Une spécification du protocole ATMR

Jean Goubault-Larrecq

January 29, 1997

1 Description informelle

ATMR (Asynchronous Transfer Mode Ring) est un protocole de transmission de données rapide sur un réseau en anneau [IIKM94]. Sa conception est fondée sur un anneau de *cases* (slots en anglais) qui circule en continu sur l'ensemble des sites connectés. (Cf. figure 1, où les cases grisées représentent des cases vides.) Nous nous inspirons du développement formel de [CP97].

Un site envoie un message à un autre en transmettant une cellule (la quantité d'informations de base) dans une case vide et en y ajoutant l'adresse de destination et sa propre adresse dans des champs spécifiques. Les noms de ces champs sont respectivement DA ("destination address") et BA ("busy address"). Lorsqu'une case pleine arrive à un site, ce dernier regarde si le champ DA contient sa propre adresse. Si oui, l'information est reçue, et le champ DA est remis à 0 pour vider la case.

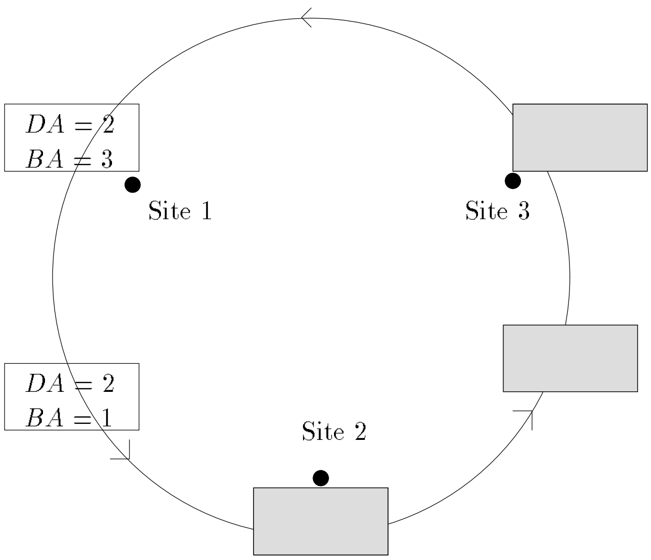

Figure 1: Exemple à trois sites et cinq cases

Le protocole ATMR possède un mécanisme de fenêtrage pour assurer une bonne distribution des ressources du réseau entre les différents sites. Lorsqu'une case visitant un site est vide, ce site peut l'utiliser pour transmettre un message. Il ne peut réellement l'utiliser que s'il a un message à transmettre, *et* si son crédit est positif. Initialement, chaque site reçoit un certain crédit appelé *taille de fenêtre*, et qui indique au site le nombre de cellules que le site a le droit de transmettre. Ce crédit est conservé dans un compteur de fenêtre local. À chaque cellule transmise, ce compteur est décrémenté de 1. Lorsque sa valeur atteint 0, le site s'arrête d'envoyer des cellules, et ce bien qu'il puisse lui rester des cellules en attente de transmission. Le compteur est réinitialisé à sa valeur initiale lorsque le site reçoit une case spéciale nommée *reset*, et l'envoi de cellules peut alors reprendre.

Les cellules voyagent comme suit. Lorsqu'un site est actif, c'est-à-dire lorsqu'il lui reste des cellules en attente d'envoi et que son compteur de fenêtre est strictement positif, il remplace le champ BA de chaque case qui lui arrive par sa propre adresse. Les cases *reset* sont envoyées par les sites qui viennent de recevoir une cellule dont le champ BA est égal à leur adresse. L'idée est que le message envoyé plus tôt par un autre site a été reçu, et cette case reset va voyager autour de l'anneau, réinitialisant tous les compteurs de fenêtre jusqu'à revenir au site qui l'a émis. Pour éviter un bouclage, le site qui a émis une case reset passe en *mode de poursuite* ("hunting state") jusqu'à ce qu'il reçoive une case reset.

2 Une spécification concrète

Nous faisons d'abord une hypothèse simplificatrice. Le protocole garantit que les cases visitent les sites circulairement, sans duplication, effacement ou permutation d'ordre. Nous allons donc supposer que le réseau tout entier passe par des états de *réception* des cases par les sites, puis d'*émission* de cases par les sites, puis de *rotation* de l'anneau des cases d'un cran sur l'anneau des sites (ce qui modélise la circulation des informations proprement dites).

$$ETAT ::= Reception \mid Emission \mid Rotation$$

Etat

$etat : ETAT$

Il est nécessaire pour que le protocole fonctionne qu'il y ait au moins autant de cases dans l'anneau que de sites. Ne pas faire cette hypothèse obligerait à supposer qu'il y a des sites qu'aucune case ne visite. Nous définissons donc le schéma suivant: les cases sont numérotées de 1 à $case\_max$, et les sites de 1 à $site\_max$. Nous réservons 0 comme numéro de site pour représenter l'absence de site (notamment la valeur du champ DA dans les cases vides).

$case\_max : \mathbb{N}$
$site\_max : \mathbb{N}$
$Cases : \mathbb{P}\mathbb{N}$
$Sites : \mathbb{P}\mathbb{N}$
$Adresses : \mathbb{P}\mathbb{N}$

$case\_max \geq site\_max$
$Cases = 1 \mathinner{..} case\_max$
$Sites = 1 \mathinner{..} site\_max$
$Adresses = 0 \mathinner{..} site\_max$

Ce genre de schḿa n'est pas décrit dans le cours : il s'agit d'une boîte de définitions d'axiomes. Ici, nous avons déclaré cinq identificateurs $case\_max$, $site\_max$, $Cases$, $Sites$ et $Adresses$, qui seront visibles dans tous les schémas de la suite. Ils sont soumis aux quatre contraintes écrites dans la deuxième partie.

Décrivons maintenant les cases. À chaque numéro de case c (un élément de $1 \mathinner{..} case\_max$), nous associons la valeur $DA(c)$ de son champ DA et la valeur $BA(c)$ de son champ BA. Nous associons aussi un booléen qui sera vrai si et seulement si cette case est la case spéciale reset; mais il est conceptuellement plus simple de définir un ensemble $reset$ de toutes les cases qui sont effectivement des cases reset. Finalement, nous associons à chaque case c un message $msg(c)$, dont le contenu nous importe peu. Le type des messages sera donc un type générique $Message$:

$$[Message]$$

ContenuCases

$DA : Cases \rightarrow Adresses$
$BA : Cases \rightarrow Adresses$
$reset : \mathbb{P} Cases$
$msg : Cases \rightarrow Message$

De même, nous décrivons l'état courant des sites. À chaque site s, nous associons le numéro $case(s)$ de la case qui visite le site et la valeur du compteur de fenêtre $credit(s)$. L'ensemble $poursuite$ est celui des sites qui sont en mode de poursuite. L'ensemble $attente$ est celui des sites qui sont en attente d'envoi d'un message (ou plusieurs). À chaque site s en attente, nous associons le message $envoi(s)$ qu'il est en attente d'envoyer et l'adresse $dest(s)$ à laquelle il doit l'envoyer. On associe aussi aux sites s le dernier message $recu(s)$ qu'ils ont reçu, s'ils en ont reçu un.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{ContenuSites}} \\
case : Sites \rightarrow Cases \\
credit : Sites \rightarrow \mathbb{N} \\
poursuite : \mathbb{P}\, Sites \\
attente : \mathbb{P}\, Sites \\
envoi : Sites \nrightarrow Message \\
dest : Sites \nrightarrow Sites \\
recu : Sites \nrightarrow Message \\
\hline
\mathrm{dom}\ envoi = attente \\
\mathrm{dom}\ dest = attente \\
\hline
\end{array}
$$

Initialement, le réseau est en état de réception. Les champs DA et BA de chaque case sont vides (égaux à 0), et aucune case n'est la case spéciale reset. (Le champ message est quelconque.) On s'assure aussi qu'initialement, deux sites ne peuvent pas être visités par la même case. (Il s'agit d'une propriété qui devrait rester vraie tout le temps; il faudra le prouver pour en être sûr; cf. section 3.) Initialement, de plus, le crédit attribué à chaque site est la taille de fenêtre $FENETRE$. (Noter que l'on pose $FENETRE \geq 1$, sinon jamais aucun message ne pourrait être émis par aucun site.) Finalement, aucun site n'est en mode de poursuite.

$$
\begin{array}{|l}
FENETRE : \mathbb{N} \\
\hline
FENETRE \geq 1
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{Initial}} \\
Etat \\
ContenuCases \\
ContenuSites \\
\hline
etat = Reception \\
\forall c : Cases \bullet DA(c) = 0 \\
\forall c : Cases \bullet BA(c) = 0 \\
reset = \emptyset \\
\forall s_1, s_2 : Sites \bullet case(s_1) = case(s_2) \Rightarrow s_1 = s_2 \\
\forall s : Sites \bullet credit(s) = FENETRE \\
poursuite = \emptyset \\
\hline
\end{array}
$$

Décrivons maintenant le comportement dynamique du réseau. Tout d'abord, nous allons définir quelques abréviations, et définir les ensembles $actifs$ des processus actifs, et $passifs$ des processus non actifs.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{ActiviteSites}} \\
ContenuSites \\
actifs : \mathbb{P}\, Sites \\
passifs : \mathbb{P}\, Sites \\
\hline
actifs = \{ s : Sites \mid credit(s) > 0 \wedge s \in attente \bullet s \} \\
passifs = Sites \setminus actifs \\
\hline
\end{array}
$$

La phase de réception de messages se déroule lorsque l'état global est à $Reception$. À la fin de cette phase, l'état global passe à $Emission$. Pour chaque site s, les changements d'états suivants doivent être effectués, que nous écrivons en notation informelle :

1. Si la case qui visite s a pour destinataire s lui-même, c'est-à-dire si $DA(case(s)) = s$, alors:

$$
\begin{array}{rll}
DA(case(s)) & := 0 & \text{— vider la case} \\
recu(s) & := msg(case(s)) & \text{— récupérer le contenu du message}
\end{array}
$$

(Le message est arrivé à sa destination.)

2. Si la case qui visite s est une case reset, et s n'est pas en mode de poursuite, c'est-à-dire si $case(s) \in reset \wedge s \notin poursuite$:

$$credit(s) \quad := FENETRE \quad \text{— réinitialiser le compteur de fenêtre à sa valeur initiale}$$

   (Le site reçoit un message venant d'un autre site l'autorisant à réémettre à concurrence de la taille de fenêtre.)

3. Si la case qui visite s est une case reset, et s est en mode de poursuite, c'est-à-dire si $case(s) \in reset \wedge s \in poursuite$, alors :

$$\begin{array}{rll} reset & := reset \setminus \{case(s)\} & \text{— faire de cette case une case non-reset} \\ poursuite & := poursuite \setminus \{s\} & \text{— sortir } s \text{ du mode de poursuite} \end{array}$$

   (Intuitivement, le site reçoit une case reset qu'il a envoyée plus tôt, et qui revient après un tour. Cette explication intuitive est incorrecte : pouvez-vous trouver ce qui manque ?)

4. Si s est passif (c'est-à-dire que $credit(s) = 0$ ou s n'est pas en $attente$), le champ BA de la case qui visite s est s lui-même, s n'est pas en mode de poursuite et la case qui visite s n'est pas une case reset, alors :

$$\begin{array}{rll} reset & := reset \cup \{case(s)\} & \text{— créer une case reset} \\ credit(s) & := FENETRE & \text{— réinitialiser le compteur de fenêtre à sa valeur initiale} \\ poursuite & := poursuite \cup \{s\} & \text{— entrer en mode de poursuite} \end{array}$$

   (Intuitivement, si le champ BA de la case qui visite s est s lui-même, c'est que ce champ n'a été changé par aucun autre site qui l'a transmis avant s, ce qui ne peut arriver que si ces autres sites sont tous passifs. En effet, si un seul de ces sites était actif, il aurait remplacé le champ BA par sa propre adresse. On en conclut que le message a fait un tour complet sans rencontrer de site actif, et on peut donc envoyer un message reset à tous les sites — soi en premier — pour réinitialiser les compteurs de fenêtre. Cette explication est, malheusement, erronée : voyez-vous pourquoi ?)

Ces cas ne sont pas exclusifs. Les autres champs ne changent pas.

Pour spécifier tout ceci, nous allons utiliser le calcul des schémas. Tout d'abord, le passage de l'état $Reception$ vers l'état $Emission$ peut se faire par un schéma $ReceptionVersEmission$ séparé :

$$\begin{array}{|l} \multicolumn{1}{l}{\text{— } ReceptionVersEmission \text{ —}} \\ \Delta Etat \\ \hline etat = Reception \\ etat' = Emission \\ \hline \end{array}$$

Ensuite, les divers traitements à effectuer en réception se décomposent en les quatre cas décrits ci-dessus, que l'on décrit par quatre ensembles respectifs de sites $cas1$, $cas2$, $cas3$ et $cas4$:

$$\begin{array}{|l} \multicolumn{1}{l}{\text{— } CasReception \text{ —}} \\ ActiviteSites \\ ContenuCases \\ cas1, cas2, cas3, cas4 : \mathbb{P} Sites \\ \hline cas1 = \{s : Sites \mid DA(case(s)) = s \bullet s\} \\ cas2 = \{s : Sites \mid case(s) \in reset \wedge s \notin poursuite \bullet s\} \\ cas3 = \{s : Sites \mid case(s) \in reset \wedge s \in poursuite \bullet s\} \\ cas4 = \{s : Sites \mid case(s) \notin reset \wedge s \notin poursuite \wedge s \in passifs \wedge BA(case(s)) = s \bullet s\} \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{\text{— } TraitementsReception \text{ —}} \\ \Delta ContenuSites \\ \Delta ContenuCases \\ cas1, cas2, cas3, cas4 : \mathbb{P} Sites \\ \hline case' = case \\ credit' = credit \oplus \{s : Sites \mid s \in cas2 \vee s \in cas4 \bullet (s, FENETRE)\} \\ poursuite' = poursuite \setminus cas3 \cup cas4 \\ attente' = attente \\ envoi' = envoi \\ dest' = dest \\ recu' = recu \oplus \{s : Sites \mid s \in cas1 \bullet (s, msg(case(s)))\} \\ DA' = DA \oplus \{s : Sites \mid s \in cas1 \bullet (case(s), 0)\} \\ BA' = BA \\ reset' = reset \setminus \{s : Sites \mid s \in cas3 \bullet case(s)\} \cup \{s : Sites \mid s \in cas4 \bullet case(s)\} \\ msg' = msg \\ \hline \end{array}$$

Nous combinons le tout, et masquons les identificateurs auxiliaires $cas1$, $cas2$, $cas3$, $cas4$, $actifs$ et $passifs$ en utilisant le quantificateur existentiel sur les schémas :

$$\begin{array}{l} PhaseReception \mathrel{\widehat=} \exists cas1, cas2, cas3, cas4, actifs, passifs : \mathbb{P}\, Sites \bullet \\ \quad CasReception \land TraitementsReception \land ReceptionVersEmission \end{array}$$

Les phases d'émission et de rotation sont plus simples. La phase d'émission de déroule lorsque l'état global est à $Emission$. À la fin de cette phase, l'état global passe à $Rotation$, ce qui est décrit par le schéma $EmissionVersRotation$ suivant :

$$\begin{array}{|l} \multicolumn{1}{l}{EmissionVersRotation} \\ \Delta Etat \\ \hline etat = Emission \\ etat' = Rotation \\ \hline \end{array}$$

Les changements informels à effectuer, pour chaque site s, sont :

1. Si la case qui visite s est vide (ou a été vidée par la phase de réception précédente) et que s est actif (c'est-à-dire qu'il a des messages en attente et n'a pas épuisé son crédit), alors :

$$\begin{array}{rll} msg(case(s)) & := envoi(s) & \text{— envoyer le message en attente} \\ DA(case(s)) & := dest(s) & \text{— à qui de droit} \\ credit(s) & := credit(s) - 1 & \text{— décrémenter le crédit d'émission} \\ attente & := attente \setminus \{s\} & \text{— sortir du mode d'attente} \end{array}$$

2. Si s est actif, en général :

$$BA(case(s)) \quad := s \quad \text{— inscrire sa propre adresse dans le champ } BA \text{ de la case à envoyer}$$

Encore une fois, ces deux cas ne s'excluent pas. De plus, l'opération $BA(case(s)) := s$ est effectuée que la case courante $case(s)$ soit vide ou pleine. Nous nommons ces deux cas comme dans la phase de réception :

$$\begin{array}{|l} \multicolumn{1}{l}{CasEmission} \\ ActiviteSites \\ ContenuCases \\ cas1, cas2 : \mathbb{P}\, Sites \\ \hline cas1 = \{ s : Sites \mid DA(case(s)) = 0 \land s \in actifs \bullet s \} \\ cas2 = actifs \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{TraitementsEmission} \\ \Delta ContenuSites \\ \Delta ContenuCases \\ cas1, cas2 : \mathbb{P}\, Sites \\ \hline case' = case \\ credit' = credit \oplus \{ s : Sites \mid s \in cas1 \bullet (s, credit(s) - 1) \} \\ poursuite' = poursuite \\ attente' = attente \setminus cas1 \\ envoi' = attente' \lhd envoi \\ dest' = attente' \lhd dest \\ recu' = recu \\ DA' = DA \oplus \{ s : Sites \mid s \in cas1 \bullet (case(s), dest(s)) \} \\ BA' = BA \oplus \{ s : Sites \mid s \in cas2 \bullet (case(s), s) \} \\ reset' = reset \\ msg' = msg \oplus \{ s : Sites \mid s \in cas1 \bullet (case(s), envoi(s)) \} \\ \hline \end{array}$$

(Exercice : pourquoi a-t-on écrit les deux lignes $envoi' = attente' \lhd envoi$ et $dest' = attente' \lhd dest$ ci-dessus, alors qu'aucun effet de bord sur $envoi$ ou sur $dest$ n'était décrit ni dans le cas 1 ni dans le cas 2 plus haut ?)

Nous combinons le tout comme dans la phase de réception :

$$PhaseEmission \mathrel{\widehat=} \exists cas1, cas2, actifs, passifs : \mathbb{P} Sites \bullet$$
$$CasEmission \land TraitementsEmission \land EmissionVersRotation$$

La phase de rotation est encore plus simple. Elle consiste juste à faire que tout site qui avait la case 1 ait la case 2, tout site qui avait la case 2 ait la case 3, ..., tout site qui avait la case $case\_max$ ait la case 1. On notera le $\Xi ContenuCases$ dans le schéma $TraitementsRotation$: pendant une rotation, le contenu des cases ne change pas. (Exercice : que décrivent les trois lignes $attente \subseteq attente'$, $attente \lhd envoi' = envoi$, $attente \lhd dest' = dest$? Pourquoi n'ai-je pas eu besoin de spécifier aussi $\mathsf{dom}\ envoi' = attente'$ et $\mathsf{dom}\ dest' = attente'$?)

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{— } RotationVersReception \text{ ———}} \\
\Delta Etat \\
\hline
etat = Rotation \\
etat' = Reception \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{— } TraitementsRotation \text{ ———}} \\
\Delta ContenuSites \\
\Xi ContenuCases \\
\hline
case' = \{s : Sites \bullet (s, (case(s) \ \mathsf{mod}\ case_max) + 1)\} \\
credit' = credit \\
poursuite' = poursuite \\
attente \subseteq attente' \\
attente \lhd envoi' = envoi \\
attente \lhd dest' = dest \\
recu' = recu \\
\hline
\end{array}$$

$$PhaseRotation \mathrel{\widehat=} TraitementsRotation \land RotationVersReception$$

Nous pouvons maintenant spécifier ce qu'est une phase du protocole :

$$Phase \mathrel{\widehat=} PhaseReception \lor PhaseEmission \lor PhaseRotation$$

Nous voulons maintenant spécifier ce qu'est une trace du protocole. Une trace est une suite infinie de configurations du système global telle que la configuration numéro 0 soit une configuration initiale (décrite par le schéma $Initial$), et que l'on passe de la configuration i à la configuration $i+1$ par une phase du protocole (décrite par le schéma $Phase$).

Ceci demande de passer à une vue plus globale du protocole. Les spécifications qui suivent sont ardues à lire; de plus, elles utilisent la notation θ, dont il n'a pas été question dans le poly du cours.

Nous définissons d'abord un schéma $Config$, définissant le type des configurations. Il s'agit d'un schéma contenant toutes les valeurs de l'état $etat$, ainsi que des identificateurs DA, BA, $reset$ et msg (décrits dans $ContenuCases$), et des identificateurs $case$, $credit$, $poursuite$, $attente$, $envoi$, $dest$ et $recu$ (décrits dans $ContenuSites$), soumis aux invariants hérités de ces schémas, soit $\mathsf{dom}\ envoi = attente$ et $\mathsf{dom}\ dest = attente$.

$$\begin{array}{|l}
\multicolumn{1}{l}{\text{— } Config \text{ ———}} \\
Etat \\
ContenuCases \\
ContenuSites \\
\hline
\end{array}$$

Une chose que nous n'avons pas mentionnée dans le cours est qu'un nom de schéma dénote aussi l'ensemble de tous les enregistrements (records) dont les champs sont décrits dans la signature du schéma et sont soumis à l'invariant décrits dans la contrainte du schḿa. Ainsi $Initial$ dénote-t-il aussi un sous-ensemble de $Config$, à savoir l'espace de toutes les configurations initiales.

La relation de transition est une relation entre configurations à une étape du protocole et les configurations possibles du système à l'étape suivante. Cette relation de transition, $relation\_de\_transition$, doit donc être de type $\mathbb{P}(Config \times Config)$. Nous la décrivons par le schéma suivant :

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{RelationDeTransition}} \\ \hline
relation_de_transition : \mathbb{P}(Config \times Config) \\ \hline
relation_de_transition = \{Phase \bullet (\theta Config, \theta Config')\} \\ \hline
\end{array}$$

Expliquons ce schéma sibyllin. La notation θS, où S est un schéma de la forme :

$$\begin{array}{|l}
\multicolumn{1}{l}{S} \\ \hline
x_1 : T_1 \\
\ldots \\
x_n : T_n \\ \hline
P(x_1, \ldots, x_n) \\ \hline
\end{array}$$

dénote juste l'enregistrement formé des champs de noms $x_1, \ldots, x_n$, avec leurs valeurs dans l'environnement courant. De même, $\theta S'$ dénote l'enregistrement formé des champs de noms $x_1, \ldots, x_n$, avec comme valeurs celles de $x'_1, \ldots, x'_n$ dans l'environnement courant. (Note : si e est une expression de type S, par exemple θS ou $\theta S'$, et si x est un identificateur déclaré dans la signature de S, c'est-à-dire le nom d'un champ de e, alors $e.x$ dénote la valeur de ce champ dans e.)

Si R est un autre schéma, de la forme :

$$\begin{array}{|l}
\multicolumn{1}{l}{R} \\ \hline
x_1, x'_1 : T_1 \\
\ldots \\
x_n, x'_n : T_n \\ \hline
Q(x_1, \ldots, x_n, x'_1, \ldots, x'_n) \\ \hline
\end{array}$$

alors rappelons (cf. cours) que $\{R \bullet f(x_1, \ldots, x_n, x'_1, \ldots, x'_n)\}$ est une abréviation de :

$$\{x_1, x'_1 : T_1; \ \ldots; \ x_n, x'_n : T_n \mid Q(x_1, \ldots, x_n, x'_1, \ldots, x'_n) \bullet f(x_1, \ldots, x_n, x'_1, \ldots, x'_n)\}$$

Le lecteur en déduira aisément la signification de l'expression $\{Phase \bullet (\theta Config, \theta Config')\}$ du schéma $RelationDeTransition$.

Une *trace* d'exécution du protocole est alors n'importe quelle suite infinie de configurations, dont le premier élément est une configuration initiale, et telle que deux configurations consécutives (numéros i et $i+1$) sont en relation par la relation de transition. Les traces sont donc des applications de $\mathbb{N}$ (l'indice, ou temps i) vers $Config$ (la configuration au temps i) :

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{ATMRTraces}} \\ \hline
init? : Initial \\
trace! : \mathbb{N} \to Config \\ \hline
trace!(0) = init? \\
\exists RelationDeTransition \bullet \forall i : \mathbb{N} \bullet (trace!(i), trace!(i+1)) \in relation_de_transition \\ \hline
\end{array}$$

On notera que l'on a décrit les traces possibles $trace!$ comme des paramètres de sortie : le schémas $ATMRTraces$ doit fournir une trace ATMR en sortie. Nous utilisons un paramètre d'entrée $init?$ qui représente un état initial. La quantification existentielle sur $RelationDeTransition$ est une abréviation de :

$$\begin{array}{l}
\exists relation_de_transition : \mathbb{P}(Config \times Config) \\
\quad \mid relation_de_transition = \{Phase \bullet (\theta Config, \theta Config')\} \bullet \ldots
\end{array}$$

Utiliser l'existentielle ainsi est un moyen classique d'introduire une abréviation. En ML, par exemple (et même en Z complet, en fait), on écrirait :

$$\mathbf{let}\ relation\_de\_transition = \{Phase \bullet (\theta Config, \theta Config')\} \bullet \ldots$$

(Exercice : on se convaincra, en le prouvant, que $\exists x : T \mid x = e \bullet P(x)$ est équivalent à $P(e)$, lorsque $e \in T$.)

3 Les spécifications abstraites

Nous allons vouloir vérifier un certain nombre de propriétés du protocole ATMR (dans la version synchrone présentée en section 2). Le plus simple est d'écrire ces propriétés comme des spécifications abstraites, et de demander à vérifier que la spécification concrète *raffine* toutes ces spécifications abstraites.

3.1 Il n'y a jamais qu'une case par site

Nous avons dit par exemple qu'initialement (cf. schéma *Initial*), deux cases ne pouvaient pas visiter deux sites différents. Nous voulons être sûrs que cette propriété reste toujours vraie. Écrivons-la donc :

$$
\begin{array}{|l}
\hline
\textit{Coherence1} \\
\textit{init?} : \textit{Initial} \\
\textit{trace!} : \mathbb{N} \rightarrow \textit{Config} \\
\hline
\forall i : \mathbb{N} \bullet \forall s_1, s_2 : \textit{Sites} \bullet (\textit{trace!}(i)).\textit{case}(s_1) = (\textit{trace!}(i)).\textit{case}(s_2) \Rightarrow s_1 = s_2 \\
\hline
\end{array}
$$

Le fait que *ATMRTraces* raffine *Coherence*1 est donné par les conditions du cours :

$$
\begin{array}{ll}
(D) & \forall \textit{init?} : \textit{Initial} \bullet \mathsf{pre}\ \textit{Coherence1} \Rightarrow \mathsf{pre}\ \textit{ATMRTraces} \\
(R) & \forall \textit{init?} : \textit{Initial};\ \textit{trace!} : \mathbb{N} \rightarrow \textit{Config} \bullet \mathsf{pre}\ \textit{Coherence1} \wedge \textit{ATMRTraces} \Rightarrow \textit{Coherence1}
\end{array}
$$

qui se simplifient du fait que l'état dans ces spécifications est vide.

Maintenant, pre *Coherence*1 est le schéma *Coherence*1 quantifié existentiellement sur ses variables d'état après et ses paramètres de sortie :

$$\mathsf{pre}\ \textit{Coherence1} \mathrel{\widehat=} \exists \textit{trace!} : \mathbb{N} \rightarrow \textit{Config} \bullet \textit{Coherence1} \tag{1}$$

Mais alors la contrainte du schéma pre *Coherence*1 ne dépend d'aucun paramètre (le seul qui est libre dans *Coherence*1 est *trace*!, mais c'est celui sur lequel porte la quantification existentielle ci-dessus). En fait, sous l'hypothèse $\textit{init?} \in \textit{Initial}$ (donnée par la quantification universelle de (D)), on peut prouver pre *Coherence*1. En effet, on vérifie que :

$$\textit{Coherence1}[\{j : \mathbb{N} \bullet (j, \textit{init?})\}/\textit{trace!}] \tag{2}$$

c'est-à-dire que :

$$\forall i : \mathbb{N} \bullet \forall s_1, s_2 : \textit{Sites} \bullet (\{j : \mathbb{N} \bullet (j, \textit{init?})\}(i)).\textit{case}(s_1) = (\{j : \mathbb{N} \bullet (j, \textit{init?})\}(i)).\textit{case}(s_2) \Rightarrow s_1 = s_2$$

est prouvable, puisque cette dernière expression est équivalente à :

$$\forall i : \mathbb{N} \bullet \forall s_1, s_2 : \textit{Sites} \bullet \textit{init?}.\textit{case}(s_1) = \textit{init?}.\textit{case}(s_2) \Rightarrow s_1 = s_2$$

(exercice : essayez de le démontrer en utilisant les règles de preuve du cours) et que, par des règles de preuve non mentionnées dans le cours (exercice : exprimez-les), cette dernière propriété est une conséquence de la propriété :

$$\forall s_1, s_2 : \textit{Sites} \bullet \textit{case}(s_1) = \textit{case}(s_2) \Rightarrow s_1 = s_2$$

mentionnée dans le schéma *Initial*. D'autre part, on vérifie que :

$$\{j : \mathbb{N} \bullet (j, \textit{init?})\} \in \mathbb{N} \rightarrow \textit{Config} \tag{3}$$

(Exercice : démontrez-le en utilisant les règles de preuve du cours.) De (2) et (3) on déduit :

$$\exists \textit{trace!} : \mathbb{N} \rightarrow \textit{Config} \bullet \textit{Coherence1}$$

par la règle $[\exists I]$ du cours. De par la définition (1), ceci prouve pre *Coherence*1 — rappelons-le, sous l'hypothèse que $\textit{init?} \in \textit{Initial}$.

De même, pre *ATMRTraces* est prouvable sous cette même hypothèse. Intuitivement, *ATMRTraces* donne une façon constructive de fabriquer *trace*!, partant de *init*? comme configuration initiale pour $\textit{trace!}(0)$ et construisant $\textit{trace!}(i)$ par récurrence sur i, en utilisant le fait que toute configuration a au moins une image par la relation de transition. (Exercice : essayez de prouver l'existence de *trace*! de cette façon; il s'agit d'un problème plus difficile qu'il n'en a l'air.) Donc pre *ATMRTraces* est aussi identiquement vrai dès que $\textit{init?} \in \textit{Initial}$.

Sous l'hypothèse $\textit{init?} \in \textit{Initial}$, $\mathsf{pre}\ \textit{Coherence1} \Rightarrow \mathsf{pre}\ \textit{ATMRTraces}$ est donc prouvable. On en déduit, par la règle $[\forall I]$ du cours, que (D) est prouvable.

Quant à la condition de raffinement proprement dite (R), sachant que sous l'hypothèse $\textit{init?} \in \textit{Initial}$, pre *Coherence*1 est toujours vraie, elle est équivalente à :

$$(R_1) \quad \forall \textit{init?} : \textit{Initial};\ \textit{trace!} : \mathbb{N} \rightarrow \textit{Config} \bullet \textit{ATMRTraces} \Rightarrow \textit{Coherence1}$$

qui signifie que toute trace engendrée par une exécution du protocole obéit au critère de cohérence *Coherence*1 — ce qui était la condition que nous souhaitions écrire depuis le début.

Nous aurions pu écrire la propriété (R_1) directement, et nous lancer dans sa preuve. Le but du développement ci-dessus était de montrer que (R_1) n'est pas n'importe quelle propriété : il s'agit d'une propriété de raffinement telle que définie dans le cours.

Maintenant, (R_1) peut être prouvée comme suit. Informellement, nous posons les hypothèses $init? \in Initial$, $trace! \in \mathbb{N} \rightarrow Config$ et supposons que la contrainte de $ATMRTraces$ est vérifiée, et nous démontrons que la contrainte de $Coherence1$ est vraie (qui est une propriété de la forme $\forall i : \mathbb{N} \bullet \ldots$), par récurrence sur le temps i (l'axiome $[\mathbb{N}Ind._1]$ du cours). La propriété est vraie au temps 0, parce que $trace!(0) = init?$ (car $ATMRTraces$ est vérifiée) et parce que $init?$ vérifie la contrainte de $Initial$. Supposant la propriété vraie au temps i, nous la montrons pour le temps $i+1$. il y a trois cas à considérer : en phase de réception, le schéma $TraitementsReception$ spécifie $case' = case$, ce qui implique que $(trace?(i+1)).case = (trace?(i)).case$; comme $(trace?(i)).case(s_1) = (trace?(i)).case(s_2) \Rightarrow s_1 \Rightarrow s_2$ par hypothèse de récurrence pour tous $s_1, s_2 \in Sites$, il s'ensuit que $(trace?(i+1)).case(s_1) = (trace?(i+1)).case(s_2) \Rightarrow s_1 \Rightarrow s_2$ pour tous $s_1, s_2 \in Sites$ (nous utilisons l'axiome de remplacement des égaux $[= Subst]$ du cours). En phase d'émission, le schéma $TraitementsEmission$ spécifie aussi $case' = case$, et le raisonnement est le même. Et en phase de rotation, le schéma $TraitementsRotation$ spécifie $case' = \{s : Sites \bullet (s, (case(s) \bmod case\_max) + 1)\}$, d'où l'on déduit $(trace?(i+1)).case = \{s : Sites \bullet (s, ((trace?(i)).case(s) \bmod case\_max) + 1)\}$. Donc si $(trace?(i+1)).case(s_1) = (trace?(i+1)).case(s_2)$, alors $((trace?(i)).case(s_1) \bmod case\_max) + 1 = ((trace?(i)).case(s_2) \bmod case\_max) + 1$, donc $(trace?(i)).case(s_1) \bmod case\_max = (trace?(i)).case(s_2) \bmod case\_max$; mais comme $trace?(i).case(s_1) \in Cases$, $trace?(i).case(s_1) \in 1..case\_max$ par définition, donc $trace?(i).case(s_1) \in 1..case\_max-1$ ou $trace?(i).case(s_1) = case\_max$; dans le premier cas $trace?(i).case(s_1) \bmod case\_max = trace?(i).case(s_1)$, dans le deuxième $trace?(i).case(s_1) = 0$, et de même avec s_2. On a donc quatre cas :

$$\begin{array}{ll}
(1) & trace?(i).case(s_1) \in 1\,..\,case_max - 1,\ trace?(i).case(s_2) \in 1\,..\,case_max - 1 \\
(2) & trace?(i).case(s_1) \in 1\,..\,case_max - 1,\ trace?(i).case(s_2) = 0 \\
(3) & trace?(i).case(s_1) = 0,\ trace?(i).case(s_2) \in 1\,..\,case_max - 1 \\
(4) & trace?(i).case(s_1) = 0,\ trace?(i).case(s_2) = 0
\end{array}$$

Dans les cas (2) et (3), on obtient une contradiction avec le fait que $((trace?(i)).case(s_1) \bmod case\_max) + 1 = ((trace?(i)).case(s_2) \bmod case\_max) + 1$ (on en déduit le faux, donc la conclusion désirée $s_1 = s_2$). Dans les cas (1) et (4), on a $trace?(i).case(s_1) = trace?(i).case(s_2)$, donc par hypothèse de récurrence $s_1 = s_2$, ce qui finit de prouver (R_1). (La preuve formelle est évidemment bien plus longue, mais l'argument présenté ici en est le squelette.)

Remarquons aussi que nous n'avons jamais eu à démontrer qu'il existait une configuration initiale. (Exercice : en chercher une preuve informelle. Noter que l'axiome $case\_max \geq site\_max$ est indispensable. En fait, ceci n'est pas prouvable tel quel : quel axiome manque-t-il ?)

3.2 Correction du protocole

La principale propriété que l'on attend du protocole est le fait que tout message en attente d'envoi sur un site finisse par arriver, intact, à son destinataire correct. Ceci recouvre plusieurs propriétés, une de *sûreté* ("safety" en anglais) :

- Toute case contenant un message voit son message et son adresse de destination DA inchangée tant qu'elle n'a pas atteint sa destination. (Note : ceci ne spécifie pas que la case doive finir par atteindre sa destination.)

Et deux propriétés de *progrès* ("liveness" en anglais) :

- Tout site qui possède un message en attente finira par remplir une case vide avec le bon message et la bonne adresse de destination. (Autrement dit, aucun site n'est bloqué par le trafic pour toujours.)
- Tout site est visité par toute case infiniment souvent. (Autrement dit, l'anneau de cases ne s'arrête pas de circuler indéfiniment longtemps.)

Voici la propriété globale de correction du système, exprimée comme une spécification abstraite :

$$\begin{array}{|l}
\multicolumn{1}{l}{ProtocoleCorrect} \\
\hline
init? : Initial \\
trace! : \mathbb{N} \rightarrow Config \\
\hline
\forall s : Sites;\ i : \mathbb{N} \bullet \\
\quad s \in (trace!(i)).attente \Rightarrow \\
\quad (\exists j : \mathbb{N} \mid j \geq i \bullet \\
\qquad (trace!(j)).recu((trace!(i)).dest(s)) = (trace!(i)).envoi(s)) \\
\hline
\end{array}$$

La propriété de sûreté ci-dessus s'écrit :

$$\begin{array}{|l}
\multicolumn{1}{l}{Surete1} \\
\hline
init? : Initial \\
trace! : \mathbb{N} \rightarrow Config \\
\hline
\forall c : Cases;\ s : Sites;\ m : Message;\ i : \mathbb{N} \bullet \\
\quad (trace!(i)).DA(c) = s \wedge (trace!(i)).msg(c) = m \wedge (trace!(i)).case(s) \neq c \Rightarrow \\
\quad (trace!(i+1)).DA(c) = s \wedge (trace!(i+1)).msg(c) = m \vee (trace!(i+1)).case(s) = c \\
\hline
\end{array}$$

La première propriété de progrès est :

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad Progres1 \quad}} \\
init? : Initial \\
trace! : \mathbb{N} \rightarrow Config \\
\hline
\forall s : Sites;\ i : \mathbb{N} \bullet \\
\quad s \in (trace!(i)).attente \Rightarrow \\
\quad (\exists j : \mathbb{N};\ c : Cases \mid j \geq i \bullet \\
\quad\quad (trace!(j)).DA(c) = (trace!(i)).dest(s) \wedge (trace!(j)).msg(c) = (trace!(i)).envoi(s)) \\
\hline
\end{array}
$$

Et la deuxième propriété de progrès est :

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad Progres2 \quad}} \\
init? : Initial \\
trace! : \mathbb{N} \rightarrow Config \\
\hline
\forall s : Sites;\ c : Cases;\ i : \mathbb{N} \bullet \\
\quad \exists j : \mathbb{N} \mid j \geq i \bullet \\
\quad\quad (trace!(j)).case(s) = c \\
\hline
\end{array}
$$

Nous devons prouver que $ATMRTraces$ raffine $ProtocoleCorrect$. La relation entre la correction du protocole et les propriétés de sûreté et de progrès ci-dessus indique qu'il suffira de vérifier que $ATMRTraces$ raffine $Surete1$, $Progres1$ et $Progres2$, et que la conjonction $Surete1 \wedge Progres \wedge Progres2$ elle-même raffine $ProtocoleCorrect$.

La raison pour laquelle $Surete1 \wedge Progres \wedge Progres2$ raffine $ProtocoleCorrect$ est intuitivement claire. De plus, $ATMRTraces$ raffine $Surete1$ parce qu'une case ayant un champ DA non nul ne peut être modifié que par le site spécifié dans ce champ DA. Ces arguments peuvent être transformées en preuves informelles dans le style de celle de la section 3.1, et de là en preuves formelles.

Le fait que $ATMRTraces$ raffine $Progres2$, autrement dit que toute case visite tout site infiniment souvent, est intuitivement aisé, mais déjà plus difficile à justifier. Remarquons seulement que la raison pour laquelle c'est vrai est qu'en fait chaque case progresse d'un site à chaque rotation, et qu'il y a une rotation tous les 3 instants. On peut donc montrer que $ATMRTraces$ raffine une spécification $Progres3$ elle-même plus concrète que $Progres2$, et qui dit que tout site est visité par toute case *avant* $site\_max$ *rotations*, soit au plus $3 * site\_max + 2$ instants.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad Progres3 \quad}} \\
init? : Initial \\
trace! : \mathbb{N} \rightarrow Config \\
\hline
\forall s : Sites;\ c : Cases;\ i : \mathbb{N} \bullet \\
\quad \exists j : \mathbb{N} \mid j \geq i \wedge j \leq i + 3 * site\_max + 2 \bullet \\
\quad\quad (trace!(j)).case(s) = c \\
\hline
\end{array}
$$

Pour montrer que $ATMRTraces$ raffine $Progres3$, on peut définir une distance $dist(i, c, s)$ entre une case c et un site s à l'instant i qui soit 1 si la case c est juste avant le site s, 2 si c est à deux rotations du site s, ..., $site\_max$ si c vient juste de quitter s; cette distance nous permet de prédire le temps d'arrivée d'un message : si au temps i, le message est à distance d de son site de destination, il arrivera au temps $3 * (d + (i \operatorname{div} 3))$. Nous faisons ensuite comme pour $ProtocoleCorrect$ et décomposons $Progres3$ en une propriété de sûreté $Surete2$ qui exprime que la distance est toujours entre 1 et $site\_max$ (un *invariant* de l'exécution), et une propriété de progrès $Progres4$ qui exprime que si la distance entre c et s est d au temps i, alors la case c atteint s au bout de d rotations exactement.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad Distance \quad}} \\
trace! : \mathbb{N} \rightarrow Config \\
dist : \mathbb{N} \times Cases \times Sites \rightarrow \mathbb{N} \\
arrivee : \mathbb{N} \times Cases \times Sites \rightarrow \mathbb{N} \\
\hline
dist = \{i : \mathbb{N};\ c : Cases;\ s : Sites \bullet \\
\quad ((i, c, s), ((c - (trace!(i)).case(s) + site\_max - 1) \operatorname{mod} site\_max) + 1)\} \\
arrivee = \{i : \mathbb{N};\ c : Cases;\ s : Sites \bullet \\
\quad ((i, c, s), 3 * (dist(i, c, s) + i \operatorname{div} 3))\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Surete2} \\
\textit{Distance} \\
\hline
\mathsf{ran}\ dist \subseteq 1 \mathrel{..} site\_max \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Progres4} \\
\textit{Distance} \\
\hline
\forall c : Cases;\ s : Sites;\ i : \mathbb{N} \bullet \\
\quad (trace!(arrivee(i, c, s))).case(s) = c \\
\hline
\end{array}
$$

Le fait que $ATMRTraces$ raffine $Progres1$, autrement dit que tout site en attente d'envoi d'un message finit par le déposer dans une case, est plus compliqué. Il se décompose en deux propriétés de progrès. La première dit que si un site s a un crédit strictement positif, il finira par émettre un message :

$$
\begin{array}{|l}
\hline
\textit{Progres5} \\
init? : Initial \\
trace! : \mathbb{N} \rightarrow Config \\
\hline
\forall s : Sites;\ i : \mathbb{N} \bullet \\
\quad s \in (trace!(i)).attente \wedge (trace!(i)).credit(s) > 0 \Rightarrow \\
\quad (\exists j : \mathbb{N};\ c : Cases \mid j \geq i \bullet \\
\qquad (trace!(j)).DA(c) = (trace!(i)).dest(s) \wedge (trace!(j)).msg(c) = (trace!(i)).envoi(s)) \\
\hline
\end{array}
$$

La deuxième dit que si tout site finit toujours par réobtenir des crédits :

$$
\begin{array}{|l}
\hline
\textit{Progres6} \\
init? : Initial \\
trace! : \mathbb{N} \rightarrow Config \\
\hline
\forall s : Sites;\ i : \mathbb{N} \bullet \\
\quad s \in (trace!(i)).attente \wedge (trace!(i)).credit(s) = 0 \Rightarrow \\
\quad (\exists j : \mathbb{N} \mid j \geq i \bullet (trace!(j)).credit(s) > 0) \\
\hline
\end{array}
$$

Il serait trop long de décrire encore comment prouver ces derniers schémas pour montrer qu'ils se raffinent en $ATMRTraces$. La méthode de [CP97] consiste à montrer que tout site actif finit par devenir passif, ce qui entraînera que $ATMRTraces$ raffine $Progres5$. Pour cela, on suppose qu'il existe un site restant actif de l'instant i à la fin des temps, et on montre : (1) tout site actif finit par ne plus générer de cases reset, (2) toutes les cases reset qui circulent finissent par disparaître, (3) les cases pleines finissent par être vidées, et (4) les cases vides finissent par êtres remplies, d'où l'on déduit que la somme des crédits de chaque site finit toujours par décroître à partir de chaque instant i. Donc cette somme décroît une infinité de fois, ce qui est impossible. (Cette impossibilité est en fait une conséquence de l'axiome de récurrence sur les entiers.) Le fait que $ATMRTraces$ raffine $Progres6$ se montre en prouvant que : (1) une case reset finit par être fabriquée, et (2) si une case reset est fabriquée, alors les crédits de tous les sites finiront par acquérir une valeur strictement positive. Nous laissons tous ces développements à faire au lecteur intéressé ... et courageux.

Finalement, les relations entre les diverses spécifications à prouver sont décrites en figure 2; les flèches $\longrightarrow$ dénotent les relations "est raffiné par" à prouver.

References

[CP97] Michel Charpentier and Gérard Padiou. A UNITY-based model and proof of the ATMR protocol. In *Soumis à FMPPTA'97*, 1997.

[IIKM94] K. Imai, T. Ito, H. Kasahara, and N. Morita. ATMR: Asynchronous transfer mode ring protocol. *Computer Networks and ISDN Systems*, 26:785–798, 1994.

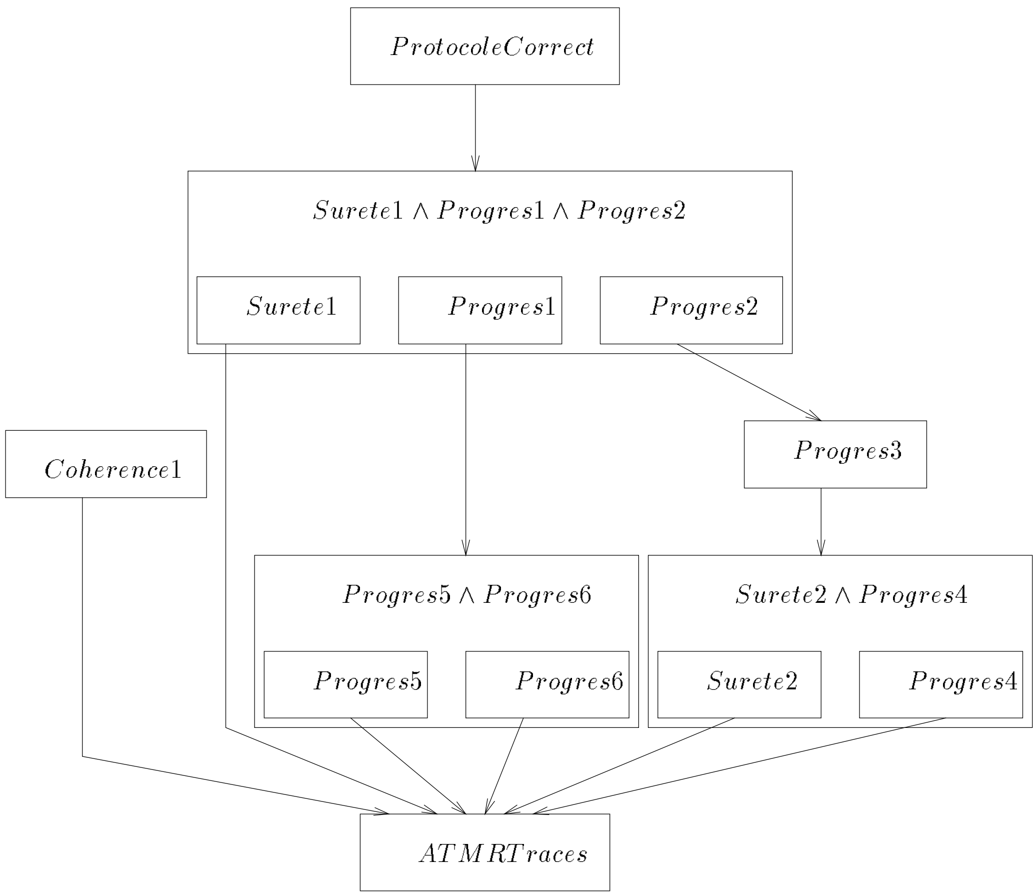

Figure 2: Relations de raffinement à prouver