Annexe C

Application 1 : protocole du bit alterné

Je suis bi-alternatif
Alternativement positif
Je suis bi-bi-bi-bi-bi-bi-alternatif
Alternativement positif
Je suis alternativement bi-dégénéré, ça c'est super
Alternativement hyper-positif

Gogol I^{er}
poète, prophète, barbare

C.1 Description du service

Le protocole du bit alterné (*alternating bit protocol*) fait partie de la couche transport (4ème couche du modèle OSI). Il permet le transfert de données entre une paire d'entités pour lesquelles une connexion bi-directionnelle a été préalablement établie.

Pour simplifier le problème, on crée une disymétrie entre les deux entités : la première (T, comme *transmitter*) émet des messages à destination de la seconde (R, comme *receiver*).

Les messages sont modélisés par des numéros compris entre 1 et un entier maximal `N` ; ils sont spécifiés par le type abstrait suivant, qui ne précise pas leur nature exacte :

```
type MESSAGE is
   sorts MSG   (* type MSG = 1..N *)
endtype
```

Vu de la couche supérieure, le service fourni par le protocole du bit alterné est l'acheminement d'une série de messages de T vers R. La transmission est fiable : les messages ne peuvent pas être perdus ni dupliqués et ils sont reçus dans l'ordre où ils ont été émis. La spécification Lotos suivante décrit ce comportement :

```
specification ALTERNATING_BIT_SERVICE [PUT, GET] : noexit behaviour
   SERVICE [PUT, GET]
where
   process SERVICE [PUT, GET] : noexit :=
      PUT ?M:MSG;  (* acquisition d'un message *)
         GET !M;   (* livraison du message *)
            SERVICE [PUT, GET]
   endproc
endspec
```

C.2 Description du protocole

Le fonctionnement "idéal" du protocole du bit alterné est le suivant : T envoie un message à R ; à la réception de ce message, R renvoie un acquittement à T.

La liaison entre T et R n'est pas fiable : il est possible que des messages ou des acquittements soient perdus. En cas de perte, le medium peut, de manière facultative, signaler cette perte au destinataire (T ou R) en envoyant une indication de perte.

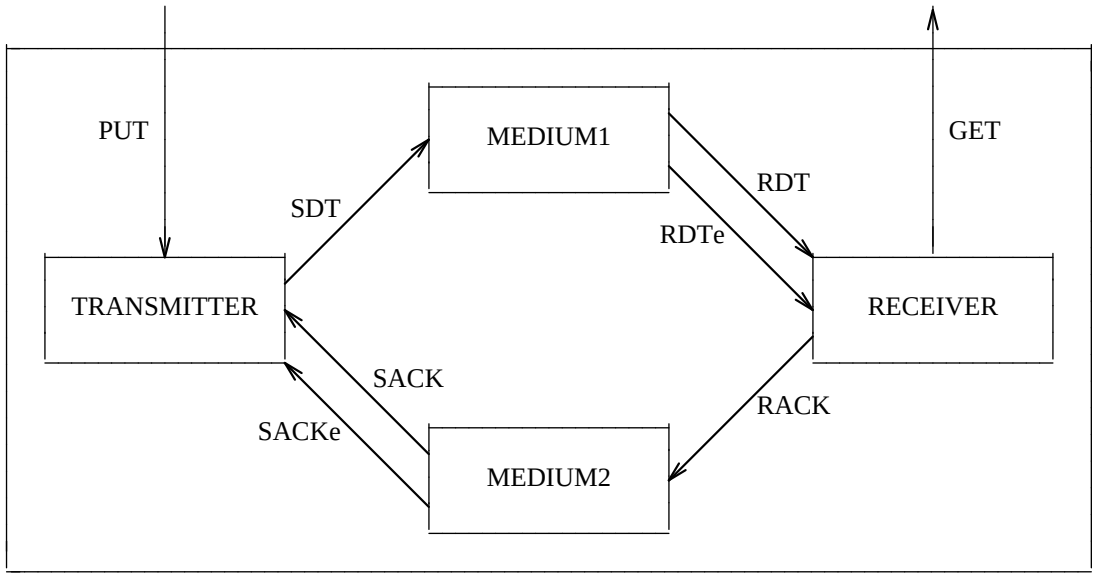
Pour détecter les pertes non signalées, les messages et les acquittements contiennent un bit de contrôle. Le bit de contrôle de chaque acquittement est égal au bit de contrôle du message qu'il acquitte. Les bits de contrôle de deux messages successivement émis ont des valeurs distinctes (la valeur du bit alterne à chaque émission).

Si l'entité T reçoit une indication de perte d'acquittement ou un acquittement avec un bit de contrôle erroné, elle réémet le dernier message envoyé.

Si l'entité R reçoit une indication de perte de message ou un message avec un bit de contrôle erroné, elle réémet le dernier acquittement envoyé.

C.3 Architecture du protocole

On choisit de décrire le protocole par quatre processus parallèles communicants :



processus	*signification*
TRANSMITTER	entité émettrice T
RECEIVER	entité réceptrice R
MEDIUM1	transmission des messages de T vers R
MEDIUM2	transmission des acquittements de R vers T

Le tableau suivant donne la liste des signaux utilisés. Les seuls signaux fournis par le service sont PUT et GET ; tous les autres désignent des signaux internes. Dans la suite M désigne un message (essentiellement un bloc de données) et B le bit de contrôle d'un message.

signal	*origine*	*destination*	*signification*
PUT !M	service	TRANSMITTER	émission d'un message
SDT !M !B	TRANSMITTER	MEDIUM1	envoi du message
RDT !M !B	MEDIUM1	RECEIVER	transmission du message
RDTe	MEDIUM1	RECEIVER	perte du message
GET !M	RECEIVER	service	réception du message
RACK !B	RECEIVER	MEDIUM2	renvoi d'un acquittement
SACK !B	MEDIUM2	TRANSMITTER	transmission de l'acquittement
SACKe	MEDIUM2	TRANSMITTER	perte de l'acquittement

L'émetteur et le récepteur fonctionnent de manière complètement asynchrone. Il en est de même pour les deux media. On peut donc spécifier l'architecture du protocole par le programme LOTOS ci-dessous, dans lequel les définitions des processus TRANSMITTER, RECEIVER, MEDIUM1 et MEDIUM2 ne sont pas explicitées :

```
specification ALTERNATING_BIT_PROTOCOL [PUT, GET] : noexit behaviour
   hide SDT, RDT, RDTe, RACK, SACK, SACKe in
      (
         (
         TRANSMITTER [PUT, SDT, SACK, SACKe] (0)
         |||
         RECEIVER [GET, RDT, RDTe, RACK] (0)
         )
      |[SDT, RDT, RDTe, RACK, SACK, SACKe]|
         (
         MEDIUM1 [SDT, RDT, RDTe]
         |||
         MEDIUM2 [RACK, SACK, SACKe]
         )
      )
where
   type BIT is
      sorts BIT
      opns 0 : -> BIT
           1 : -> BIT
           not : BIT -> BIT
   endtype
endspec
```

Remarque C-1
Le paramètre effectif 0 des processus TRANSMITTER et RECEIVER sert à initialiser la valeur du bit de contrôle ; par convention le bit de contrôle du premier message est égal à 0. ■

C.4 Spécification du medium des messages

En recevant un message M avec un bit de contrôle égal à B, le medium n^{o} 1 peut réagir de trois façons différentes :

- transmettre correctement le message et son bit de contrôle. En aucun cas le medium ne peut changer la valeur du bit de contrôle
- perdre le message et envoyer une indication de perte à l'entité réceptrice
- perdre silencieusement le message

```
process MEDIUM1 [SDT, RDT, RDTe] : noexit :=
   SDT ?M:MSG ?B:BIT;   (* reception d'un message *)
      (
      RDT !M !B;        (* transmission correcte *)
         MEDIUM1 [SDT, RDT, RDTe]
      []
      RDTe;             (* perte avec indication *)
         MEDIUM1 [SDT, RDT, RDTe]
      []
      i;                (* perte silencieuse *)
         MEDIUM1 [SDT, RDT, RDTe]
      )
endproc
```

C.5 Spécification du medium des acquittements

Le fonctionnement du medium n^{o} 2 est analogue à celui du medium n^{o} 1. La seule différence réside dans les noms de signaux et dans le fait que les acquittements, contrairement aux messages, ne portent pas d'information autre que le bit de contrôle.

```
process MEDIUM2 [RACK, SACK, SACKe] : noexit :=
   RACK ?B:BIT;   (* reception d'un acquittement *)
      (
      SACK !B;    (* transmission correcte *)
         MEDIUM2 [RACK, SACK, SACKe]
      []
      SACKe;      (* perte avec indication *)
         MEDIUM2 [RACK, SACK, SACKe]
      []
      i;          (* perte silencieuse *)
         MEDIUM2 [RACK, SACK, SACKe]
      )
endproc
```

C.6 Spécification de l'émetteur

L'entité émettrice acquiert un message via `PUT` et le transmet au medium n^{o} 1 après lui avoir ajouté la valeur courante B du bit de contrôle. Si elle reçoit en réponse un acquittement avec un bit de contrôle B la transmission a réussi, sinon il faut réémettre le message. Il y a 3 causes possibles de réémission :

- l'émetteur a reçu un acquittement ayant $(\neg B)$ comme bit de contrôle

- l'émetteur a reçu une indication de perte d'acquittement `SACKe`
- l'émetteur peut réémettre spontanément le message afin d'éviter le blocage dans le cas où le medium n^{o} 1 (*resp.* n^{o} 2) aurait perdu silencieusement un message (*resp.* un acquittement). En réalité, cette réémission n'a lieu que si une certaine contrainte de délai (*timeout*) est vérifiée (*cf.* [RSV86] et [RV87]) mais, LOTOS ne permettant pas d'exprimer le délai, on le modélise par un événement silencieux "**i**"

Remarque C-2
Pour décrire l'entité émettrice on a choisi une structure de contrôle paramétrée par des données (essentiellement la valeur B du bit de contrôle). D'autres solutions auraient été également viables, en particulier une modélisation sans données, avec une structure de contrôle complètement développée. Il s'agit de la dualité classique entre le contrôle et les données. ■

```
process TRANSMITTER [PUT, SDT, SACK, SACKe] (B:BIT) : noexit :=
   PUT ?M:MSG;  (* acquisition d'un message *)
      TRANSMIT [PUT, SDT, SACK, SACKe] (B, M)
where
   process TRANSMIT [PUT, SDT, SACK, SACKe] (B:BIT, M:MSG) : noexit :=
      SDT !M !B;           (* emission du message *)
         (
         SACK !B;          (* bit de controle correct *)
            TRANSMITTER [PUT, SDT, SACK, SACKe] (not (B))
         []
         SACK !(not (B));  (* bit de controle incorrect => reemission *)
            TRANSMIT [PUT, SDT, SACK, SACKe] (B, M)
         []
         SACKe;            (* indication de perte => reemission *)
            TRANSMIT [PUT, SDT, SACK, SACKe] (B, M)
         []
         i;                (* timeout => reemission *)
            TRANSMIT [PUT, SDT, SACK, SACKe] (B, M)
         )
   endproc
endproc
```

C.7 Spécification du récepteur

Remarque C-3
Le comportement de l'entité réceptrice est parfaitement symétrique de celui de l'entité paire, bien que la modélisation "contrôle+données" choisie puisse faire croire à la disymétrie. ■

Lorsqu'elle reçoit un message avec un bit de contrôle B correct, l'entité réceptrice délivre le message via `GET` et renvoie un acquittement avec un bit de contrôle égal à B. Dans les autres cas, elle renvoie un acquittement incorrect (ayant $(\neg B)$ comme bit de contrôle) ; ces cas sont au nombre de trois :

- le récepteur a reçu un message ayant $(\neg B)$ comme bit de contrôle
- le récepteur a reçu une indication de perte de message `RDTe`
- le récepteur peut émettre spontanément un acquittement invalide afin d'éviter le blocage dans le cas où le medium n^{o} 1 (*resp.* n^{o} 2) aurait perdu silencieusement un message (*resp.* un acquittement). Comme expliqué plus haut, il s'agit d'un *timeout* que l'on modélise par un événement silencieux "**i**"

```
process RECEIVER [GET, RDT, RDTe, RACK] (B:BIT) : noexit :=
   RDT ?M:MSG !B;          (* bit de controle correct *)
      GET !M;              (* livraison du message *)
         RACK !B;          (* envoi d'un acquittement correct *)
            RECEIVER [GET, RDT, RDTe, RACK] (not (B))
   []
   RDT ?M:MSG !(not (B));  (* bit de controle incorrect => *)
      RACK !(not (B));     (* envoi d'un acquittement incorrect *)
         RECEIVER [GET, RDT, RDTe, RACK] (B)
   []
   RDTe;                   (* indication de perte => *)
      RACK !(not (B));     (* envoi d'un acquittement incorrect *)
         RECEIVER [GET, RDT, RDTe, RACK] (B)
   []
   i;                      (* timeout => *)
      RACK !(not (B));     (* envoi d'un acquittement incorrect *)
         RECEIVER [GET, RDT, RDTe, RACK] (B)
endproc
```

C.8 Validation

A l'aide de la version 3.1 de CÆSAR on a produit, pour diverses valeurs de N, les réseaux et les graphes qui correspondent aux descriptions en LOTOS du service et du protocole.

Le réseau du service comporte 2 places, 2 transitions et 1 variable ; celui du protocole comporte 17 places, 21 transitions et 7 variables. Ces valeurs sont indépendantes de N.

Les deux tableaux suivants indiquent les résultats obtenus respectivement pour le service et le protocole. Dans chaque cas on indique la taille du graphe (nombre d'états et nombre d'arcs) ainsi que le temps total mis par CÆSAR pour produire ce graphe ; cette durée est exprimée sous la forme d'un couple *max–min* où :

- *max* est mesuré sur une station de travail SUN 3/50 avec 4Mo de mémoire principale, pour un seul utilisateur qui a ouvert 3 fenêtres sous X-windows version 10. On relève ainsi les performances de CÆSAR dans un environnement "normal"
- *min* est mesuré sur une station de travail SUN 3/60 avec 8Mo de mémoire principale, pour un seul utilisateur qui exécute seulement CÆSAR

Les durées *max* et *min* sont données sous la forme *minutes*:*secondes* ; il s'agit de temps utilisateur et non de temps virtuel. La dernière colonne du table contient les *débits*, c'est-à-dire le nombre d'états produits par seconde, calculés pour *max* et *min*.

N	états	arcs	temps	débit
5	11	35	0:27–0:22	0.4–0.5
10	21	120	0:28–0:18	0.8–1.2
15	31	255	0:30–0:17	1.0–1.8
20	41	440	0:30–0:17	1.4–2.4
25	51	675	0:34–0:19	1.5–2.7
30	61	960	0:32–0:17	1.9–3.5
35	71	1295	0:38–0:17	1.9–4.2
40	81	1680	0:29–0:15	2.8–5.4
45	91	2115	0:30–0:16	3.0–6
50	101	2600	0:31–0:19	3.3–5.3
70	141	5040	0:33–0:22	4.2–6.4

N	états	arcs	temps	débit
5	3 576	11 317	1:09–0:43	52–83
10	12 346	39 527	1:33–1:37	133–127
15	26 316	84 637	2:28–2:02	177–130
20	45 486	146 647	4:01–2:56	189–258
25	69 856	225 557	5:35–4:05	209–285
30	99 426	321 367	8:13–4:51	201–342
35	134 196	434 077	11:08–6:32	200–342
40	174 166	563 687	23:39–9:11	122–316
45	219 336	710 197	159:49–11:11	23–326
50	269 706	873 607	?–14:48	?–304
70	523 186	1 696 247	?–80:53	?–108

Pour `N` valant 5, 10 et 15, l'utilisation du logiciel ALDEBARAN [Fer88] a permis de montrer que le graphe du protocole était observationnellement équivalent à celui du service.

Pour des valeurs élevées de `N` la taille du graphe croît rapidement et avec elle le temps nécessaire pour comparer le protocole au service. Dans de telles situations, le recours aux logiques temporelles s'impose. Voici un exemple de propriétés, exprimées dans la logique R*ICO, qui devraient être vérifiées par le graphe correspondant au protocole du bit alterné :

- il n'y a pas de blocage :

  not stop

- il est impossible d'avoir deux actions `PUT` successives :

  never `(PUT ?M1:MSG . i* . PUT ?M2:MSG)`

- il est impossible d'avoir deux actions `GET` successives :

  never `(GET ?M1:MSG . i* . GET ?M2:MSG)`

- il est impossible qu'une action `PUT !M1` soit suivie d'une action `GET !M2` si la valeur de `M2` est différente de celle de `M1` :

  all `M1:MSG` **in never** `(PUT !M1:MSG . i* . GET ?M2:MSG [not (M1 = M2)])`

- toute action `PUT !M` est inévitablement[46] suivie d'une action `GET !M` :

all `M:MSG` **in** `(PUT !M =>` **finev** `(`**i**`,GET !M))`

C.9 Notes bibliographiques

Le protocole du *bit alterné* a longtemps eu la faveur des concepteurs de systèmes de vérification automatique, principalement en raison de la petite taille de son graphe (quelques dizaines d'états). La version en LOTOS proposée ici permet, lorsque l'on donne à `N` une valeur élevée, d'obtenir des graphes de grande taille, démontrant ainsi les capacités réelles de CÆSAR.

La modélisation proposée s'inspire principalement de deux sources :

- [RSV86] et [RV87] : il s'agit d'une description du protocole, dans le formalisme ATP, qui prend en compte les contraintes de délai. Le medium entre les deux entités est défini sous la forme de deux processus identiques fonctionnant en parallèle. Les messages et les acquittements peuvent être perdus sans que le medium le signale
- [Lon87] : cette description représente le medium comme un processus unique. Les messages et les acquittements peuvent être perdus, mais le medium doit alors envoyer au destinataire une indication de perte de message ou d'acquittement

Par rapport à [RSV86], [RV87] et [Lon87], la transmission des messages (considérés comme des valeurs entières) est effectivement modélisée. Les deux cas d'erreur de transmission ont été modélisés : perte silencieuse et perte signalée au destinataire.

Enfin les contraintes temporelles n'ont pas été conservées parce qu'elles ne peuvent pas s'exprimer simplement en LOTOS. Ceci appelle quelques remarques :

- lorsqu'on remplace une contrainte de délai par un événement "**i**", le nouveau comportement est un sur-ensemble de l'ancien
- toutefois, en règle générale, les "bons algorithmes" ne dépendent pas des valeurs des délais et conservent leurs "bonnes propriétés" (*speed independance*)
- autrement dit l'ajustage des délais permet d'améliorer les performances du système mais il ne doit pas servir à rendre correct un algorithme qui serait faux pour des valeurs de délais quelconques
- toutefois, la modélisation d'un délai par un événement "**i**" peut modifier le comportement du système. Dans le cas du bit alterné, si l'on choisit judicieusement les valeurs des délais ([RSV86], [RV87]), à chaque instant, au plus un message ou acquittement circule sur l'ensemble des deux media (la liaison est *half-duplex*). Si les délais sont mal choisis ou modélisés de manière asynchrone, plusieurs messages ou acquittements peuvent transiter simultanément, dans les deux sens (la liaison devient *full-duplex*)

[46] sous l'hypothèse d'équité