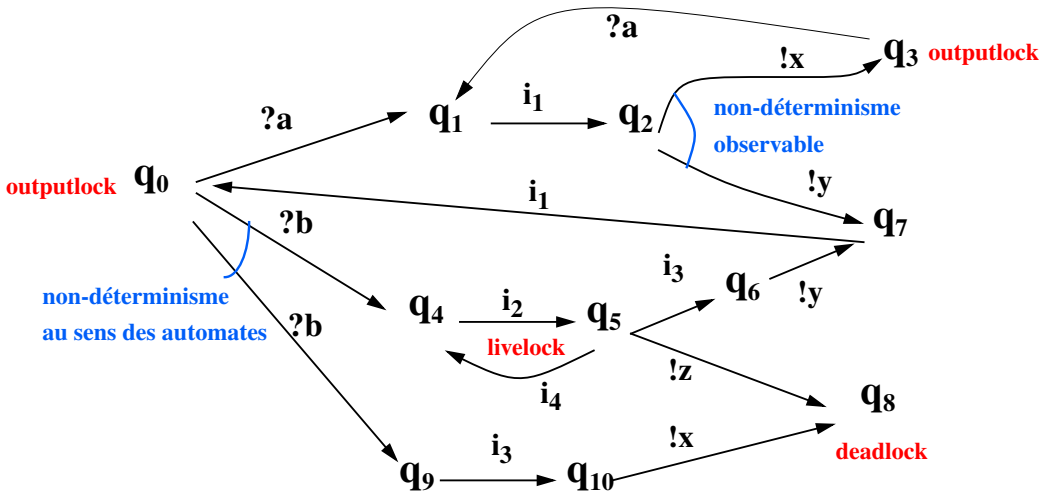
L'approche TGV

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

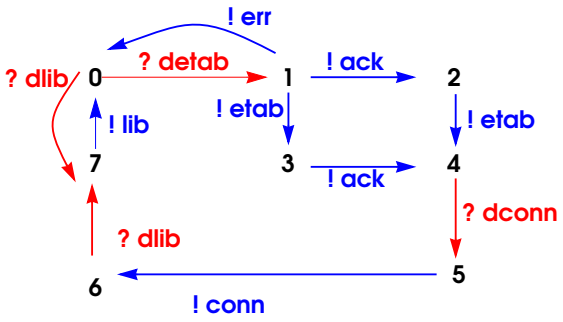
- **Spécification: S= (Q^S, A^S, $\rightarrow_S$, q_0^S) avec $A^S = A_I^S \cup A_O^S \cup I^S$**
- **Implémentation: I= (Q^I, A^I, $\rightarrow_I$, q_0^I) avec $A^I = A_I^I \cup A_O^I \cup I^S$**
  et $A_I^I \supseteq A_I^S$, $A_O^I \supseteq A_O^S$

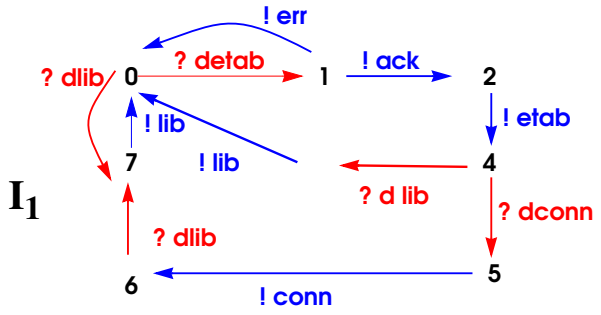
  On suppose que I ne peut refuser aucune entrée
- **Relation de conformité:**

  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.

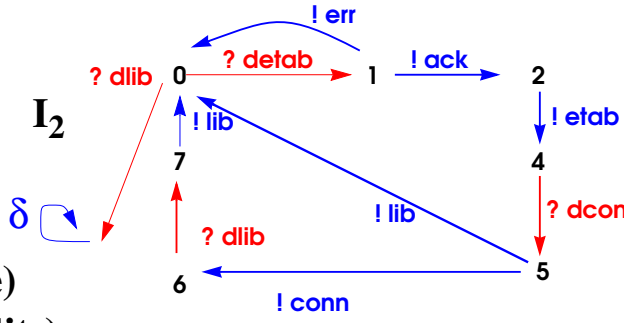
I ioco S $\Leftrightarrow \forall \sigma \in$ Traces(S^δ), Out(I after σ) $\subseteq$ Out(S^δ after σ)

Conformité: exemple





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

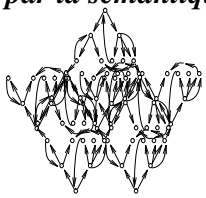


I_1 *ioco* S (choix d'implantation, spécification partielle)

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

Principes généraux de la génération

Spécification S
comportements donnés par la sémantique du langage



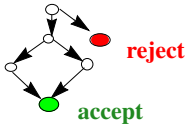
conformité ioco

Implémentation boîte noire I

cohérence (préordre)

GÉNÉRATION

Objectif de test TP
automate permettant la sélection des tests

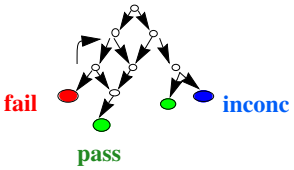


IUT I

Application du test sur IUT

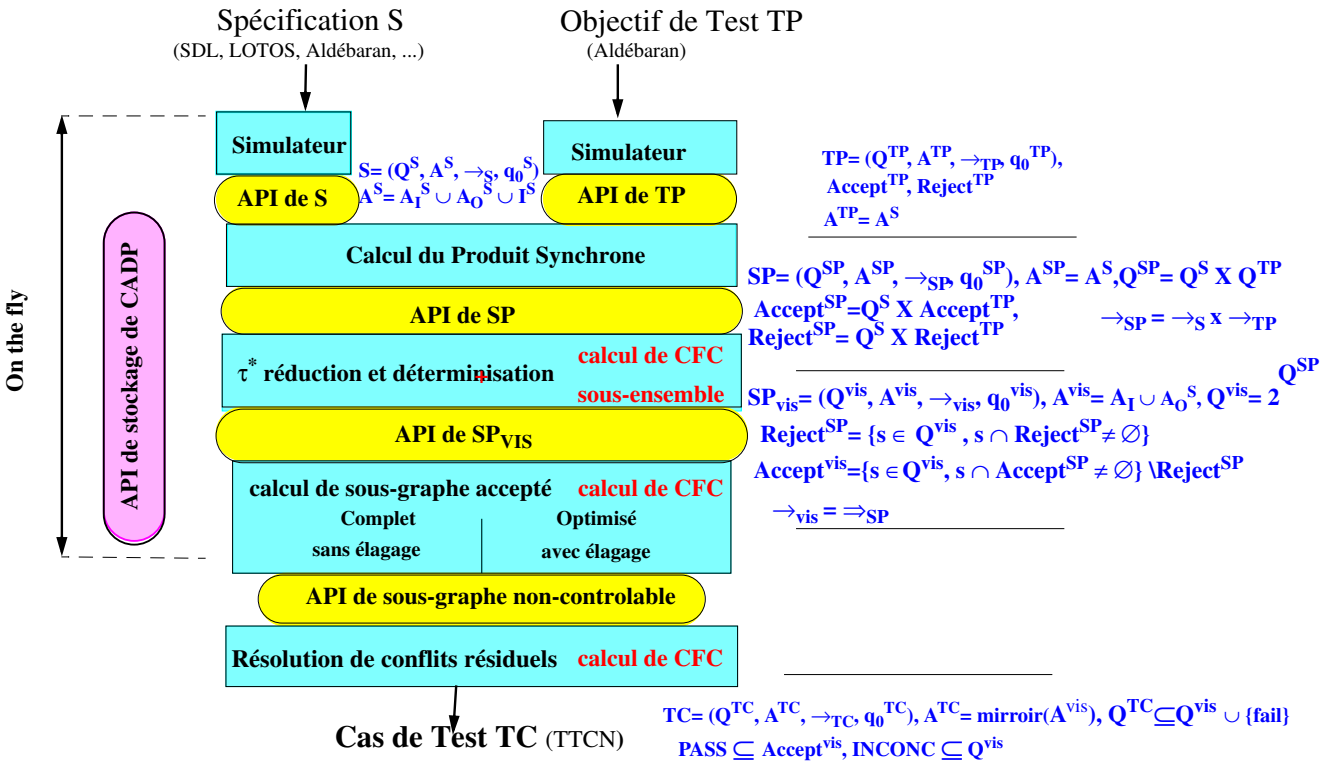
Verdict

Cas de test TC



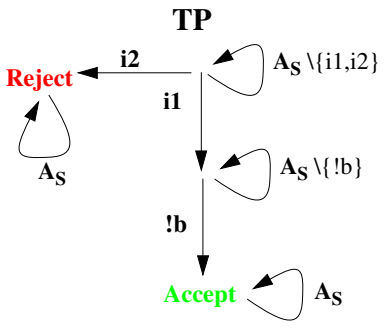
Propriétés : non-biais: I || TC -> fail => not (I ioco S)
exhaustivité: si équité de I, not(I ioco S) => exists TP tq TC(S,TP) || I -> fail

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**

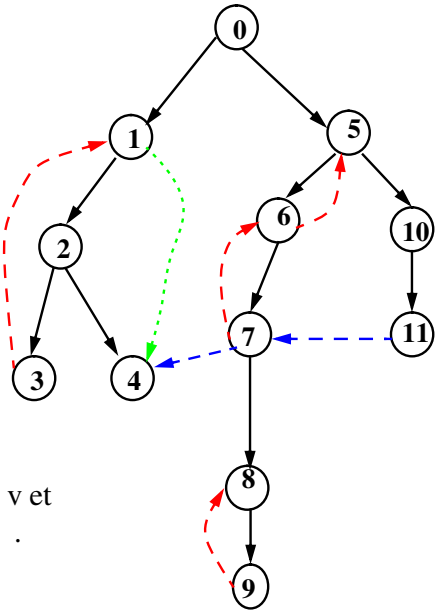
Composantes Fortement Connexes

- 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

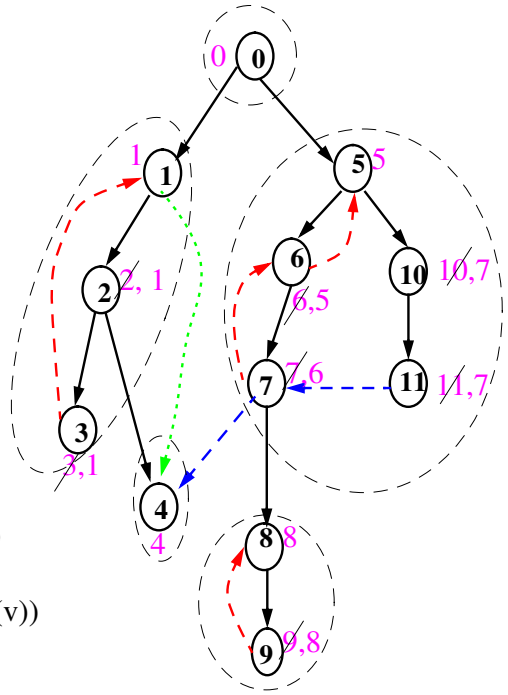
Lowlink(v)=plus petit numéro d'un sommet de même CFC que v et atteignable par des ⟶ et au plus un - - -▸ ou un - - ▸ .

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

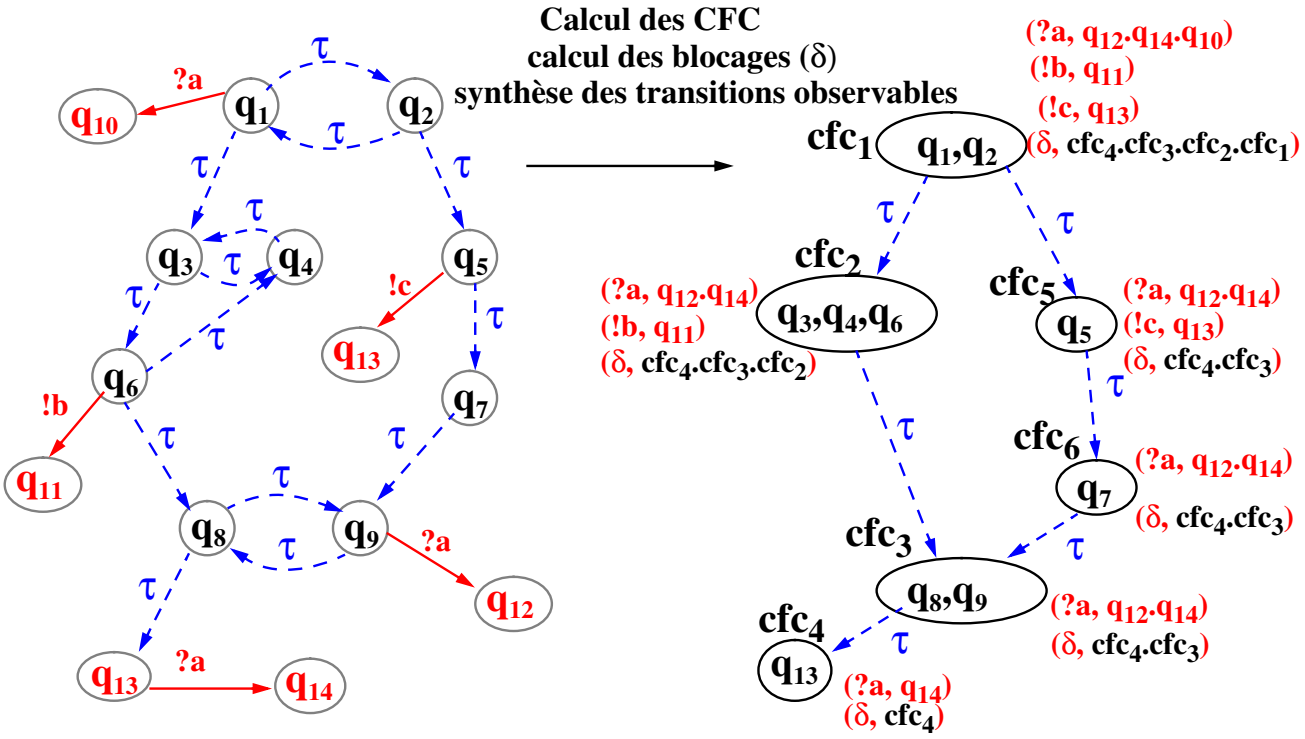
Tarjan (suite)

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

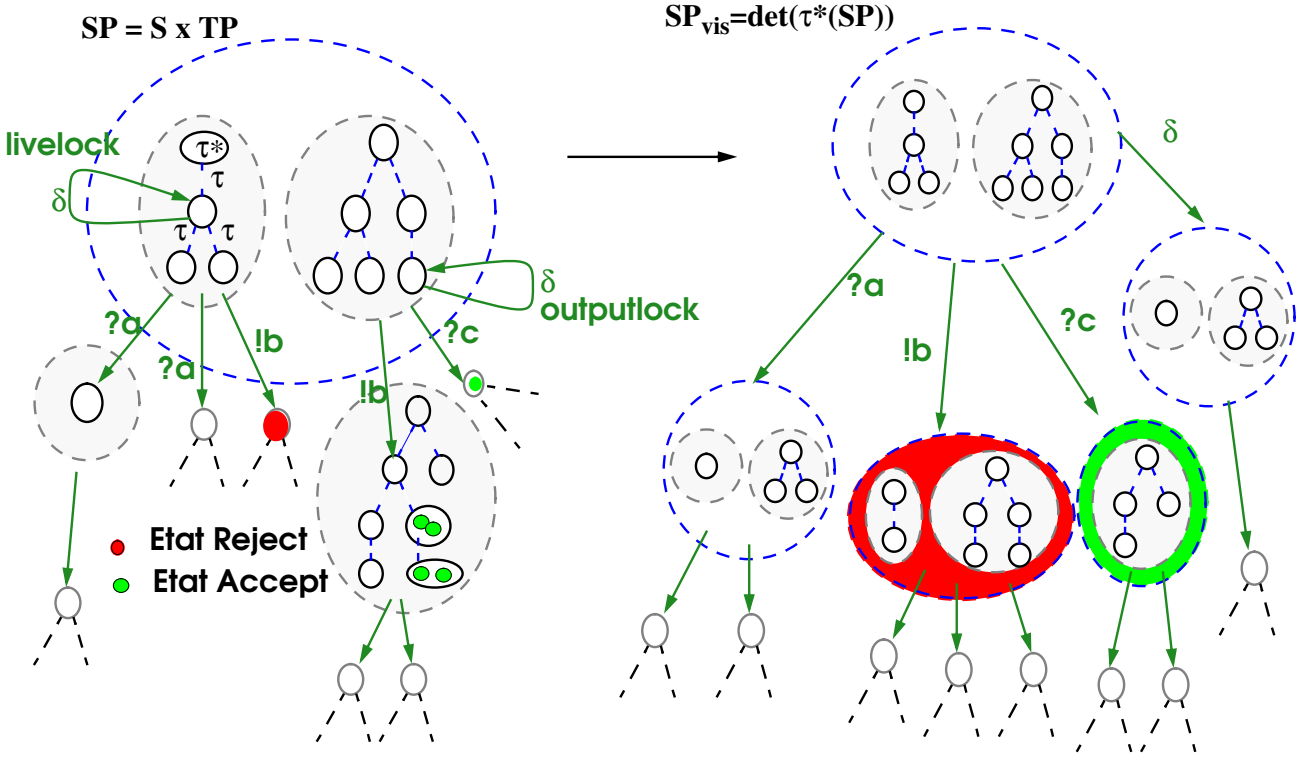
Lowlink est synthétisé le long des ⟶, - - -▸ et - - ▸
- initialement i:=0;
- v ⟶ w : lowlink(w) := number(w) := i := i+1;
  push(w) dans DFS-stack et CFC-stack
- v - - -▸ w ou v - - ▸ w et w in CFC-stack:
  lowlink(v):=min(lowlink(v), number(w))
- pop(v) de DFS-stack: si lowlink(v) = number(v) alors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

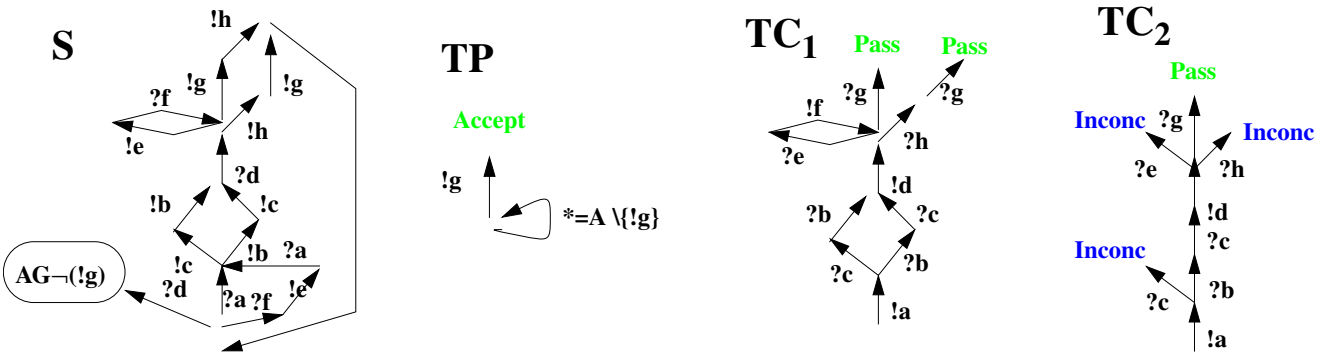


τ*-réduction et déterminisation (vue globale)



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

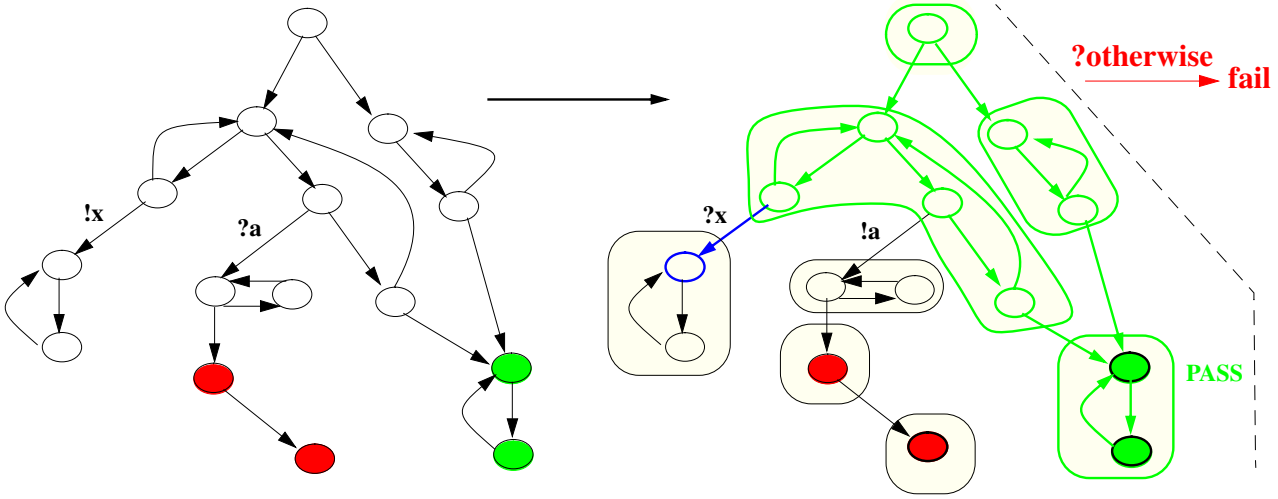
CTG = (L2A ∪ Inconc ∪ {fail} ,$\overline{A}_{vis}$, s_0, →)

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 $\xrightarrow{\overline{a}}$ 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-arc$_{in}$	Tree-arc$_{out}$
cross-link$_{in}$	cross-link$_{out}$
short-cut$_{in}$	short-cut$_{out}$
frond	

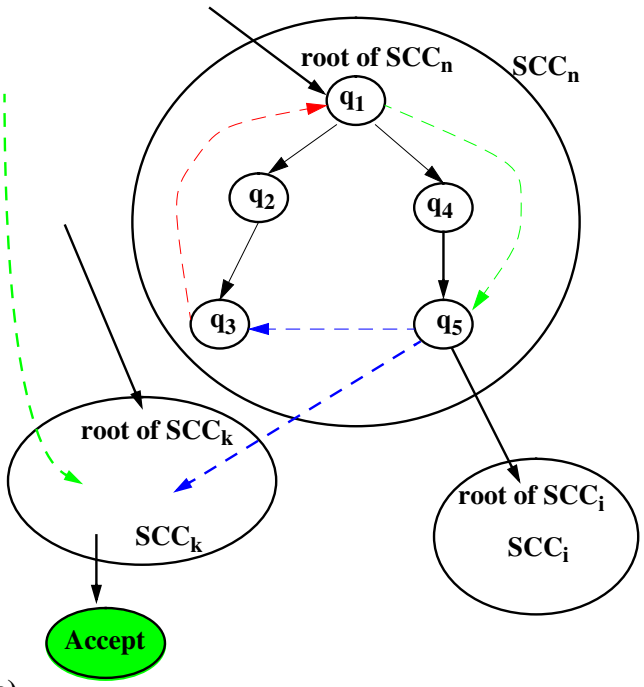
Pendant le calcul des CFC, L2A est synthétisé vers les racines de CFC le long des:

Tree-arc$_{in}$ Tree-arc$_{out}$

(cross-link$_{in}$) cross-link$_{out}$

(frond)

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



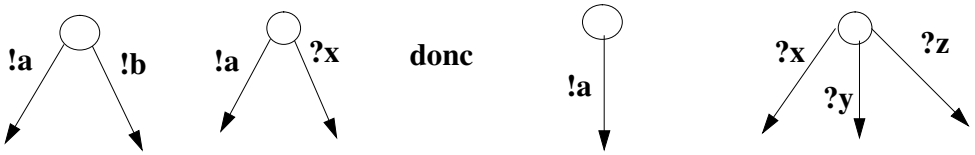
Comparaison avec le model-checking

- 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)

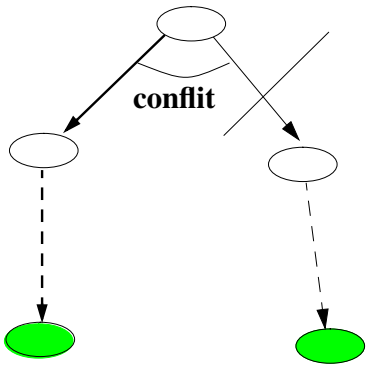
Résolution de conflits

- 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

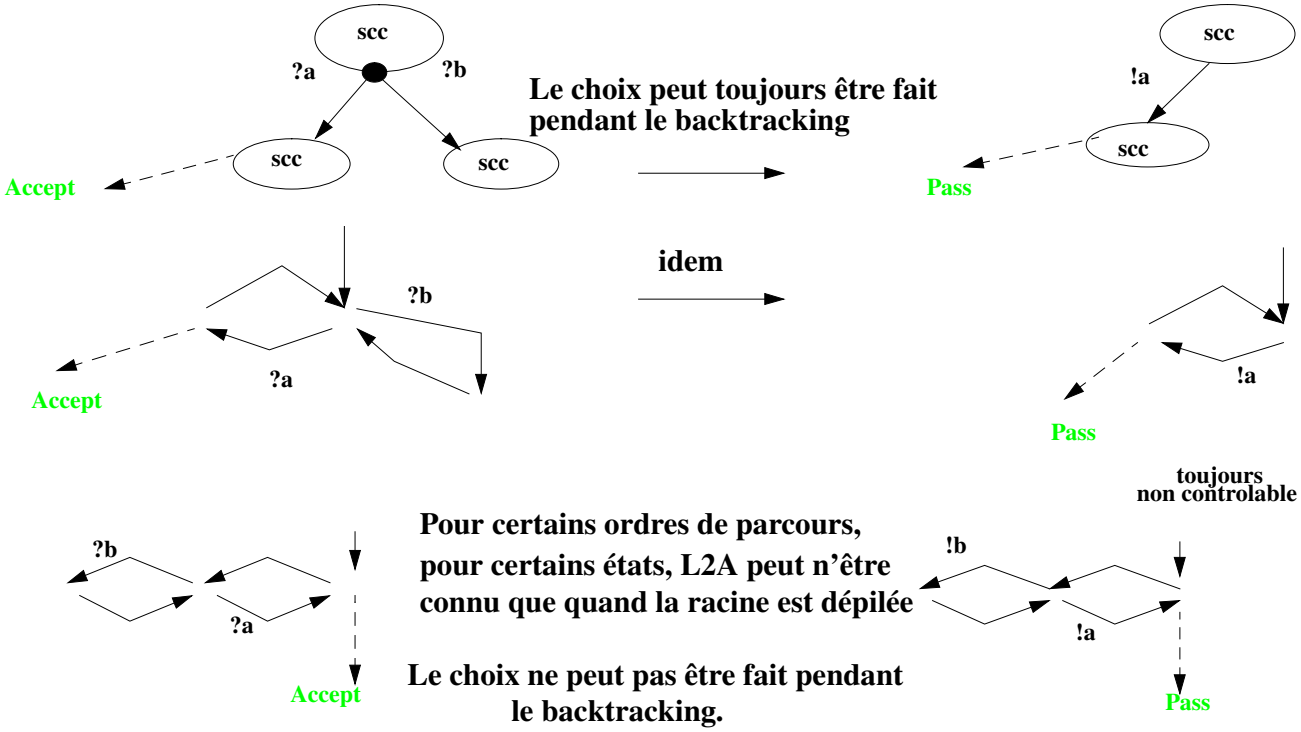
- profondeur avec CFC (Tarjan) :
synthèse de l'information L2init ds $\rightarrow_{CTG}^{-1}$
-> garbage collection

En résumé

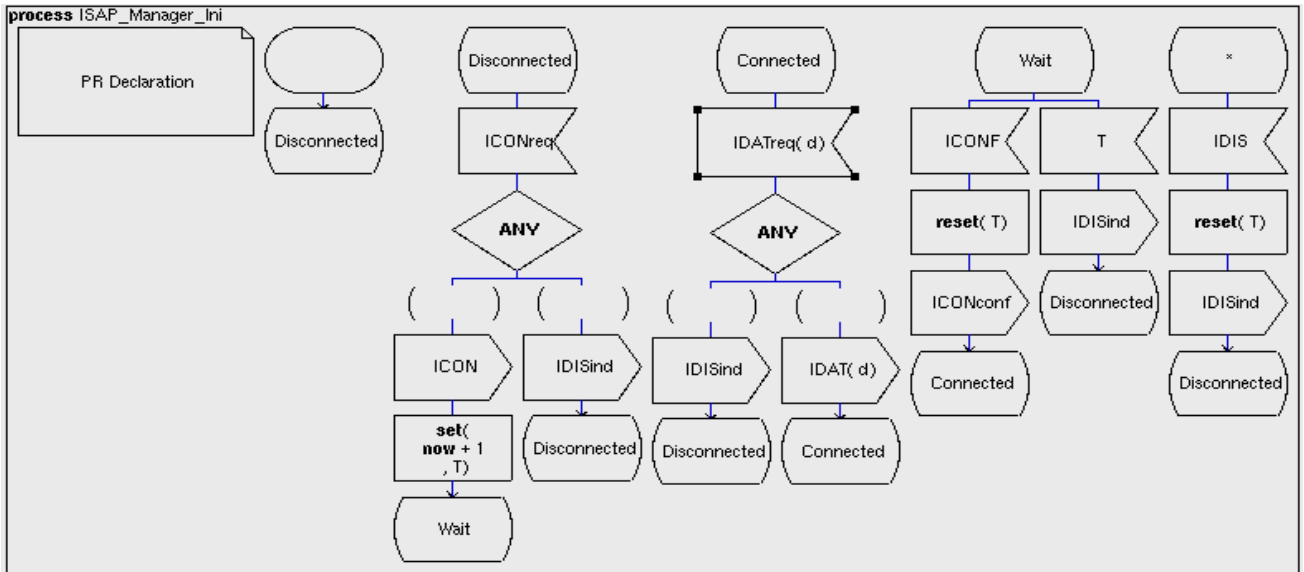
- CTG n'est pas controlable.
- Le calcul de CTG fournit un graphe contenant tous les tests pour une spécification S et un objectif de test TP donnés.
- CTG peut ensuite être élagué pour fournir un cas de test contrôlable.
- Mais le calcul de CTG nécessite de parcourir tout PS_{vis} donc tout S (au moins 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

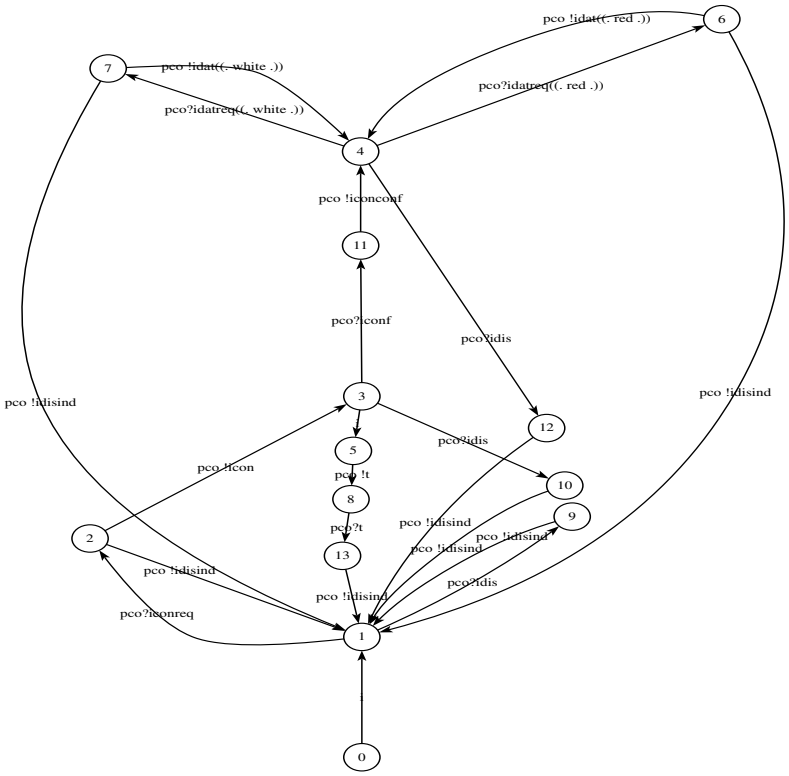


Exemple: processus initiateur d'Inres en SDL



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.

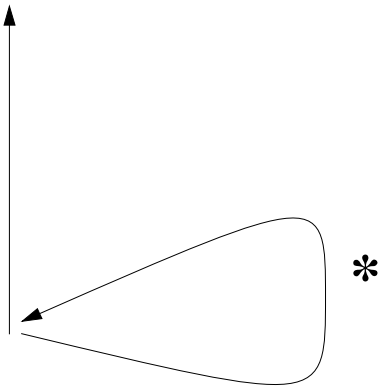
Graphe d'état (non minimal)



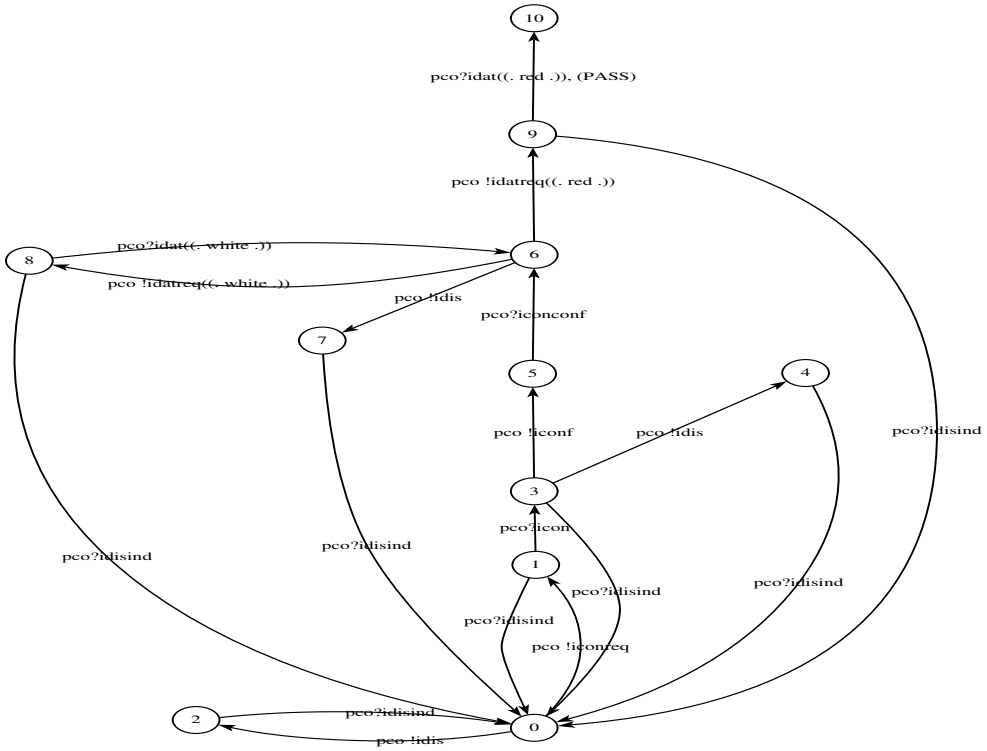
Objectif de test

Accept

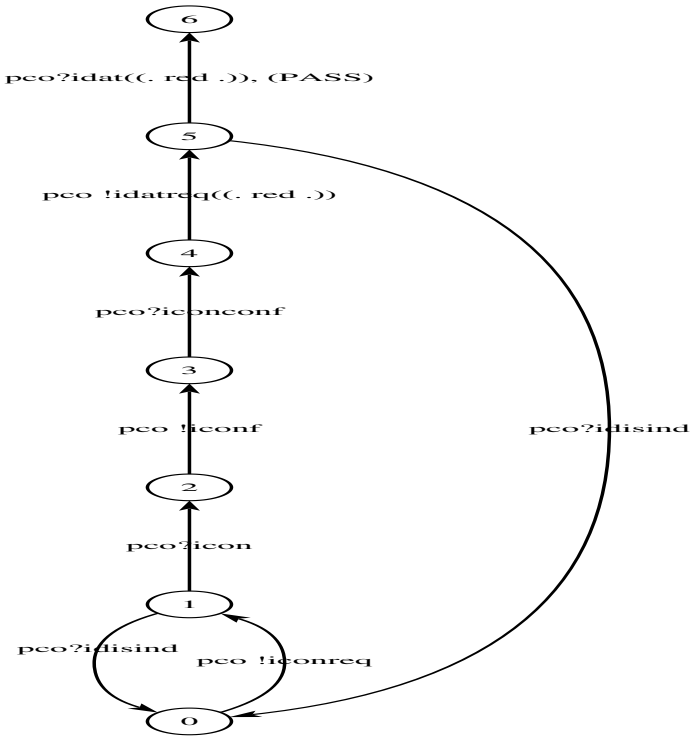
idat(0)



CTG généré à la volée par TGV



Cas de test produit à la volée



Cas de test TTCN

Test Case Dynamic Behaviour

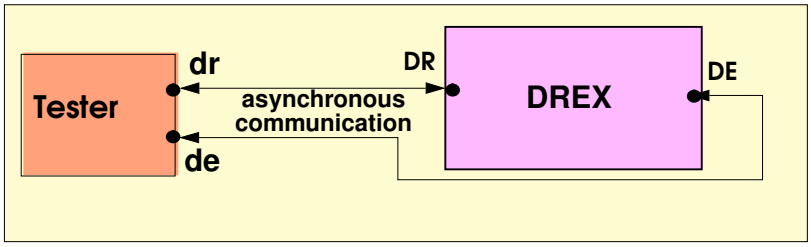
Test Case Name	: graphes/t01_fly_obs_2
Group	:
Purpose	:
Default	:
Comments	:

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	

Expériences d'utilisation de TGV

❑ **EGT/STEI in 94: protocole DREX, version militaire D en SDL**

- 5 spec (1 par service): 1 process, 35 transitions SDL, 1800 lignes de SDL PR, 50 pages de SDL GR.



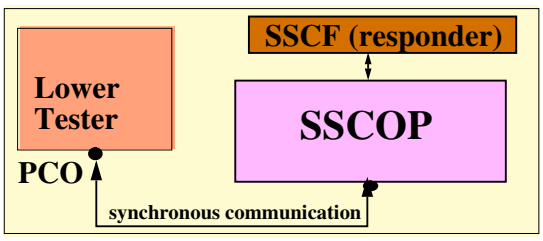
- 1ère version de TGV: graphe d'état explicites
- sélection de 50 Objectifs de test et génération des tests
- comparaison avec les tests manuels (détection d'erreurs).
- comparaison avec TVeda(CNET) et TTCgeN(Verilog)
- démonstrait clairement l' intérêt de l'approche en termes d'efficacité et de qualité des tests.

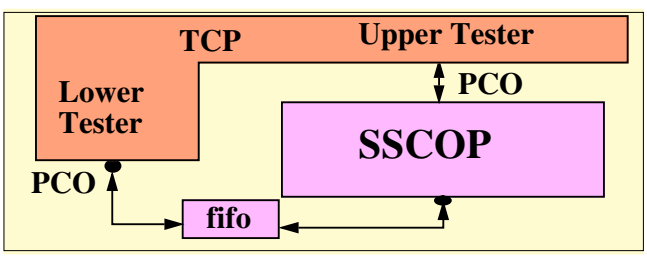
❑ **Action Vasy du GIE Bull-Inria Dyade 96-98 avec Inria Rhones-Alpes:**
protocol de cohérence de cache de l'architecture Polykid en Lotos
2000 lignes de Lotos (1800 ADT, 200 contrôle), 1 process.

- ✍ experimentation de la 1ère version de TGV "à la volée" sur une spec Lotos, connecté à Open Caesar.
- ✍ quelques difficultés du au contexte hardawre <> Telecom: pratique différente du test, adaptation des tests à la plateforme de test de Bull (drivers de test en C, batch testing <>interactif).
- ✍ les cas de test produits par TGV ont été passé sur l'architecture Polykid.
- ✍ bonne appréciation de Bull

❑ **Forma 97-99: protocol SSCOP de l'ATM en SDL**

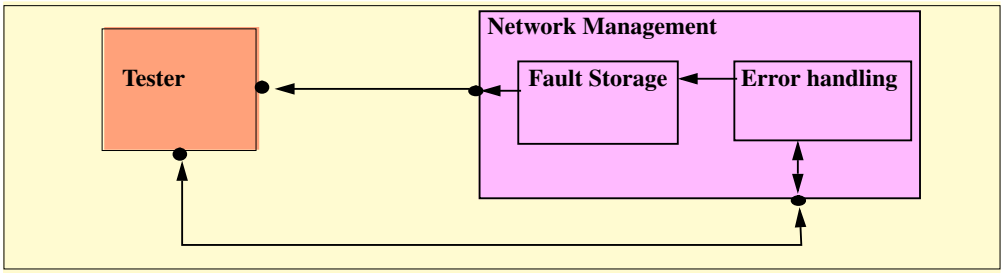
- 1 process (2 dans le cas asynchrone), 7000 lignes de SDL PR, 100 pages de SDL GR, 175 transitions SDL
- conception et implémentation de la première version à la volée connectée au simulateur ObjectGéode de Verilog.
- description de 50 objectifs complexes.
- plusieurs campagnes de génération avec des architectures de test différentes (locale avec 1PCO, distante avec 2PCOs).





❑ **Modistarc 97-98: protocole de gestion de réseau embarqué dans les automobiles en SDL.**

- draft du protocole: 2 processus, 1000 lignes de SDL PR, 25 pages de SDL GR



- faisabilité de la génération à la volée montrée sur quelques exples.
- connexion de TGV “explicite” à SDT de Telelogic par traduction des graphes d’état de SDT au format Aldebaran.

Transfert industriel

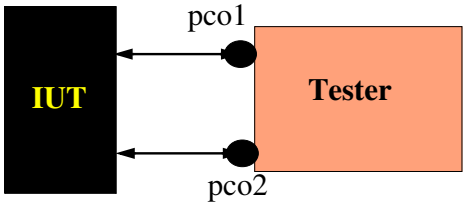
❑ **GAT project 98-99: France Telecom, Verilog, Verimag, Pampa**

Conception et développement d’un outil industriel fondé sur TVéda (CNET), TTCgeN (Verilog) et TGV dans le simulateur ObjectGéode.

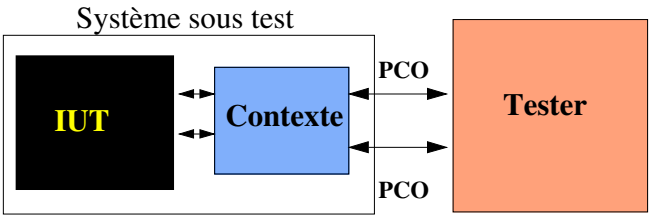
- génération automatique d’objectifs de test (MSC et scenarii) avec des techniques issues de TVeda incorporant une stratégie de couverture (tous les chemins observables)
- génération de cas de test avec des algorithmes issus de TGV
  - depuis les objectifs générés
  - depuis des objectifs specifiques en Goal

La conception et le codage et une partie de la validation des algorithms de TGV est faite à l’Irisa.

Test dans un contexte



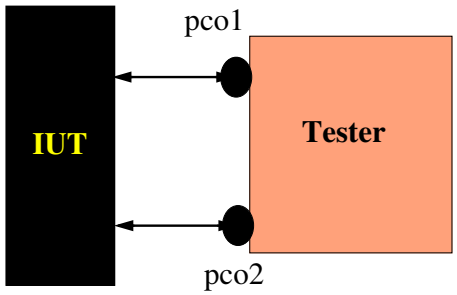
Interaction synchrone



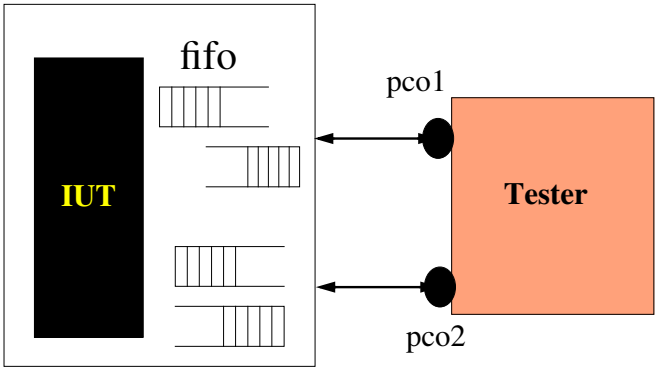
- **Le contexte C transforme la perception de l'IUT I en C || I en modifiant l'observabilité et la controlabilité.**
- **Conformité dans un contexte:**
  I ioco S in context C ⇔ C || I ioco C || S
- **Attention, en général**:
  I ioco S in context C ≠> I ioco S (perte d'information)
  I ioco S ≠> I ioco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

  Pas très satisfaisant mais peut on faire mieux ?

L'exemple du test en environnement asynchrone



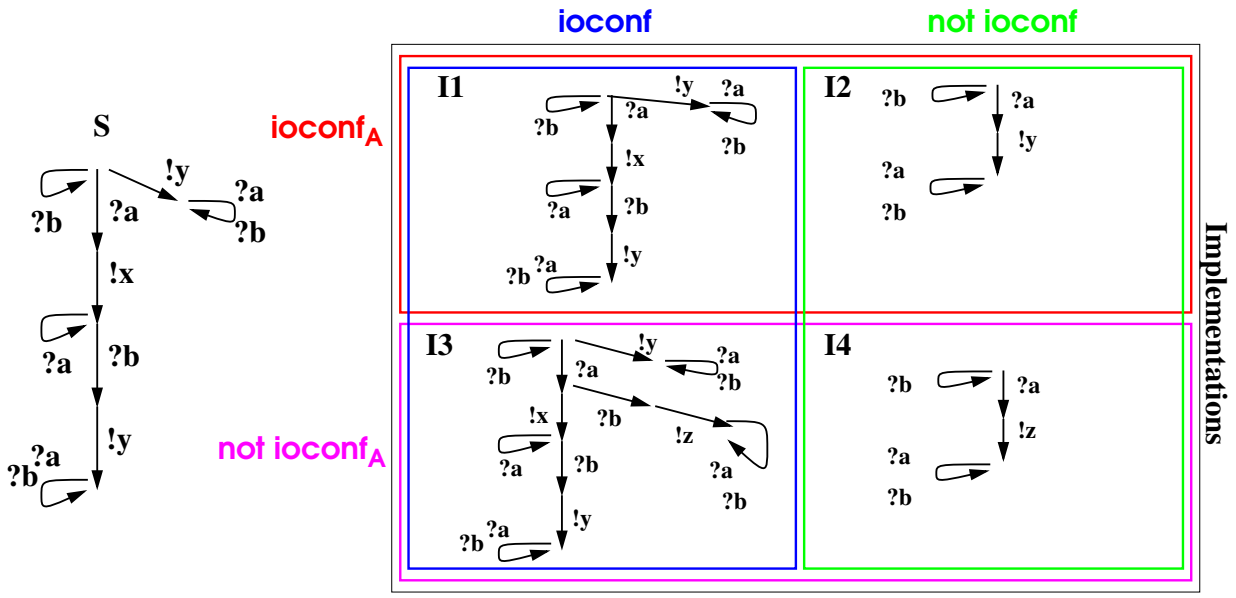
Interaction synchrone



Interaction asynchrone

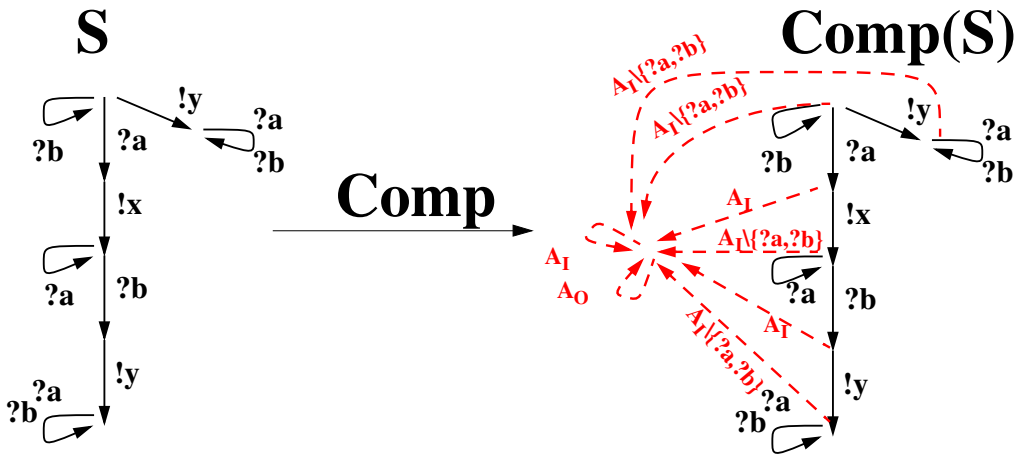
I ioconf S ⇔
∀ σ ∈ Traces(S), Out(I after σ) ⊆ Out(S after σ)

$I\ ioconf_A\ S \Leftrightarrow I \| A\ ioconf\ S \| A$



La non conservation de la conformité dans un contexte vient de la non-complétude de la spécification qui autorise l'IUT à faire ce qu'elle veut après une réception non spécifiée.

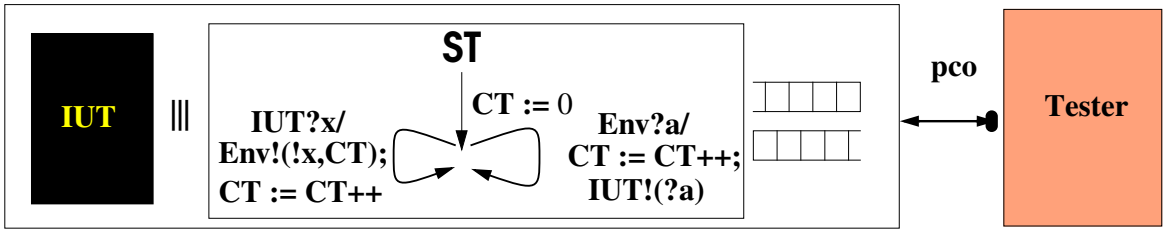
Complétion

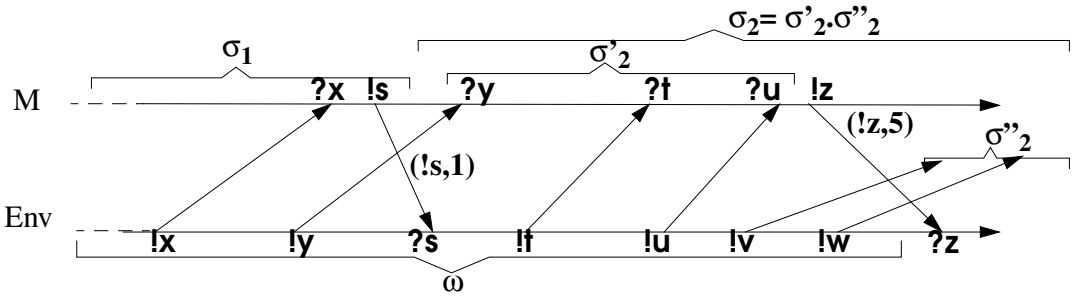


Pour S input-complete: I ioconf S ⇔ Trace(I) ⊆ Trace(S)
De plus: I ioconf S ⇔ I ioconf Comp(S)
Monotonie: Trace(M) ⊆ Trace(M') => Trace(C(M)) ⊆ Trace(C(M'))
Pour C monotone, I, S input-complètes: I ioconf S => C(I) ioconf C(S).
Cas asynchrone, I, S input-complètes: I ioconf S => I ioconf$_A$ S
Donc I ioconf S => I ioconf$_A$ Comp(S)

Test asynchrone estampillé (cas d'un PCO)

- Numéroter les émissions de l'IUT pour affiner le diagnostic en environnement asynchrone.





Puissance du test asynchrone estampillé

Pour S, I séquentielles, input-complètes, 1 PCO fifo,
le test estampillé asynchrone a la même puissance que le test synchrone:

I ioconf S ⇔ st(I) ioconf$_A$ st(S)

Génération:

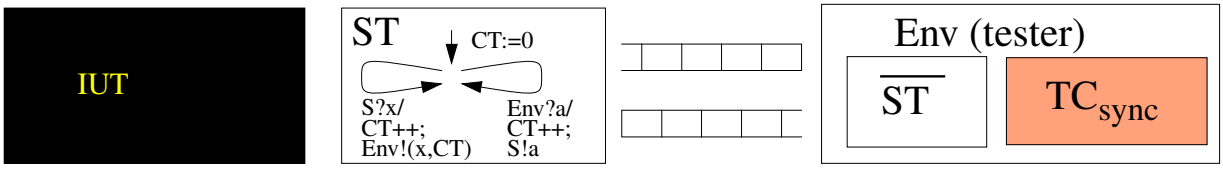
S $\xrightarrow{\text{complétion}}$ Comp(S) $\xrightarrow{\text{estampillage}}$ st(Comp(S)) $\xrightarrow{\text{transfo asynchrone}}$ st(Comp(S)) || fifo

st(Comp(S)) || fifo $\xrightarrow{\text{génération}}$ TC: Cas de test non biaisé

TC non biaisé pour ioconf$_A$ donc
TC rejette I => not(I ioconf$_A$ Comp(S)) => not(I ioconf S)

Test synchrone en asynchrone

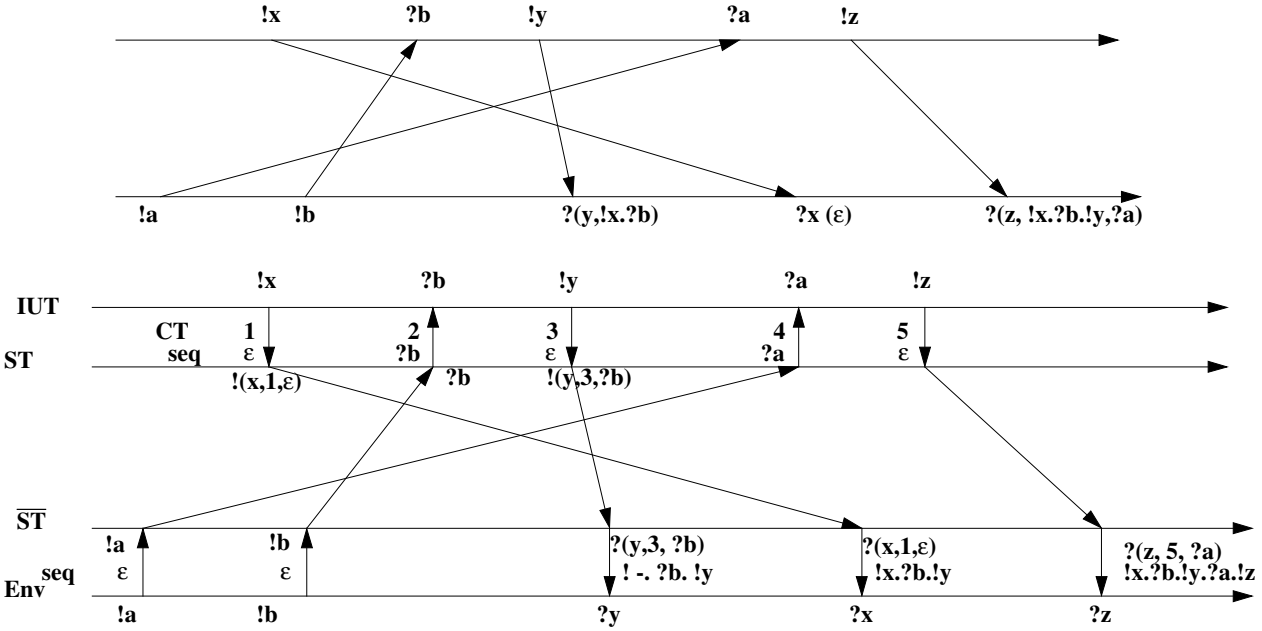
- **Idée:** utiliser les tests synchrones en asynchrone en décodant l'estampille. ST est inversible en $\overline{ST}$



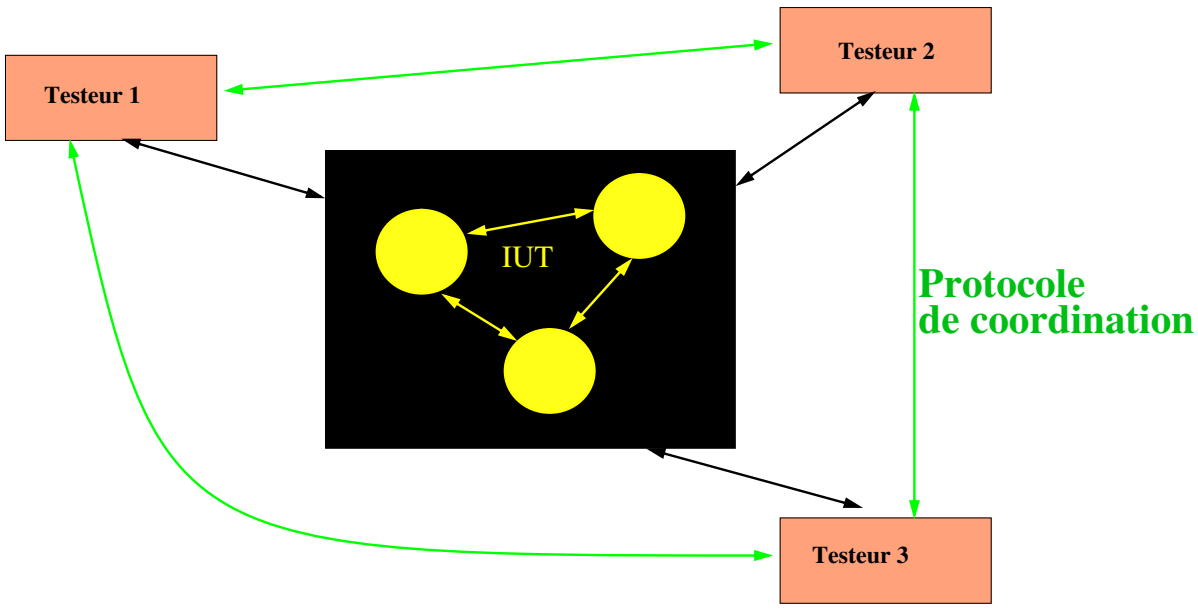
■ **Input:** TC: test case. **Output:** verdict
Invariant $\sigma_1 = \overline{st}(\omega)[1]$ **and** $\sigma_2 = \overline{st}(\omega)[2]$
Initialization: $\omega := \epsilon$; $\sigma_1 := \epsilon$; $\sigma_2 := \epsilon$; verdict:= nil
(* $\overline{st}(\epsilon) = (\epsilon, \epsilon)$ *)
while verdict = nil **do**
non-deterministic choice
if $Out(TC\ after\ \sigma_1.\sigma_2) \neq \emptyset$
send(a) where $\{a\} = Out(TC\ after\ \sigma_1.\sigma_2)$
(* a is unique as test cases are semi-controllable *)
$\omega := \omega.?a$; $\sigma_2 := \sigma_2.?a$; (* i.e. $\overline{st}(\omega.?a) = (\overline{st}(\omega)[1], \overline{st}(\omega)[2].?a$*)
if input queue is not empty
receive((z, i))
$\omega = \omega.(!z, i)$
let σ_2', σ_2'' s.t. $length(\sigma_1.\sigma_2') = i$ and $\sigma_2 = \sigma_2'.\sigma_2''$
$\sigma_1 := \sigma_1.\sigma_2'.!z$; $\sigma_2 := \sigma_2''$; (* $\overline{st}(\omega.(!z, i)) = (\overline{st}(\omega)[1].\sigma_2'.!z, \sigma_2'')$ *)
verdict := verdict $(TC\ after\ \sigma_1)$
end

Cas de PCOs multiples

- **Même si chaque PCO est fifo la communication est globalement non fifo.**



Test réparti



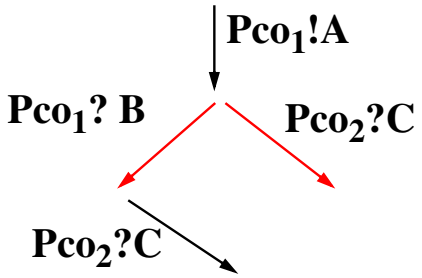
Génération de tests distribués

- **Problème difficile dans sa généralité:**
  - **asynchronisme**
  - **concurrrence entre testeurs**

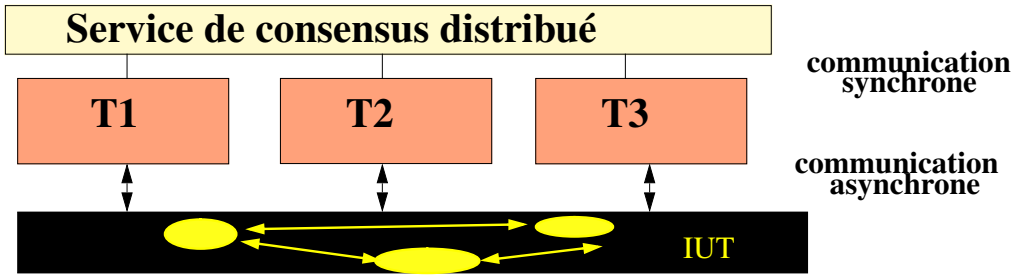
- **Problème traité: distribution d'un test séquentiel en testeurs répartis:**
  - **générer un test à partir de la spécification du système global.**
  - **Distribuer ce test en tests séquentiels en fonction de la localité des actions sur les PCOs.**

Problème des choix non-locaux

- Séquentialité simulée par l'échange de messages
- Problème des choix non locaux:

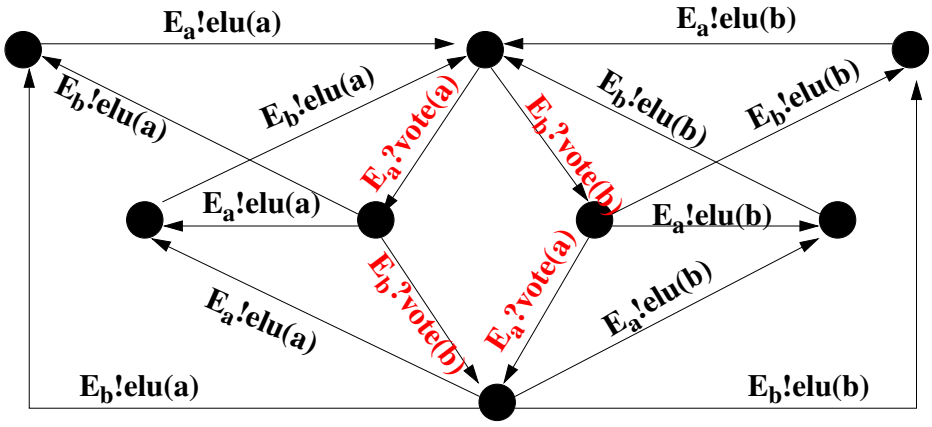


Solution: utiliser un service de consensus



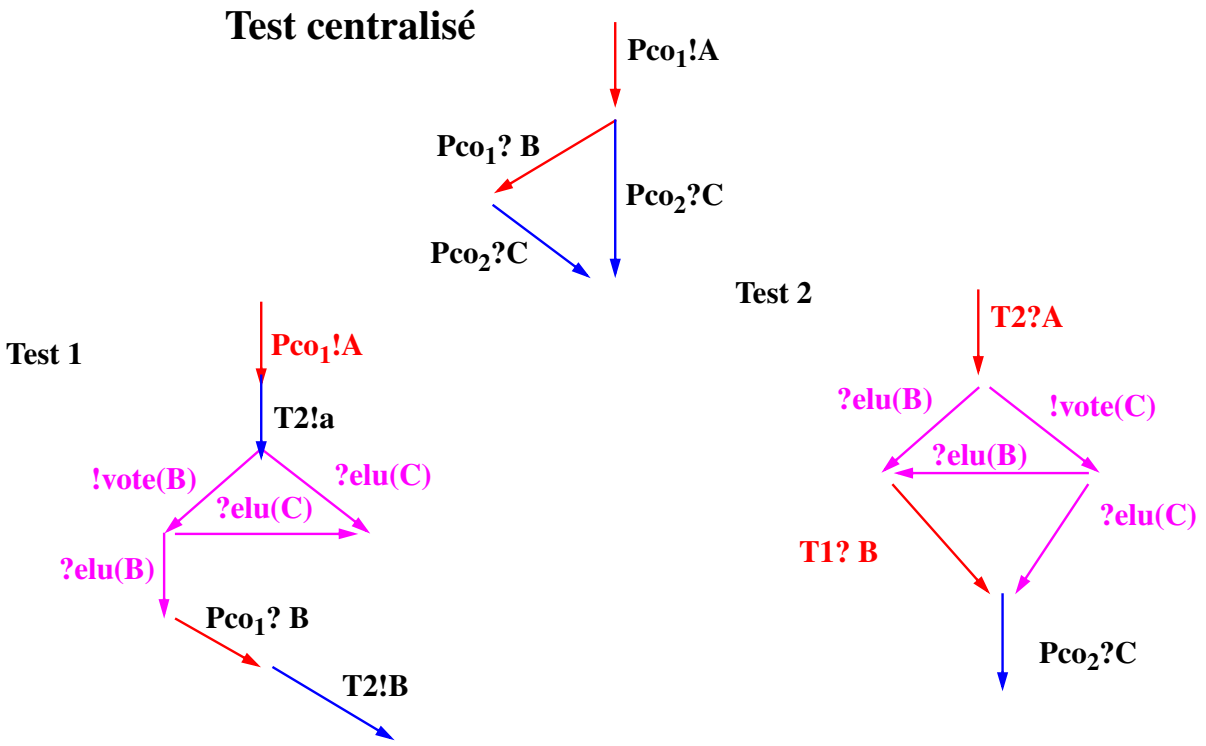
Service de consensus

Cas de 2 labels



Il existe des implémentations distribuées du consensus.

Distribution



Résultats

- **Règles de distribution:**
  TC: Test asynchrone séquentiel -> TC_i: Test distribués
- **Correction de la distribution:**
  Trace((//TC_i x CS) /PCO) = Trace (TC)

  Ce qui montre que
  (//TCi x CS) /PCO a la même puissance de test que TC

  En particulier
  si TC non biaisé pour $ioconf_A$
  alors (//TCi x CS) /PCO non biaisé pour $ioconf_A$.

Perspectives

- **Test de logiciel objet distribué:**
  - **connexion de TGV à UML.**
  - **strategies de test d'intégration**
- **Génération de tests symboliques:**
  - **connexion de TGV à un solveur de contraintes.**
  - **utilisation d'abstractions**
  - **combinaison test et preuve**

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

ITU-T SG~10/Q.8 ISO/IEC JTC1/SC21~WG7. Information retrieval, transfer and management for OSI; framework: Formal Methods in Conformance Testing. Committee Draft 13245-1, ITU-T proposed recommendation Z500. ISO/ITU-T, 1996.

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.

D.Lee and M.Yannakakis. Principles and methods of testing finite state machines - a survey. Proc. of the IEEE}, 84(8):1090--1123, August 1996.

A. Gill. Introduction to the theory of finite-state machines. Mc Graw-Hill. 1962 (difficilement trouvable)

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.

R.G. De Vries and J.Tretmans. On-the-Fly Conformance Testing using Spin. In G.Holzmann, E.Najm, and A.Serhrouchni, editors, Fourth Workshop on Automata Theoretic Verification with the Spin Model Checker, ENST 98 S 002, pages 115--128, Paris, France, November 2 1998.

.

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. Jeron, 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. Jeron, 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

L. Doldi, V.Encontre, J.C.Fernandez, T.Jeron, 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.

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.

T. Jéron, P. Morel: Test generation derived from model-checking, to appear in CAV 99, Trento Italy, Juillet 99

M. Bozga, J.-C. Fernandez, L. Ghirvu, C. Jard, T. Jeron, A. Kerbrat, P. Morel, and L.Mounier. Verification and test generation for the SSCOP protocol. Journal of Science of Computer Programming, Special Issue on The Application of Formal Methods in Industrial Critical Systems , To appear, 1999.

A. Kerbrat, T. Jeron, and R. Groz. Automated Test Generation from SDL Specifications. In Proc. 9th SDL FORUM (Montréal, Quebec, Canada)}, June 1999.

❑ 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.

Voir aussi les travaux sur les tests de systèmes temporisés.En particulier:

J.G. Springintveld, F.W. Vaandrager and P.R. D'Argenio. Testing timed automata. Technical Report CSI-R9712, Computing Science Institute, University of Nijmegen, August 1997. Also available as CTIT Technical Report 97-17, Centre for Telematics and Information Technology, University of Twente, September 1997. Full version to appear in Theoretical Computer Science.

Thèse de Patrice Laurencot au Labri (Bordeaux)

❑ Algorithmes

R.T arjan. Depth-first search and linear graph algorithms.SIAM Journal Computing, 1(2):146--160, June 1972.

J.E. Hopcroft and J.D. Ullman. Introduction to automata theory, languages, and computation. Addison-Wesley series in computer science, 1979.

M. Gondran, M. Minoux. Graphes et algorithmes. Eyrolles. 2nde édition,1985