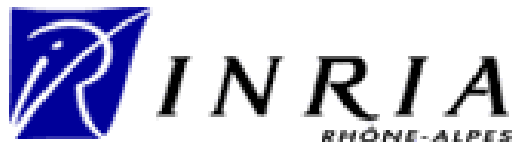

NTIF

Un modèle symbolique général
pour les processus séquentiels communicants
contenant des données

Hubert Garavel, Frédéric Lang

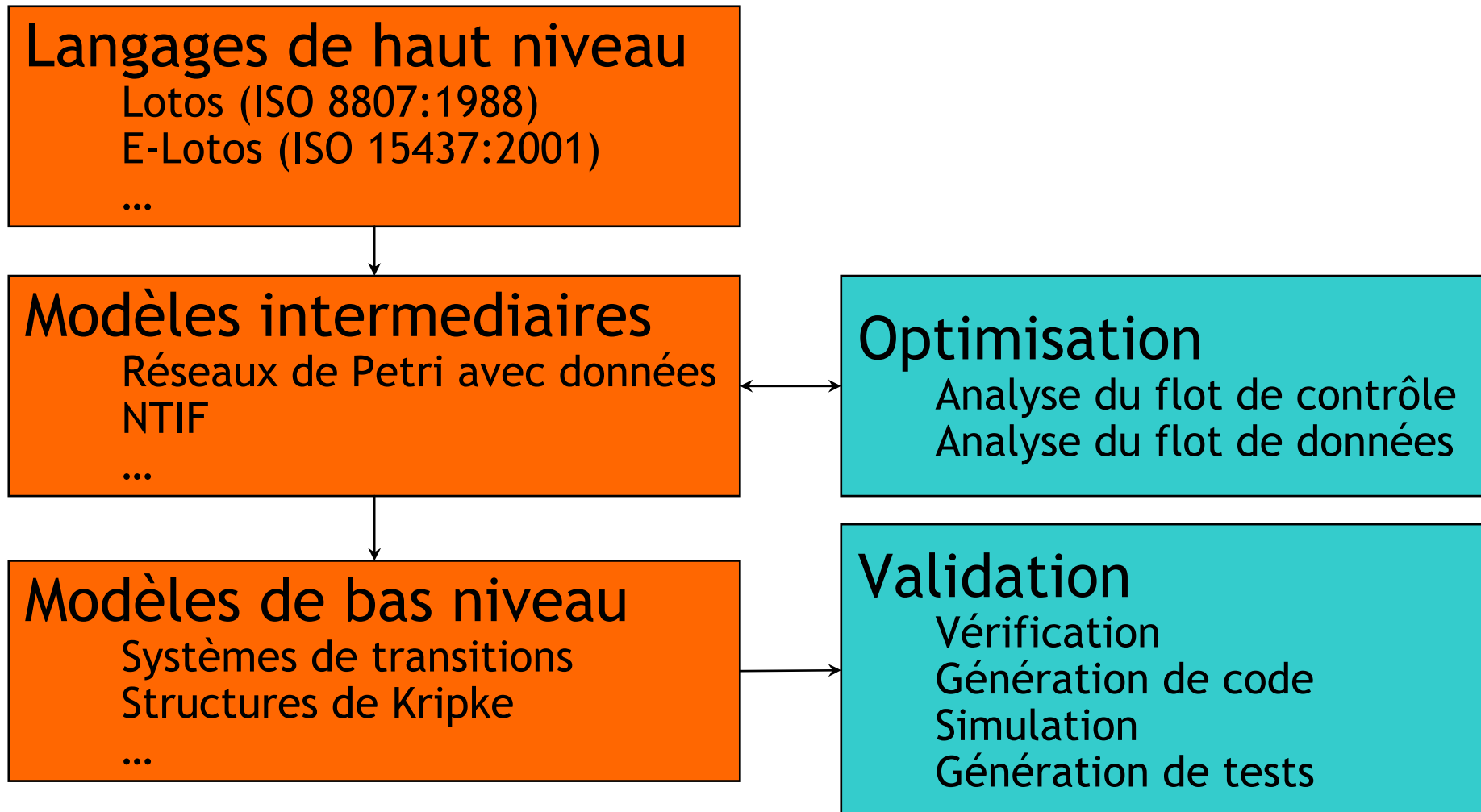
INRIA Rhône-Alpes / VASY
655, avenue de l'Europe
F-38330 Montbonnot Saint Martin



Introduction



Modèles intermédiaires



Un des premiers exemples de modèle intermédiaire

- Réseaux de Petri interprétés avec données (Garavel & Sifakis, 1990)
 - Variables d'état globales
 - Transitions étiquetées par des communications
 - Transitions étiquetées par des actions (gardes, affectations de variables, resets, etc.)
- Avantages
 - Séparer les aspects dépendants/indépendants du langage
 - Améliorer l'efficacité des algorithmes de vérification en se basant sur un modèle sémantique plus simple
 - Peut être utilisé pour plusieurs langages de haut niveau



Choix d'un modèle intermédiaire

Deux critères de choix qui s'opposent:

- Généralité et expressivité
 - Le modèle doit pouvoir servir pour plusieurs langages
 - Le modèle doit préserver la sémantique de ces langages
- Simplicité
 - Le modèle ne doit pas contenir de détails non-nécessaires (*sucres syntaxiques*, etc.)
 - Le modèle ne doit pas induire de surcoût lors de l'analyse



Formalismes existants basés sur condition/action



Les modèles à base de condition/action

- Nombreux modèles basés sur condition/action

Input/Output Automata, Linear Process Operators (muCRL), Symbolic Transition Systems (CCS), IF (SDL), Machines à états communicantes (Basic LOTOS), etc.

- Condition/action : transitions $s \xrightarrow{E \Rightarrow A / C} s'$ où

- s, s' sont des états
- E est une condition de franchissement
- A (action) est une séquence d'affectations de variables
- C est un événement de communication
 - Entrée : $G ? V$ (porte G , variable V)
 - Sortie : $G ! E$ (porte G , expression E)
 - Événement silencieux : τ



Limitations des modèles condition/action

4 limitations essentielles (développées dans les transparents suivants)

- Le franchissement d'une transition est déterminé par une condition unique
- Les conditions et les actions ne peuvent pas être entrelacées
- Le langage d'actions est trop réduit pour préserver les *sémantiques à grands pas*
- Les conditions booléennes présentes dans le langage de haut niveau sont dupliquées



1. Condition de franchissement unique (1/2)

- En fonction du formalisme la garde E est évaluée :

- Soit *avant* » la communication

Exemple $V > 2 \Rightarrow \text{null} / G ?V$

signifie « si $V > 2$ alors lire sur la porte G une nouvelle valeur pour V »

- Soit « *pendant* » la communication

Le même exemple signifie « ne tirer la transition que s'il est possible de lire sur la porte G une valeur pour V supérieure à 2 »

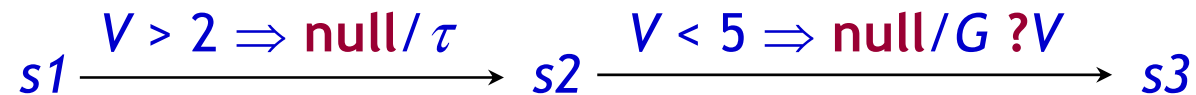


Condition de franchissement unique (2/2)

- Les deux types de conditions sont possibles au niveau langage

Exemple if $V > 2$ then $G ?V$ where $V < 5$

- Non-modélisable si évaluation « *avant* »
- Deux transitions nécessaires si évaluation « *pendant* »



- Les deux types de conditions doivent co-exister pour éviter des transitions supplémentaires



2. Condition et action ne peuvent pas être entrelacées

- Dans tous les modèles la condition est évaluée avant l'action
- En pratique des actions (affectations) précédant la condition seraient utiles

Exemple **if** $F(F'(V))$ **then** $G !V; V := F'(V)$

devrait pouvoir être écrit

$W := F'(V);$ **if** $F(W)$ **then** $G !V; V := W$

sans ajouter de transitions dans le modèle

- Plus généralement : des entrelacements entre conditions et actions sont nécessaires



3. Sémantique (1/3)

- Deux sortes de sémantiques existent pour les langages concurrents
 - *A petits pas* : une transition par affectation
Exemple $V_1 := 0; V_2 := 0; V_3 := 0$
 - 3 transitions par défaut en PROMELA
 - possibilité d'agréger explicitement: `dstep`
 - *A grands pas* : transitions induites par les communications
Exemple $V_1 := 0; V_2 := 0; V_3 := 0; G !V_1$
 - 1 transition en E-LOTOS
 - Les variables sont locales aux processus
 - Les affectations à elles seules n'induisent pas de transitions



Sémantique (2/3)

- Sémantique des modèles condition/action
 - Si l'action contient au plus une affectation alors la sémantique est purement à **petits pas**
 - Si l'action peut contenir des séquences d'affectations alors la sémantique est une combinaison de **petits pas** et de **grands pas**
- Dans les deux cas le langage d'actions **ne convient pas** pour les sémantiques à grands pas




Sémantique (3/3)

- Exemple boucles et sémantique à grands pas
 - $V[1] := 0; V[2] := 0; V[3] := 0$ peut se traduire par

$$s_0 \xrightarrow{V[1] := 0; V[2] := 0; V[3] := 0 / \tau} s_1$$

- Mais **for** i **in** $1..3$ **do** $V[i] := 0$ doit être traduit par

$$s_0 \xrightarrow{i := 1 / \tau} s_1 \xrightarrow{i > 3 \Rightarrow \text{null} / \tau} s_2$$


 $i \leq 3 \Rightarrow i := i+1; V[i] := 0 / \tau$

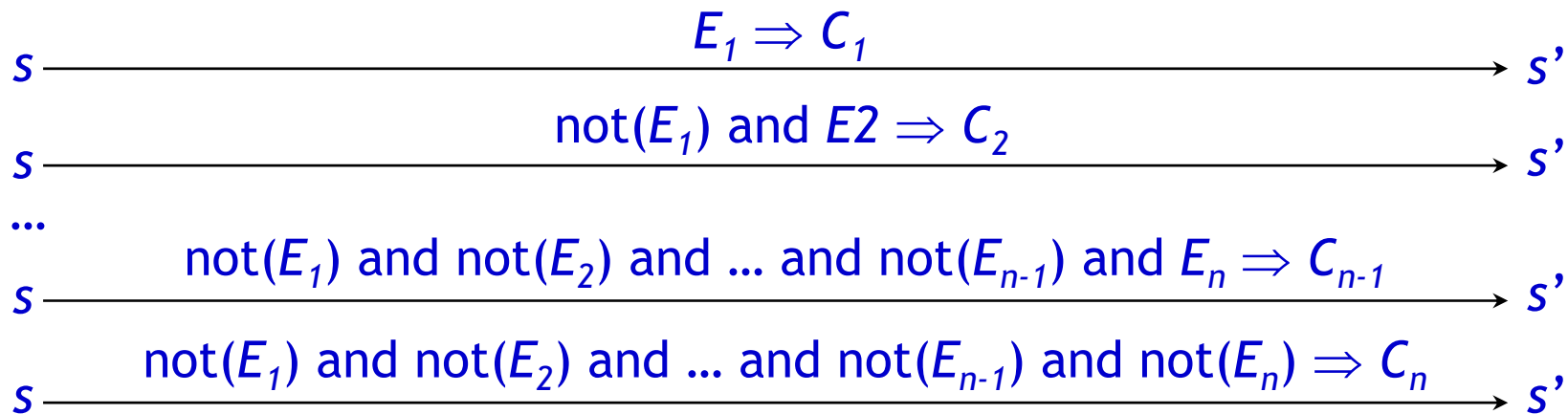
- Le langage d'action doit être étendu pour préserver les sémantiques à grands pas



4. Duplication des conditions (1/2)

- La traduction de conditionnelles (**if-else**, **case**) entraîne des duplications de conditions

Exemple **if** E_1 **then** C_1 **elsif** E_2 **then** C_2 ... **else** C_n
se traduit en



soit $n(n+1) / 2$ conditions à évaluer au lieu de n



Duplication des conditions (2/2)

- La duplication des conditions pénalise la facilité d'utilisation par des personnes
 - Modèles laborieux à écrire (risque élevé d'erreur)
 - Modèles difficiles à relire et corriger
 - Elle pénalise surtout l'efficacité de l'analyse
 - La condition et sa négation doivent être évaluées pendant le model checking ou la simulation
 - Des propriétés évidentes vues du programme source deviennent difficiles à prouver au niveau intermédiaire
- Exemple vérifier qu'au plus une (ou exactement une) transition peut être franchie d'un état donné



Conclusion sur les modèles condition/action

- Bien que très souvent utilisés dans la littérature les modèles condition/action ne sont bons :
 - Ni pour l'écriture à la main
 - Ni pour la lecture et le débogage
 - Ni pour les traitements automatisés
- Un meilleur formalisme est nécessaire : **NTIF** (créé en avril 2001)



NTIF : *The New Technology Intermediate Form*



Programme NTIF

- Un programme NTIF est un ensemble de processus séquentiels communicants avec des données
 - Le parallélisme et le temps seront traités plus tard
- Un processus NTIF est composé
 - D'états s, s', \dots
 - De paramètres typés avec condition de validité
 - De variables d'état (locales) typées
 - De transition à *branchement multiple* entre les états de la forme "**from** s A " où A est une action contenant des structures de contrôle et des sauts à l'état suivant



Actions NTIF

- Les actions sont construites à partir des éléments syntaxiques suivants
 - Types notés T
 - Variables V
 - Portes G
 - Expressions E
 - Motifs de filtrage P
 - Offres $O ::= !E \mid ?P \text{ where } E$



Syntaxe des actions

- $A ::= \text{null}$

- | $V_0, \dots, V_n := E_0, \dots, E_n$

- | $V_0, \dots, V_n := \text{any } T_0, \dots, T_n$

- | **reset** V_0, \dots, V_n

- | $G O_1 \dots O_n$

- | **to** s

- | $A_1; A_2$

- | **select** $A_1 [] \dots [] A_n$ **end select**

- | **case** E **is**

- $P_1 \rightarrow A_1 \mid \dots \mid P_n \rightarrow A_n$

- end case**

- | **while** E **do** A_0 **end while**

Inaction

Affectation

Affectation non-déterministe

Désactivation de variable

Communication (rendez-vous)

Saut vers un état

Composition séquentielle

Choix non-déterministe

Choix déterministe

Boucle while



Constructions NTIF dérivées

- **for** est dérivé de **while**
- **if-then** et **if-then-else** sont dérivés de **case**

if E then A_1 else A_2 end if =

case E is true $\rightarrow A_1$ | false $\rightarrow A_2$ end case

if E then A_1 end if = if E then A_1 else null end if

- **stop** est dérivé de **select**
stop = select end select



Sémantique statique de NTIF

- Assure la bonne formation des programmes
- Plusieurs vérifications
 - Filtres et affectations lient les variables sans ambiguïtés
 - Toute variable est définie avant d'être utilisée
 - Au plus un événement de communication a lieu sur chaque chemin d'exécution d'une transition (e.g., pas de communications dans les boucles **while**)
 - Pas de blocage entre une communication et le saut vers l'état suivant
 - Certains "**case**" couvrent toutes les valeurs possibles d'un type donné



Sémantique dynamique de NTIF

- Sémantique formelle et intuitive
- Exprimée sous forme SOS (*Structured Operational Semantics*)
 - Associe un système de transitions étiquetées (temporisé) à chaque instance d'un processus
 - $[A], \rho \Rightarrow^l s', \rho'$ signifie que dans l'environnement ρ :
 - A s'exécute sans blocage
 - Effectue l'événement de communication l
 - Saute vers l'état s' avec l'environnement ρ'
 - « **from** s A » et « $[A], \rho \Rightarrow^l s', \rho'$ » induisent une transition $\langle s, \rho \rangle \rightarrow^l \langle s', \rho' \rangle$



NTIF résoud les limitations des modèles condition/action

- Les conditions peuvent être utilisées avant et pendant les communications grâce aux actions **if** et aux motifs de filtrage conditionnels
Exemple **if** $V > 2$ **then** $G ?V$ **where** $V < 5$ **end if**
- Conditions et actions peuvent être entrelacées
- Les sémantiques à grands pas sont préservées grâce aux boucles **while**
- Les conditions ne sont pas dupliquées grâce à la structure à branchement multiple des transitions



Exemple NTIF : if-else-elsif

```
from Cep_Test_NT
```

```
  if Deactivated then
```

```
    Init_Load_Resp.Status := x9106;
```

```
    Cep_Reply !Init_Load_Resp;
```

```
    to Cep_Init
```

```
  elsif Locked then
```

```
    Init_Load_Resp.Status := x9110;
```

```
    Cep_Reply !Init_Load_Resp;
```

```
    to Cep_Init
```

```
  elsif NT >= NT_Limit then
```

```
    Init_Load_Resp.Status := x9102;
```

```
    Cep_Reply !Init_Load_Resp;
```

```
    to Cep_Init
```

```
  else
```

```
    Load_Amount := Inquiry.Load_Amt;
```

```
    Slot_Index := 0;
```

```
    Currency_Sought := Inquiry.Currency;
```

```
    Slots_Available := 0;
```

```
    Last_Avail_Slot := SlotCount;
```

```
    to Cep_IFL_Locate_Slot
```

```
  end if
```



Même exemple en IF (condition/action)

```
from Cep_Test_NT
  if Deactivated
  sync tau
  do {Init_Load_Resp.Status := x9106}
to S_00017 ;

from S_00017
  if Reply_Type_Value_0 = Init_Load_Resp
  sync Cep_Reply!(Reply_Type_Value_0)
to Cep_Init ;

from Cep_Test_NT
  if not Deactivated and Locked
  sync tau
  do {Init_Load_Resp.Status := x9110}
to S_00018 ;

from S_00018
  if Reply_Type_Value_0 = Init_Load_Resp
  sync Cep_Reply!(Reply_Type_Value_0)
to Cep_Init ;
```

```
from Cep_Test_NT
  if not Deactivated and
  not Locked and (NT >= NT_Limit)
  sync tau
  do { Init_Load_Resp.Status := x9102 }
to S_00019 ;

from S_00019
  if Reply_Type_Value_0 = Init_Load_Resp
  sync Cep_Reply!(ReplyType_Value_0)
to Cep_Init ;

from Cep_Test_NT
  if not Deactivated and not Locked and
  not (NT >= NT_Limit)
  sync tau
  do {
    Load_Amount := Inquiry.Load_Amt,
    Slot_Index := 0,
    Currency_Sought := Inquiry.Currency,
    Slots_Available := 0 ,
    Last_Avail_Slot := Slot_Count
  }
to Cep_IFL_Locate_Slot ;
```



Exemple NTIF : case

```
from Cep_Command_Case
  Cep_Command ?Inquiry;
  case Inquiry.Command is
    ALLSLOTS00 ->
      Slots_Reported := 0;
      Slot_Index := 0;
      to Cep_Slot_Inquiry_Sequence
| ALLSLOTS01 ->
      to Cep_SIQ_Reply
| any ->
      to Cep_Command_Out_Of_Sequence
  end case
```



Même exemple en IF

```
from Cep_Command_Case
  sync Cep_Command ?Command_Type_Value_0
  do {Inquiry := Command_Type_Value_0}
to S_00023
```

```
from S_00023
  if Inquiry.Command = ALLSLOTS00
  sync tau
  do { Slots_Reported := 0, Slot_Index := 0 }
to Cep_Slot_Inquiry_Sequence ;
```

```
from S_00023
  if (Inquiry.Command <> ALLSLOTS00) and (Inquiry.Command = ALLSLOTS01)
  sync tau
to Cep_SIQ_Reply ;
```

```
from S_00023
  if (Inquiry.Command <> ALLSLOTS00) and (Inquiry.Command <> ALLSLOTS01)
  sync tau
to Cep_Command_Out_Of_Sequence ;
```



Exemple NTIF : while

```
from Cep_Slot_Inquiry_Sequence
while (Slot_Index < Slot_Count) do
  if (Slots[Slot_Index].In_Use) then
    Slots[Slot_Index].Reported := false;
    Slot_Index := Slot_Index + 1
  else
    Slots[Slot_Index].Reported := true;
    Slot_Index := Slot_Index + 1;
    Slots_Reported := Slots_Reported + 1
  end if
end while;
Cep_Reply !Slot_Info;
to Cep_SIQ_Reply
```



Même exemple en IF

```
from Cep_Slot_Inquiry_Sequence
  sync tau
to S_00009 ;

from S_00009
  if (Slot_Index < Slot_Count) and
     Slots[Slot_Index].In_Use
  sync tau
  do {
    Slots[Slot_Index].Reported := false,
    Slot_Index := Slot_Index + 1
  }
to S_00009 ;
```

```
from S_00009
  if (Slot_Index < Slot_Count) and
     not Slots[Slot_Index].In_Use
  sync tau
  do {
    Slots[Slot_Index].Reported := true,
    Slot_Index := Slot_Index + 1,
    Slots_Reported := Slots_Reported + 1
  }
to S_00009 ;

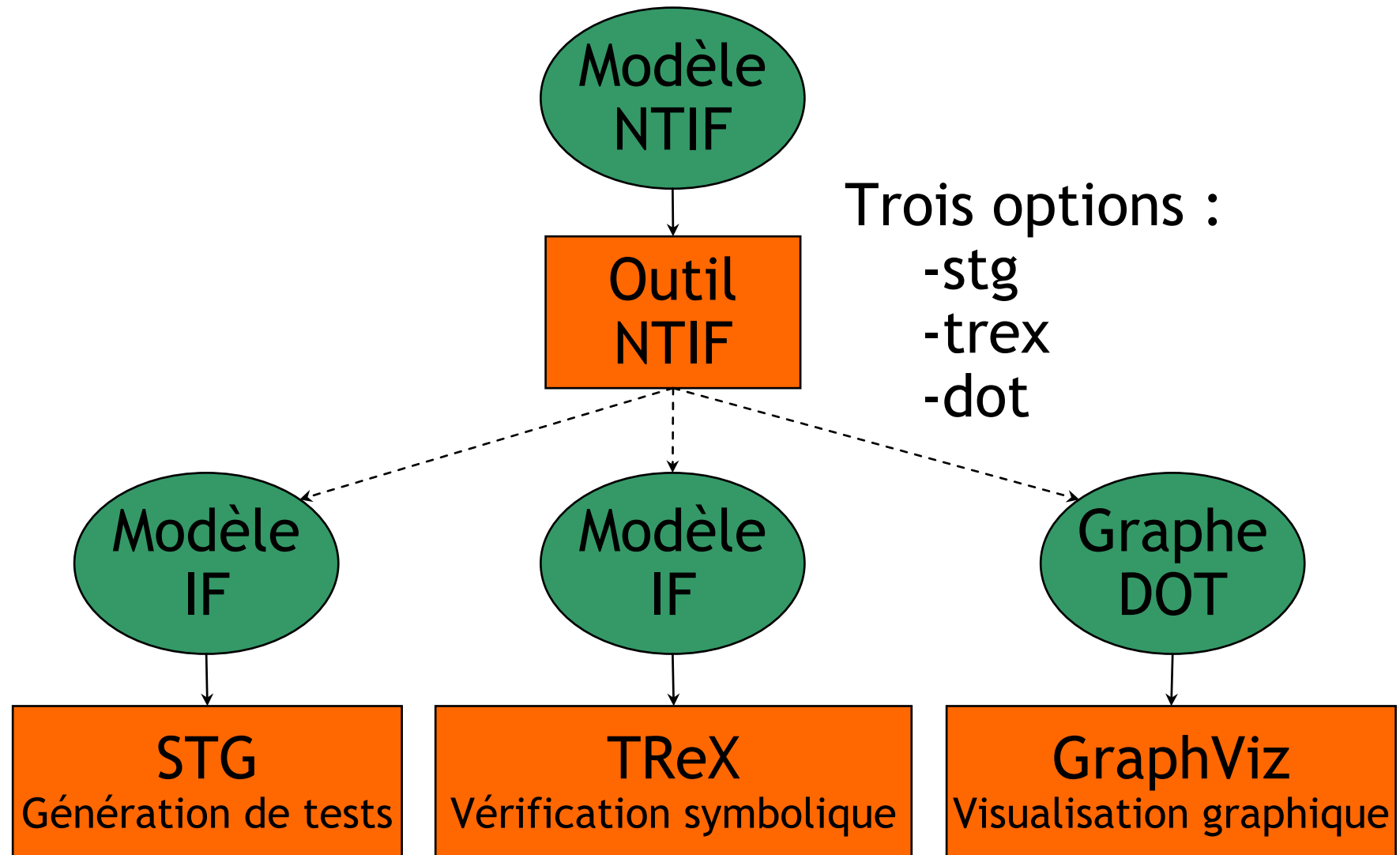
from S_00009
  if not (SlotIndex < SlotCount) and
     (Reply_Type_Value_0 = Slot_Info)
  sync Cep_Reply !Reply_Type_Value_0
to Cep_SIQ_Reply ;
```



Outils pour NTIF



L'outil NTIF



L'outil NTIF

- Dépliage symbolique des transitions **NTIF** vers deux dialectes de IF 1.0
 - IF pour STG (INRIA, Rennes)
utilisé pour la génération de tests symboliques
 - IF pour TReX (LIAFA, Paris)
utilisé pour l'analyse symbolique d'atteignabilité
- Visualisation graphique de descriptions NTIF à l'aide du format DOT fourni dans le paquetage GraphViz (AT&T)



Développement de l'outil NTIF

- Développement initié en avril 2001
- Utilisation de la technologie de construction de compilateur SYNTAX + TRAIAN
<http://www.inrialpes.fr/vasy/Publications/Garavel-Lang-Mateescu-02.html>
- 12 000 lignes de code
 - 2 200 lignes de code SYNTAX
 - 8 300 lignes de code LOTOS NT
 - 1 500 lignes de code C
- Versions pour Solaris, Linux et Windows



Etudes de cas



Porte-monnaie électronique

- Spécification d'un porte-monnaie électronique multi-devises (norme CEPS) à partir d'une description existante en IF 1.0 (Février 2001)
- Nombreuses bogues détectées dans le code IF liées :
 - A la duplication des conditions : conditions non-exclusives, cas non-couverts
 - A l'utilisation de variables non-définies
- Traduction en IF avec NTIF et génération de tests symboliques avec STG
- En cours : vérification symbolique avec TReX



Systeme d'exploitation pour carte à puce

- Commandes administratives d'un OS carte à puce dédié à la téléphonie mobile 3GPP (F.-X. Ponscarne, INRIA, Rennes, Juillet 2001)
- Etude de cas réalisée en contexte industriel (fournie par Schlumberger)
- Traduction en IF avec NTIF et génération de tests symboliques avec STG



Statistiques NTIF vs IF

- NTIF conduit à des descriptions plus concises, contenant moins d'états et de transitions que IF

	CEPS			OS 3GPP		
	IF	NTIF	% ↓	IF	NTIF	% ↓
# lignes	598	418	30 %	697	498	28 %
# transitions	63	23	63 %	78	22	71 %
# états	31	21	32 %	34	20	41 %
Branchement	1	2.21		1	2.77	



Visualisation graphique

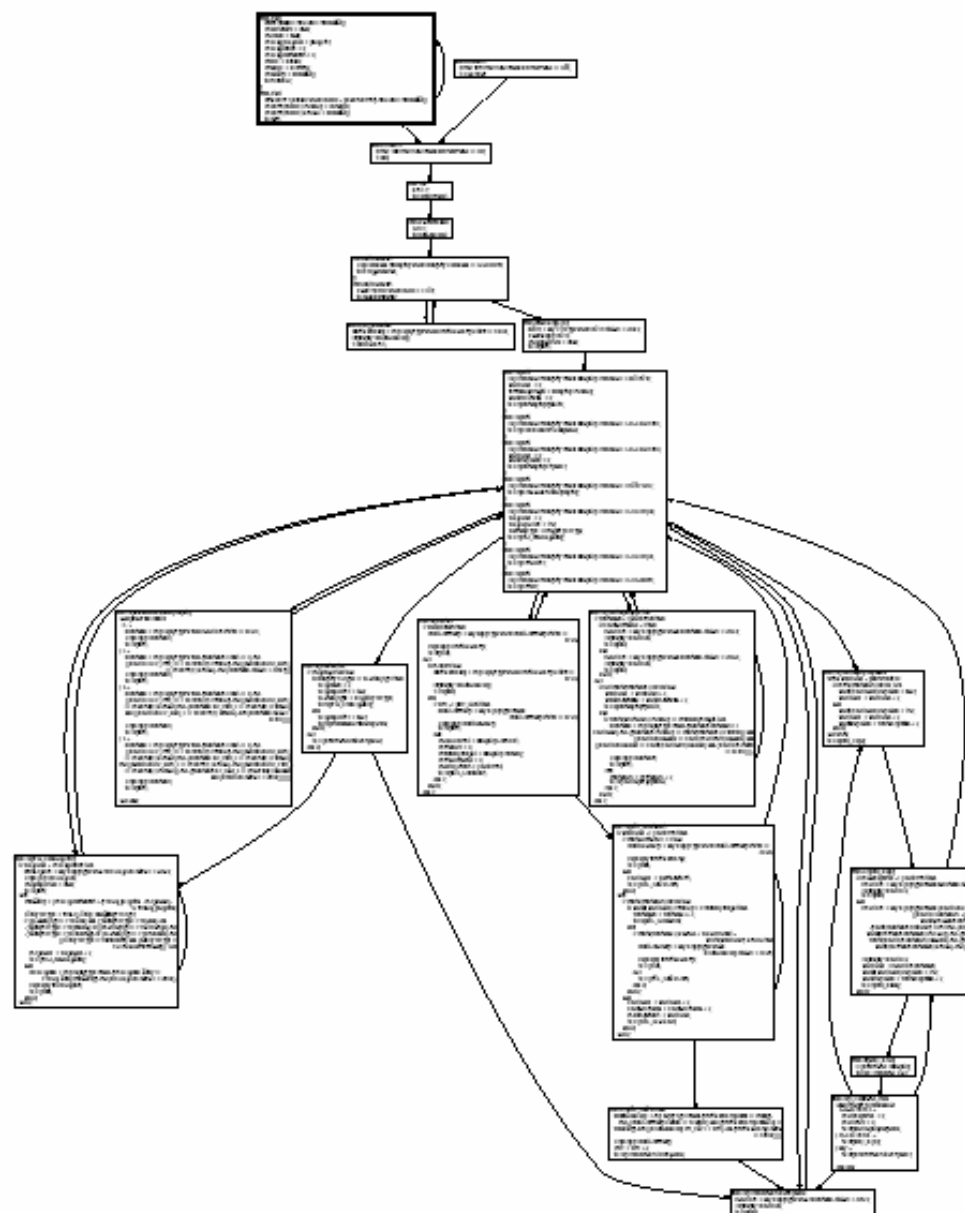
- NTIF conduit à des descriptions plus structurées comme cela peut être constaté à l'aide de DOT

Visualisation graphique du CEPS codé en IF
(Produite par STG)



Visualisation graphique

Visualisation graphique du CEPS codé en NTIF (outil NTIF)



Zoom sur un état

Etat du
CEPS codé
en NTIF

```
from CepIFL_LocateSlot
if vSlotIndex >= pSlotCount then
  if vSlotsAvailable == 0 then
    mInitLoadResp := any ReplyType where mInitLoadResp.Status ==
                                                                x9401;

    CepReply !mInitLoadResp;
    to CepInit;
  else
    vSlotIndex := vLastAvailSlot;
    to CepIFL_InitForLoad;
  end if;
else
  if vSlots[vSlotIndex].InUse then
    if vSlots[vSlotIndex].Currency != vCurrencySought then
      vSlotIndex := vSlotIndex + 1;
      to CepIFL_LocateSlot;
    else
      if vSlots[vSlotIndex].Balance + vLoadAmount >
                                                                vSlots[vSlotIndex].BalMax then
        mInitLoadResp := any ReplyType where
                                                                mInitLoadResp.Status == x9402;

        CepReply !mInitLoadResp;
        to CepInit;
      else
        to CepIFL_InitForLoad;
      end if;
    end if;
  else
    vSlotIndex := vSlotIndex + 1;
    vSlotsAvailable := vSlotsAvailable + 1;
    vLastAvailSlot := vSlotIndex;
    to CepIFL_LocateSlot;
  end if;
end if;
```



Conclusion



Conclusion

- Les modèles condition/action sont mal adaptés pour la description et l'analyse (symbolique ou énumérative) des systèmes
- Créé en avril 2001, **NTIF** est un modèle intermédiaire structuré qui résoud les problèmes
- Des outils existent et ont été appliqués à plusieurs études de cas



Travaux en cours

- Mise en œuvre du langage de données complet (records, tableaux, listes, arbres, types à constructeurs)
- Extension de NTIF avec des constructions temporisées (délais, urgence, etc.)
- Vérification des contraintes de temps
 - Extension de la connexion à TReX
 - Connexion à UppAal



Travaux futurs

- Compilation de LOTOS et E-LOTOS via NTIF, qui nécessite des extensions pour prendre en charge
 - Le parallélisme
 - Les exceptions
- Connexion à CADP pour la vérification énumérative, la simulation et le test



Pour en savoir plus...

Article publié à la conférence FORTE 2002

<http://www.inrialpes.fr/vasy/Publications/Garavel-Lang-02.html>

