Un modèle intermédiaire pour la vérification des systèmes asynchrones embarqués temps réel : ATLANTIF

Soutenance de thèse
Jan Stöcker

Montbonnot, 10 décembre 2009
INRIA Grenoble — Rhône-Alpes
équipe-projet VASY

Systèmes embarqués

- Systèmes spécialisés qui contrôlent des équipements
- Souvent *critiques* : une erreur pourrait avoir de graves conséquences, humaines (ex. : avionique) et/ou économiques (ex. : microélectronique)
- Nécessité de valider au plus tôt dès la phase de conception

Utilisation des méthodes formelles

- Vérification du comportement à partir d'un modèle formel
- Modéliser des systèmes réalistes requiert un langage permettant d'exprimer :
  - Données complexes : listes, structures, unions, etc.
    Ex. : protocole de communication qui échange des données
  - Contrôle et concurrence : événements, synchronisation, communication, etc.
    Ex. : messages entre systèmes
  - Temps réel : délais, urgence, etc.
    Ex. : pilote automatique d'avion

Formalismes existants

- Algèbres de processus temporisées
  - Extensions temporisées de CCS, ACP, CSP, LOTOS (T-LOTOS, RT-LOTOS, ET-LOTOS, …)
  - Nouveaux langages : E-LOTOS, LOTOS NT
  - Expressifs et concis (donc *langages de description*) mais peu d'outils
- Modèles de graphe
  - Automates temporisés, réseaux de Petri temporisés, …
  - Existence d'outils : Uppaal, Red, Tina, Roméo, …
  - Mais : difficulté à modéliser des systèmes réalistes

Exemple de langage de description : ET-LOTOS

Temps d'exécution de la tâche A

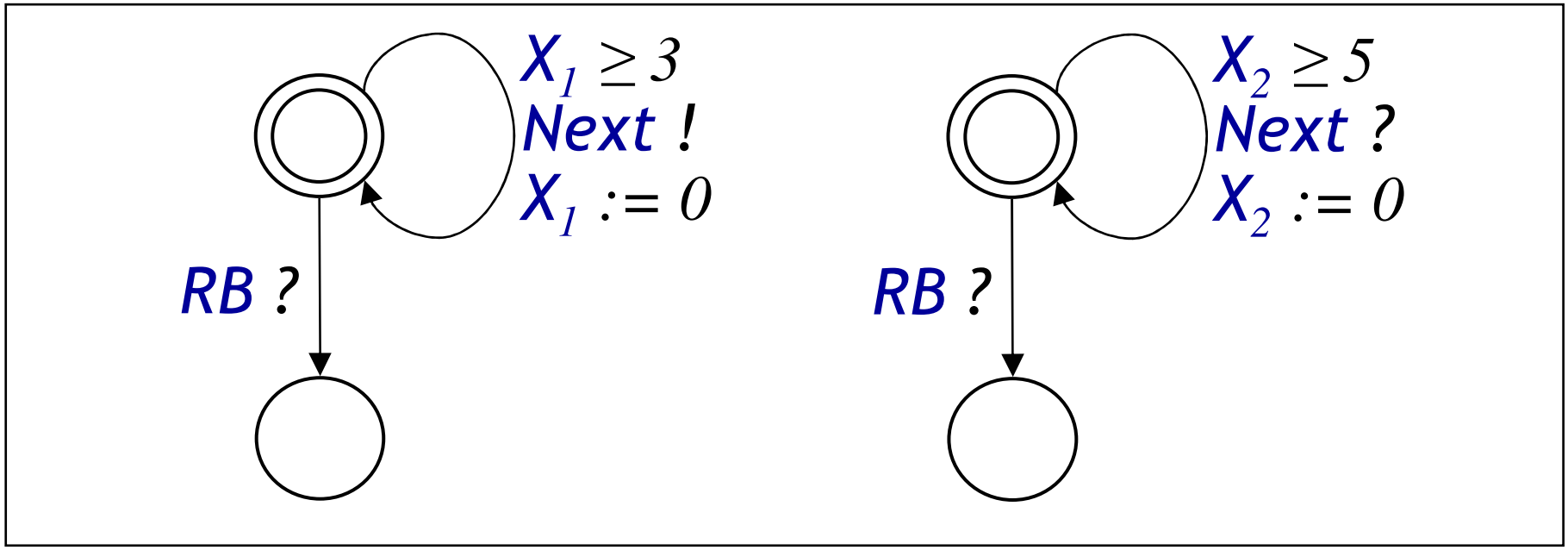
« red button », Arrêt d'urgence

```
specification Two_Robots is
        gates Next, RB
        behaviour (loop wait 3; Next endloop
                                  |[Next]|
                   loop wait 5; Next endloop) [> RB; null
endspec
```

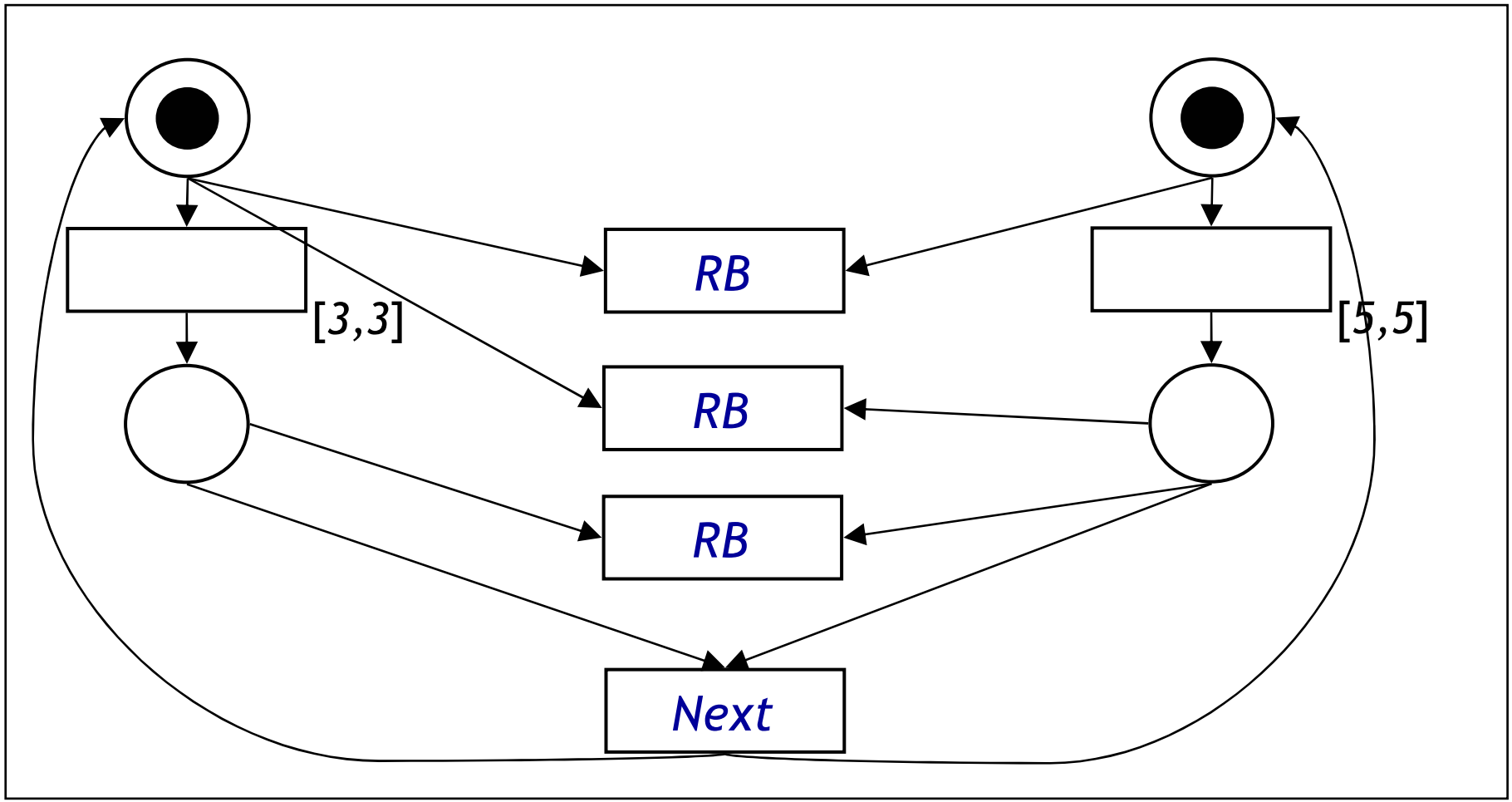
Temps d'exécution de la tâche B

Rendez-vous sur la fin des tâches

Exemple de modèle de graphe : automates temporisés



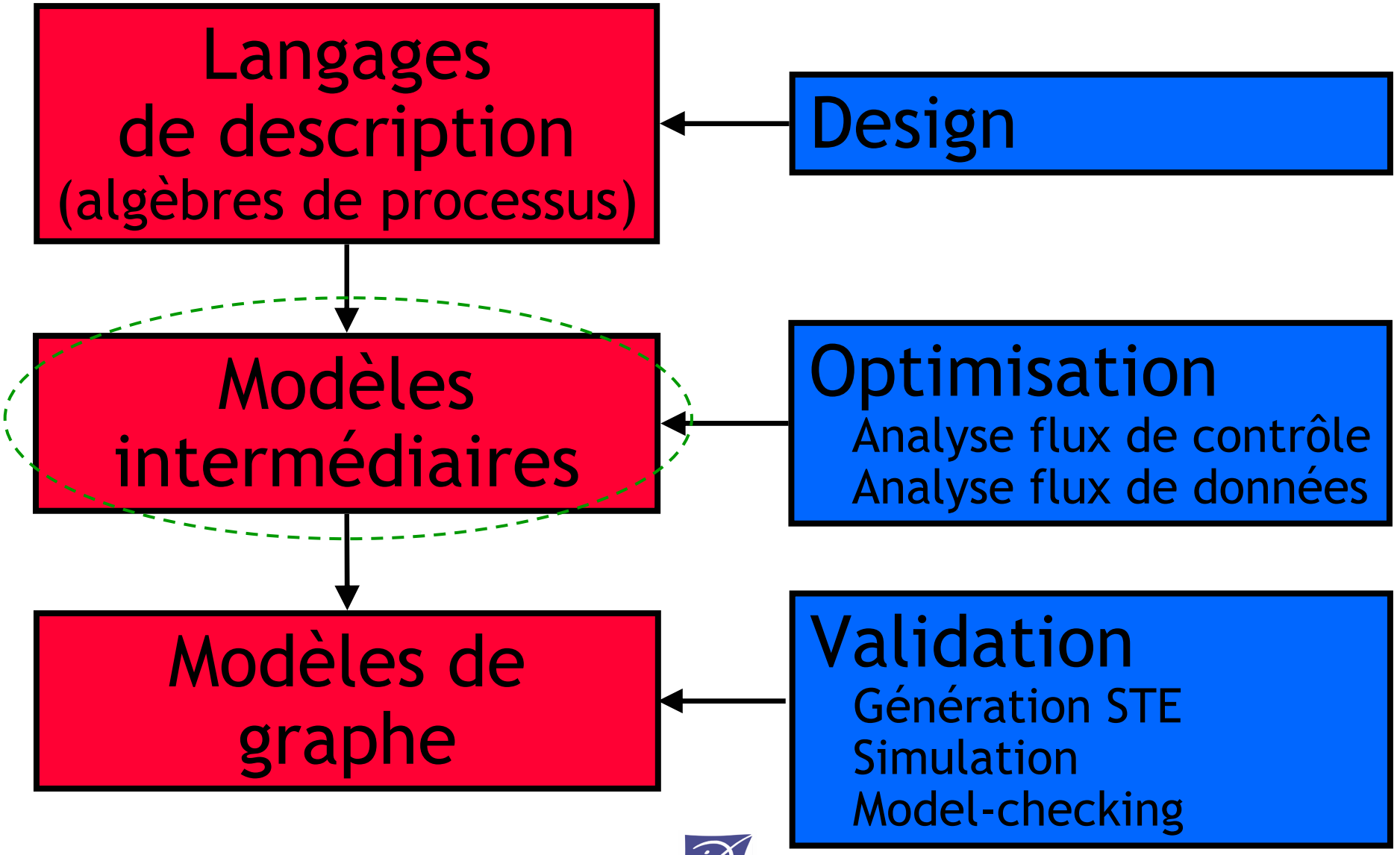
Exemple de modèle de graphe : réseau de Petri temporisé



Objectif de la thèse

- Réduire le fossé entre :
  - **langages de description** permettant de modéliser des systèmes réalistes
  - **modèles de graphe**, pour lesquels des algorithmes de vérification efficaces existent
- Nécessité d'un ***modèle intermédiaire*** qui permet une traduction des langages de description :
  - Automatisée
  - Efficace
  - Sémantiquement correcte
- Définition du modèle **ATLANTIF**

Modèles intermédiaires



Modèles intermédiaires existants

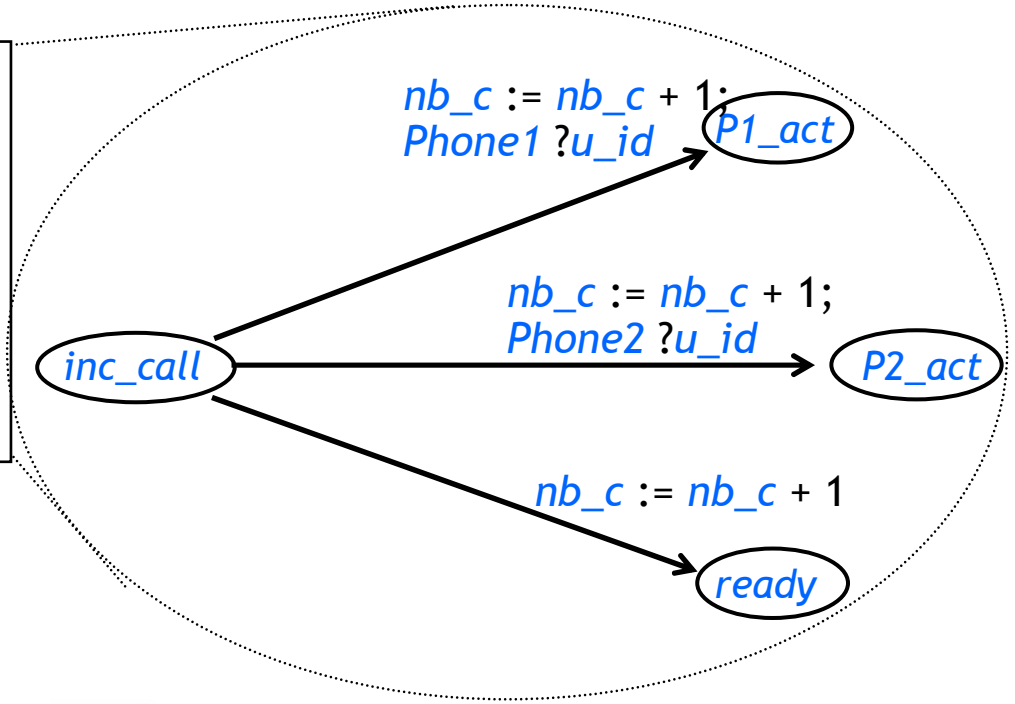
- MoDeST [D'Argenio_Hermanns_Katoen_Klaren_2001]
  - Modèle probabiliste
  - Synchronisation de processus concurrentes limité aux opérateurs basiques
- BIP [Gößler_Sifakis_2005, Basu_Bozga_Sifakis_2006]
  - Modèle concurrent
  - Processus représentés sous forme d'automates
- Fiacre [Berthomieu_Bodeveix_Farail_et_al_2008]
  - Langage pivot dans la traduction vers des outils de vérification (CADP, Tina)
  - Sans constructions de temps-réel à l'intérieur des processus

Le modèle NTIF [Garavel-Lang-2002]

- Possibilité de définir
  - des types de données complexes
  - des fonctions
  - des processus séquentiels
- Adapté à la traduction de processus séquentiels des langages de description
- Mais sans concurrence, ni temps réel

Exemple NTIF

```
process phone_system [Phone1, Phone2]

init system_boot is
var number_calls : int, user_id : int

from system_boot
    (…);
    to ready
from incoming_call
    number_calls := number_calls + 1;
    select
        Phone1 ?user_id; to Phone1_active
    []
        Phone2 ?user_id; to Phone2_active
    []
         null; to ready
    end select
from Phone1_active
    (…);
    to ready
(…)
end process
```



Exemple NTIF

- Plusieurs actions atomiques dans un seul terme
- Une action exprime un ou plusieurs *chemins d'exécution* possibles
- Ici : trois chemins différents
- On parle d'une *transition à branchement multiple*

Approche

- Extension du modèle NTIF

- Permettre la vérification formelle

Extensions de NTIF

- Extension des actions avec des constructions temporisées
- Définition de processus hiérarchiques (désormais appelés « *unités* »)
- Introduction de « *synchroniseurs* » qui définissent :
  - les synchronisations entre les unités
  - le démarrage et l'arrêt des unités

  → le modèle *ATLANTIF*

Constructions temporisées

- Domaine de temps
  - Formalismes existants ont un domaine de temps dense (ex. : ACP_ρ) ou discret (ex. : Temporal CCS)
  - ATLANTIF permet de choisir : mots-clés **no time**, **discrete time**, ou **dense time**
- Délai
  - Ecoulement d'un temps déterminé dans une unité
  - Exemple : temps d'attente pour une réponse
  - Nouvelle action « **wait** *E* » comme dans plusieurs langages existants (Ex. : TCSP)

Constructions temporisées

- Restrictions temporelles sur les actions de communication
  - Limiter les instants où une communication est possible (Exemple : timeout)
  - Introduction d'une *fenêtre de temps*, définie comme une composition d'intervalles :

$$W ::= [E_1, E_2] \mid]E_1, E_2] \mid [E_1, E_2[\mid]E_1, E_2[$$
$$\mid [E_1, \ldots[\mid]E_1, \ldots[\mid W_1 \textbf{ or } W_2 \mid W_1 \textbf{ and } W_2$$

  - Temps relatif à l'instant où l'action est atteinte

Exemple

- L’envoi d’un message ne peut se faire qu’après un délai minimal :

Send_Message !*Data_Package* **in**]0, …[

Sémantique des échéances

- Échéance = *dernier instant* où la communication est possible
  Exemple : $W =]3,7]$ → l'échéance est 7
- Plusieurs comportements possibles à l'échéance [Boyer-Roux-2007] :
  - Sémantique faible : le temps continue à s'écouler
  - Sémantique forte : impossibilité de dépasser l'échéance, le temps s'arrête
  - ATLANTIF laisse le choix de la sémantique : mots-clés **may** et **must**

Exemples

- Signal émis irrégulièrement dans des intervalles de 3 à 5 secondes

  from *Sending*
  Send **must in** [3,5]; **to** *Sending*

- Procédure de log-in attendant un mot de passe pendant 10 secondes

  from *Authentification*
  select *Password* **may in** [0,10]; **to** *Logging_in*
  [] **wait** 10; to *Error_state* **end select**

Résumé extension d'actions

A ::= **null**

| **wait** E ← action de délai

| $V_0,\ldots,V_n := E_0,\ldots,E_n$

| $V_0,\ldots,V_n :=$ **any** $T_0,\ldots,T_n$ [**where** E]

| **reset** $V_0,\ldots,V_n$

manipulation de variables

| $G\ O_1 \ldots O_n$ [[**must** | **may**] **in** W]

communication par porte avec restriction de temps

| **to** s ← saut

| $A_1; A_2$

| **select** $A_1[]\ldots[]A_n$ **end**

| **case** E **is** $P_1 \rightarrow A_1 | \ldots | P_n \rightarrow A_n$ **end**

| **while** E **do** A **end**

composition

deux extensions par rapport à la syntaxe d'actions NTIF

Hiérarchisation des unités

- Dans les langages de description, l'architecture des processus n'est pas toujours statique
- Exemple : combinaison de compositions parallèles et séquentielles

```
var x : nat in
    A; (B || C); D
end var
```

- ATLANTIF propose un découpage des processus en unités et *sous-unités*

Hiérarchisation des unités

```
var x : nat in
    A; (B || C); D
end var
```

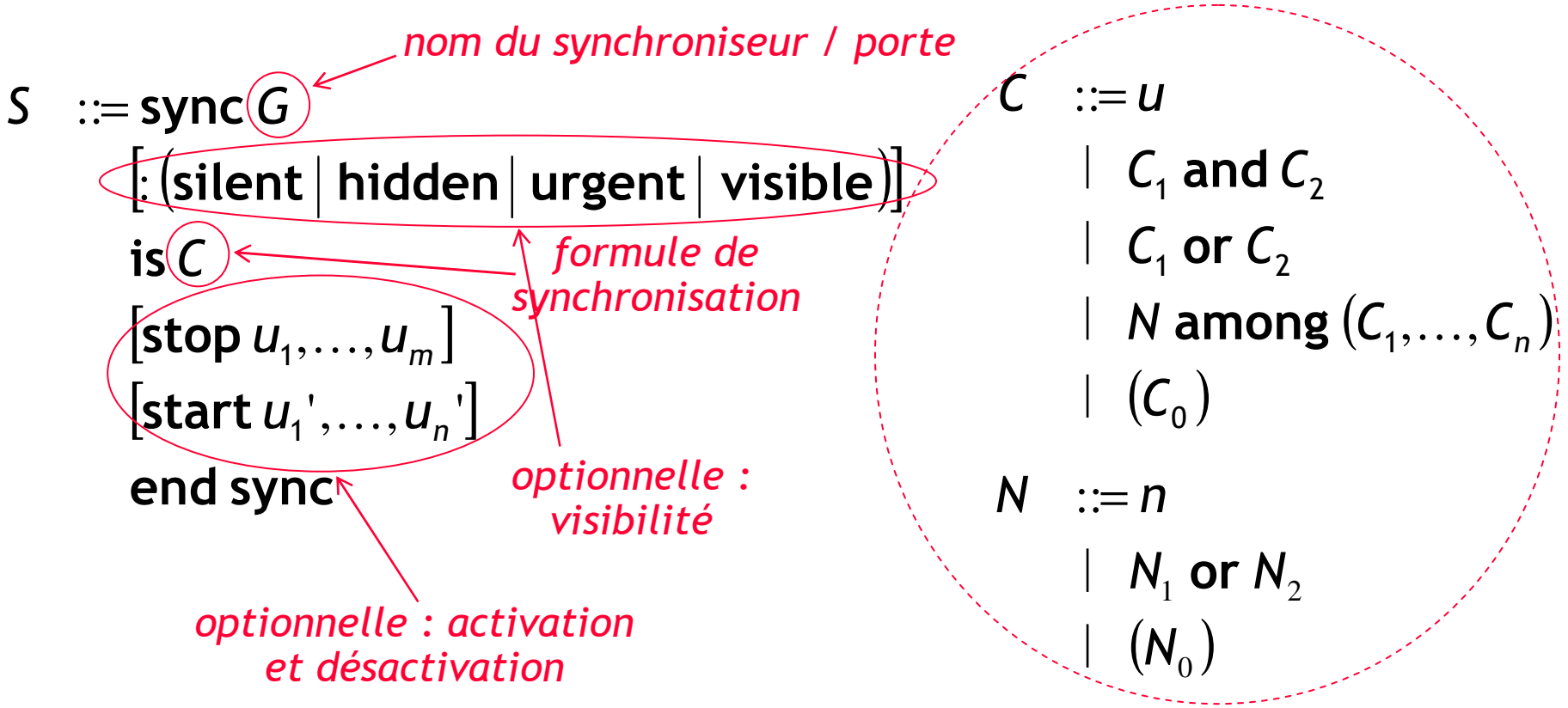
```
unit ABCD is
variables x : nat
(...)
    unit A is
    (...)
    end unit
    unit BC is
    (...)
        unit B is
        (...)
        end unit
        unit C is
        (...)
        end unit
    end unit
    unit D is
    (...)
    end unit
end unit
```

→ *x* peut être accédée dans toutes les sous-unités de *ABCD*

Synchroniseurs

- Les unités s'exécutent en parallélisme asynchrone
- Besoin d'une construction pour synchroniser, arrêter et démarrer les unités
- Introduction des *synchroniseurs*
- Associé à une porte pour laquelle la *visibilité* est définie
  - **visible** : porte et ses offres visibles depuis l'environnement
  - **hidden** : porte non pas visible mais induit un changement d'état interne du système
  - **urgent** : comme hidden, mais l'écoulement du temps est impossible
  - **silent** : pas de changement d'état interne du système

Syntaxe des synchroniseurs



Exemples synchroniseurs

- **Compétition**
  synchronisation de u_1 avec soit u_2 soit u_3

  sync *G* is u_1 and (u_2 or u_3) end sync

- **Synchronisation « multiway »**
  synchronisation simultanée de n unités

  Exemple pour *n* = *3*

  sync *G* is u_1 and u_2 and u_3 end sync

Exemples synchroniseurs

- **Composition parallèle généralisée**
  synchronisation entre plusieurs sous-ensembles d'unités [Garavel-Sighireanu-1999]

  Exemple : deux ou trois unités parmi u_1,u_2,u_3

  sync *G* **is** (*2* **or** *3*) **among** (u_1,u_2,u_3) **end sync**

- **Arrêt** et **Démarrage** dynamique des unités
  Exemple : arrêt de u_1, u_2 et démarrage de u_3, u_4 lors d'une synchronisation sur *G*

  sync *G* **is** u_1 **and** u_2 **stop** u_1,u_2 **start** u_3,u_4 **end sync**

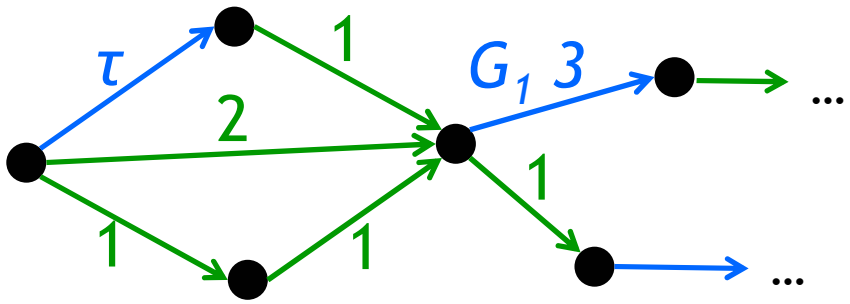
Accès aux variables dans les unités hiérarchiques

- Plusieurs unités concurrentes peuvent accéder à une même variable
  → Ambiguïté de sa valeur si deux unités peuvent l'écrire en même temps
- Exclusion de ce problème par une vérification statique
  - Un algorithme assure que dans tout état, chaque variable ne peut être écrite que par une unité

Sémantique dynamique

Définition d'un **système à transitions temporisées** comportant deux types de transitions :

- Les transitions discrètes, étiquetées par une porte et ses offres ou le symbole τ représentant un changement d'état interne
- Les transitions temporisées, étiquetées par les éléments d'un domaine de temps (dense ou discret)



Sémantique dynamique

- Plusieurs « bonnes » propriétés souhaitées :
  - *additivité de temps* : deux délais consécutifs équivalent leur somme
  - *déterminisme de temps* : l'écoulement du temps ne résoud pas des choix
  - *progrès maximal d'actions urgentes* : les actions urgentes sont prioritaires sur l'écoulement du temps
- Certains formalismes ne les satisfont pas
  Exemple : RT-LOTOS

Sémantique dynamique

- ATLANTIF définit une sémantique sur deux niveaux :
  - règles SOS définissent les transitions à l'intérieur des unités
  - une règle regroupe ces transitions pour définir des transitions discrètes
  - une autre règle définit les transitions temporisées
- Cette définition structurée permet de facilement satisfaire les trois bonnes propriétés (propositions 4.1, 4.2, 4.3)

Exemple

```
module Two_Robots is
dense time

sync Next is
        Rob1 and Rob2
end sync

sync RB is
        Rob1 and Rob2
        stop Rob1, Rob2
end sync

init Rob1, Rob2
```

```
unit Rob1 is
        from Task_A
                select
                        wait 3; Next
                        []
                        RB
                end;
                to Task_A
end unit

unit Rob2 is
        from Task_B
                select
                        wait 5; Next
                        []
                        RB
                end;
                to Task_B
end unit

end module
```

Approche

- Extension du modèle NTIF

- Permettre la vérification formelle

Traductions d'ATLANTIF

- Objectif : vérifier formellement les propriétés d'un programme ATLANTIF
- Utilisation des outils existants :
  - Uppaal permet de vérifier des *automates temporisés*
  - Tina permet de vérifier des *réseaux de Petri temporisés*
- Définition de traductions de sous-ensembles d'ATLANTIF vers Uppaal et Tina

Traduction vers Uppaal

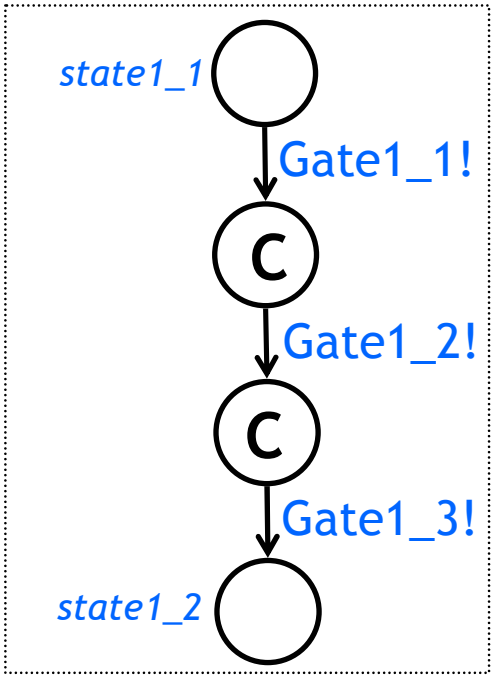
construction ATLANTIF	construction Uppaal
module	réseau d'automates temp.
synchroniseur	un ou plusieurs canaux
toute synchronisation possible avec 1 ou 2 unités	un canal binaire
toute synchronisation possible avec $n > 2$ unités	(n - 1) canaux pour émuler la synchronisation « multiway »
unité	un automate temporisé
état discret	location
transition à branchement multiple	une transition par chemin
communication par porte	étiquette d'action sur transition
restrictions temporelles	gardes et invariants
offres de communication	émulées par variables globales

Traduction des synchronisations

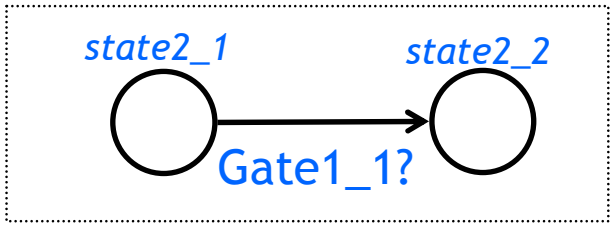
```
sync Gate1 is
   Unit1 and Unit2 and Unit3 and Unit4
end sync

...

unit Unitn is (* n = 1,2,3,4 *)
   ...
   from staten_1
      Gate1;
      to staten_2
   ...
end unit

...
```

- Uppaal ne permet que la synchronisation *binaire*
- Le traducteur émule la synchronisation *n-aire*
  - Décomposition en n-1 synchronisations
  - Utilisation des états « commited »

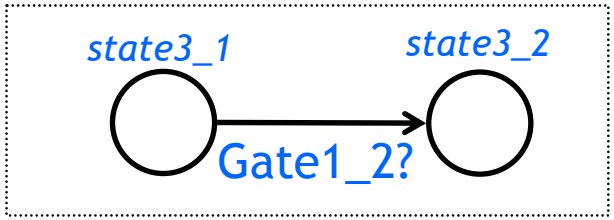
Traduction d'Unit1 :



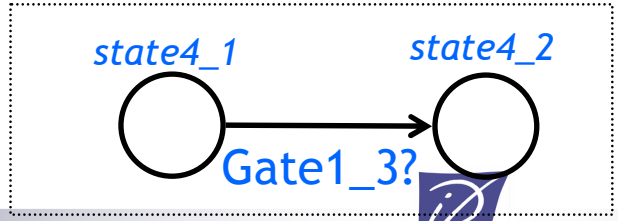
Traduction d'Unit2 :



Traduction d'Unit3 :



Traduction d'Unit4 :



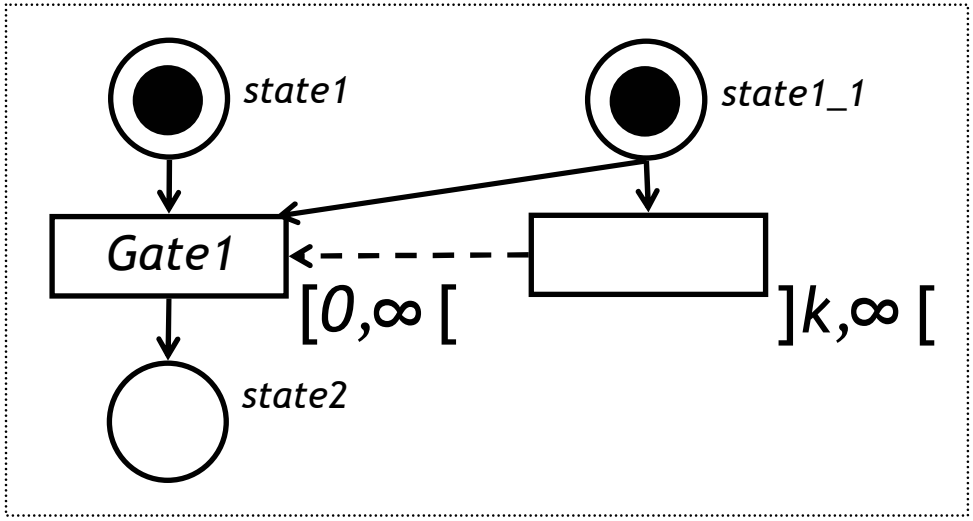
Traduction vers Tina

construction ATLANTIF	construction Tina
module	un réseau de Petri temporisé (TPN)
synchroniseur	pas de traduction directe
unité	sous-ensemble du TPN
état discret	place
transition à branchement multiple	d'abord une transition par chemin, puis multiplications et fusions par synchronisations
communication par porte	étiquette d'action sur transition
restrictions temporelles	transitions auxiliaires, priorités et arcs inhibiteurs
manipulation de données, offres de communication	traduction vers des fonctions C

Traduction des restrictions temporelles

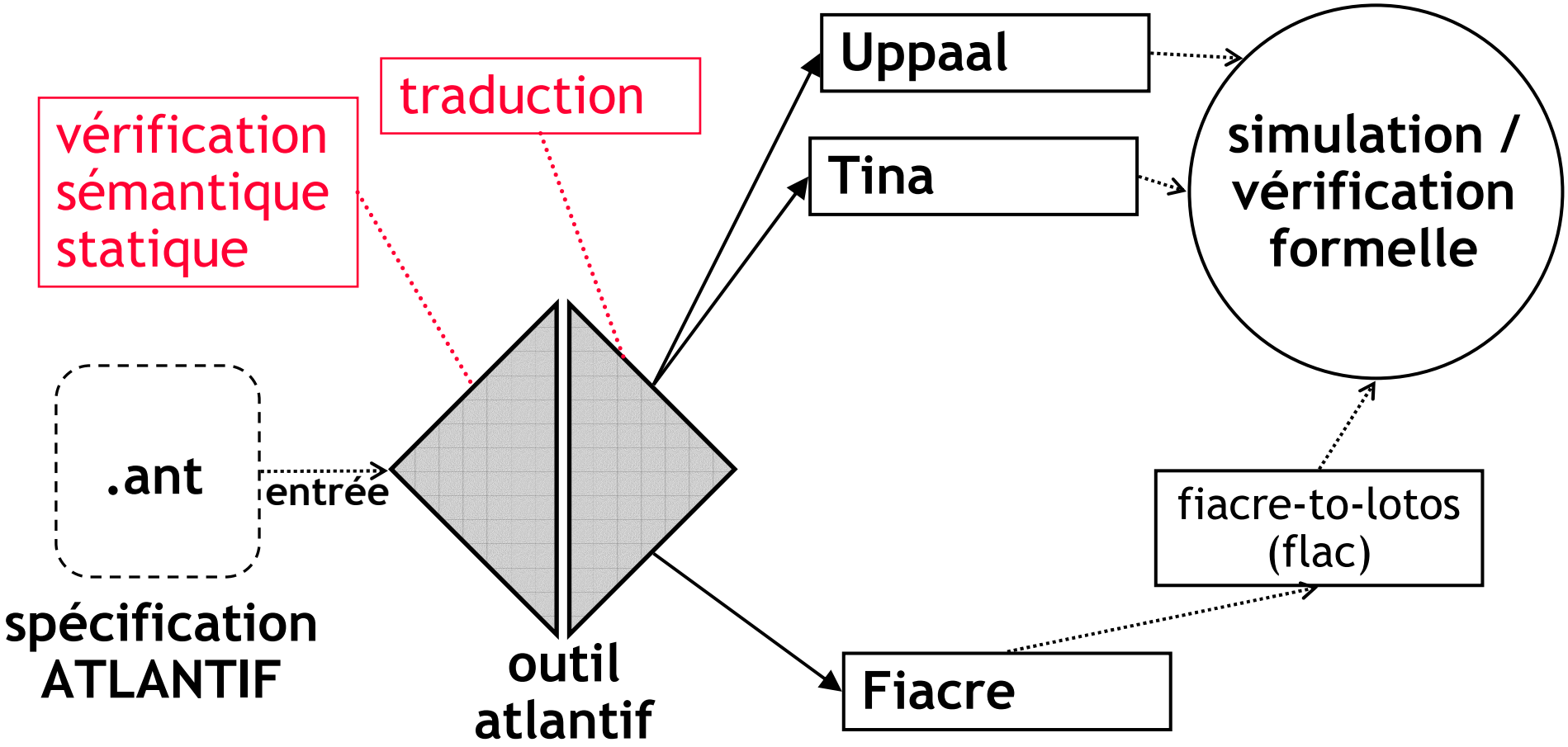
from *state1*
Gate1 **may in** [*0*,*k*];
to *state2*
(où k≥ 0)

Traduction :



- Restrictions temporelles dans Tina toujours avec sémantique forte
- Le traducteur émule les limites faibles
  - Définition des états et transitions auxiliaires
  - Utilisation des priorités

L'outil atlantif



- ~16 000 lignes de code (LOTOS NT, C, Syntax)

Contributions

- Définition d'un nouveau modèle intermédiaire
  - avec une combinaison unique de constructions, dont des constructions originales
    - formules de synchronisation
    - fenêtre de temps avec choix de la sémantique des échéances
  - structuré et proche des nouveaux langages comme LOTOS NT et E-LOTOS
- Des traductions permettent la vérification formelle

Perspectives

- Utiliser ATLANTIF sur une étude de cas de grande taille
- Etendre les traductions pour couvrir un maximum d'ATLANTIF
  - Compléter les traductions des constructions couvertes
  - Proposer des abstractions/approximations pour les constructions non-couvertes
- Traduire vers ATLANTIF un langage de description complet (exemple : LOTOS NT)
- Analyser statiquement du code ATLANTIF
  - Optimiser les traductions vers les modèles de graphe
  - Vérifier des propriétés sans passer par des modèles de graphe

Merci de votre attention