

Tutorial RT-LOTOS

tutorial.pdf

Christophe Lohr

1 Présentation des outils

1.1 Prérequis

Ce tutorial a pour objectif de vous familiariser avec les utilisations classiques de RT-LOTOS (manipulation des outils, compréhension d'une spécification donnée, écriture d'une spécification, etc.).

Avant de commencer assurez vous que vous disposez des outils suivants :

1. `rtl rtlfig dta2dot` <http://www.laas.fr/RT-LOTOS>
2. `xfig` <http://www.xfig.org>
3. `dotty` <http://www.research.att.com/sw/tools/graphviz>
4. un éditeur de texte quelconque

1.2 rtl

L'outil `rtl` permet de :

- simuler une spécification,
- générer l'automate temporisé équivalent (DTA),
- générer le graphe minimal de régions (RG).

Le fichier `stopw.lot` (annexe A) donne une spécification RT-LOTOS du protocole Stop&Wait. Recopiez-le dans votre répertoire de travail, et au passage, jetez-y un coup d'œil.

1.3 Simulation

Exécutez la commande `rtl -max-spec-t-500 stopw.lot`. On réalise ainsi une simulation de cette spécification sur une durée de 500 unités de temps. La trace de la simulation apparaît alors dans le fichier `stopw.lot.sim`. Regardez-la.

L'outil `rtlfig` permet de donner une représentation graphique de cette trace de simulation sous la forme de *lignes temporelles*.

Les options de `rtlfig` sont :

<code>-tl-action-ACT</code>	réalise la ligne temporelle de l'action ACT
<code>-xfig-max-N</code>	trace jusqu'à l'instant N
<code>-xfig-min-N</code>	trace a partir de l'instant N
<code>-xfig-size-N</code>	ligne temporelle de taille N inch
<code>-xfig-scale-N</code>	mise à l'échelle d'un facteur N

Exécutez la commande `figall stopw.lot`. Ce petit script exécute la commande `rtlfig` sur toutes les actions de la simulation. On obtient ainsi un fichier `stopw.lot.fig` (annexe B) pouvant être visualisé avec `xfig` (essayez `xfig stopw.lot.fig`), ainsi qu'un fichier PostScript `stopw.lot.ps` pouvant être visualisé avec `ghostview`, ou imprimé. Libre à vous de jouer avec les options de ces divers outils pour avoir la meilleure vue de votre simulation.

1.4 Dynamic Timed Automata

Exécutez la commande `rtl -TG1 -p1 stopw.lot`. Vous obtiendrez alors, sous forme textuelle, l'automate temporisé correspondant à votre spécification.

Vous pouvez en avoir une représentation graphique avec l'outil `dotty` après avoir converti l'automate dans le format approprié avec la commande `dta2dot`. Procédez ainsi :

```
rtl -TG1 -p1 stopw.lot > stopw.dta
dta2dot < stopw.dta > stopw.dta.dot
```

ou bien :

```
rtl -TG1 -p1 stopw.lot | dta2dot > stopw.dta.dot
```

Puis exécutez `dotty stopw.dta.dot` (annexe C).

Vous pouvez obtenir une version PostScript de votre automate par la commande :

```
dot -Tps < stopw.dta.dot > stopw.dta.ps
```

1.5 Minimal Reachability Graph

Exécutez la commande `rtl -TG3 stopw.lot`. Vous obtiendrez alors, sous forme textuelle, le graphe minimal d'accessibilité de votre spécification.

Vous pouvez obtenir le DTA en même temps en ajoutant l'option `-p1`.

Vous avez aussi la possibilité de réaliser l'analyse d'accessibilité *à la volée* : c'est à dire que `rtl`, avec l'option `-TG2`, peut construire les états du DTA un par un, et construire le graphe de régions au fur et à mesure. La spécification `on_the_fly.lot` (annexe D) est un cas pathologique qui ne peut être analysé qu'à la volée (le DTA est infini, alors que le RG est fini).

Vous remarquerez que certains noeuds de ce graphe font référence à un code et un numéro : par exemple dans l'arc ($3-(7\ 6)$, `i(no)`, $5-(7) \Rightarrow N380-L11$) (en fait, le code $N380-L11$ correspond à la position de l'état dans l'arbre syntaxique du code RT-LOTOS). Vous noterez aussi, en consultant les équations bornant chaque région, que la configuration $5-(7)$ appartient en fait à la région $5-(3)$. L'outil `rgstrap` réalise les redirections adéquates automatiquement (dans notre exemple, remplacer $5-(7) \Rightarrow N380-L11$ par $5-(3)$).

Par ailleurs, vous avez toujours la possibilité, comme précédemment, d'obtenir une vue graphique de ce graphe avec les outils `dta2dot` et `dotty` (annexe E) :

```
rtl -TG3 stopw.lot | rgstrap > stopw.rg
dta2dot < stopw.rg > stopw.rg.dot
dotty stopw.rg.dot
```

Par contre, les équations des régions n'apparaissent pas sur cette représentation graphique.

2 Compréhension d'une spécification

Le fichier `stopw.lot` (annexe A) donne une spécification RT-LOTOS du protocole Stop&Wait. Pourriez-vous expliquer de façon informelle le fonctionnement de ce protocole ?

Pour répondre à une telle question, je vous invite à suivre le raisonnement suivant.

2.1 Paradigme de boîtes noires

Un processus RT-LOTOS peut être vue comme une *boîte noire* communiquant avec l'extérieur par des *portes* (ou *actions*). À l'intérieur de chaque processus (ou boîte), on peut trouver soit d'autres sous-processus (et donc d'autres boîtes noires emboîtées à l'intérieur) composés entre eux en séquence ou en parallèle avec éventuellement de la synchronisation, soit des automates élémentaires.

Notre processus Stop&Wait est composé de trois sous-processus synchronisés entre eux. L'un d'eux est lui-même composé de deux processus.

Sur le papier, tentez de représenter la composition de ces processus par des boîtes emboîtées les unes dans les autres, et reliées entre elles par les actions de synchronisation.

Une réponse est proposée en annexe F.

2.2 Automates élémentaires

Les processus qui ne peuvent plus être décomposés en sous-processus sont en fait des automates élémentaires composés de délais, latence, actions, etc... et qui parfois rebouclent.

Sur le papier, proposez une représentation pour chacun des automates élémentaires que vous avez identifiés.

Notez que je ne vous demande pas de respecter un formalisme d'automate donné. Vous pouvez inventer le vôtre à condition qu'il soit suffisamment intuitif et convainquant.

Une réponse est proposée en annexe G.

2.3 Spécification en langage naturel

À la lumière de ce que vous venez de comprendre du protocole Stop&Wait, expliquez son principe de fonctionnement de façon clair et non ambiguë en quelques phrases.

Une réponse est proposée en annexe H.

3 Réalisation de spécifications RT-LOTOS

À l'inverse de ce que l'on vient de faire, la démarche consistera à partir d'une spécification en langage naturel pour arriver à une spécification en RT-LOTOS sur laquelle on pourra réaliser des simulations et une analyse d'accessibilité.

3.1 Protocole de communication

On se propose de développer le protocole de communication suivant : deux machines, communiquent entre elles par un médium sur lequel elles émettent et reçoivent des paquets.

L'application s'exécutant sur les machines émet ou reçoit des paquets périodiquement, avec une période comprise entre 10 et 20ms.

Lorsqu'une machine veut envoyer un paquet, elle le place dans le buffer de la carte réseau. Si la carte ne détecte aucune transmission de donnée sur le réseau pendant 3ms, le paquet est envoyé, si non, il y a "collision" : on devra attendre entre 5 et 6 ms avant de tenter de le re-émettre.

Les cartes réseau sont uni-directionnelles : lorsqu'un paquet est en attente d'émission dans le buffer, les cartes ne peuvent pas lire et mémoriser des paquets en provenance de réseau. Dans ce cas, si un paquet arrive, il est alors perdu.

- Structurez ce protocole en *boîtes noires*. Une réponse est proposée en annexe I.
- Implémentez cette structure en composant des processus RT-LOTOS. Une réponse est proposée en annexe J.
- Écrivez en RT-LOTOS les automates élémentaires. Petites astuces : sur vos automates, lorsque plusieurs transitions partent d'un nœud, vous utiliserez certainement l'opérateur de choix RT-LOTOS, et lorsque plusieurs transitions arrivent sur un nœud, vous utiliserez certainement un sous-processus RT-LOTOS pour donner un nom à ce nœud.
- Utilisez les fonctionnalités de simulation pour déboguer votre spécification.
- Réalisez l'analyse d'accessibilité et cherchez les éventuelles situations de dysfonctionnement, blocage, etc... de ce protocole.

3.2 Document multimédia

On souhaite ordonnancer entre eux les éléments d'un document multimédia.

Le document débute par un texte accompagné d'un fond musical de 10s.

Puis apparaît une séquence audio-vidéo. La vidéo a une durée de 30s. Le fond musical est une séquence qui tourne en boucle et qui peut donc être arrêtée n'importe quand. Par contre, juste avant la fin de la vidéo, on devra présenter en même temps l'enregistrement audio d'une voix.

Par contre, certaines machines ne permettent pas de mixer deux sources audio. Le document prévoit dans ce cas un texte en remplacement de la voix.

La dernière séquence est composée d'une vidéo, d'un fond musical, et d'une animation graphique. Ces éléments peuvent être accélérés ou ralentis pour les besoins de la présentation. La vidéo peut être jouée entre 9s et 18s, l'audio entre 9s et 15s et l'animation entre 9s et 24s. La séquence débute avec le début de la vidéo et se termine avec la fin de l'animation. La vidéo et le son se terminent en même temps. Le son et l'animation débutent ensemble.

En suivant la démarche précédente, spécifiez ce document en RT-LOTOS, et tentez d'en dégager certaines propriétés par l'analyse d'accessibilité.

A stopw.lot

```
Contenu du fichier stopw.lot.
(* Protocole Stop and Wait *)

specification StopWait : noexit

type boolean is
  sorts bool
  opns
    not : bool->bool
    and : bool, bool->bool
    or  : bool, bool->bool
endtype

type natural is boolean
  sorts nat
  opns
    + : nat, nat->nat
    - : nat, nat->nat
    * : nat, nat->nat
    < : nat, nat->bool
    > : nat, nat->bool
    <= : nat, nat->bool
    >= : nat, nat->bool
endtype

behaviour

  hide send, rec_ack, timeout, receive, snd_ack, yes, no in

  (
    Sender[send, rec_ack, timeout](false)
    |[send, rec_ack]|
    Medium[send, rec_ack, receive, snd_ack]
  )
  |[receive, snd_ack]|
  Receiver[receive, snd_ack, yes, no]

where

  process Medium[send, rec_ack, receive, snd_ack] : noexit :=

    Slot[send, receive](1,6)
    |||
    Slot[snd_ack, rec_ack](1,6)

  where

  process Slot[in_slot, out_slot](dmin, dmax:time) : noexit :=
    in_slot; delay(dmin, dmax) out_slot;
    Slot[in_slot, out_slot](dmin, dmax)
```

```

    endproc (* SSlot *)
endproc (* Medium *)

process Sender[send,rec_ack,timeout](wait:bool):noexit:=
  [not(wait)]-> ( delay(1,3) send; Sender[send,rec_ack,timeout](true) )
  []
  [wait]-> (
    ( rec_ack; Sender[send,rec_ack,timeout](false) )
    []
    ( delay(19) timeout; send; Sender[send,rec_ack,timeout](true) )
  )
endproc (* Sender *)

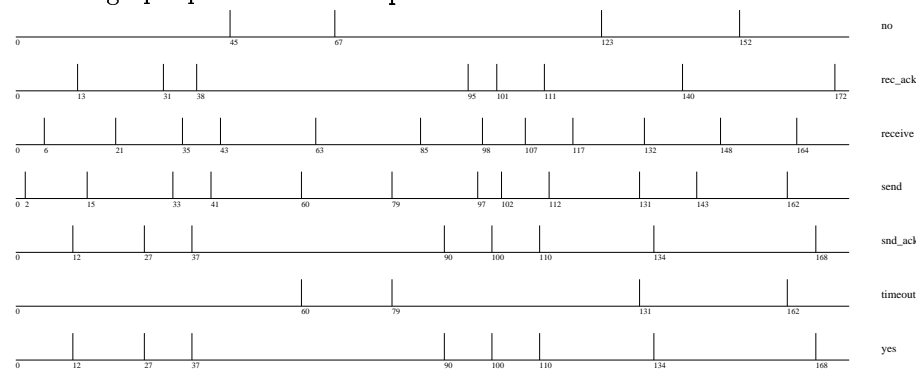
process Receiver[receive,snd_ack,yes,no]:noexit:=
  receive;
  delay(2,6) (
    ( no; Receiver[receive,snd_ack,yes,no] )
    []
    ( yes; snd_ack; Receiver[receive,snd_ack,yes,no] )
  )
endproc (* Receiver *)

endspec (* StopWait *)

```

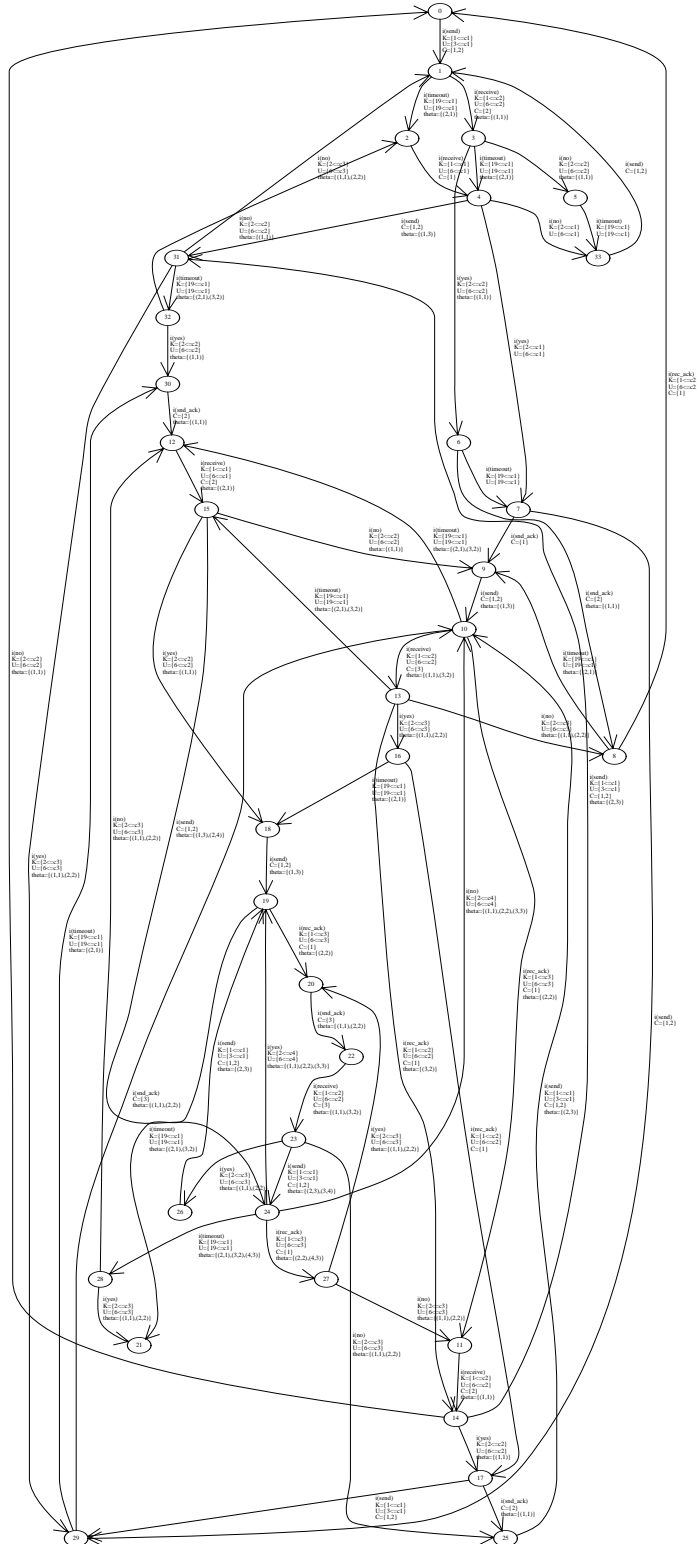
B stopw.sim

Vue graphique du fichier stopw.lot.sim.



C stopw.dta

Vue graphique du fichier stopw.dta.



D on_the_fly.lot

```
Contenu du fichier on_the_fly.lot.
(* Exemple academique dont on ne *)
(* peut realiser l'accessibilite *)
(* que a la volee. *)

specification On_The_Fly : noexit

type boolean is
  sorts bool
  opns
    not : bool->bool
    and : bool, bool->bool
    or : bool, bool->bool
endtype

type natural is boolean
  sorts nat
  opns
    + : nat, nat->nat
    - : nat, nat->nat
    * : nat, nat->nat
    < : nat, nat->bool
    > : nat, nat->bool
    <= : nat, nat->bool
    >= : nat, nat->bool
endtype

behaviour

  hide a, b in

    P1[a](1)
    |[a]|
    P2[a, b](0)

where

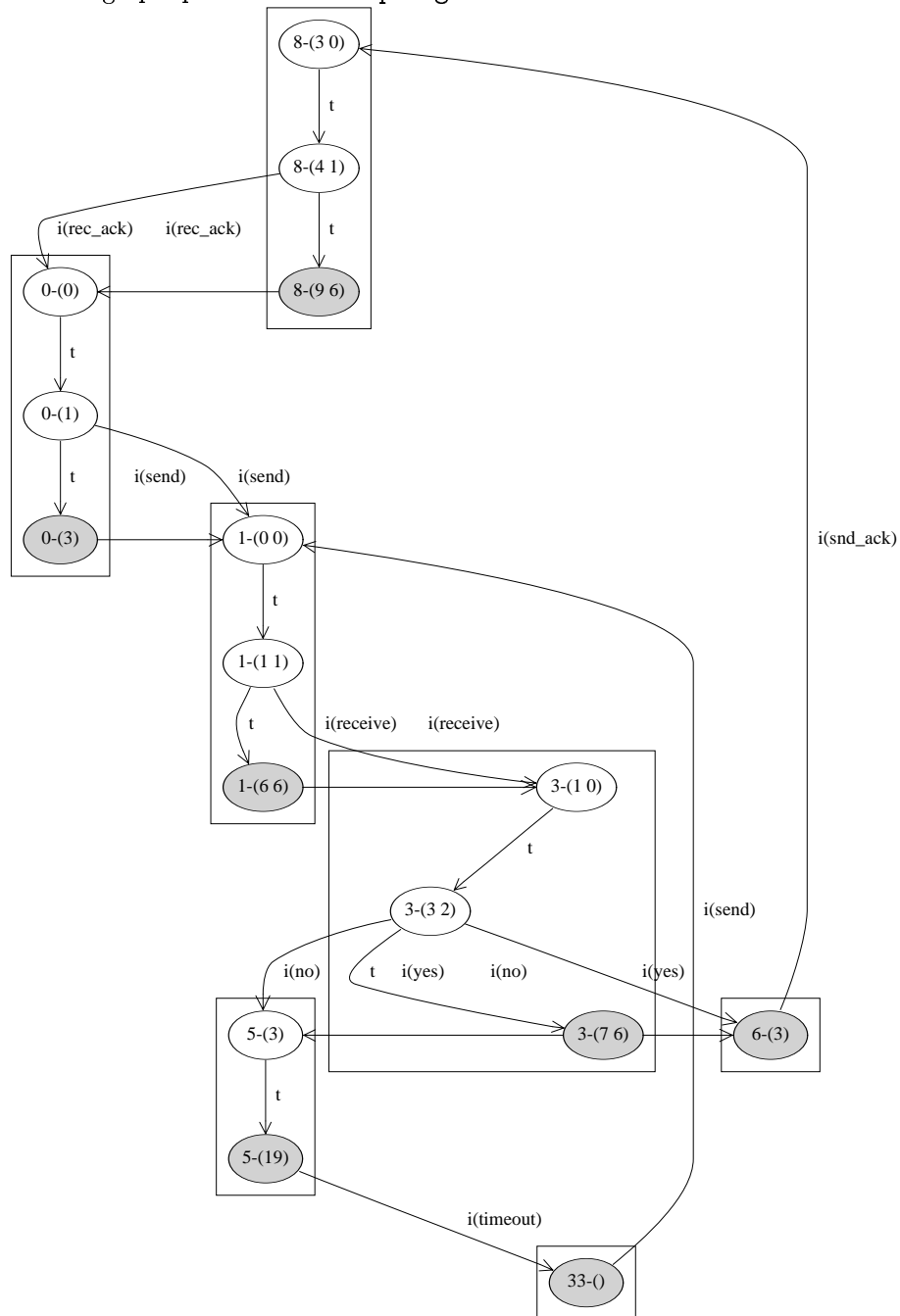
  process P1[a](per:time) : noexit :=
    delay(per) a; P1[a](per)
  endproc

  process P2[a, b](n:nat) : noexit :=
    ( a; P2[a, b](n+1) )
    []
    [n>0] -> ( b; P2[a, b](n-1) )
  endproc

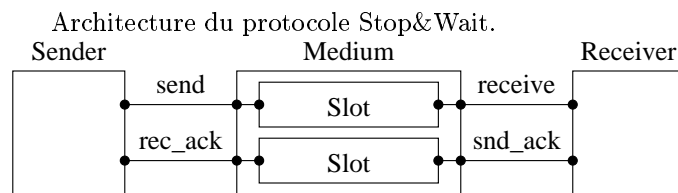
endspec
```

E stopw.rg

Vue graphique du fichier stopw.rg.

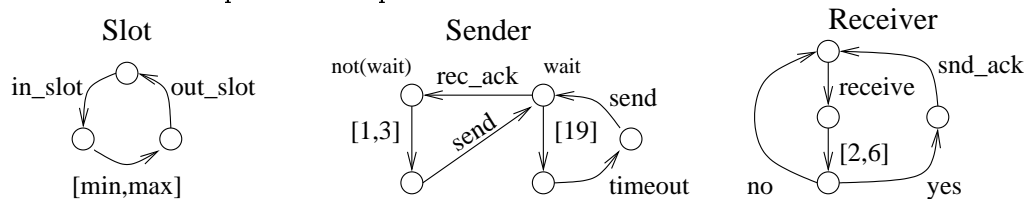


F Proposition de réponse à la question 2.1



G Proposition de réponse à la question 2.2

Automates du protocole Stop&Wait.



H Proposition de réponse à la question 2.3

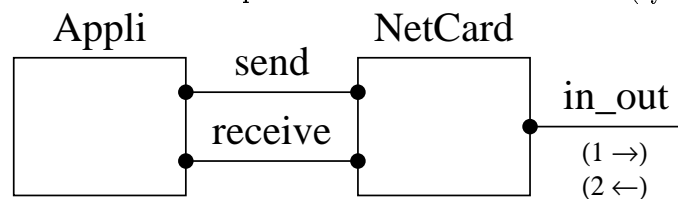
Deux canaux de communication sont utilisés : l'un pour les données, l'autre pour les acquittements.

L'émetteur envoie un paquet et attend son acquittement au plus 19 unités de temps. Si le paquet est acquitté, le prochain est envoyé. Si le paquet n'est pas acquitté, il est ré-émis. Et ainsi de suite.

Le récepteur reçoit un paquet, et de manière non déterministe décide de l'acquitter ou non.

I Proposition de réponse à la question 3.1

Architecture du protocole du côté d'une machine (symétrie).



J Proposition de réponse à la question 3.1

Automates du protocole.

