Numéro d'ordre 2083
Année 1998

Mémoire de D.E.A.
d'Informatique, Automatique et Productique

Classification de propriétés dynamiques et de leurs raffinements à partir de quelques études de cas

Par Stéphane Galland

Stage encadré par le Professeur Jacques Julliand

Présenté le	17/07/98
Jury :	
Prof. J. Julliand	Laboratoire d'Informatique de Besançon
Prof. F. Bellegarde	Laboratoire d'Informatique de Besançon
Prof. B. Legeard	Laboratoire d'Informatique de Besançon
Mdc H. Mountassir	Laboratoire d'Informatique de Besançon
Mdc C. Varnier	Laboratoire d'Automatique de Besançon

Table des matières

Remerciements

Je remercie en premier lieu Monsieur Jacques Julliand, mon responsable de stage, pour le temps qu'il m'a consacré durant ce stage et pour ses précieux conseils dans l'orientation de mes travaux et dans la conception de ce rapport.

Je tiens également à remercier F. Bellegarde, B. Legeard, H. Mountassir et C. Varnier pour l'intérêt qu'ils portent à ce mémoire et pour avoir bien voulu participer à ce jury.

Je remercie Benoit Parreaux et Bruno Tatibouët pour les informations qu'ils ont eu la gentillesse de me donner.

Je remercie également tous les membres du laboratoire qui m'ont acceuilli avec bonne humeur et plus particulièrement Christophe Darlot et Jean-Yves Pierron, membres du D.E.A. option 4, qui m'ont aidé à mettre une ambience bon enfant dans notre bureau.

J'aimerais aussi remercier mon entourage (Laurent, Laly, ...) pour son soutien durant cette période.

Ces remerciements ne seraient pas complets sans un petit mot pour Jean-Michel Hufflen qui m'a donné la possibilité de faire connaissance avec LaTeX.

Chapitre 1

Un problème et sa spécification en B étendu par de la Logique Temporelle Linéaire

Ce rapport met l'accent sur une méthode de construction et de vérification de systèmes réactifs concurrents. Le premier objectif des travaux présentés est de définir une extension de la méthode de spécification B afin de pouvoir exprimer et vérifier des propriétés dynamiques[1]. Le second objectif est de mesurer l'impact du raffinement sur cette extension. Le raffinement permet de diminuer la complexité de vérification par "model checking" grâce à une vérification modulaire et à une vérification par raffinement des propriétés dynamiques. Le dernier objectif de ce rapport est de présenter un algorithme permettant de vérifier qu'il y a un raffinement entre deux graphes d'accessibilité. Ces derniers représentent sous forme graphique l'ensemble des états atteignables par le système ainsi que les transitions pouvant y mener.

Concernant le premier objectif, la méthode B, telle qu'elle est décrite dans [Abr96, Abr96a] consiste à spécifier des propriétés invariantes. Celles-ci permettent de spécifier des propriétés de sûreté de fonctionnement des systèmes en limitant l'espace des états atteignables. C'est insuffisant pour spécifier les propriétés dynamiques des systèmes qui sont souvent prédominantes dans le cas de systèmes réactifs. Spécifier des propriétés dynamiques consiste à limiter les transitions entre les états atteignables. Ces propriétés sont complémentaires des propriétés de sûreté et ne peuvent pas s'exprimer par des invariants. Nous proposons de les exprimer en Logique Temporelle Linéaire (LTL) [Man92, MP96]. D'autres approches temporelles comme les logiques temporelles d'actions [Lam91, Mer96, CWB94], ou les logiques temporelles arborescentes [EC82] permettent également d'atteindre ce but.

L'extension de la méthode B présentée dans [AM98] propose d'exprimer les propriétés dynamiques sous deux formes, des invariants dynamiques qui expriment des propriétés sur les transitions c'est-à-dire entre deux états successifs, et une généralisation appelée modalité qui exprime des propriétés sur des séquences de transitions. Du point de vue de la puissance d'expression, cette approche est une restriction de la Logique Temporelle Linéaire. Les invariants dynamiques permettent d'exprimer l'opérateur *Next*($\bigcirc$). Bien que limité à deux schémas, les modalités *Leadsto*($\leadsto$) et *Until*($\mathcal{U}$) permettent d'exprimer deux compositions des opérateurs *Always*($\Box$), *Eventually*($\Diamond$) et *Until* couramment rencontrées: $\Box\,(\ p \Rightarrow \Diamond q\ )$ et $\Box\,(\ p \Rightarrow p'\,\mathcal{U}\,q\ )$. Du point de vue de la facilité d'expression l'approche B, fondée sur une technique de vérification par preuve, oblige l'utilisateur à décrire des *variants* et des *invariants* de boucle qui sont nécessaires pour effectuer les preuves de terminaison des comportements itératifs. Par contre *variants* et *invariants* sont inutiles pour effectuer la vérification des propriétés de Logique Temporelle Linéraire par une méthode de "model checking" [GH93, GPVW95].

[1] aussi appelées propriétés de sûreté et de vivacité (ou de progrès)

Du point de vue de la technique de vérification l'approche preuve de type B permet d'adresser des systèmes infinis mais elle a recours à l'aide de l'utilisateur. De plus elle peut carrément échouer compte tenu de l'incomplétude de l'automatisation liée à l'indécibilité des logiques. L'approche par "model checking" de type SPIN [Hola] est limitée aux systèmes finis et bute sur les limites en temps et en espace dues à l'explosion combinatoire du nombre d'états du modèle. Par contre l'automatisation du "model checking" est complète et présente l'intérêt d'exhiber des scénarios d'exécution violant les propriétés, ce qui est très intéressant pour la mise au point des modèles.

Concernant le second objectif, l'approche de spécification de propriétés dynamiques en LTL et de leurs vérifications par "model checking" n'est pas nouvelle. Par exemple les systèmes SPIN [Holb] ou STeP [MP96] proposent cette approche. Beaucoup d'autres systèmes effectuent des vérifications par "model checking". C'est le cas de systèmes fondés sur les langages LDS, Estelle, ou sur les systèmes de transitions comme Mec [Arn95]. La force de ces approches est d'être à un niveau d'expression proche des langages d'implantation. Mais c'est aussi leur faiblesse du point de vue de la complexité algorithmique de la vérification. L'originalité de notre approche est de se situer dans le contexte d'un langage ensembliste de haut niveau et d'être dans le cadre d'une méthode supportant le raffinement. Ce dernier point ouvre des perspectives intéressantes pour faire décroître la complexité de la vérification. D'autre part la préservation des propriétés du modèle abstrait par raffinement la réduit car la vérification sur un modèle abstrait est de plus faible complexité que sur un modèle détaillé. D'autre part la complexité diminue en vérifiant des propriétés par morceaux sur le graphe grâce à la modularité induite par le raffinement qui permet de décomposer le graphe d'accessibilité. Ici nous nous intéresserons plus particulièrements aux différentes formes des propriétés dynamiques et de leurs raffinements. Ceci nous procure une base intéressante concernant le développement de la méthode d'analyse au niveau humain. En effet, un nombre limité de formes temporelles pourrait être avantageux pour le prouveur mais à contrario elle nécessitera un effort suplémentaire pour l'analyste dans l'écriture des propriétés dynamiques.

Le dernier objectif de ce rapport s'inscrit dans le cadre du développement d'une méthode de vérification modulaire par "model checking". Cet algorithme permet d'avoir une base intéressante pour l'implémentation de la vérification du raffinement à l'aide d'une méthode de type "model checking".

Au chapitre 2 nous présentons un exemple de spécification d'un protocole de transfert de fichier. Chapitres 3 et 4 sont décrites les formes de propriétés dynamiques rencontrés dans les études de cas analysées, ainsi que les différentes possibilités de raffinement utilisées. Aux chapitres 5, 6 et 7 est présenté l'algorithme de détection de raffinement de graphes accompagné d'un exemple d'éxécution. Enfin nous décrirons les résultats de ce travail ainsi que les perspectives envisagées.

Partie I

Classification des propriétés temporelles

Chapitre 2

L'exemple du protocole BRP: spécification par raffinement

Durant ce chapitre, nous allons nous attacher à la spécification d'un protocole de communication : le BRP[1]. Cette étude est basée sur celle de [AM97]. Nous y ajoutons des formules temporelles pour spécifier les propriétés dynamiques.

2.1 Présentation informelle du BRP

Le protocole BRP représenté par la figure 2.1 permet de copier un fichier d'un site vers un autre que nous appelons l'émetteur et le récepteur.

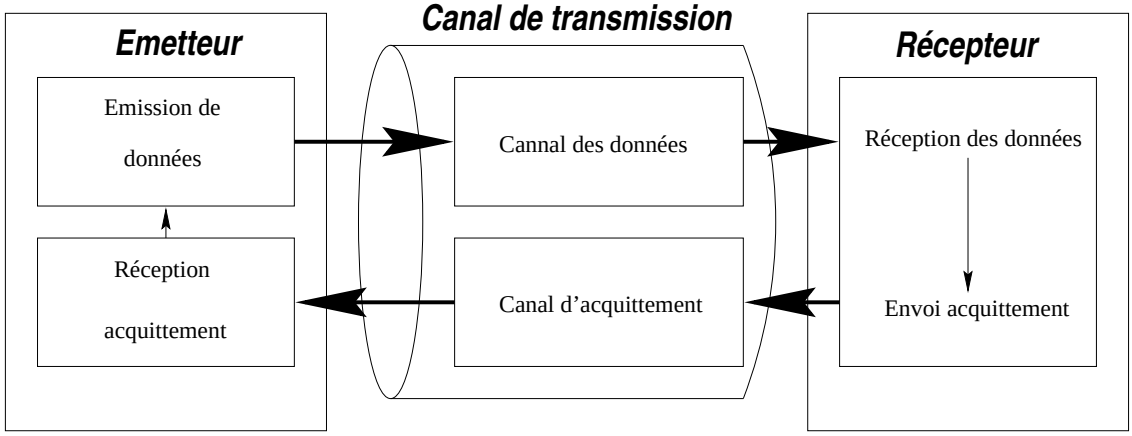


Figure 2.1: Schéma de fonctionnement du protocole BRP

Le fichier est tranféré morceau par morceau. L'émetteur envoie une donnée sur le support de communication (canal de transmission). Lorsque le récepteur reçoit une donnée en provenance du canal de transmission, il la stocke dans un fichier et envoie un acquittement vers l'émetteur. Une fois que ce dernier a reçu une confirmation du transfert, il envoie le paquet de données suivant. On suppose que les paquets transitant par le canal possèdent une information particulière permettant au récepteur de savoir s'il s'agit des dernières données du fichier. Lorsque c'est le cas, le récepteur envoie quand même un acquittement.

[1] Bounded Retransmission Protocol

Nous venons de décrire le fonctionnement normal du protocole c'est à dire lorsque le fichier est tranféré totalement de l'émetteur vers le récepteur. Nous allons maintenant nous attacher à décrire le fonctionnement du protocole lorsqu'un problème apparait sur le canal de transmissiom.

Erreur de communication

Le canal de transmission situé entre l'émetteur et le récepteur peut avoir une défaillance, c'est à dire que des données envoyées par l'émetteur ou un acquittement provenant du récepteur peuvent être perdus. Pour résoudre ce problème, l'émetteur démarre un mécanisme qui aura pour rôle de le réveiller lorsqu'un délai D_{max} se sera écoulé et qu'il n'aura pas reçu d'acquittement pour le paquet de données envoyé. Nous considérerons que le délai sera toujours plus grand que le temps maximal nécessaire pour que les données arrivent chez le récepteur, que ce dernier les traite et qu'il renvoie un acquittement. L'émetteur conclura donc que son message s'est définitivement perdu lorsqu'il a attendu pendant un temps D_{max}. Bien sûr, l'émetteur ne peut savoir si l'erreur de transmission est due au canal de transmission des données ou à celui de l'acquittement. Pour résoudre ce problème, l'émetteur renvoie le message et attend l'acquittement correspondant.

Arrêt anormal du protocole

Lorsqu'il y a plusieurs pertes de messages, l'émetteur retransmet ces derniers N_{max} fois. Si le nombre de retransmission d'un message atteint N_{max}, l'émetteur considère que la transmission est définitivement stoppée. Il reste toutefois un petit problème concernant le récepteur car celui-ci est toujours actif lorsque l'émetteur s'arrête. Pour répondre à cette difficulté, il faut ajouter un compteur sur le site récepteur qui le réveillera s'il n'a pas reçu de nouveau message (message non re-transmis) au bout d'un temps supérieur à $N_{max} \times D_{max}$. Si le récepteur se réveille à cause de ce compteur, il considère que la transmission est terminée. Cette méthode permet d'avoir une synchronisation indirecte entre les deux sites c'est à dire que lorsque l'un d'entre eux se désactive, l'autre se désactivera immédiatement après.

Bit d'alternance

Un autre problème se pose lors des re-transmissions. En effet, l'émetteur peut envoyer le message deux fois de suite mais aussi deux messages différents ayant la même valeur. Cette situation ne permet pas au récepteur de savoir si un message est une retransmission ou non. La solution à ce petit problème est l'introduction d'un bit dont la valeur est alternée pour chaque nouveau message à envoyer. Le récepteur saura dès lors qu'un message est une re-transmission si son bit d'alternance est identique à celui du message précédent.

Situation finale du protocole

Lorsque le protocole s'arrête, il peut être dans trois états différents :

1. Le fichier a été entièrement transféré du site émetteur vers le site récepteur et le premier a reçu l'acquittement correspondant au dernier message.

2. Le fichier a été entièrement transféré du site émetteur vers le site récepteur mais le premier n'a pas reçu le message d'acquittement provenant du récepteur. Le site émetteur s'arrête sur une erreur de transmission alors que le site récepteur s'est déjà arrêté en considérant que le fichier a été entièrement transféré.

3. Les deux sites s'arrêtent sur une erreur.

Principes de fonctionnement

Le protocole BRP respecte les principes de fonctionnement suivants :

a) Le protocole est un protocole de transfert de fichiers. Il doit copier partiellement ou totalement des données d'un site vers un autre,

b) A la fin du protocole, les deux sites savent si le transfert est terminé,

c) Si l'émetteur est inactif alors le récepteur deviendra fatalement inactif. Si le récepteur est inactif alors l'émetteur deviendra fatalement inactif,

d) Un site peut se désactiver quand :

- le protocole s'est terminé correctement,
- le procotole s'est mal terminé;

e) Quand l'expéditeur sait que le protocole s'est bien terminé alors le récepteur le sait déjà. Quand le récepteur sait que le protocole s'est mal terminé alors l'émetteur le sait déjà. Ce principe n'est pas décrit explicitement dans le cahier des charges mais apparait après une première réflexion sur le comportement du protocole,

f) Il est possible que l'expéditeur croie que le protocole s'est mal terminé alors que le récepteur sait que la copie s'est effectuée entièrement (perte du dernier acquittement),

g) Le récepteur croit que le protocole s'est bien terminé lorsque le fichier source a été entièrement copié. L'émetteur croit que le protocole s'est mal terminé lorsque le fichier source n'a certainement pas été entièrement copié ou que le dernier acquittement ne lui est pas parvenu,

h) Lorsque le protocole a terminé de copier un fichier, il attend une nouvelle demande de transfert. Ce principe n'est pas directement issu du cahier des charges mais d'un choix de modélisation.

2.2 Spécifications descriptives et opérationnelles

2.2.1 Spécification formelle

But

On ne considère dans cette première spécification que l'initialisation du transfert ainsi que sa terminaison. Le fichier sera copié instantanément en une seule fois. Le support de communication n'est pas pris en compte.

Ce niveau de spécification possède deux événements : $InitTransfert_0$ et $ArrêtTransfert_0$. Le premier correspond à l'initialisation du transfert et à l'activation des deux sites. Le second se produit lorsque le protocole s'arrête.

Données

On introduit les variables suivantes :

- $fich\_emet_0$: fichier source lu par l'émetteur. Ce fichier est modélisé par une séquence.
- $fich\_recep_0$: fichier destination écrit par le récepteur. Ce fichier est aussi modélisé par une séquence.
- $emet\_actif_0$: état d'activation de l'émetteur. Cette variable permet de savoir si un transfert est en cours de traitement par l'émetteur.
- $recep\_actif_0$: état d'activation du récepteur. Cette variable permet de savoir si un transfert est en cours de traitement par le récepteur.
- $emet\_transfert\_ok_0$: état du transfert pour l'émetteur. Cette variable indique si l'émetteur croit que le transfert s'est bien terminé.
- $recep\_transfert\_ok_0$: état du transfert pour le récepteur. Cette variable indique si le récepteur croit que le transfert s'est bien terminé.

Invariant

– – **Typage des variables, soit S le type des éléments des fichiers**
$fich\_emet_0 \in \mathrm{seq}_1(\mathrm{S}) \wedge$
$fich\_recep_0 \in \mathrm{seq}(\mathrm{S}) \wedge$
$emet\_actif_0 \in \{oui, non\} \wedge$
$recep\_actif_0 \in \{oui, non\} \wedge$
$emet\_transfert\_ok_0 \in \{oui, non\} \wedge$
$recep\_transfert\_ok_0 \in \{oui, non\} \wedge$

– – ***Principe de fonctionnement a:*** **Le protocole transférant un fichier, le fichier destination est un préfixe du fichier source**
$fich\_recep_0 \subseteq fich\_emet_0 \wedge$

– – ***Principe de fonctionnement b:*** **Tant qu'un site est actif il ne peut pas savoir si le transfert s'est bien terminé.**
$emet\_actif_0 = oui \Leftrightarrow emet\_transfert\_ok_0 = non \wedge$
$recep\_actif_0 = oui \Leftrightarrow recep\_transfert\_ok_0 = non$

– – ***Principe de fonctionnement e:*** **Lorsque l'émetteur sait que le protocole s'est bien terminé, le récepteur le sait déjà**
$(\ emet\_actif_0 = non \wedge emet\_transfert\_ok_0 = oui \ )$
$\Rightarrow (\ recep\_actif_0 = non \wedge recep\_transfert\_ok_0 = oui \ ) \wedge$

– – ***Principe de fonctionnement e:*** **Lorsque le récepteur sait que le protocole s'est mal terminé, l'émetteur le sait déjà**
$(\ recep\_actif_0 = non \wedge recep\_transfert\_ok_0 = non \ )$
$\Rightarrow (\ emet\_actif_0 = non \wedge emet\_transfert\_ok_0 = non \ )$

– – **Comme le fichier est transféré instantanément, le fichier destination est vide tant que les deux sites sont actifs. Cet invariant n'exprime pas directement un principe de fonctionnement et ne sera plus vrai dans les raffinements suivants**
$(\ emet\_actif_0 = oui \wedge recep\_actif_0 = oui \ ) \Rightarrow fich\_recep_0 = [\,] \wedge$

– – ***Principe de fonctionnement g:*** **On considère que le protocole s'est bien terminé lorsque le récepteur est inactif et que les deux fichiers sont identiques**
$recep\_actif_0 = non \Rightarrow (\ recep\_transfert\_ok_0 = oui \Leftrightarrow fich\_emet_0 = fich\_recep_0 \ ) \wedge$

Propriétés temporelles

– – **Le protocole est modélisé comme un démon. Il attend une nouvelle demande de transfert après qu'il ait terminé de copier le fichier**
$${}_hB_0^1 \quad \Box \left(\begin{array}{l} emet\_actif_0 = non \wedge recep\_actif_0 = non \\ \Rightarrow \bigcirc (\ emet\_actif_0 = oui \wedge recep\_actif_0 = oui \ ) \end{array} \right)$$

– – **Le fichier est transféré instantanément. Les deux sites se désactivent immédiatement. Ces propriétés n'expriment que partiellement le principe de fonctionnement c. A ce niveau d'abstraction, les deux sites s'arrêtent simultanément.**
$${}_cB_0^2 \quad \Box (\ emet\_actif_0 = oui \Rightarrow \bigcirc (\ emet\_actif_0 = non \ ) \ )$$
$${}_cB_0^3 \quad \Box (\ recep\_actif_0 = oui \Rightarrow \bigcirc (\ recep\_actif_0 = non \ ) \ )$$

– – **Lorsque le protocole s'arrête, il peut être dans l'un des trois états cités page 14. Cette propriété dynamique exprime le principe de fonctionnement f et la section situation finale. Le principe de fonctionnement d est modélisé aussi par B_0^4.**

$$_{1,2,3,f,d}B_0^4 \quad \Box \left(\begin{array}{l} emet\_actif_0 = oui \wedge recep\_actif_0 = oui \wedge \\ \bigcirc \left(\begin{array}{l} emet\_actif_0 = non \wedge recep\_actif_0 = non \end{array} \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} emet\_transfert\_ok_0 = oui \wedge \\ recep\_transfert\_ok_0 = oui \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_0 = non \wedge \\ recep\_transfert\_ok_0 = non \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_0 = non \wedge \\ recep\_transfert\_ok_0 = oui \end{array} \right) \end{array} \right) \end{array} \right)$$

Evénements

NOTATION UTILISÉE POUR LES GRAPHES D'ACCESSIBILITÉ

Afin de permettre une meilleure compréhension des graphes d'accessibilité, une notation va être utilisée dans la suite de ce chapitre. Les valeurs des principales variables utilisées durant la spécification sont représentées à l'aide de pastilles à l'intérieur des états. La figure 2.2 montre un exemple de représentation d'état.

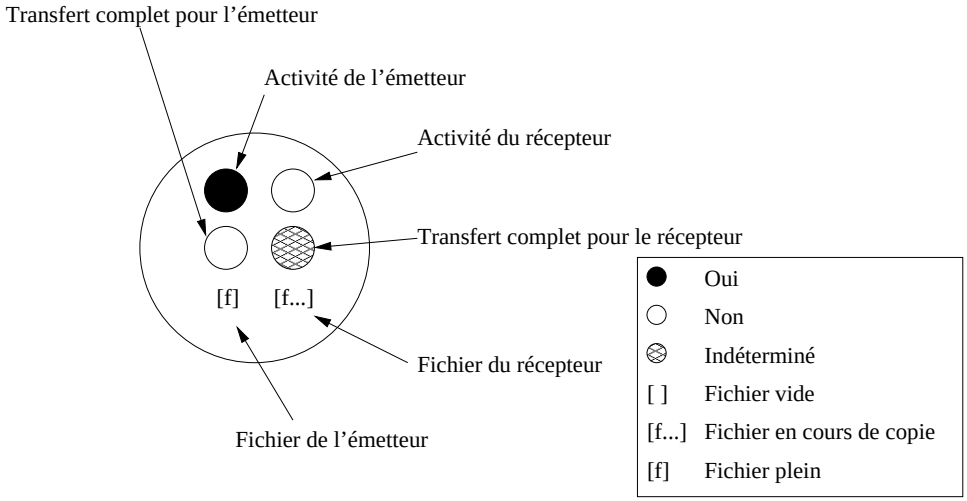


Figure 2.2: BRP - Description de la notation utilisée au niveau formel

Les états contiennent six variables dont voici la description:

- Activitié de l'émetteur (resp. du récepteur): cette pastille correspond à l'état de la variable $emet\_actif_0$ (resp. $recep\_actif_0$). Elle est noire si $emet\_actif_0 = oui$ (resp. $recep\_actif_0 = oui$) et blanche sinon.
- Transfert complet pour l'émetteur (resp. le récepteur): cette pastille représente la variable $emet\_transfert\_ok_0$ (resp. $recep\_transfert\_ok_0$). Elle est noire si la variable est égale à *oui* et blanche sinon. Elle peut aussi être hachurée dans le cas où la valeur de la variable est indéterminée.
- Fichier de l'émetteur (resp. du récepteur): cette partie représente l'état du fichier source (resp. destination). Elle peut prendre plusieurs valeurs:

- [ ] = le fichier est vide,
- $[f]$ = le fichier n'est pas vide (resp. est identique au fichier source),
- $[f...]$ = le fichier correspond à une partie du fichier à copier. A ce niveau d'abstraction seul le fichier du récepteur utilise cette valeur.

Les pastilles sont hachurées si la valeur de la variable correspondante est indéterminée, c'est à dire si elle n'a pas encore été initialisée. Les variables représentant les fichiers en sont un exemple. Ces dernières ne sont initialisées qu'au lancement du protocole (évenement $InitTransfert_0$) et non pas lors du démarrage du système (événement Initialisation).

Description des événements

Ce niveau possède deux événements: $InitTransfert_0$ et $ArrêtTransfert_0$. Le protocole est modélisé sous la forme d'un démon, c'est à dire que lorsqu'il a terminé de copier un fichier, il attend qu'une nouvelle demande de transfert lui parvienne. Le graphe d'accessibilité de la figure 2.3 représente les différentes possibilités d'évolution du système. La signification des notations utilisées à l'intérieur des états est décrite page 17.

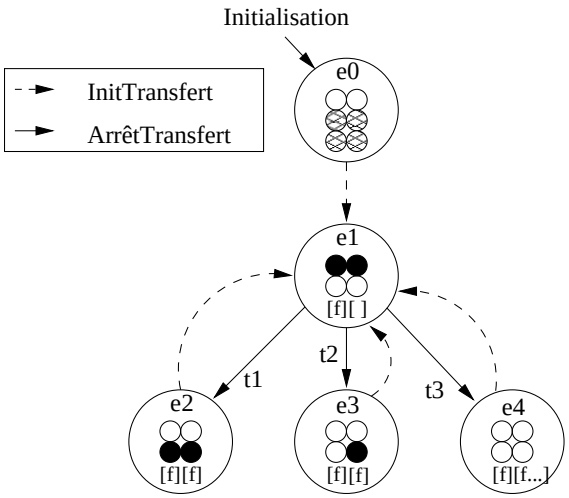


Figure 2.3: BRP - Graphe d'accessibilité du niveau formel

Trois chemins sont mis en évidence sur la figure 2.3 : $t1$, $t2$ et $t3$. Ils représentent les trois cas d'arrêt du transfert et seront utilisés par la suite pour montrer l'existence du raffinement des graphes d'accessibilité.

A l'initialisation du système, les deux sites sont inactifs. Il n'est pas nécessaire d'initialiser les valeurs des autres variables car cela sera réalisé par l'événement $InitTransfert_0$.

Initialisation $\triangleq$ $emet\_actif_0 := non \ \| \ recep\_actif_0 := non$

L'événement $InitTransfert_0$ se produit lorsqu'un transfert doit commencer. Les deux sites doivent donc s'activer et ne considèrent pas le transfert comme correctement terminé. Les fichiers source et destination sont initialisés à ce moment. Le premier ne peut pas être une séquence vide alors que le second en est une.

$InitTransfert_0 \triangleq$
SELECT
$emet\_actif_0 = non \wedge$
$recep\_actif_0 = non$
THEN
ANY f
WHERE
$f \in \mathrm{seq}_1(\mathrm{S})$
THEN
$fich\_emet_0 := f$
END ||
$fich\_recep_0 := [\,]$ ||
$emet\_actif_0 := oui$ ||
$recep\_actif_0 := oui$ ||
$emet\_transfert\_ok_0 := non$ ||
$recep\_transfert\_ok_0 := non$
END

L'événement $ArrêtTransfert_0$ se produit lorsque le protocole s'arrête. Il met à jour le fichier du récepteur en le copiant soit en partie soit en totalité. Ce choix est réalisé de manière non déterministe afin de modéliser simplement les résultats d'une erreur de transfert. Une fois cette tâche réalisée, les deux sites sont désactivés. Il faut toutefois constater qu'à ce moment l'état du transfert correspond à l'un des trois cas cités section 2.1 page 14 :

- le fichier est copié entièrement et les deux sites le savent,
- le fichier est copié entièrement mais seul l'émetteur n'a jamais reçu l'acquittement du dernier paquet,
- les deux sites se sont arrêtés en considérant que le fichier n'a pas été entièrement copié.

$ArrêtTransfert_0 \triangleq$
SELECT
$emet\_actif_0 = oui \wedge$
$recep\_actif_0 = oui$
THEN
ANY f
WHERE
$f \subseteq fich\_emet_0$
THEN
$fich\_recep_0 := f$
END;
IF $fich\_emet_0 = fich\_recep_0$
THEN
CHOICE
$emet\_transfert\_ok_0 := oui$
OR
$emet\_transfert\_ok_0 := non$
END ||
$recep\_transfert\_ok_0 := oui$
ELSE
$emet\_transfert\_ok_0 := non$ ||
$recep\_transfert\_ok_0 := non$
END
END

2.2.2 Raffinement n ° 1

But

Le but de ce raffinement est de modéliser le comportement du protocole de manière plus précise. Pour cela, on s'intéressera aux différentes possibilités d'arrêt des deux sites.

On raffinera l'événement $ArrêtTransfert_0$ en cinq événements :

- $ArrêtEmetteur_1$: événement représentant l'arrêt normal de l'émetteur, c'est à dire lorsqu'aucune erreur ne s'est produite.
- $ArrêtRécepteur_1$: Idem pour le récepteur.
- $ErreurEmetteur_1$: événement représentant l'arrêt anormal de l'émetteur. Il se produit lorsque le fichier n'a pas pu être entièrement copié.
- $ErreurRécepteur_1$: Idem pour le récepteur.
- $ArrêtTransfert_1$: cet événement se produit uniquement lorsque les deux sites ne sont plus actifs c'est à dire lorsque le protocole s'arrête.

Données

On garde les variables du niveau précédent au changement d'indice près. On introduit la variable $global\_actif_1$ qui indique l'état du protocole. Plus simplement, cette variable est égale à *oui* si le protocole est actif[2] et à *non* sinon.

Invariant de collage

– – **Le fichier de l'émetteur est identique à son abstraction**
$fich\_emet_1 = fich\_emet_0 \wedge$

– – **Les autres variables ne sont pas identiques aux abstractions.**
$fich\_recep_1 \in \text{seq(S)} \wedge$
$emet\_actif_1 \in \{oui, non\} \wedge$
$recep\_actif_1 \in \{oui, non\} \wedge$
$emet\_transfert\_ok_1 \in \{oui, non\} \wedge$
$recep\_transfert\_ok_1 \in \{oui, non\} \wedge$

– – **Nouvelle variable représentant l'état d'activation du protocole**
$global\_actif_1 \in \{oui, non\} \wedge$

– – ***Principe de fonctionnement a:*** **Le protocole transférant un fichier, le fichier destination est un préfixe du fichier source**
$fich\_recep_1 \subseteq fich\_emet_1 \wedge$

– – **Les deux sites peuvent être inactifs avant que le protocole se termine dans le système concret alors que dans la spécification abstraite, cet arrêt n'est considéré que lorsque le protocole est terminé. On peut donc dire que les sites raffinés s'arrêtent *avant* leurs abstractions**
$(\ emet\_actif_0 = non \wedge recep\_actif_0 = non \ ) \Rightarrow (\ emet\_actif_1 = non \wedge recep\_actif_1 = non \ ) \wedge$

[2] protocole actif = au moins un site actif

– – **Quand le protocole est inactif, le fichier du récepteur ainsi que les états d'activation sont identiques à leurs abstractions**

$$\left(\begin{array}{l} emet\_actif_1 = non \wedge recep\_actif_1 = non \end{array} \right) \Rightarrow \left(\begin{array}{l} fich\_recep_0 = fich\_recep_1 \wedge \\ emet\_actif_0 = emet\_actif_1 \wedge \\ recep\_actif_0 = recep\_actif_1 \wedge \\ emet\_transfert\_ok_0 = emet\_transfert\_ok_1 \wedge \\ recep\_transfert\_ok_0 = recep\_transfert\_ok_1 \end{array} \right) \wedge$$

Propriétés temporelles

Propriétés temporelles raffinées

– – **Le protocole est modélisé comme un démon. Il attend une nouvelle demande de transfert après qu'il ait terminé de copier le fichier**

$${}_hB_1^1 \quad \Box \left(\begin{array}{l} global\_actif_1 = non \wedge emet\_actif_1 = non \wedge recep\_actif_1 = non \\ \Rightarrow \bigcirc \left(\begin{array}{l} global\_actif_1 = oui \wedge \\ emet\_actif_1 = oui \wedge recep\_actif_1 = oui \end{array} \right) \end{array} \right)$$

– – **Les deux sites ne se désactivent plus immédiatement mais fatalement. Ces deux propriétés dynamiques expriment toujours partiellement le principe de fonctionnement c. Ce dernier sera exprimé complètement dans les propriétés introduites à ce niveau de raffinement**

$${}_cB_1^2 \quad \Box \left(\; emet\_actif_1 = oui \Rightarrow \Diamond \left(\; emet\_actif_1 = non \; \right) \; \right)$$

$$B_1^c \quad \Box \left(\; 3 \; \right) recep\_actif_1 = oui \Rightarrow \Diamond \left(\; recep\_actif_1 = non \; \right)$$

– – **Lorsque le protocole s'arrête, il peut être dans l'un des trois états cités page 14**

$${}_{1,2,3,f,d}B_1^4 \quad \Box \left(\begin{array}{l} global\_actif_1 = oui \wedge \bigcirc \left(\; global\_actif_1 = non \; \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} emet\_transfert\_ok_1 = oui \wedge \\ recep\_transfert\_ok_1 = oui \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_1 = non \wedge \\ recep\_transfert\_ok_1 = non \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_1 = non \wedge \\ recep\_transfert\_ok_1 = oui \end{array} \right) \end{array} \right) \end{array} \right)$$

Il est possible d'écrire cette propriété sous la forme :

$${}_{1,2,3,f,d}B_1^{4'} \quad \Box \left(\begin{array}{l} global\_actif_1 = oui \Rightarrow \\ \bigcirc \left(\begin{array}{l} global\_actif_1 = oui \vee \\ \left(\; emet\_transfert\_ok_1 = oui \wedge recep\_transfert\_ok_1 = oui \; \right) \vee \\ \left(\; emet\_transfert\_ok_1 = non \wedge recep\_transfert\_ok_1 = non \; \right) \vee \\ \left(\; emet\_transfert\_ok_1 = non \wedge recep\_transfert\_ok_1 = oui \; \right) \end{array} \right) \end{array} \right)$$

Cette transformation est rendue possible par l'équivalence suivante $\Box \left(\; p \wedge \bigcirc p' \Rightarrow \bigcirc q \; \right) \Leftrightarrow \Box \left(\; p \Rightarrow \bigcirc \left(\; \neg p' \vee q \; \right) \; \right)$ car

$$\begin{array}{l} \neg (p \wedge \bigcirc p') \vee \bigcirc q \\ \Leftrightarrow \neg p \vee \neg \bigcirc p' \vee \bigcirc q \\ \Leftrightarrow \neg p \vee \bigcirc \neg p' \vee \bigcirc q \\ \Leftrightarrow \neg p \vee \bigcirc \neg p' \vee q \\ \Leftrightarrow \neg p \Rightarrow \bigcirc \left(\; \neg p' \vee q \; \right) \end{array}$$

Mais, même si mathématiquement ces deux expressions sont équivalentes, la seconde parait moins claire vis à vis de la description informelle. En effet, la première propriété exprime directement la phrase informelle. Tandis que l'autre pourrait se traduire par "Si le protocole est actif alors à l'état suivant il restera actif ou sera dans l'un des trois états finaux". D'autre part, cette formulation n'exprime plus une propriété sur une transition mais une propriété sur un état.

NOUVELLES PROPRIÉTÉS TEMPORELLES

– – **Lorsque les deux sites sont inactifs, le protocole doit devenir lui aussi inactif. Le protocole devient inactif *après* l'arrêt du dernier site en fonctionnement. L'arrêt du protocole ne se produit pas simultanément à celui du dernier site car durant ce raffinement on différencie l'arrêt du protocole et ceux des sites.**

$$
{}_cB_1^5 \quad \Box \left(\begin{array}{l} global\_actif_1 = oui \wedge emet\_actif_1 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_1 = oui \wedge emet\_actif_1 = non \wedge \\ recep\_actif_1 = non \end{array} \right) \\ \Rightarrow \bigcirc \bigcirc \left(\; global\_actif_1 = non \; \right) \end{array} \right)
$$

$$
{}_cB_1^6 \quad \Box \left(\begin{array}{l} global\_actif_1 = oui \wedge recep\_actif_1 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_1 = oui \wedge emet\_actif_1 = non \wedge \\ recep\_actif_1 = non \end{array} \right) \\ \Rightarrow \bigcirc \bigcirc \left(\; global\_actif_1 = non \; \right) \end{array} \right)
$$

– – **Les deux sites se désactivent l'un après l'autre et consécutivement. Cette propriété dynamique est plus forte que le principe de fonctionnement c. Ceci est propre au niveau d'abstraction. La propriété subira des modifications la rapprochant du principe de fonctionnement dans les raffinements ultérieurs**

$$
{}_cB_1^7 \quad \Box \left(\begin{array}{l} emet\_actif_1 = oui \wedge recep\_actif_1 = oui \wedge \\ \bigcirc \left(\; emet\_actif_1 = non \wedge recep\_actif_1 = oui \; \right) \\ \Rightarrow \bigcirc \bigcirc \left(\; recep\_actif_1 = non \; \right) \end{array} \right)
$$

$$
{}_cB_1^8 \quad \Box \left(\begin{array}{l} emet\_actif_1 = oui \wedge recep\_actif_1 = oui \wedge \\ \bigcirc \left(\; recep\_actif_1 = non \wedge emet\_actif_1 = oui \; \right) \\ \Rightarrow \bigcirc \bigcirc \left(\; emet\_actif_1 = non \; \right) \end{array} \right)
$$

– – **Quand l'expéditeur sait que le protocole s'est bien terminé alors le récepteur le savait déjà.**

$$
{}_eB_1^9 \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} emet\_actif_1 = non \wedge emet\_transfert\_ok_1 = oui \wedge \\ global\_actif_1 = oui \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} recep\_actif_1 = non \wedge recep\_transfert\_ok_1 = oui \wedge \\ global\_actif_1 = oui \end{array} \right) \end{array} \right)
$$

– – **Quand le récepteur sait que le protocole s'est mal terminé alors l'émetteur le savait déjà**

$$
{}_eB_1^{10} \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} recep\_actif_1 = non \wedge recep\_transfert\_ok_1 = non \wedge \\ global\_actif_1 = oui \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} emet\_actif_1 = non \wedge emet\_transfert\_ok_1 = non \wedge \\ global\_actif_1 = oui \end{array} \right) \end{array} \right)
$$

On constate que les propriétés dynamiques B_1^9 et B_1^{10} sont des raffinements de l'invariant du niveau abstrait. Ce type de raffinement est rendu possible par le fait que l'expression des propriétés invariantes est plus forte que celle des propriétés dynamiques. En effet, les premières ne posent des contraintes que sur un seul état alors que les propriétés de comportement le font sur exactement deux états.

Ici aussi nous pouvons tenter que reformuler les deux dernières expressions:

$$_eB_1^{9'} \quad \Box \left(\begin{array}{l} recep\_actif_1 = oui \vee recep\_transfert\_ok_1 = non \vee global\_actif_1 = non \\ \Rightarrow \bigcirc \left(\ emet\_actif_1 = oui \vee emet\_transfert\_ok_1 = non \vee global\_actif_1 = non \ \right) \end{array} \right)$$

$$_eB_1^{10'} \quad \Box \left(\begin{array}{l} emet\_actif_1 = non \vee emet\_transfert\_ok_1 = oui \vee global\_actif_1 = non \\ \Rightarrow \bigcirc \left(\ emet\_actif_1 = oui \vee emet\_transfert\_ok_1 = oui \vee global\_actif_1 = non \ \right) \end{array} \right)$$

Ces deux nouvelles expressions sont respectivement équivalentes à B_1^9 et B_1^{10} car $\left(\ \bigcirc a \Rightarrow b \ \right) \Leftrightarrow \left(\ \neg b \Rightarrow \bigcirc \neg a \ \right)$ car:

$$\begin{aligned} & \bigcirc a \Rightarrow b \\ \Leftrightarrow & \neg \bigcirc a \vee b \\ \Leftrightarrow & \bigcirc \neg a \vee b \\ \Leftrightarrow & \neg b \Rightarrow \bigcirc \neg a \end{aligned}$$

Mais là aussi, leurs significations intuitives ne paraissent pas représenter directement la propriété exprimée informellement.

Evénements

Modification de la notation

Dans ce niveau de raffinement, la variable $global\_actif_1$ a été introduite. Afin de pouvoir la représenter sur le graphe d'accessibilité, il est nécessaire de modifier la notation mise en place page 17. La figure 2.4 correspond à la nouvelle représentation graphique des états du système.

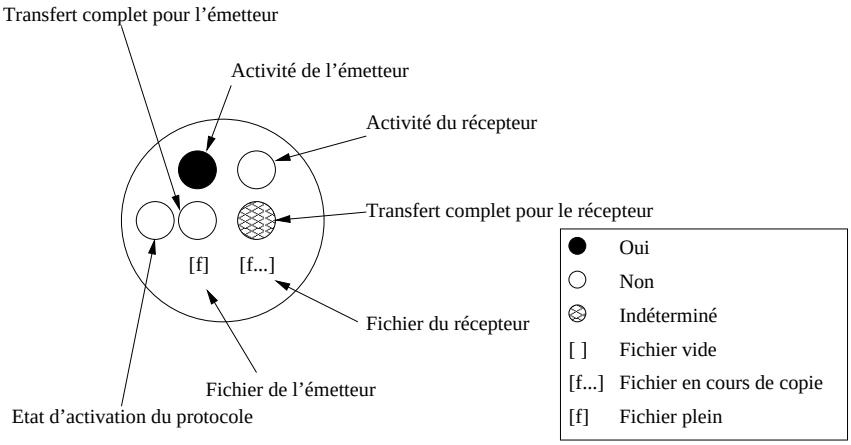


Figure 2.4: BRP - Description de la notation utilisée au raffinement n° 1

L'état d'activation du protocole apparait comme une nouvelle pastille qui sera noire si le protocole est actif et blanche dans l'autre cas. On peut rappeler que le protocole est considéré comme actif lorsqu'au moins un des deux sites est actif.

Description des événements

Ce niveau de raffinement introduit quatre nouveaux événements: $ArrêtEmetteur_1$, $ArrêtRécepteur_1$, $ErreurEmetteur_1$ et $ErreurRécepteur_1$. Chacun d'entre eux permet de modéliser l'arrêt des sites sur un cas d'erreur ou non. La figure 2.5 montre bien les différentes possibilités d'évolution du système:

- L'émetteur s'arrête normalement seulement si le récepteur l'a déjà fait,

- L'émetteur considère que le protocole s'est mal terminé alors que le récepteur sait que le transfert s'est bien déroulé. Ce cas est dû à la perte de l'acquittement du dernier paquet du fichier.

- Le récepteur considère que le protocole a échoué seulement si l'autre site s'est arrêté sur un cas d'erreur,

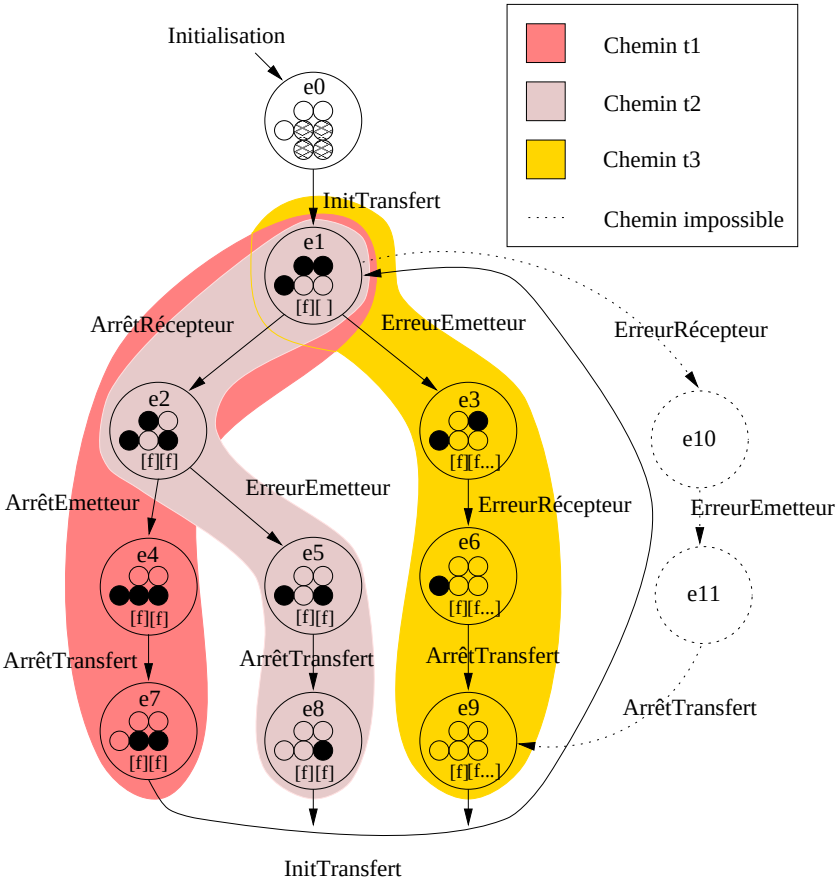


Figure 2.5: BRP - Graphe d'accessibilité du raffinement n ° 1

Le graphe d'accessibilité représenté sur la figure 2.5 est un raffinement du graphe abstrait de la figure 2.3. Les trois transitions $t1$, $t2$ et $t3$ du niveau abstrait sont raffinées par trois chemins portant le même nom. Il est impossible que le chemin passant par les états e1, e10, e11 et e9 sur la figure 2.5 existe. En effet, le site émetteur considère qu'il y a une erreur s'il n'a pas reçu un acquittement au bout d'un délai $D_{max} \times N_{max}$[3]. Le site récepteur lui ne considère qu'il y a erreur lorsque qu'il n'a pas reçu de paquet pendant un temps supérieur à $D_{max} \times N_{max}$. Il est impossible au site récepteur de considérer que le protocole a échoué si l'émetteur n'est pas arrivé aux mêmes conclusions. D'autre part, les délais font que l'arrêt du site récepteur sur une erreur ne peut se produire que si le site émetteur s'est arrêté lui aussi sur une situation anormale.

L'initialisation est proche de son abstraction. Les deux sites sont toujours inactifs au démarrage du système, ainsi que l'état global du protocole représenté par la variable $global\_actif_1$.

[3] D_{max} est le délai de transmission maximale d'un paquet et N_{max} est le nombre maximal de réémissions

Initialisation	$\triangleq$	$emet\_actif_1 := non \ \|\|$ $recep\_actif_1 := non \ \|\|$ $global\_actif_1 := non$

L'événement $InitTransfert_1$ est identique à son abstraction. Toutefois il est nécessaire de mettre à jour la valeur de la variable $global\_actif_1$.

$InitTransfert_1$	$\triangleq$	SELECT $emet\_actif_1 = non \wedge$ $recep\_actif_1 = non \wedge$ $global\_actif_1 = non$ THEN ANY f WHERE $f \in \text{seq}_1(\text{S})$ THEN $fich\_emet_1 := f$ END $\|\|$ $fich\_recep_1 := [\,] \ \|\|$ $emet\_actif_1 := oui \ \|\|$ $recep\_actif_1 := oui \ \|\|$ $emet\_transfert\_ok_1 := non \ \|\|$ $recep\_transfert\_ok_1 := non \ \|\|$ $global\_actif_1 := oui$ END

L'événement $ArrêtTransfert_1$ est différent de son abstraction. Il ne représente ici que l'arrêt du protocole. Ce dernier ne peut se produire que lorsque les deux sites sont devenus inactifs. La copie du fichier n'est plus réalisée dans cet événement mais répartie dans $ArrêtRécepteur_1$ et $ErreurRécepteur_1$.

$ArrêtTransfert_1$	$\triangleq$	SELECT $emet\_actif_1 = non \wedge$ $recep\_actif_1 = non \wedge$ $global\_actif_1 = oui$ THEN $global\_actif_1 := non$ END

Afin de représenter les deux possibilités d'arrêt du site émetteur, les deux événements suivants sont introduits.

$ArrêtEmetteur_1$ ne se produit que lorsque le fichier a été entièrement copié. Le récepteur s'est déjà arrêté et sait si le fichier a été entièrement copié. Le résultat de cet événement est l'arrêt du site source.

$ArrêtEmetteur_1$	$\triangleq$	SELECT $emet\_actif_1 = oui \wedge$ $recep\_actif_1 = non \wedge$ $recep\_transfert\_ok_1 = oui$ THEN $emet\_actif_1 := non \ \|\|$ $emet\_transfert\_ok_1 := oui$ END

L'événement $ErreurEmetteur_1$ diffère d'$ArrêtEmetteur_1$ par le fait que le site source considère que le protocole s'est terminé sur une erreur.

$ErreurEmetteur_1 \triangleq$
SELECT
$emet\_actif_1 = oui$
THEN
$emet\_actif_1 := non \ \|$
$emet\_transfert\_ok_1 := non$
END

Tout comme l'émetteur, le récepteur possède deux nouveaux événements permettant de modéliser les différentes possibilités d'arrêt du site.

$ArrêtRécepteur_1$ représente l'arrêt du site destination avec la copie du fichier entièrement réalisée. L'effet de cet événement est la désactivation du récepteur mais surtout la copie du fichier source vers son homologue. La garde de cet événement permet d'éviter que le site récepteur s'arrête normalement alors que l'émetteur a détecté une erreur.

$ArrêtRécepteur_1 \triangleq$
SELECT
$recep\_actif_1 = oui \wedge emet\_actif_1 = oui$
THEN
$fich\_recep_1 := fich\_emet_1 \ \|$
$recep\_actif_1 := non \ \|$
$recep\_transfert\_ok_1 := oui$
END

L'événement $ErreurRécepteur_1$ permet de stopper le récepteur sur un cas d'erreur. Il copiera une partie seulement du fichier source et désactivera le récepteur.

$ErreurRécepteur_1 \triangleq$
SELECT
$recep\_actif_1 = oui \wedge$
$emet\_actif_1 = non \wedge$
$emet\_transfert\_ok_1 = non$
THEN
ANY f
WHERE
$f \in \text{seq(S)} \wedge f \subset fich\_emet_1$
THEN
$fich\_recep_1 := f$
END $\|$
$emet\_actif_1 := non \ \|$
$recep\_transfert\_ok_1 := non$
END

2.2.3 Raffinement n ° 2

But

Durant ce raffinement, on s'intéresse à l'arrêt du site émetteur sur un cas d'erreur. Pour cela, l'émetteur doit recevoir un acquittement de la part du récepteur pour lui indiquer que ce dernier a bien reçu le fichier. Si l'acquittement n'arrive pas au site source, celui-ci peut renvoyer le fichier encore une fois. Afin qu'il n'y ait pas de comportement infini, on introduit une variable comptant

le nombre de réémissions. Lorsque ce compteur atteint sa valeur maximale, l'émetteur considère que le protocole est définitivement arrêté.

L'acquittement du transfert de fichier est modélisé par l'événement $ArrêtRécepteur_2$. On introduit un nouvel événement permettant de gérer le compteur du nombre de réémissions: $DelaiEcoulé_2$. Cet événement se produit lorsqu'un acquittement attendu n'est jamais arrivé. Il a pour effet d'incrémenter le compteur cité ci-dessus.

Données

On garde les variables du niveau précédent au changement d'indice près. On introduit la variable $NbRéémis_2$ qui permet de compter le nombre de réémissions du fichier par le site source. Cette variable est bornée par la constante MAX. Nous allons utiliser une astuce concernant cette nouvelle variable; si elle atteint la valeur MAX alors l'émetteur s'arrête et positionne sa valeur à (MAX + 1). Cette dernière est utilisée pour dire que le récepteur doit s'arrêter sur un cas d'erreur. Ce petit protocole permet de déclencher l'événement $ErreurRécepteur_2$ seulement lorsque $ErreurEmetteur_2$ est arrivé.

Invariant de collage

– – **Les variables suivantes sont identiques à leurs abstractions**
$fich\_emet_2 = fich\_emet_1 \wedge$
$fich\_recep_2 = fich\_recep_1 \wedge$
$emet\_actif_2 = emet\_actif_1 \wedge$
$recep\_actif_2 = recep\_actif_1 \wedge$
$emet\_transfert\_ok_2 = emet\_transfert\_ok_1 \wedge$
$recep\_transfert\_ok_2 = recep\_transfert\_ok_1 \wedge$
$global\_actif_2 = global\_actif_1 \wedge$

– – **La variable $NbRéémis_2$ est introduite est représente le nombre de *time out* se produisant sur le site émetteur**
$NbRéémis_2 \in 0..(\text{MAX} + 1) \wedge$

– – **Lorsque le compteur atteint le maximum, alors l'émetteur considère que le protocole s'est mal terminé**
$NbRéémis_2 = (\text{MAX} + 1) \Leftrightarrow \left(\; emet\_actif_2 = non \wedge emet\_transfert\_ok_2 = non \; \right) \wedge$

Propriétés temporelles

PROPRIÉTÉS TEMPORELLES RAFFINÉES

– – **Le protocole est modélisé comme un démon. Il attend une nouvelle demande de transfert après qu'il ait terminé de copier le fichier**

$$ {}_hB_2^1 \quad \Box \left(\begin{array}{l} global\_actif_2 = non \wedge emet\_actif_2 = non \wedge recep\_actif_2 = non \\ \Rightarrow \bigcirc \left(\begin{array}{l} global\_actif_2 = oui \wedge \\ emet\_actif_2 = oui \wedge recep\_actif_2 = oui \end{array} \right) \end{array} \right) $$

– – **Les deux sites se désactivent fatalement**
$${}_cB_2^2 \quad \Box \left(\; emet\_actif_2 = oui \Rightarrow \Diamond \left(\; emet\_actif_2 = non \; \right) \; \right)$$
$${}_cB_2^3 \quad \Box \left(\; recep\_actif_2 = oui \Rightarrow \Diamond \left(\; recep\_actif_2 = non \; \right) \; \right)$$

– – **Lorsque le protocole est arrêté, il peut être dans l'un des trois états cités page 14**

$$_{1,2,3,f,d}B_2^4 \quad \Box \left(\begin{array}{l} global\_actif_2 = oui \wedge \bigcirc (\ global\_actif_2 = non \ ) \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} emet\_transfert\_ok_2 = oui \wedge \\ recep\_transfert\_ok_2 = oui \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_2 = non \wedge \\ recep\_transfert\_ok_2 = non \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_2 = non \wedge \\ recep\_transfert\_ok_2 = oui \end{array} \right) \end{array} \right) \end{array} \right)$$

– – **Lorsque les deux sites sont inactifs, le protocole est lui aussi inactivé. Le protocole devient inactif après l'arrêt du dernier site en fonctionnement.**

$$_cB_2^5 \quad \Box \left(\begin{array}{l} global\_actif_2 = oui \wedge emet\_actif_2 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_2 = oui \wedge emet\_actif_2 = non \wedge \\ recep\_actif_2 = non \end{array} \right) \\ \Rightarrow \bigcirc \bigcirc (\ global\_actif_2 = non \ ) \end{array} \right)$$

$$_cB_2^6 \quad \Box \left(\begin{array}{l} global\_actif_2 = oui \wedge recep\_actif_2 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_2 = oui \wedge emet\_actif_2 = non \wedge \\ recep\_actif_2 = non \end{array} \right) \\ \Rightarrow \bigcirc \bigcirc (\ global\_actif_2 = non \ ) \end{array} \right)$$

– – **Les deux sites se désactivent l'un après l'autre et consécutivement**

$$_cB_2^7 \quad \Box \left(\begin{array}{l} emet\_actif_2 = oui \wedge recep\_actif_2 = oui \wedge \\ \bigcirc (\ emet\_actif_2 = non \wedge recep\_actif_2 = oui \ ) \\ \Rightarrow \bigcirc \bigcirc (\ recep\_actif_2 = non \ ) \end{array} \right)$$

$$_cB_2^8 \quad \Box \left(\begin{array}{l} emet\_actif_2 = oui \wedge recep\_actif_2 = oui \wedge \\ \bigcirc (\ recep\_actif_2 = non \wedge emet\_actif_2 = oui \ ) \\ \Rightarrow \left(\begin{array}{l} NbRéémis_2 < \text{MAX} \wedge \\ emet\_actif_2 = oui \end{array} \right) \mathcal{U} (\ emet\_actif_2 = non \ ) \end{array} \right)$$

– – **Quand l'expéditeur sait que le protocole s'est bien terminé alors le récepteur le savait déjà**

$$_eB_2^9 \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} emet\_actif_2 = non \wedge emet\_transfert\_ok_2 = oui \wedge \\ global\_actif_2 = oui \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} recep\_actif_2 = non \wedge recep\_transfert\_ok_2 = oui \wedge \\ global\_actif_2 = oui \end{array} \right) \end{array} \right)$$

– – **Quand le récepteur sait que le protocole s'est mal terminé alors l'émetteur le savait déjà**

$$_eB_2^{10} \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} recep\_actif_2 = non \wedge recep\_transfert\_ok_2 = oui \wedge \\ global\_actif_2 = oui \wedge NbRéémis_2 = (\ \text{MAX} + 1 \ ) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} emet\_actif_2 = non \wedge emet\_transfert\_ok_2 = oui \wedge \\ global\_actif_2 = oui \wedge NbRéémis_2 = \text{MAX} \end{array} \right) \end{array} \right)$$

– – **Lorsque le compteur du nombre de *time out* arrive à son maximum, l'émetteur est arrêté sur une erreur et le compteur est de nouveau incrémenté pour permettre l'arrêt du récepteur. Cette propriété exprime en partie les principes de fonctionnement c et g**

$$_{c,g}B_2^{11} \quad \Box \left(\begin{array}{l} NbRéémis_2 = \text{MAX} \wedge emet\_actif_2 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} NbRéémis_2 = (\ \text{MAX} + 1\ ) \wedge \\ emet\_actif_2 = non \wedge \\ emet\_transfert\_ok_2 = non \end{array} \right) \end{array} \right)$$

– – **Lorsque le site émetteur s'est arrêté sur une erreur, le récepteur ne peut que détecter lui aussi une anomalie. Cette propriété exprime en partie le principe de fonctionnement e**

$$_eB_2^{12} \quad \Box \left(\begin{array}{l} NbRéémis_2 = (\text{MAX} + 1) \wedge recep\_actif_2 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} recep\_actif_2 = non \wedge \\ recep\_transfert\_ok_2 = non \end{array} \right) \end{array} \right)$$

Evénements

MODIFICATION DE LA NOTATION

Dans ce niveau de raffinement, on a introduit un compte du nombre de réémissions sous la forme de la variable $NbRéémis_2$. Il est nécessaire de modifier encore une fois la notation utilisée pour la représentation des états. La figure 2.6 montre les modifications apportées.

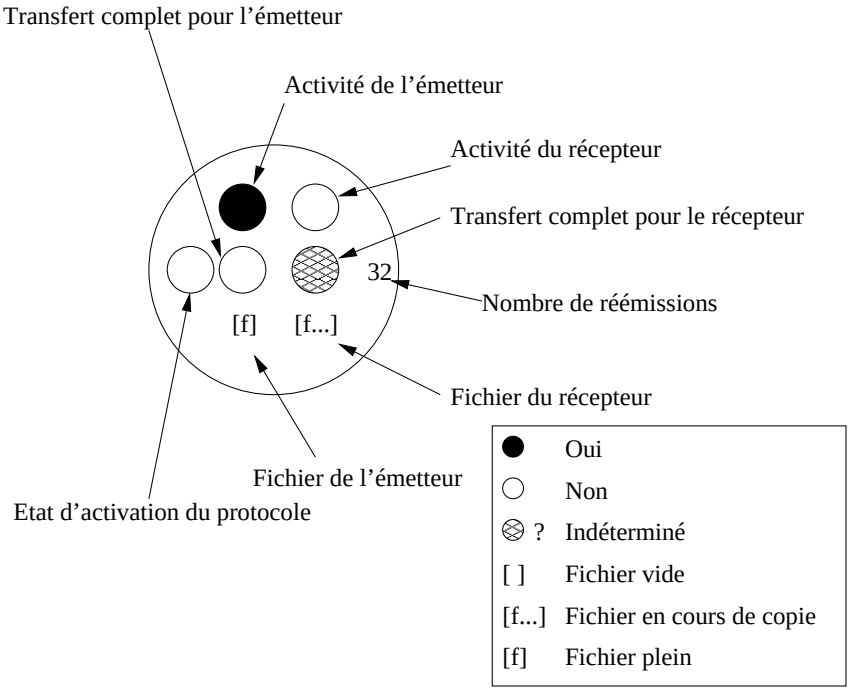


Figure 2.6: BRP - Description de la notation utilisée au raffinement n ° 2

Le nombre de réémissions apparait directement dans la partie droite des états. Il prend une valeur dans $0..(\text{MAX} + 1)$. Dans le premier état du système, la valeur du nombre de réémissions est inconnue ou indéterminée. Ce fait est représenté par l'utilisation de ? en lieu et place de la

valeur de $NbRéémis_2$.

Description des événements

Un seul événement est introduit à ce niveau de raffinement: $DelaiEcoulé_2$. Ce dernier, représenté sur la figure 2.7, correspondent au réveil du site émetteur après un délai d'attente sans recevoir d'acquittement.

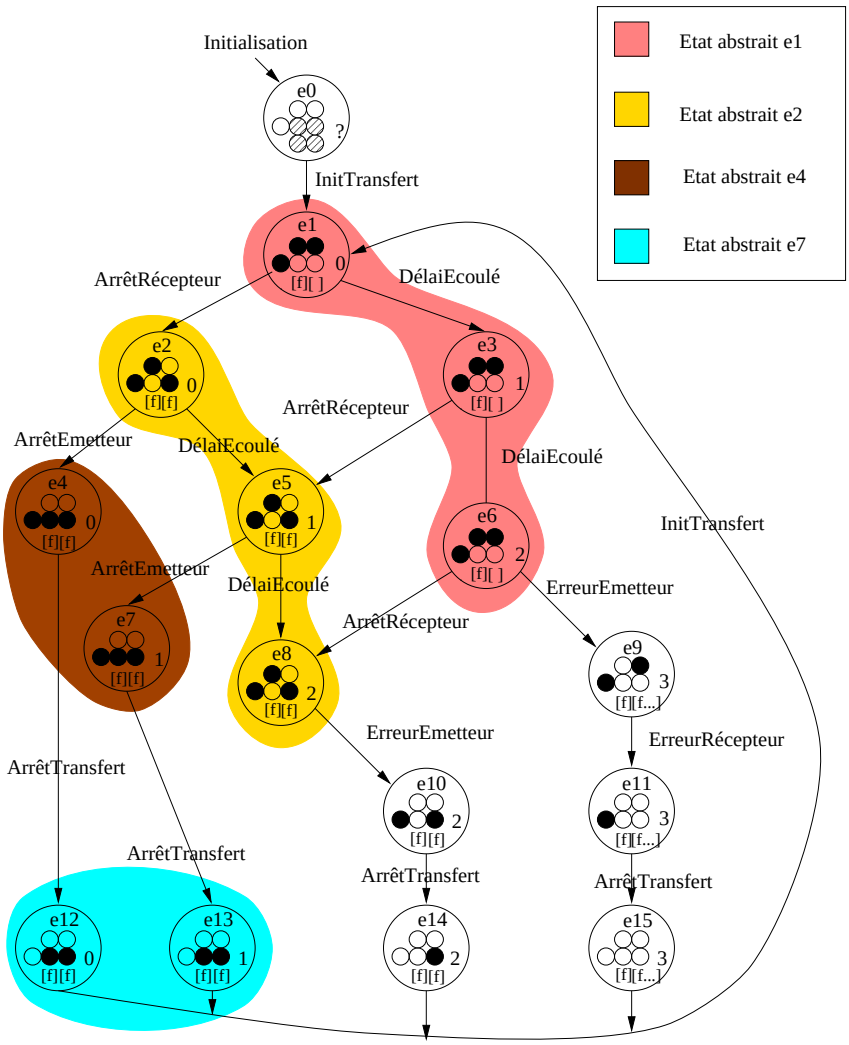


Figure 2.7: BRP - Graphe d'accessibilité du raffinement n ° 2 avec Max = 2

La figure 2.7 montre qu'à ce niveau, l'introduction du nombre de réémissions provoque le raffinement des états abstraits $e1$, $e2$, $e4$ et $e7$. Le graphe voit sa complexité explosée proportionellement à la valeur de la variable Max.

L'initialisation du système est identique à celle du système abstrait.

Initialisation	$\triangleq$	$emet\_actif_2 := non \ \|\|$ $recep\_actif_2 := non \ \|\|$ $global\_actif_2 := non$

L'événement $InitTransfert_2$ est similaire à son abstraction. La seule différence est la présence de

l'initialisation de la variable $NbRéémis_2$.

$InitTransfert_2 \triangleq$
SELECT
$emet\_actif_2 = non \wedge$
$recep\_actif_2 = non \wedge$
$global\_actif_2 = non$
THEN
ANY f
WHERE
$f \in \text{seq}_1(\text{S})$
THEN
$fich\_emet_2 := f$
END ||
$fich\_recep_2 := [\,]$ ||
$emet\_actif_2 := oui$ ||
$recep\_actif_2 := oui$ ||
$emet\_transfert\_ok_2 := non$ ||
$recep\_transfert\_ok_2 := non$ ||
$global\_actif_2 := oui$ ||
$NbRéémis_2 := 0$
END

Les événements $ArrêtTransfert_2$, $ArrêtEmetteur_2$ et $ArrêtRécepteur_2$ ne comporte aucune modification par rapport à leurs abstractions.

$ArrêtTransfert_2 \triangleq$
SELECT
$emet\_actif_2 = non \wedge$
$recep\_actif_2 = non \wedge$
$global\_actif_2 = oui$
THEN
$global\_actif_2 := non$
END

$ArrêtEmetteur_2 \triangleq$
SELECT
$emet\_actif_2 = oui \wedge$
$recep\_actif_2 = non \wedge$
$recep\_transfert\_ok_2 = oui$
THEN
$emet\_actif_2 := non$ ||
$emet\_transfert\_ok_2 := oui$
END

$ArrêtRécepteur_2 \triangleq$
SELECT
$recep\_actif_2 = oui \wedge emet\_actif_2 = oui$
THEN
$fich\_recep_2 := fich\_emet_2$ ||
$recep\_actif_2 := non$ ||
$recep\_transfert\_ok_2 := oui$
END

$ErreurEmetteur_2$ possède la même sémantique que son abstraction. Toutefois il ne peut être déclenché que lorsque le compteur $NbRéémis_2$ à atteint son maximum. Une fois cet événement activable, le compteur doit être modifié afin de permettre à $ErreurRécepteur_2$ d'être éxécuté.

$ErreurEmetteur_2 \triangleq$
SELECT
$emet\_actif_2 = oui \wedge$
$NbRéémis_2 = \text{MAX}$
THEN
$emet\_actif_2 := non \ \|$
$emet\_transfert\_ok_2 := non \ \|$
$NbRéémis_2 := \text{MAX} + 1$
END

L'événement $ErreurRécepteur_2$ est lui aussi presque identique à son abstraction. A ce niveau, il ne peut pas être déclenché que si une erreur est déjà survenue chez l'émetteur c'est à dire si la variable $NbRéémis_2$ est égale à $\text{MAX} + 1$.

$ErreurRécepteur_2 \triangleq$
SELECT
$recep\_actif_2 = oui \wedge$
$emet\_actif_2 = non \wedge$
$emet\_transfert\_ok_2 = non \wedge$
$NbRéémis_2 = (\text{MAX} + 1)$
THEN
ANY f
WHERE
$f \in \text{seq(S)} \wedge f \subset fich\_emet_2$
THEN
$fich\_recep_2 := f$
END $\|$
$emet\_actif_2 := non \ \|$
$recep\_transfert\_ok_2 := non$
END

Le nouvel événement $DelaiEcoulé_2$ est déclenchable lorsque l'émetteur a envoyé le fichier au récepteur et lorsqu'aucun acquittement n'a été reçu. Toutefois il ne doit pas arrêter l'émetteur mais mettre à jour le compteur.

$DelaiEcoulé_2 \triangleq$
SELECT
$emet\_actif_2 = oui \wedge$
$NbRéémis_2 < \text{MAX}$
THEN
$NbRéémis_2 := NbRéémis_2 + 1$
END

2.2.4 Raffinement n ° 3

But

Le but de ce raffinement est d'introduire le transfert du fichier par petits paquets. Pour réaliser cette tâche, on introduit un nouvel événement : $RéceptionPaquet_3$. Il permettra avec $ArrêtRécepteur_3$ de modéliser le transfert morceau par morceau. $RéceptionPaquet_3$ réalise la copie des paquets ne se trouvant pas à la fin du fichier source, alors qu'$ArrêtRécepteur_3$ copie

le dernier. Le transfert de la totalité du fichier est modélisé par une suite éventuellement vide d'événements $RéceptionPaquet_3$ suivie d'un $ArrêtRécepteur_3$ (si aucune erreur ne s'est produite). La variable $fich\_emet_3$ ne représente plus la totalité du fichier à copier mais la partie de celui-ci qu'il reste à traiter. Dans ce niveau de raffinement, on ne force pas encore le système à avoir un acquittement après l'envoi d'un bloc du fichier. Cette notion sera introduite plus tard.

Données

Les variables du niveau précédent sont conservées au changement d'indice près.

Invariant de collage

– – **Les variables suivantes sont identiques à leurs abstractions**
$emet\_actif_3 = emet\_actif_2 \wedge$
$recep\_actif_3 = recep\_actif_2 \wedge$
$emet\_transfert\_ok_3 = emet\_transfert\_ok_2 \wedge$
$recep\_transfert\_ok_3 = recep\_transfert\_ok_2 \wedge$
$global\_actif_3 = global\_actif_2 \wedge$
$NbRéémis_3 = NbRéémis_2 \wedge$

– – **Les fichiers ne sont pas les mêmes que leurs abstractions.**
$fich\_emet_3 \in \text{seq(S)} \wedge$
$fich\_recep_3 \in \text{seq(S)} \wedge$

– – **La concaténation du fichier source et du fichier destination correspond au fichier source abstrait**
$fich\_recep_3 \frown fich\_emet_3 = fich\_emet_2$

– – **Quand le récepteur est actif, c'est à dire lorsqu'il est en train de copier, le fichier destination est un préfixe strict du fichier source abstrait**
$emet\_actif_3 = oui \Rightarrow fich\_recep_3 \subset fich\_emet_2$

– – **Lorsque le récepteur a terminé de copier, le fichier destination est identique à son abstraction**
$emet\_actif_3 = non \Rightarrow fich\_recep_3 = fich\_recep_2$

– – **Lorsque l'émetteur est en cours de copie, le fichier source n'est pas vide, mais il peut diminuer(déduction à partir des clauses précédents)**
$emet\_actif_3 = oui \Rightarrow fich\_emet_3 \neq [\,]$

Propriétés temporelles

Propriétés temporelles raffinées

– – **Le protocole est modélisé comme un démon. Il attend une nouvelle demande de transfert après qu'il ait terminé de copier le fichier**

$$ {}_hB_3^1 \quad \Box \left(\begin{array}{l} global\_actif_3 = non \wedge emet\_actif_3 = non \wedge recep\_actif_3 = non \\ \Rightarrow \bigcirc \left(\begin{array}{l} global\_actif_3 = oui \wedge \\ emet\_actif_3 = oui \wedge recep\_actif_3 = oui \end{array} \right) \end{array} \right) $$

– – **Les deux sites ne se désactivent plus fatalement**

$${}_cB_3^2 \quad \Box \left(\ emet\_actif_3 = oui \Rightarrow \Diamond \left(\ emet\_actif_3 = non \ \right) \ \right)$$

$${}_cB_3^3 \quad \Box \left(\ recep\_actif_3 = oui \Rightarrow \Diamond \left(\ recep\_actif_3 = non \ \right) \ \right)$$

– – **Lorsque le protocole est arrêté, il peut être dans l'un des trois états cités page 14**

$${}_{1,2,3,f,d}B_3^4 \quad \Box \left(\begin{array}{l} global\_actif_3 = oui \wedge \bigcirc \left(\ global\_actif_3 = non \ \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} emet\_transfert\_ok_3 = oui \wedge \\ recep\_transfert\_ok_3 = oui \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_3 = non \wedge \\ recep\_transfert\_ok_3 = non \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_3 = non \wedge \\ recep\_transfert\_ok_3 = oui \end{array} \right) \end{array} \right) \end{array} \right)$$

– – **Lorsque les deux sites sont inactifs, le protocole est lui aussi inactivé. Le protocole devient inactif après l'arrêt du dernier site en fonctionnement.**

$${}_cB_3^5 \quad \Box \left(\begin{array}{l} global\_actif_3 = oui \wedge emet\_actif_3 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_3 = oui \wedge emet\_actif_3 = non \wedge \\ recep\_actif_3 = non \end{array} \right) \\ \Rightarrow \bigcirc \bigcirc \left(\ global\_actif_3 = non \ \right) \end{array} \right)$$

$${}_cB_3^6 \quad \Box \left(\begin{array}{l} global\_actif_3 = oui \wedge recep\_actif_3 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_3 = oui \wedge emet\_actif_3 = non \wedge \\ recep\_actif_3 = non \end{array} \right) \\ \Rightarrow \bigcirc \bigcirc \left(\ global\_actif_3 = non \ \right) \end{array} \right)$$

– – **Les deux sites se désactivent l'un après l'autre et consécutivement**

$${}_cB_3^7 \quad \Box \left(\begin{array}{l} emet\_actif_3 = oui \wedge recep\_actif_3 = oui \wedge \\ \bigcirc \left(\ emet\_actif_3 = non \wedge recep\_actif_3 = oui \ \right) \\ \Rightarrow \bigcirc \bigcirc \left(\ recep\_actif_3 = non \ \right) \end{array} \right)$$

$${}_cB_3^8 \quad \Box \left(\begin{array}{l} emet\_actif_3 = oui \wedge recep\_actif_3 = oui \wedge \\ \bigcirc \left(\ recep\_actif_3 = non \wedge emet\_actif_3 = oui \ \right) \\ \Rightarrow \left(\begin{array}{l} NbRéémis_3 < \textsc{Max} \wedge \\ emet\_actif_3 = oui \end{array} \right) \mathcal{U} \left(\ emet\_actif_3 = non \ \right) \end{array} \right)$$

– – **Quand l'expéditeur sait que le protocole s'est bien terminé alors le récepteur le savait déjà**

$${}_eB_3^9 \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} emet\_actif_3 = non \wedge emet\_transfert\_ok_3 = oui \wedge \\ global\_actif_3 = oui \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} recep\_actif_3 = non \wedge recep\_transfert\_ok_3 = oui \wedge \\ global\_actif_3 = oui \end{array} \right) \end{array} \right)$$

– – **Quand le récepteur sait que le protocole s'est mal terminé alors l'émetteur le savait déjà**

$${}_eB_3^{10} \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} recep\_actif_3 = non \wedge recep\_transfert\_ok_3 = oui \wedge \\ global\_actif_3 = oui \wedge NbRéémis_3 = \left(\ \textsc{Max} + 1 \ \right) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} emet\_actif_3 = non \wedge emet\_transfert\_ok_3 = oui \wedge \\ global\_actif_3 = oui \wedge NbRéémis_3 = \textsc{Max} \end{array} \right) \end{array} \right)$$

– – **Lorsque le compteur du nombre de *time out* arrive à son maximum, l'émetteur est arrêté sur une erreur et le compteur est de nouveau incrémenté pour permettre l'arrêt du récepteur**

$$_{c,g}B_3^{11} \quad \Box \left(\begin{array}{l} NbRéémis_3 = \text{MAX} \wedge emet\_actif_3 = oui \wedge recep\_actif_3 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} NbRéémis_3 = (\ \text{MAX} + 1\ ) \wedge \\ emet\_actif_3 = non \wedge \\ emet\_transfert\_ok_3 = non \end{array} \right) \end{array} \right)$$

– – **Lorsque le site émetteur s'est arrêté sur une erreur, le récepteur ne peut que détecter lui aussi une anomalie.**

$$_eB_3^{12} \quad \Box \left(\begin{array}{l} NbRéémis_3 = (\text{MAX} + 1) \wedge recep\_actif_3 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} recep\_actif_3 = non \wedge \\ recep\_transfert\_ok_3 = non \end{array} \right) \end{array} \right)$$

NOUVELLES PROPRIÉTÉS TEMPORELLES

– – **Les données ajoutées au fichier destination proviennent du fichier source. On considère que l'envoi d'un paquet est instantané tant qu'un canal de transmission ne sera pas introduit**

$$\forall r \in \text{seq(S)}, \forall e \in \text{seq(S)}$$

$$_aB_3^{13} \quad \Box \left(\begin{array}{l} r = fich\_recep_3 \wedge e = fich\_emet_3 \wedge e \neq [\,] \wedge \\ \bigcirc (\ fich\_emet_3 = \text{tail}\ (\ e\ )\ ) \\ \Rightarrow \bigcirc (\ fich\_recep_3 = r \frown \text{first}\ (\ e\ )\ ) \end{array} \right)$$

Evénements

MODIFICATION DE LA NOTATION

L'introduction du transfert par paquets nécessite une légère modification de la notation utilisée pour le représentation des états des graphes d'accessibilité. La figure 2.8 montre que l'on représente l'état des fichiers source et destination par un nombre entier.

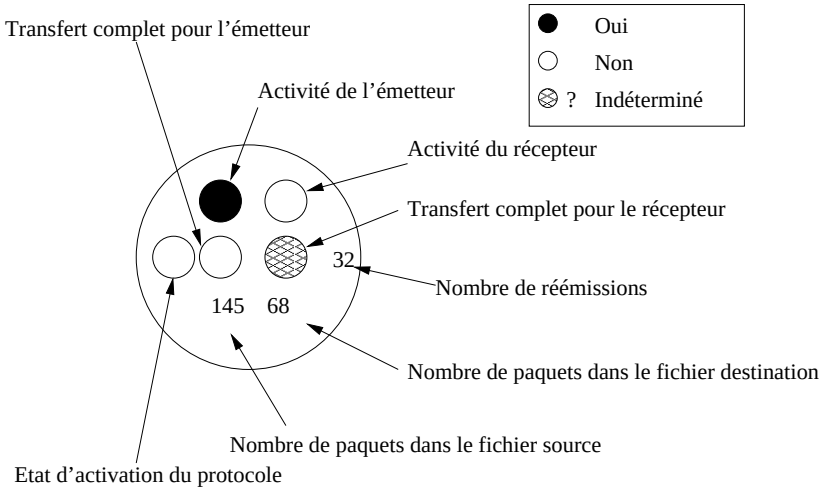


Figure 2.8: BRP - Description de la notation utilisée au raffinement n° 3

Ce nombre entier représente la quantité de paquets composant les fichiers. Le nombre de

paquets du fichier destination sera toujours inférieur ou égal à celui du fichier source.

Description des événements

Ce niveau de raffinement voit l'apparition d'un nouvel événement : $RéceptionPaquet_3$. Il représente le réception d'une partie du fichier source. Le dernier paquet composant le fichier du site émetteur n'est pas reçu par ce nouvel événement mais par $ArrêtRécepteur_3$.

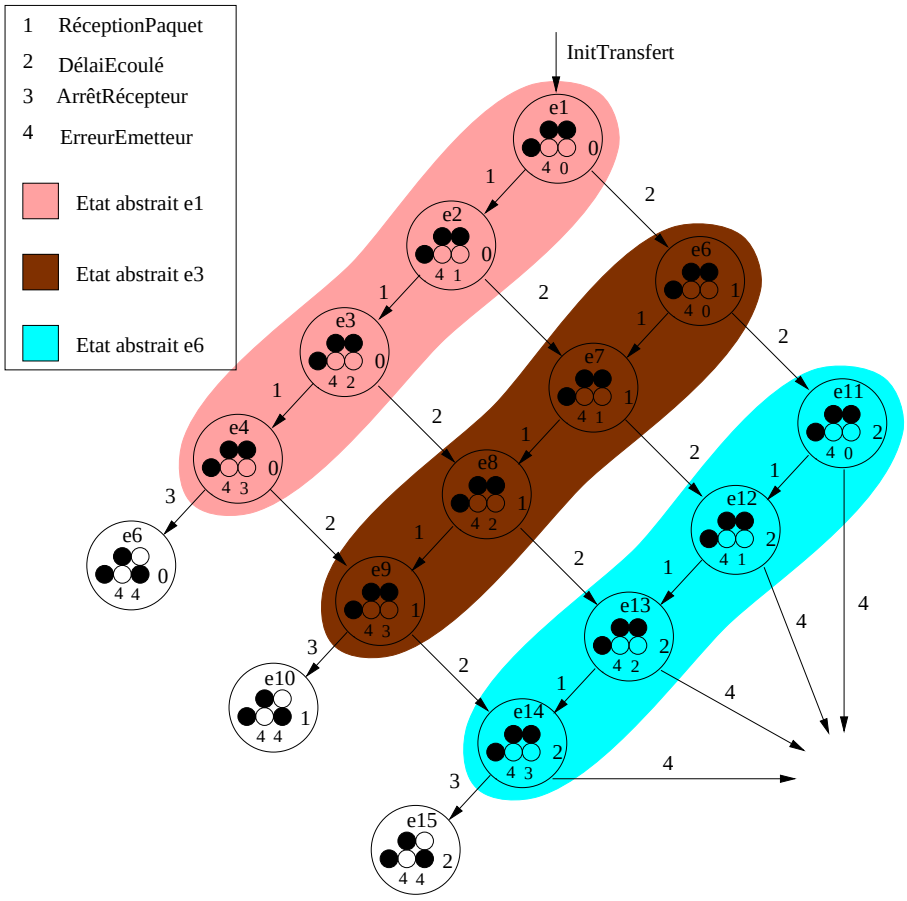


Figure 2.9: BRP - Partie du graphe d'acessibilité du raffinement n°3, raffinement des états abstraits $e1$, $e3$ et $e6$ avec Max = 2 et quatres paquets dans le fichier source

Dans ce niveau de spécification, les états abstraits $e1$, $e3$, $e6$, $e9$, $e11$ et $e15$ sont raffinés en un ensemble détats afin de représenter le transfert par paquets. L'augmentation du nombre d'état est proportionnelle au nombre de paquets dans le fichier source. La figure 2.9 montre le raffinement des trois premiers états abstraits cité ci-dessus en considérant que le nombre de réémission maximal est de deux et le nombre de paquets dans le fichier source est de quatres.

L'initialisation du système est identique à celle du système abstrait.

Initialisation	$\triangleq$	$emet\_actif_3 := non \ \|\|$ $recep\_actif_3 := non \ \|\|$ $global\_actif_3 := non$

L'événement d'initialisation du transfert ne possèdent aucune modification par rapport à son abstraction.

$InitTransfert_3 \triangleq$
SELECT
$emet\_actif_3 = non \wedge$
$recep\_actif_3 = non \wedge$
$global\_actif_3 = non$
THEN
ANY f
WHERE
$f \in \mathrm{seq}_1(\mathrm{S})$
THEN
$fich\_emet_3 := f$
END ||
$fich\_recep_3 := [\,]$ ||
$emet\_actif_3 := oui$ ||
$recep\_actif_3 := oui$ ||
$emet\_transfert\_ok_3 := non$ ||
$recep\_transfert\_ok_3 := non$ ||
$global\_actif_3 := oui$ ||
$NbRéémis_3 := 0$
END

Les événements $ArrêtTransfert_3$ et $ArrêtEmetteur_3$ sont identiques à leurs abstractions.

$ArrêtTransfert_3 \triangleq$
SELECT
$emet\_actif_3 = non \wedge$
$recep\_actif_3 = non \wedge$
$global\_actif_3 = oui$
THEN
$global\_actif_3 := non$
END

$ArrêtEmetteur_3 \triangleq$
SELECT
$emet\_actif_3 = oui \wedge$
$recep\_actif_3 = non \wedge$
$recep\_transfert\_ok_3 = oui$
THEN
$emet\_actif_3 := non$ ||
$emet\_transfert\_ok_3 := oui$
END

L'arrêt du récepteur diffère de son abstraction. Cet événement ne peut être déclenché que lors de la réception du dernier paquet composant le fichier source. Le transfert du dernier paquet est réalisé en partie par cet événement.

$ArrêtRécepteur_3 \triangleq$ SELECT
$recep\_actif_3 = oui \wedge emet\_actif_3 = oui \wedge$
tail ($fich\_emet_3$) = [ ]
THEN
$fich\_recep_3 := fich\_recep_3 \frown$ first ($fich\_emet_3$) ||
$fich\_emet_3 :=$ tail ($fich\_emet_3$) ||
$recep\_actif_3 := non$ ||
$recep\_transfert\_ok_3 := oui$
END

Les arrêts des deux sites sur des cas d'erreur sont réalisés de la même façon que dans le niveau abstrait. Les événements correspondant ne sont pas modifiés.

$ErreurEmetteur_3 \triangleq$ SELECT
$emet\_actif_3 = oui \wedge$
$NbRéémis_3 =$ MAX
THEN
$emet\_actif_3 := non$ ||
$emet\_transfert\_ok_3 := non$ ||
$NbRéémis_3 :=$ MAX + 1
END

$ErreurRécepteur_3 \triangleq$ SELECT
$recep\_actif_3 = oui \wedge$
$emet\_actif_3 = non \wedge$
$emet\_transfert\_ok_3 = non \wedge$
$NbRéémis_3 =$ (MAX + 1)
THEN
ANY f
WHERE
$f \in$ seq(S) $\wedge f \subset fich\_emet_3$
THEN
$fich\_recep_3 := f$
END ||
$emet\_actif_3 := non$ ||
$recep\_transfert\_ok_3 := non$
END

L'événement $DelaiEcoulé_3$ est identique à son abstraction.

$DelaiEcoulé_3 \triangleq$ SELECT
$emet\_actif_3 = oui \wedge$
$NbRéémis_3 <$ MAX
THEN
$NbRéémis_3 := NbRéémis_3 + 1$
END

Le transfert des fichiers est réalisé paquet par paquet. Il existe deux événements pour mettre en oeuvre cette tâche: $ArrêtRécepteur_3$ et $RéceptionPaquet_3$. Le premier correspond à la réception du dernier paquet du fichier source alors que le second ne s'occupe que des autres paquets du

même fichier.

$RéceptionPaquet_3$	$\triangleq$	SELECT $recep\_actif_3 = oui \wedge$ $emet\_actif_3 = oui \wedge$ tail ($fich\_emet_3$) $\neq [\,]$ THEN $fich\_recep_3 := fich\_recep_3 \frown$ first ($fich\_emet_3$) $\|$ $fich\_emet_3 :=$ tail ($fich\_emet_3$) END

2.2.5 Raffinement n ° 4

But

Durant ce raffinement, on s'interesse au transport des données. Jusqu'à présent, les fichiers source et destination étaient modifiés simultanément dans le même événement. Maintenant, seul l'émetteur pourra décrémenter le fichier source et le récepteur se contentera d'incrémenter le fichier destination. Afin d'éviter toute erreur durant le transfert, on introduit un bit global permettant de savoir quel site peut travailler sur les fichiers. Ceci évitera que le fichier source soit décrémenté avant que le fichier destination ne soit modifié. On introduit le nouvel événement $RéceptionAck_4$ qui correspond à la réception de l'acquittement d'un paquet différent du dernier composant le fichier à copier.

Données

On garde les variables du niveau précédent au changement d'indice près. On introduit la variable $Site\_Actif_4$ qui indique le site pouvant modifier un des fichiers.

Invariant de collage

– – **Les variables suivantes sont identiques à leurs abstractions**
$fich\_emet_4 = fich\_emet_3 \wedge$
$fich\_recep_4 = fich\_recep_3 \wedge$
$emet\_actif_4 = emet\_actif_3 \wedge$
$recep\_actif_4 = recep\_actif_3 \wedge$
$emet\_transfert\_ok_4 = emet\_transfert\_ok_3 \wedge$
$recep\_transfert\_ok_4 = recep\_transfert\_ok_3 \wedge$
$global\_actif_4 = global\_actif_3 \wedge$
$NbRéémis_4 = NbRéémis_3 \wedge$

– – **La variable suivante est utilisée pour connaître le site autorisé à la manipulation des fichiers**
$Site\_Actif_4 \in \{émetteur, récepteur\}$

– – **Lorsque le récepteur possède le contrôle, le fichier de l'émetteur est identique à son abstraction**
$Site\_Actif_4 = récepteur \Rightarrow fich\_emet_4 = fich\_emet_3$

– – **Lorsque le récepteur possède le contrôle, le fichier de l'émetteur est plus grand que son abstraction**
$Site\_Actif_4 = émetteur \Rightarrow fich\_emet_4 \neq [\,] \wedge$
$Site\_Actif_4 = récepteur \Rightarrow$ tail ($fich\_emet_4$) $= fich\_emet_3$

Propriétés temporelles

PROPRIÉTÉS TEMPORELLES RAFFINÉES

– – Le protocole est modélisé comme un démon. Il attend une nouvelle demande de transfert après qu'il ait terminé de copier le fichier

$$ {}_hB_4^1 \quad \Box \left(\begin{array}{l} global\_actif_4 = non \wedge emet\_actif_4 = non \wedge recep\_actif_4 = non \\ \Rightarrow \bigcirc \left(\begin{array}{l} global\_actif_4 = oui \wedge \\ emet\_actif_4 = oui \wedge recep\_actif_4 = oui \end{array} \right) \end{array} \right) $$

– – Les deux sites se désactivent fatalement

$$ {}_cB_4^2 \quad \Box \left(\; emet\_actif_4 = oui \Rightarrow \Diamond \left(\; emet\_actif_4 = non \; \right) \right) $$

$$ {}_cB_4^3 \quad \Box \left(\; recep\_actif_4 = oui \Rightarrow \Diamond \left(\; recep\_actif_4 = non \; \right) \right) $$

– – Lorsque le protocole est arrêté, il peut être dans l'un des trois états cités page 14

$$ {}_{1,2,3,f,d}B_4^4 \quad \Box \left(\begin{array}{l} global\_actif_4 = oui \wedge \bigcirc \left(\; global\_actif_4 = non \; \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} emet\_transfert\_ok_4 = oui \wedge \\ recep\_transfert\_ok_4 = oui \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_4 = non \wedge \\ recep\_transfert\_ok_4 = non \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_4 = non \wedge \\ recep\_transfert\_ok_4 = oui \end{array} \right) \end{array} \right) \end{array} \right) $$

– – Lorsque les deux sites sont inactifs, le protocole est lui aussi inactivé. Le protocole devient inactif après l'arrêt du dernier site en fonctionnement.

$$ {}_cB_4^5 \quad \Box \left(\begin{array}{l} global\_actif_4 = oui \wedge emet\_actif_4 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_4 = oui \wedge emet\_actif_4 = non \wedge \\ recep\_actif_4 = non \end{array} \right) \\ \Rightarrow \bigcirc \bigcirc \left(\; global\_actif_4 = non \; \right) \end{array} \right) $$

$$ {}_bB_4^6 \quad \Box \left(\begin{array}{l} global\_actif_4 = oui \wedge recep\_actif_4 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_4 = oui \wedge emet\_actif_4 = non \wedge \\ recep\_actif_4 = non \end{array} \right) \\ \Rightarrow \bigcirc \bigcirc \left(\; global\_actif_4 = non \; \right) \end{array} \right) $$

– – Les deux sites se désactivent l'un après l'autre et consécutivement

$$ {}_cB_4^7 \quad \Box \left(\begin{array}{l} emet\_actif_4 = oui \wedge recep\_actif_4 = oui \wedge \\ \bigcirc \left(\; emet\_actif_4 = non \wedge recep\_actif_4 = oui \; \right) \\ \Rightarrow \bigcirc \bigcirc \left(\; recep\_actif_4 = non \; \right) \end{array} \right) $$

$$ {}_bB_4^8 \quad \Box \left(\begin{array}{l} emet\_actif_4 = oui \wedge recep\_actif_4 = oui \wedge \\ \bigcirc \left(\; recep\_actif_4 = non \wedge emet\_actif_4 = oui \; \right) \\ \Rightarrow \left(\begin{array}{l} NbRéémis_4 < \text{MAX} \wedge \\ emet\_actif_4 = oui \end{array} \right) \mathcal{U} \left(\; emet\_actif_4 = non \; \right) \end{array} \right) $$

– – Quand l'expéditeur sait que le protocole s'est bien terminé alors le récepteur le savait déjà

$$ {}_eB_4^9 \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} emet\_actif_4 = non \wedge emet\_transfert\_ok_4 = oui \wedge \\ global\_actif_4 = oui \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} recep\_actif_4 = non \wedge recep\_transfert\_ok_4 = oui \wedge \\ global\_actif_4 = oui \end{array} \right) \end{array} \right) $$

– – **Quand le récepteur sait que le protocole s'est mal terminé alors l'émetteur le savait déjà**

$$ {}_eB_4^{10} \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} recep\_actif_4 = non \wedge recep\_transfert\_ok_4 = oui \wedge \\ global\_actif_4 = oui \wedge NbRéémis_4 = \left(\ \textsc{Max} + 1 \ \right) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} emet\_actif_4 = non \wedge emet\_transfert\_ok_4 = oui \wedge \\ global\_actif_4 = oui \wedge NbRéémis_4 = \textsc{Max} \end{array} \right) \end{array} \right) $$

– – **Lorsque le compteur du nombre de *time out* arrive à son maximum, l'émetteur est arrêté sur une erreur et le compteur est de nouveau incrémenté pour permettre l'arrêt du récepteur**

$$ {}_{c,g}B_4^{11} \quad \Box \left(\begin{array}{l} NbRéémis_4 = \textsc{Max} \wedge emet\_actif_4 = oui \wedge recep\_actif_4 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} NbRéémis_4 = \left(\ \textsc{Max} + 1 \ \right) \wedge \\ emet\_actif_4 = non \wedge \\ emet\_transfert\_ok_4 = non \end{array} \right) \end{array} \right) $$

– – **Lorsque le site émetteur s'est arrêté sur une erreur, le récepteur ne peut que détecter lui aussi une anomalie.**

$$ {}_eB_4^{12} \quad \Box \left(\begin{array}{l} NbRéémis_4 = (\textsc{Max} + 1) \wedge recep\_actif_4 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} recep\_actif_4 = non \wedge \\ recep\_transfert\_ok_4 = non \end{array} \right) \end{array} \right) $$

– – **Les données ajoutées au fichier destination proviennent du fichier source**

$$ \forall r \in \text{seq(S)}, \forall e \in \text{seq(S)} $$

$$ {}_aB_4^{13} \quad \Box \left(\begin{array}{l} r = fich\_recep_4 \wedge e = fich\_emet_4 \wedge e \neq [\,] \wedge \\ emet\_actif_4 = oui \wedge recep\_actif_4 = oui \wedge Site\_Actif_4 = récepteur \wedge \\ \left(\ fich\_emet_4 = e \ \right) \ \mathcal{U} \ \left(\ fich\_emet_4 = \text{tail} \left(\ e \ \right) \ \right) \\ \Rightarrow \left(\begin{array}{l} Site\_Actif_4 = récepteur \wedge \\ fich\_recep_4 = r \wedge \\ fich\_emet_4 = e \end{array} \right) \mathcal{U} \left(\begin{array}{l} Site\_Actif_4 = émetteur \wedge \\ fich\_recep_4 = r \frown \text{first} \left(\ e \ \right) \wedge \\ fich\_emet_4 = e \end{array} \right) \end{array} \right) $$

NOUVELLES PROPRIÉTÉS TEMPORELLES

– – **Lorsqu'un paquet est ajouté au fichier destination, le contrôle est donné à l'émetteur pour qu'il puisse retirer le paquet reçu du fichier source**

$$ \forall r \in \text{seq(S)}, \forall e \in \text{seq(S)} $$

$$ {}_aB_4^{14} \quad \Box \left(\begin{array}{l} emet\_actif_4 = oui \wedge recep\_actif_4 = oui \wedge \\ e = fich\_emet_4 \wedge e \neq [\,] \wedge \\ r = fich\_recep_4 \wedge \\ Site\_Actif_4 = récepteur \wedge \\ \bigcirc \left(\begin{array}{l} emet\_actif_4 = oui \wedge recep\_actif_4 = oui \wedge \\ fich\_recep_4 = r \frown \text{first} \left(\ e \ \right) \end{array} \right) \\ \Rightarrow \bigcirc \left(\ Site\_Actif_4 = émetteur \ \right) \end{array} \right) $$

– – **Lorsque le fichier source est décrémenté, le contrôle est donné au récepteur pour qu'il puisse lire le prochain paquet**

$$\forall e \in \mathrm{seq(S)}$$
$${}_aB_4^{15} \quad \Box \left(\begin{array}{l} emet\_actif_4 = oui \wedge \\ e = fich\_emet_4 \wedge e \neq [\,] \wedge \\ Site\_Actif_4 = émetteur \wedge \\ \bigcirc \left(\begin{array}{l} fich\_emet_4 = \mathrm{tail}\,(\ e\ ) \wedge \\ emet\_actif_4 = oui \end{array} \right) \\ \Rightarrow \bigcirc (\ Site\_Actif_4 = récepteur\ ) \end{array} \right)$$

– – **Lorsque le protocole est initialisé, le contrôle est automatiquement donné au récepteur. A ce niveau de raffinement, c'est le site émetteur qui copie le premier paquet du fichier. Le site émetteur ne fait que retirer le paquet envoyé du fichier source. Le contrôle doit être donné au récepteur pour ne pas perdre le premier paquet du fichier. A priori, cette propriété n'exprime pas directement un des principes de fonctionnement**

$$B_4^{16} \quad \Box \left(\begin{array}{l} emet\_actif_4 = non \wedge recep\_actif_4 = non \wedge \\ global\_actif_4 = non \wedge \\ \bigcirc \left(\begin{array}{l} emet\_actif_4 = oui \wedge recep\_actif_4 = oui \wedge \\ global\_actif_4 = oui \end{array} \right) \\ \Rightarrow \bigcirc (\ Site\_Actif_4 = récepteur\ ) \end{array} \right)$$

Evénements

MODIFICATION DE LA NOTATION

Dans ce niveau de raffinement, la variable $Site\_Actif_4$ a été introduite. Afin de pouvoir la représenter sur le graphe d'accessibilité, il est nécessaire de modifier la notation mise en place page 35. La figure 2.10 correspond à la nouvelle représentation graphique des états du système.

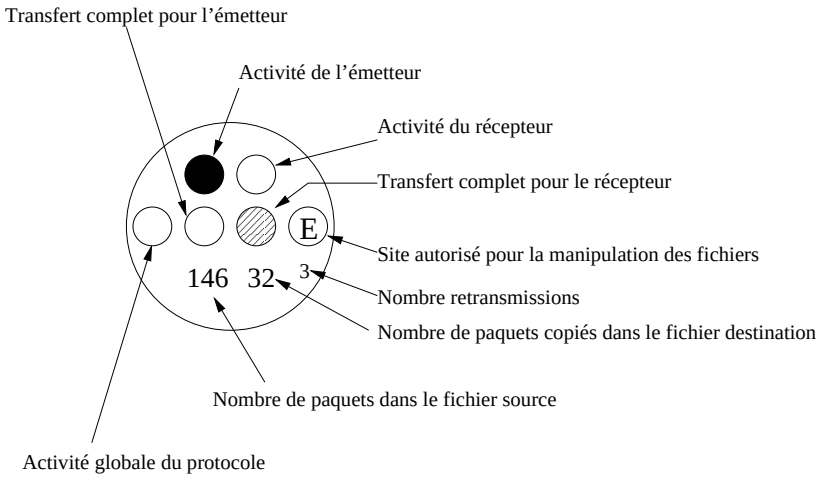


Figure 2.10: BRP - Description de la notation utilisée au raffinement n ° 4

La nouvelle pastille correspond aux deux états possibles de la variable $Site\_Actif_4$. Elle contient la lettre R si le contrôle est donné au site récepteur et la lettre E si s'est l'émetteur qui est autorisé à la manipulation des fichiers.

Comme le montre la figure 2.11, on introduit un nouvel événement nommé $\mathit{RéceptionAck}_4$. Il correspond à la réception de l'acquittement d'un paquet différent du dernier paquet composant le fichier à copier. Il ne peut y avoir de réception d'acquittement que lorsqu'un paquet à été envoyé vers le site récepteur. L'événement $\mathit{ArrêtEmetteur}_4$ correspond non seulement à l'arrêt du site émetteur mais aussi à la réception de l'acquittement du dernier paquet du fichier.

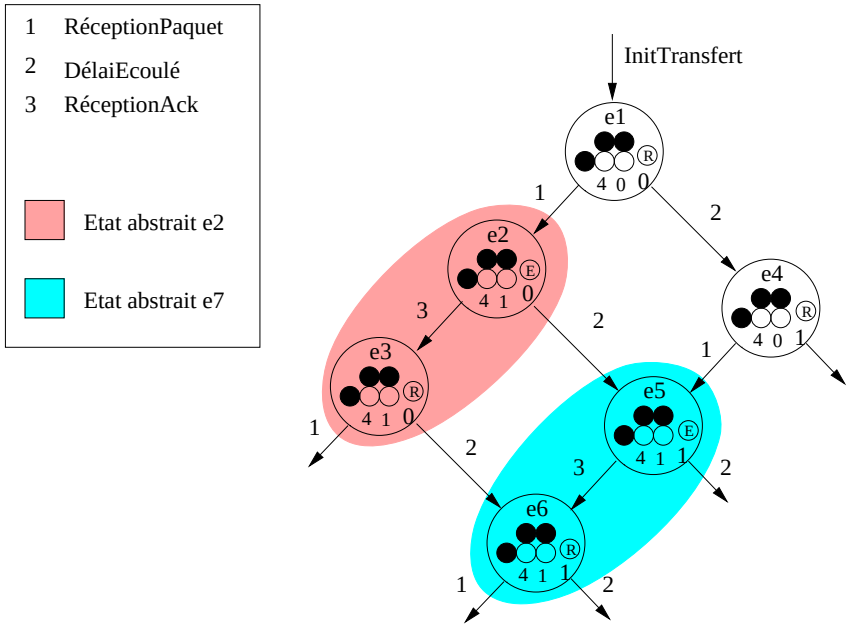


Figure 2.11: BRP - Partie du graphe d'accessibilité du raffinement n ° 4, raffinement des états abstraits $e1$, $e2$, $e6$ et $e7$

La figure 2.11 représente une partie seulement du graphe d'accessiblité de ce niveau de raffinement. Max $*$ (Nombre de parquets dans le fichier source $-$ 1) nouveaux états sont venus s'ajoutés au graphe du niveau précédent.

L'initialisation du système est identique à celle du système abstrait.

Initialisation	$\triangleq$	$emet\_actif_4 := non \ \|\|$ $recep\_actif_4 := non \ \|\|$ $global\_actif_4 := non$

L'initialisation du protocole est légèrement modifiée par rapport à son abstraction. Le seul site autorisé à travailler sur les fichiers est le récepteur. En effet, il est nécessaire que le fichier du récepteur soit incrémenté avant que celui de l'émetteur ne soit modifié afin de ne pas provoquer des pertes de données.

$InitTransfert_4 \triangleq$ SELECT
$emet\_actif_4 = non \wedge$
$recep\_actif_4 = non \wedge$
$global\_actif_4 = non$
THEN
ANY f
WHERE
$f \in \mathrm{seq}_1(\mathrm{S})$
THEN
$fich\_emet_4 := f$
END ||
$fich\_recep_4 := [\,]$ ||
$emet\_actif_4 := oui$ ||
$recep\_actif_4 := oui$ ||
$emet\_transfert\_ok_4 := non$ ||
$recep\_transfert\_ok_4 := non$ ||
$global\_actif_4 := oui$ ||
$NbRéémis_4 := 0$ ||
$Site\_Actif_4 = récepteur$
END

L'événement $ArrêtTransfert_4$est identique à son abstraction.

$ArrêtTransfert_4 \triangleq$ SELECT
$emet\_actif_4 = non \wedge$
$recep\_actif_4 = non \wedge$
$global\_actif_4 = oui$
THEN
$global\_actif_4 := non$
END

Contrairement à son abstraction, $ArrêtEmetteur_4$ modifie le fichier à transmettre c'est à dire qu'il retire à ce dernier le paquet précédemment envoyé. Cette opération ne peut évidemment être réalisée que si le contrôle est donné au site émetteur.

$ArrêtEmetteur_4 \triangleq$ SELECT
$emet\_actif_4 = oui \wedge$
$recep\_actif_4 = non \wedge$
$recep\_transfert\_ok_4 = oui \wedge$
$Site\_Actif_4 = émetteur \wedge$
$\mathrm{tail}(\ fich\_emet_4\ ) = [\,]$
THEN
$emet\_actif_4 := non$ ||
$emet\_transfert\_ok_4 := oui$ ||
$fich\_emet_4 := \mathrm{tail}(\ fich\_emet_4\ )$
END

L'événement $ArrêtRécepteur_4$ doit lui aussi tenir compte du transfert de fichier paquet par paquet. Il ne peut être activé que si le contrôle est donné au site récepteur. D'autre part, seul le fichier destination doit être modifié par cet événement. Le fichier source est traité par l'événement $ArrêtEmetteur_4$ et n'apparait donc plus ici.

$ArrêtRécepteur_4 \triangleq$ SELECT
$recep\_actif_4 = oui \wedge emet\_actif_4 = oui \wedge$
tail $(\ fich\_emet_4 \ ) = [\,] \wedge$
$Site\_Actif_4 = récepteur$
THEN
$fich\_recep_4 := fich\_recep_4 \frown$ first $(\ fich\_emet_4 \ ) \ \|$
$recep\_actif_4 := non \ \|$
$recep\_transfert\_ok_4 := oui \ \|$
$Site\_Actif_4 := émetteur$
END

Les événements $ErreurEmetteur_4$, $DelaiEcoulé_4$ et $ErreurRécepteur_4$ sont identiques à leurs abstractions. Ce dernier n'a plus besoin de mettre à jour la variable $fich\_recep_4$ car cela est réalisé par les événements de copie morceau par morceau.

$ErreurEmetteur_4 \triangleq$ SELECT
$emet\_actif_4 = oui \wedge$
$NbRéémis_4 = \text{MAX}$
THEN
$emet\_actif_4 := non \ \|$
$emet\_transfert\_ok_4 := non \ \|$
$NbRéémis_4 := \text{MAX} + 1 \ \|$
$Site\_Actif_4 := émetteur$
END

$ErreurRécepteur_4 \triangleq$ SELECT
$recep\_actif_4 = oui \wedge$
$emet\_actif_4 = non \wedge$
$emet\_transfert\_ok_4 = non \wedge$
$NbRéémis_4 = (\text{MAX} + 1)$
THEN
$emet\_actif_4 := non \ \|$
$recep\_transfert\_ok_4 := non$
END

$DelaiEcoulé_4 \triangleq$ SELECT
$emet\_actif_4 = oui \wedge$
$NbRéémis_4 < \text{MAX}$
THEN
$NbRéémis_4 := NbRéémis_4 + 1$
END

La réception de paquets par le site récepteur est similaire à l'arrêt de ce dernier. Mais ici, cet événement ne peut être déclenché que lorsqu'un paquet peut être reçu et qu'il ne s'agisse pas du dernier composant le fichier source.

$RéceptionPaquet_4 \triangleq$ SELECT
$recep\_actif_4 = oui \wedge$
$emet\_actif_4 = oui \wedge$
tail ($fich\_emet_4$) $\neq [\,] \wedge$
$Site\_Actif_4 = récepteur$
THEN
$fich\_recep_4 := fich\_recep_4 \frown$ first ($fich\_emet_4$) ||
$Site\_Actif_4 := émetteur$
END

La réception d'un acquittement met à jour le fichier à transférer. De plus cet événement ne peut être déclenché que lorsque l'émetteur possède le contrôle. Le compteur du nombre de réémissions est remis à zéro.

$RéceptionAck_4 \triangleq$ SELECT
$emet\_actif_4 = oui \wedge recep\_actif_4 = oui \wedge$
$Site\_Actif_4 = émetteur \wedge$
tail ($fich\_emet_4$) $\neq [\,]$
THEN
$NbRéémis_4 := 0$ ||
$Site\_Actif_4 := récepteur$ ||
$fich\_emet_4 :=$ tail ($fich\_emet_4$)
END

2.2.6 Raffinement n ° 5

But

Lors de ce raffinement, on introduit une nouvelle entité entre le site récepteur et le site émetteur : le canal de transmission. On va donc ajouter des variables permettant de représenter le transport des données ainsi que des événements permettant aux sites d'utiliser le support de communication. Les nouveaux événements sont :

- $EnvoiPaquet_5$: cet événement permet à l'émetteur d'envoyer un paquet de données en sachant que celui-ci n'est pas le dernier composant le fichier.
- $EnvoiDernierPaquet_5$: idem sauf qu'il s'agit du dernier paquet composant le fichier source.

Il est aussi nécessaire de renforcer la garde des événements $RéceptionPaquet_5$ et $ArrêtRécepteur_5$.

Données

On garde les variables du niveau précédent au changement d'indice près. On introduit les variables suivantes :

- $canal\_Donnée_5$: variable représentant le paquet de données en cours de transfert.
- $canal\_Dern\_Paquet_5$: cette variable indique si le paquet contenu dans la variable précédente est le dernier composant le fichier.
- $etat\_Canal_5$: l'état du canal est représenté par cette donnée. Elle permet de savoir si le support de transmission peut recevoir ou envoyer des données ou encore s'il peut faire parvenir à l'émetteur un acquittement.

Invariant de collage

– – Les variables suivantes sont identiques à leurs abstractions
$fich\_emet_5 = fich\_emet_4 \wedge$
$fich\_recep_5 = fich\_recep_4 \wedge$
$emet\_actif_5 = emet\_actif_4 \wedge$
$recep\_actif_5 = recep\_actif_4 \wedge$
$emet\_transfert\_ok_5 = emet\_transfert\_ok_4 \wedge$
$recep\_transfert\_ok_5 = recep\_transfert\_ok_4 \wedge$
$global\_actif_5 = global\_actif_4 \wedge$
$NbRéémis_5 = NbRéémis_4 \wedge$
$Site\_Actif_5 = Site\_Actif_4 \wedge$

– – Typage des variables concernant le support de communication, soit S le type des éléments des fichiers
$canal\_Donnée_5 \in \mathrm{S} \wedge$
$canal\_Dern\_Paquet_5 \in \{oui, non\} \wedge$
$etat\_Canal_5 \in \{envoi\_donnée, récep\_donnée, envoi\_ack\} \wedge$

– – Quand le canal peut contient des données alors le fichier source abstrait n'est pas vide
$etat\_Canal_5 = récep\_donnée \Rightarrow fich\_emet_4 \neq [\,] \wedge$

– – Lorsque le canal de transmission contient des données alors il indique qu'il s'agit du dernier bloc si la queue du fichier source est vide
$etat\_Canal_5 = récep\_donnée \Rightarrow \left(\; canal\_Dern\_Paquet_5 = oui \Leftrightarrow \mathrm{tail}\left(\; fich\_emet_5 \; \right) = [\,] \; \right) \wedge$

– – La donnée contenue dans le canal de transmission correspond au premier bloc du fichier source
$etat\_Canal_5 = récep\_donnée \Rightarrow canal\_Donnée_5 = \mathrm{first}\left(\; fich\_emet_5 \; \right)$

Propriétés temporelles

Propriétés temporelles raffinées

– – Le protocole est modélisé comme un démon. Il attend une nouvelle demande de transfert après qu'il ait terminé de copier le fichier
$${}_hB_5^1 \quad \Box \left(\begin{array}{l} global\_actif_5 = non \wedge emet\_actif_5 = non \wedge recep\_actif_5 = non \\ \Rightarrow \bigcirc \left(\begin{array}{l} global\_actif_5 = oui \wedge \\ emet\_actif_5 = oui \wedge recep\_actif_5 = oui \end{array} \right) \end{array} \right)$$

– – Les deux sites se désactivent fatalement
${}_cB_5^2 \quad \Box \left(\; emet\_actif_5 = oui \Rightarrow \Diamond \left(\; emet\_actif_5 = non \; \right) \; \right)$
${}_cB_5^3 \quad \Box \left(\; recep\_actif_5 = oui \Rightarrow \Diamond \left(\; recep\_actif_5 = non \; \right) \; \right)$

– – **Lorsque le protocole est arrêté, il peut être dans l'un des trois états cités page 14**

$$ {}_{1,2,3,f,d}B_5^4 \quad \Box \left(\begin{array}{l} global\_actif_5 = oui \wedge \bigcirc \left(\ global\_actif_5 = non \ \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} emet\_transfert\_ok_5 = oui \wedge \\ recep\_transfert\_ok_5 = oui \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_5 = non \wedge \\ recep\_transfert\_ok_5 = non \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_5 = non \wedge \\ recep\_transfert\_ok_5 = oui \end{array} \right) \end{array} \right) \end{array} \right) $$

– – **Lorsque les deux sites sont inactifs, le protocole est lui aussi inactivé. Le protocole devient inactif après l'arrêt du dernier site en fonctionnement.**

$$ {}_cB_5^5 \quad \Box \left(\begin{array}{l} global\_actif_5 = oui \wedge emet\_actif_5 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_5 = oui \wedge emet\_actif_5 = non \wedge \\ recep\_actif_5 = non \end{array} \right) \\ \Rightarrow \bigcirc \bigcirc \left(\ global\_actif_5 = non \ \right) \end{array} \right) $$

$$ {}_cB_5^6 \quad \Box \left(\begin{array}{l} global\_actif_5 = oui \wedge recep\_actif_5 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_5 = oui \wedge emet\_actif_5 = non \wedge \\ recep\_actif_5 = non \end{array} \right) \\ \Rightarrow \bigcirc \bigcirc \left(\ global\_actif_5 = non \ \right) \end{array} \right) $$

– – **Les deux sites se désactivent l'un après l'autre et consécutivement**

$$ {}_cB_5^7 \quad \Box \left(\begin{array}{l} emet\_actif_5 = oui \wedge recep\_actif_5 = oui \wedge \\ \bigcirc \left(\ emet\_actif_5 = non \wedge recep\_actif_5 = oui \ \right) \\ \Rightarrow \bigcirc \bigcirc \left(\ recep\_actif_5 = non \ \right) \end{array} \right) $$

$$ {}_cB_5^8 \quad \Box \left(\begin{array}{l} emet\_actif_5 = oui \wedge recep\_actif_5 = oui \wedge \\ \bigcirc \left(\ recep\_actif_5 = non \wedge emet\_actif_5 = oui \ \right) \\ \Rightarrow \left(\begin{array}{l} NbRéémis_5 < \text{MAX} \wedge \\ emet\_actif_5 = oui \end{array} \right) \mathcal{U} \left(\ emet\_actif_5 = non \ \right) \end{array} \right) $$

– – **Quand l'expéditeur sait que le protocole s'est bien terminé alors le récepteur le savait déjà**

$$ {}_eB_5^9 \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} emet\_actif_5 = non \wedge emet\_transfert\_ok_5 = oui \wedge \\ global\_actif_5 = oui \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} recep\_actif_5 = non \wedge recep\_transfert\_ok_5 = oui \wedge \\ global\_actif_5 = oui \end{array} \right) \end{array} \right) $$

– – **Quand le récepteur sait que le protocole s'est mal terminé alors l'émetteur le savait déjà**

$$ {}_eB_5^{10} \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} recep\_actif_5 = non \wedge recep\_transfert\_ok_5 = oui \wedge \\ global\_actif_5 = oui \wedge NbRéémis_5 = \left(\ \text{MAX} + 1 \ \right) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} emet\_actif_5 = non \wedge emet\_transfert\_ok_5 = oui \wedge \\ global\_actif_5 = oui \wedge NbRéémis_5 = \text{MAX} \end{array} \right) \end{array} \right) $$

– – **Lorsque le compteur du nombre de *time out* arrive à son maximum, l'émetteur est arrêté sur une erreur et le compteur est de nouveau incrémenté pour permettre l'arrêt du récepteur**

$$ {}_{c,g}B_5^{11} \quad \Box \left(\begin{array}{l} NbRéémis_5 = \text{MAX} \wedge emet\_actif_5 = oui \wedge recep\_actif_5 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} NbRéémis_5 = \left(\ \text{MAX} + 1 \ \right) \wedge \\ emet\_actif_5 = non \wedge \\ emet\_transfert\_ok_5 = non \end{array} \right) \end{array} \right) $$

– – **Lorsque le site émetteur s'est arrêté sur une erreur, le récepteur ne peut que détecter lui aussi une anomalie.**

$$ {}_eB_5^{12} \quad \Box \left(\begin{array}{l} NbRéémis_5 = (\text{MAX} + 1) \wedge recep\_actif_5 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} recep\_actif_5 = non \wedge \\ recep\_transfert\_ok_5 = non \end{array} \right) \end{array} \right) $$

– – **Les données ajoutées au fichier destination proviennent du fichier source**

$$ \forall r \in \text{seq(S)}, \forall e \in \text{seq(S)} $$

$$ {}_aB_5^{13} \quad \Box \left(\begin{array}{l} r = fich\_recep_5 \wedge e = fich\_emet_5 \wedge e \neq [\,] \wedge \\ emet\_actif_5 = oui \wedge recep\_actif_5 = oui \wedge Site\_Actif_5 = récepteur \wedge \\ etat\_Canal_5 = envoi\_donnée \wedge \\ \left(\; fich\_emet_5 = e \wedge canal\_Donnée_5 = \text{first} \left(\; e \; \right) \; \right) \; \mathcal{U} \; \left(\; fich\_emet_5 = \text{tail} \left(\; e \; \right) \; \right) \\ \Rightarrow \left(\begin{array}{l} Site\_Actif_5 = récepteur \wedge \\ fich\_recep_5 = r \wedge \\ fich\_emet_5 = e \wedge \\ etat\_Canal_5 = envoi\_donnée \end{array} \right) \mathcal{U} \left(\begin{array}{l} Site\_Actif_5 = émetteur \wedge \\ fich\_recep_5 = r \frown canal\_Donnée_5 \wedge \\ fich\_emet_5 = e \wedge \\ etat\_Canal_5 = récep\_donnée \end{array} \right) \end{array} \right) $$

– – **Lorsque le fichier destination est incrémenté, le contrôle est donné à l'émetteur pour qu'il puisse décrémenter le fichier source**

$$ \forall r \in \text{seq(S)}, \forall e \in \text{seq(S)} $$

$$ {}_aB_5^{14} \quad \Box \left(\begin{array}{l} emet\_actif_5 = oui \wedge recep\_actif_5 = oui \wedge \\ e = fich\_emet_5 \wedge e \neq [\,] \wedge \\ r = fich\_recep_5 \wedge \\ Site\_Actif_5 = récepteur \wedge etat\_Canal_5 = récep\_donnée \wedge \\ \bigcirc \left(\begin{array}{l} emet\_actif_5 = oui \wedge recep\_actif_5 = oui \wedge \\ fich\_emet_5 = r \frown \text{first} \left(\; e \; \right) \end{array} \right) \\ \Rightarrow \bigcirc \left(\; Site\_Actif_5 = émetteur \wedge envoi\_ack \; \right) \end{array} \right) $$

– – **Lorsque le fichier source est décrémenté, le contrôle est donné au récepteur pour qu'il puisse lire le prochain paquet**

$$ \forall e \in \text{seq(S)} $$

$$ {}_aB_5^{15} \quad \Box \left(\begin{array}{l} emet\_actif_5 = oui \wedge \\ e = fich\_emet_5 \wedge e \neq [\,] \wedge \\ Site\_Actif_5 = émetteur \wedge etat\_Canal_5 = envoi\_donnée \wedge \\ \bigcirc \left(\begin{array}{l} fich\_emet_5 = \text{tail} \left(\; e \; \right) \wedge \\ emet\_actif_5 = oui \end{array} \right) \\ \Rightarrow \bigcirc \left(\; Site\_Actif_5 = récepteur \wedge etat\_Canal_5 = récep\_donnée \; \right) \end{array} \right) $$

– – **Lorsque le protocole est initialisé, le contrôle est automatiquement donné au récepteur**

$$ B_5^{16} \quad \Box \left(\begin{array}{l} emet\_actif_5 = non \wedge recep\_actif_5 = non \wedge \\ global\_actif_5 = non \wedge \\ \bigcirc \left(\begin{array}{l} emet\_actif_5 = oui \wedge recep\_actif_5 = oui \wedge \\ global\_actif_5 = oui \end{array} \right) \\ \Rightarrow \bigcirc \left(\; Site\_Actif_5 = récepteur \; \right) \end{array} \right) $$

NOUVELLES PROPRIÉTÉS TEMPORELLES

– – **Si le récepteur peut lire les données du canal alors ce dernier les a forcément lu dans le fichier source. Autrement dit, si le canal possède une donnée pouvant être lue, c'est qu'il la possédait déjà ou qu'il venait de la lire dans le fichier source. Cette propriété exprime en partie le principe de fonctionnement a**

$$ {}_aB_5^{17} \quad \Box \left(\begin{array}{l} \bigcirc \left(\; recep\_actif_5 = oui \wedge etat\_Canal_5 = récep\_donnée \; \right) \\ \Rightarrow \left(\; etat\_Canal_5 = envoi\_donnée \vee etat\_Canal_5 = récep\_donnée \; \right) \end{array} \right) $$

– – **Si l'émetteur peut recevoir un acquittement alors le récepteur a déjà reçu un paquet. Autrement dit, si le canal contient un acquittement alors soit il y avait déjà un acquittement soit il était dans un état réceptif**

$$ B_5^{18} \quad \Box \left(\begin{array}{l} \bigcirc \left(\; emet\_actif_5 = oui \wedge etat\_Canal_5 = envoi\_ack \; \right) \\ \Rightarrow \left(\; etat\_Canal_5 = récep\_donnée \vee etat\_Canal_5 = envoi\_ack \; \right) \end{array} \right) $$

– – **Si l'émetteur peut envoyer des données alors il a reçu un acquittement ou le protocole vient de débuter**

$$ B_5^{19} \quad \Box \left(\begin{array}{l} \bigcirc \left(\; emet\_actif_5 = oui \wedge etat\_Canal_5 = envoi\_donnée \; \right) \\ \Rightarrow \left(\begin{array}{l} \left(\; emet\_actif_5 = oui \wedge etat\_Canal_5 = envoi\_ack \; \right) \vee \\ \left(\begin{array}{l} emet\_actif_5 = non \wedge recep\_actif_5 = non \wedge \\ global\_actif_5 = non \end{array} \right) \end{array} \right) \end{array} \right) $$

Evénements

MODIFICATION DE LA NOTATION

L'introduction du canal de transmission et des variables le représentant nécessite une nouvelle modification de la notation utilisée pour représenter les états. La figure 2.12 correspond aux nouveaux états qui composeront le graphe d'accessiblité.

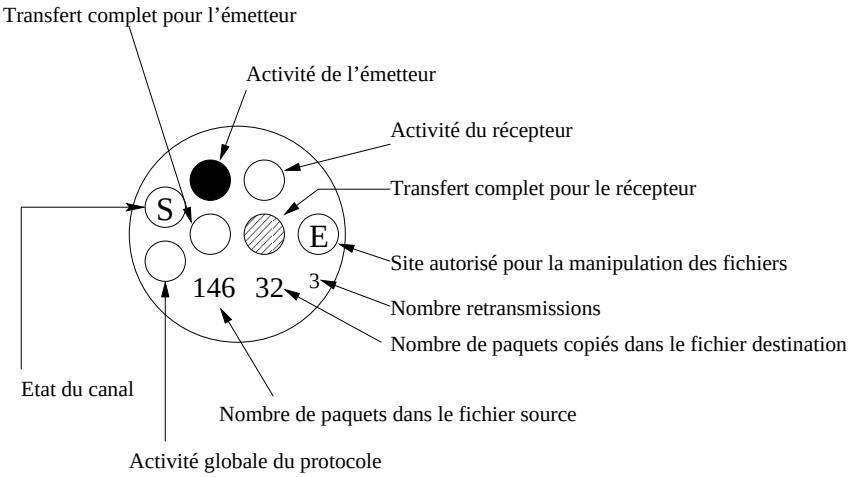


Figure 2.12: BRP - Description de la notation utilisée au raffinement n ° 5

La variable $etat\_Canal_5$ est représentée par une pastille contenant l'une des lettres S, R ou A qui correspondent au fait que le canal de transmission puisse respectivement envoyer des données, en recevoir ou faire parvenir un acquittement au site récepteur. Nous estimons qu'ils n'est pas nécessaire de repésenter les variables $canal\_Donnée_5$ et $canal\_Dern\_Paquet_5$. En effet, la première n'apporte rien de particulier au graphe d'accessiblité et la valeur de la seconde peu être déduite à partir des variables représentant les états des fichiers.

Dans ce niveau de raffinement deux nouveaux événements apparaissent : $EnvoiPaquet_5$ et $EnvoiDernierPaquet_5$. Les modifications, reportées sur le graphe d'accessibilité de la figure 2.13, permettent de modéliser un canal de transmission entre les deux sites et ainsi de distinguer totalement ces derniers : les variables d'un site ne seront pas utilisées par un autre site.

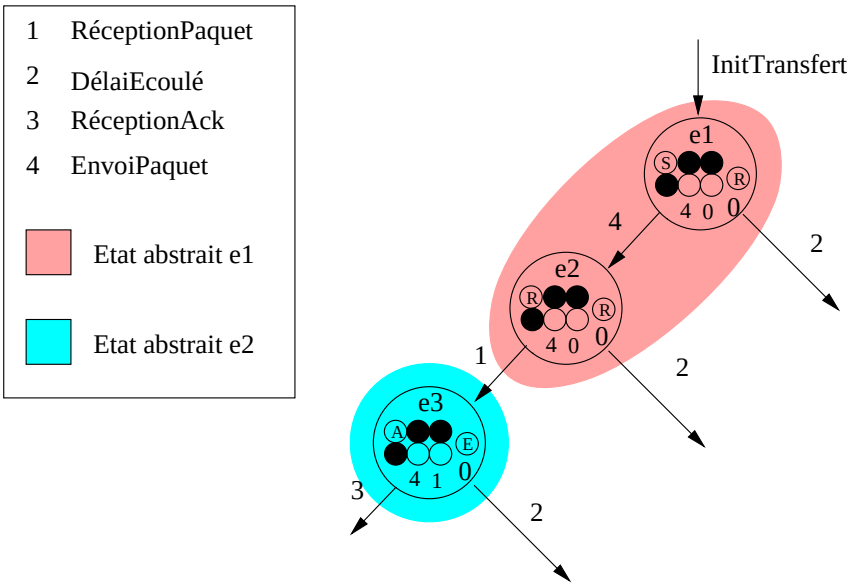


Figure 2.13: BRP - Partie du graphe d'accessibilité du raffinement n° 5, raffinement des états abstraits $e1$ et $e2$

La figure 2.13 représente le raffinement des états abstraits $e1$ et $e2$ du graphe 2.11 page 43. On constate que tous les états où l'événement $RéceptionPaquet_5$ est directement accessible sont raffinés par deux états reliés par l'événement $EnvoiPaquet_5$. Le nombre détats dans le système croit encore de Max $*$ (Nombre de paquets dans le ficheir source -2). Le graphe d'accessiblité ne représente pas tous les raffinements. En effet l'événement $EnvoiDernierPaquet_5$ est activable avant $ArrêtRécepteur_5$. Tous les états abstraits où ce dernier est directement éxécutable sont raffinés par un chemin composé de $EnvoiDernierPaquet_5$ et de $ArrêtRécepteur_5$.

L'initialisation du système est identique à celle du système abstrait.

Initialisation	$\triangleq$	$fich\_emet_5 := non \ \|\|$ $fich\_recep_5 := non \ \|\|$ $global\_actif_5 := non$

L'initialisation du protocole comporte une nouvelle initialisation. La variable $etat\_Canal_5$ doit indiquer que le canal de transmission est près à recevoir des données en provenance de l'émetteur.

$InitTransfert_5 \triangleq$

SELECT
$emet\_actif_5 = non \wedge$
$recep\_actif_5 = non \wedge$
$global\_actif_5 = non$
THEN
ANY f
WHERE
$f \in \mathrm{seq}_1(\mathrm{S})$
THEN
$fich\_emet_5 := f$
END ||
$fich\_recep_5 := [\,]$ ||
$emet\_actif_5 := oui$ ||
$recep\_actif_5 := oui$ ||
$emet\_transfert\_ok_5 := non$ ||
$recep\_transfert\_ok_5 := non$ ||
$global\_actif_5 := oui$ ||
$NbRéémis_5 := 0$ ||
$Site\_Actif_5 := récepteur$ ||
$etat\_Canal_5 := envoi\_donnée$
END

L'événements $ArrêtTransfert_5$ est identique à son abstraction.

$ArrêtTransfert_5 \triangleq$

SELECT
$emet\_actif_5 = non \wedge$
$recep\_actif_5 = non \wedge$
$global\_actif_5 = oui$
THEN
$global\_actif_5 := non$
END

$ArrêtEmetteur_5$ diffère de son abstraction car il tient compte du support de transmission et n'utilise plus les variables concernant le site récepteur.

$ArrêtEmetteur_5 \triangleq$

SELECT
$emet\_actif_5 = oui \wedge$
$recep\_transfert\_ok_5 = oui \wedge$
$Site\_Actif_5 = émetteur \wedge$
$\mathrm{tail}(\ fich\_emet_5\ ) = [\,] \wedge$
$etat\_Canal_5 = envoi\_ack$
THEN
$emet\_actif_5 := non$ ||
$emet\_transfert\_ok_5 := oui$ ||
$fich\_emet_5 := \mathrm{tail}(\ fich\_emet_5\ )$
END

L'arrêt du récepteur diffère légèrement de son abstraction. Le site récepteur envoit un acquittement par l'intermédiraire du canal de transmission.

$ArrêtRécepteur_5 \triangleq$ SELECT
$recep\_actif_5 = oui \wedge$
$canal\_Dern\_Paquet_5 = oui \wedge$
$Site\_Actif_5 = récepteur$
THEN
$fich\_recep_5 := fich\_recep_5 \frown canal\_Donnée_5 \ \|$
$recep\_actif_5 := non \ \|$
$recep\_transfert\_ok_5 := oui \ \|$
$Site\_Actif_5 := émetteur \ \|$
$etat\_Canal_5 := envoi\_ack$
END

A ce niveau de raffinement, l'événement $ErreurEmetteur_5$ ne peut se déclencher que si l'émetteur n'est pas en train d'envoyer un bloc au canal de transmission ($etat\_Canal_5 \neq envoi\_donnée$).

$ErreurEmetteur_5 \triangleq$ SELECT
$emet\_actif_5 = oui \wedge$
$NbRéémis_5 = \text{MAX} \wedge$
$etat\_Canal_5 \neq envoi\_donnée$
THEN
$emet\_actif_5 := non \ \|$
$emet\_transfert\_ok_5 := non \ \|$
$NbRéémis_5 := \text{MAX} + 1 \ \|$
$Site\_Actif_5 := émetteur$
END

L'événements $ErreurRécepteur_5$ est identique à son abstraction.

$ErreurRécepteur_5 \triangleq$ SELECT
$recep\_actif_5 = oui \wedge$
$emet\_actif_5 = non \wedge$
$emet\_transfert\_ok_5 = non \wedge$
$NbRéémis_5 = (\text{MAX} + 1)$
THEN
$emet\_actif_5 := non \ \|$
$recep\_transfert\_ok_5 := non$
END

Lorsque le temps d'attente d'un acquittement a été atteint, le site émetteur doit renvoyer le bloc qui n'a peut être pas été reçu par son vis à vis. Il est nécessaire de modifier la garde de l'événement afin qu'il ne se déclenche pas lorsque le canal est déjà dans l'état *envoi_donnée*.

$DelaiEcoulé_5 \triangleq$ SELECT
$emet\_actif_5 = oui \wedge$
$NbRéémis_5 < \text{MAX} \wedge$
$etat\_Canal_5 \neq envoi\_donnée$
THEN
$NbRéémis_5 := NbRéémis_5 + 1$
END

Le site émetteur peut recevoir un acquittement s'il a été envoyé par le récepteur. Cette condition est modélisée par le raffinement de la garde de l'événement qui n'autorise pas son éxécution si le canal de transmission ne contient pas un acquittement. Une fois ce dernier traité, le canal de transmission peut à nouveau être utilisé pour copier un paquet de données.

$RéceptionAck_5 \triangleq$
SELECT
$emet\_actif_5 = oui \wedge$
$Site\_Actif_5 = émetteur \wedge$
tail ($fich\_emet_5$) $\neq [\,] \wedge$
$etat\_Canal_5 = envoi\_ack$
THEN
$NbRéémis_5 := 0$ ||
$Site\_Actif_5 := récepteur$ ||
$fich\_emet_5 :=$ tail ($fich\_emet_5$) ||
$etat\_Canal_5 := envoi\_donnée$
END

L'événement $RéceptionPaquet_5$ doit lui aussi tenir compte de létat du canal de transmission et renvoyer un acquittement au site émetteur par l'intermédiaire du canal de transmission.

$RéceptionPaquet_5 \triangleq$
SELECT
$recep\_actif_5 = oui \wedge$
$canal\_Dern\_Paquet_5 = non \wedge$
$Site\_Actif_5 = récepteur \wedge$
$etat\_Canal_5 = récep\_donnée$
THEN
$fich\_recep_5 := fich\_recep_5 \frown canal\_Donnée_5$ ||
$Site\_Actif_5 := émetteur$ ||
$etat\_Canal_5 := envoi\_ack$
END

Le nouvel événement $EnvoiPaquet_5$ est propre au canal de transmission. Il permet à celui-ci de prendre possession du prochain paquet du fichier source. Il est toutefois déclenchable uniquement si l'envoi de données est autorisé. Cette événement ne prend en compte que les paquet différents du dernier.

$EnvoiPaquet_5 \triangleq$
SELECT
$emet\_actif_5 = oui \wedge$
$etat\_Canal_5 = envoi\_donnée \wedge$
tail ($fich\_emet_5$) $\neq [\,]$
THEN
$etat\_Canal_5 := récep\_donnée$ ||
$canal\_Donnée_5 :=$ first ($fich\_emet_5$) ||
$canal\_Dern\_Paquet_5 := non$
END

$EnvoiDernierPaquet_5$ est similaire à l'événement précédent mais ici seul le dernier bloc du fichier source est pri en compte.

$EnvoiDernierPaquet_5 \triangleq$ SELECT
$emet\_actif_5 = oui \wedge$
$etat\_Canal_5 = envoi\_donnée \wedge$
tail ($fich\_emet_5$) = [ ]
THEN
$etat\_Canal_5 := récep\_donnée \parallel$
$canal\_Donnée_5 :=$ first ($fich\_emet_5$) $\parallel$
$canal\_Dern\_Paquet_5 := oui$
END

2.2.7 Raffinement n ° 6

But

Le but de ce raffinement est de distribuer le bit de contrôle $Site\_Actif_6$ sur chaque site. Le bit de contrôle de l'émetteur sera envoyé avec les données. Le récepteur comparera le bit reçu avec le sien.

Données

On garde les variables du niveau précédent à l'exception de $Site\_Actif_5$. Cette dernière est remplacée par les variables suivantes :

- $emet\_bit\_ctrl_6$: cette variable représente le bit de contrôle situé chez l'émetteur.
- $recep\_bit\_ctrl_6$: idem chez le récepteur.
- $canal\_bit\_ctrl_6$: cette variable représente le bit de contrôle de l'émetteur qui transite par le canal pour être utilisé par le récepteur.

Invariant de collage

– – **Les variables suivantes sont identiques à leurs abstractions**
$fich\_emet_6 = fich\_emet_5 \wedge$
$fich\_recep_6 = fich\_recep_5 \wedge$
$emet\_actif_6 = emet\_actif_5 \wedge$
$recep\_actif_6 = recep\_actif_5 \wedge$
$emet\_transfert\_ok_6 = emet\_transfert\_ok_5 \wedge$
$recep\_transfert\_ok_6 = recep\_transfert\_ok_5 \wedge$
$global\_actif_6 = global\_actif_5 \wedge$
$NbRéémis_6 = NbRéémis_5 \wedge$
$canal\_Donnée_6 = canal\_Donnée_5 \wedge$
$canal\_Dern\_Paquet_6 = canal\_Dern\_Paquet_5 \wedge$
$etat\_Canal_6 = etat\_Canal_5 \wedge$

– – **Typage des variables représentant les bits de contrôle**
$emet\_bit\_ctrl_6 \in \{\alpha, \beta\} \wedge$
$recep\_bit\_ctrl_6 \in \{\alpha, \beta\} \wedge$
$canal\_bit\_ctrl_6 \in \{\alpha, \beta\} \wedge$

– – **Propriétés des nouveaux bits de contrôle**
$emet\_bit\_ctrl_6 = recep\_bit\_ctrl_6 \Leftrightarrow Site\_Actif_5 = récepteur \wedge$
$etat\_Canal_6 \neq envoi\_donnée \Rightarrow canal\_bit\_ctrl_6 = emet\_bit\_ctrl_6 \wedge$

– – **Propriété de négation nécessaire aux nouveaux bits**
$\neg\alpha \Leftrightarrow \beta\wedge$
$\neg\beta \Leftrightarrow \alpha$

Propriétés temporelles

PROPRIÉTÉS TEMPORELLES RAFFINÉES

– – **Le protocole est modélisé comme un démon. Il attend une nouvelle demande de transfert après qu'il ait terminé de copier le fichier**

$$ {}_hB_6^1 \quad \Box \left(\begin{array}{l} global\_actif_6 = non \wedge emet\_actif_6 = non \wedge recep\_actif_6 = non \\ \Rightarrow \bigcirc \left(\begin{array}{l} global\_actif_6 = oui\wedge \\ emet\_actif_6 = oui \wedge recep\_actif_6 = oui \end{array} \right) \end{array} \right) $$

– – **Les deux sites ne se désactivent fatalement**

$$ {}_cB_6^2 \quad \Box \left(\; emet\_actif_6 = oui \Rightarrow \Diamond \left(\; emet\_actif_6 = non \; \right) \right) $$
$$ {}_cB_6^3 \quad \Box \left(\; recep\_actif_6 = oui \Rightarrow \Diamond \left(\; recep\_actif_6 = non \; \right) \right) $$

– – **Lorsque le protocole est arrêté, il peut être dans l'un des trois états cités page 14**

$$ {}_{1,2,3,f,d}B_6^4 \quad \Box \left(\begin{array}{l} global\_actif_6 = oui \wedge \bigcirc \left(\; global\_actif_6 = non \; \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} emet\_transfert\_ok_6 = oui\wedge \\ recep\_transfert\_ok_6 = oui \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_6 = non\wedge \\ recep\_transfert\_ok_6 = non \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_6 = non\wedge \\ recep\_transfert\_ok_6 = oui \end{array} \right) \end{array} \right) \end{array} \right) $$

– – **Lorsque les deux sites sont inactifs, le protocole est lui aussi inactivé. Le protocole devient inactif après l'arrêt du dernier site en fonctionnement.**

$$ {}_cB_6^5 \quad \Box \left(\begin{array}{l} global\_actif_6 = oui \wedge emet\_actif_6 = oui\wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_6 = oui \wedge emet\_actif_6 = non\wedge \\ recep\_actif_6 = non \end{array} \right) \\ \Rightarrow \bigcirc \; \bigcirc \left(\; global\_actif_6 = non \; \right) \end{array} \right) $$
$$ {}_cB_6^6 \quad \Box \left(\begin{array}{l} global\_actif_6 = oui \wedge recep\_actif_6 = oui\wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_6 = oui \wedge emet\_actif_6 = non\wedge \\ recep\_actif_6 = non \end{array} \right) \\ \Rightarrow \bigcirc \; \bigcirc \left(\; global\_actif_6 = non \; \right) \end{array} \right) $$

– – **Les deux sites se désactivent l'un après l'autre et consécutivement**

$$ {}_cB_6^7 \quad \Box \left(\begin{array}{l} emet\_actif_6 = oui \wedge recep\_actif_6 = oui\wedge \\ \bigcirc \left(\; emet\_actif_6 = non \wedge recep\_actif_6 = oui \; \right) \\ \Rightarrow \bigcirc \; \bigcirc \left(\; recep\_actif_6 = non \; \right) \end{array} \right) $$
$$ {}_cB_6^8 \quad \Box \left(\begin{array}{l} emet\_actif_6 = oui \wedge recep\_actif_6 = oui\wedge \\ \bigcirc \left(\; recep\_actif_6 = non \wedge emet\_actif_6 = oui \; \right) \\ \Rightarrow \left(\begin{array}{l} NbRéémis_6 < \text{MAX}\wedge \\ emet\_actif_6 = oui \end{array} \right) \mathcal{U} \left(\; emet\_actif_6 = non \; \right) \end{array} \right) $$

– – **Quand l'expéditeur sait que le protocole s'est bien terminé alors le récepteur le savait déjà**

$$ {}_{ed}B_6^9 \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} emet\_actif_6 = non \wedge emet\_transfert\_ok_6 = oui \wedge \\ global\_actif_6 = oui \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} recep\_actif_6 = non \wedge recep\_transfert\_ok_6 = oui \wedge \\ global\_actif_6 = oui \end{array} \right) \end{array} \right) $$

– – **Quand le récepteur sait que le protocole s'est mal terminé alors l'émetteur le szvait déjà**

$$ {}_{e}B_6^{10} \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} recep\_actif_6 = non \wedge recep\_transfert\_ok_6 = oui \wedge \\ global\_actif_6 = oui \wedge NbRéémis_6 = \left(\ \textsc{Max} + 1 \ \right) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} emet\_actif_6 = non \wedge emet\_transfert\_ok_6 = oui \wedge \\ global\_actif_6 = oui \wedge NbRéémis_6 = \textsc{Max} \end{array} \right) \end{array} \right) $$

– – **Lorsque le compteur du nombre de *time out* arrive à son maximum, l'émetteur est arrêté sur une erreur et le compteur est de nouveau incrémenté pour permettre l'arrêt du récepteur**

$$ {}_{c,g}B_6^{11} \quad \Box \left(\begin{array}{l} NbRéémis_6 = \textsc{Max} \wedge emet\_actif_6 = oui \wedge recep\_actif_6 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} NbRéémis_6 = \left(\ \textsc{Max} + 1 \ \right) \wedge \\ emet\_actif_6 = non \wedge \\ emet\_transfert\_ok_6 = non \end{array} \right) \end{array} \right) $$

– – **Lorsque le site émetteur s'est arrêté sur une erreur, le récepteur ne peut que détecter lui aussi une anomalie.**

$$ {}_{e}B_6^{12} \quad \Box \left(\begin{array}{l} NbRéémis_6 = (\textsc{Max} + 1) \wedge recep\_actif_6 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} recep\_actif_6 = non \wedge \\ recep\_transfert\_ok_6 = non \end{array} \right) \end{array} \right) $$

– – **Les données ajoutées au fichier destination proviennent du fichier source**

$$ \forall r \in \mathrm{seq(S)}, \forall e \in \mathrm{seq(S)}, \forall t \in \{\alpha, \beta\} $$

$$ {}_{a}B_6^{13} \quad \Box \left(\begin{array}{l} r = fich\_recep_6 \wedge e = fich\_emet_6 \wedge e \neq [\,] \wedge \\ emet\_actif_6 = oui \wedge recep\_actif_6 = oui \wedge emet\_bit\_ctrl_6 = t \wedge \\ etat\_Canal_6 = envoi\_donnée \wedge \\ \left(\begin{array}{l} fich\_emet_6 = e \wedge \\ canal\_Donnée_6 = \mathrm{first}\left(\ e \ \right) \end{array} \right) \mathcal{U} \left(\begin{array}{l} emet\_bit\_ctrl_6 = \neg t \wedge \\ fich\_emet_6 = \mathrm{tail}\left(\ e \ \right) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} emet\_bit\_ctrl_6 = t \wedge \\ fich\_recep_6 = r \wedge \\ fich\_emet_6 = e \wedge \\ etat\_Canal_6 = envoi\_donnée \end{array} \right) \mathcal{U} \left(\begin{array}{l} emet\_bit\_ctrl_6 = t \wedge \\ fich\_recep_6 = r \frown canal\_Donnée_6 \wedge \\ fich\_emet_6 = e \wedge \\ etat\_Canal_6 = récep\_donnée \end{array} \right) \end{array} \right) $$

– – **Lorsque le fichier destination est incrémenté, le contrôle est donné à l'émetteur pour qu'il puisse décrémenter le fichier source**

$$ \forall r \in \mathrm{seq(S)}, \forall e \in \mathrm{seq(S)}, \forall t \in \{\alpha, \beta\} $$

$$ {}_{a}B_6^{14} \quad \Box \left(\begin{array}{l} fich\_recep_6 = r \wedge fich\_emet_6 = e \wedge \\ emet\_bit\_ctrl_6 = t \wedge etat\_Canal_6 = récepteur \wedge \\ emet\_actif_6 = oui \wedge recep\_actif_6 = oui \wedge \\ \bigcirc \left(\ fich\_recep_6 = r \frown \mathrm{first}\left(\ e \ \right) \wedge recep\_bit\_ctrl_6 = t \ \right) \\ \Rightarrow \bigcirc \left(\ emet\_bit\_ctrl_6 = \neg t \wedge etat\_Canal_6 = envoi\_ack \ \right) \end{array} \right) $$

– – **Lorsque le fichier source est décrémenté, le contrôle est donné au récepteur pour qu'il puisse lire le prochain paquet**

$$e \in \text{seq(S)}, t \in \{\alpha, \beta\}$$
$$_aB_6^{15} \quad \Box \left(\begin{array}{l} fich\_emet_6 = e \wedge e \neq [\,] \wedge \\ emet\_bit\_ctrl_6 = t \wedge emet\_actif_6 = oui \wedge \\ \bigcirc \left(\begin{array}{l} fich\_emet_6 = \text{tail}\,(\ e\ ) \wedge emet\_actif_6 = oui \wedge \\ emet\_bit\_ctrl_6 = \neg t \end{array} \right) \\ \Rightarrow \bigcirc (\ etat\_Canal_6 = envoi\_donnée \wedge emet\_bit\_ctrl_6 = \neg t\ ) \end{array} \right)$$

– – **Lorsque le protocole est initialisé, le contrôle est automatiquement donné au récepteur**

$$B_6^{16} \quad \Box \left(\begin{array}{l} emet\_actif_6 = non \wedge recep\_actif_6 = non \wedge global\_actif_6 = non \wedge \\ \bigcirc (\ emet\_actif_6 = oui \wedge recep\_actif_6 = oui \wedge global\_actif_6 = oui\ ) \\ \Rightarrow \bigcirc (\ emet\_bit\_ctrl_6 = \alpha \wedge recep\_bit\_ctrl_6 = \alpha\ ) \end{array} \right)$$

– – **Si le récepteur peut lire les données du canal alors ce dernier les a forcément lu dans le fichier source. Autrement dit, si le canal possède une donnée pouvant être lue, c'est qu'il la possédait déjà ou qu'il venait de la lire dans le fichier source**

$$_aB_6^{17} \quad \Box \left(\begin{array}{l} \bigcirc (\ recep\_actif_6 = oui \wedge etat\_Canal_6 = récep\_donnée\ ) \\ \Rightarrow (\ etat\_Canal_6 = envoi\_donnée \vee etat\_Canal_6 = récep\_donnée\ ) \end{array} \right)$$

– – **Si l'émetteur peut recevoir un acquittement alors le récepteur a déjà reçu un paquet**

$$B_6^{18} \quad \Box \left(\begin{array}{l} \bigcirc (\ emet\_actif_6 = oui \wedge etat\_Canal_6 = envoi\_ack\ ) \\ \Rightarrow (\ etat\_Canal_6 = récep\_donnée \vee etat\_Canal_6 = envoi\_ack\ ) \end{array} \right)$$

– – **Si l'émetteur peut envoyer des données alors il a reçu un acquittement ou le protocole vient de débuter**

$$B_6^{19} \quad \Box \left(\begin{array}{l} \bigcirc (\ emet\_actif_6 = oui \wedge etat\_Canal_6 = envoi\_donnée\ ) \\ \Rightarrow \left(\begin{array}{l} (\ emet\_actif_6 = oui \wedge etat\_Canal_6 = envoi\_ack\ ) \vee \\ \left(\begin{array}{l} emet\_actif_6 = non \wedge recep\_actif_6 = non \wedge \\ global\_actif_6 = non \end{array} \right) \end{array} \right) \end{array} \right)$$

NOUVELLES PROPRIÉTÉS TEMPORELLES

– – **Lorsque l'émetteur envoie un message, le bit de contrôle transporté par le canal est identique à celui du site expéditeur**

$$\forall t \in \{\alpha, \beta\}$$
$$B_6^{20} \quad \Box \left(\begin{array}{l} etat\_Canal_6 = envoi\_donnée \wedge t = emet\_bit\_ctrl_6 \\ \Rightarrow \bigcirc (\ canal\_bit\_ctrl_6 = t\ ) \end{array} \right)$$

– – **Le récepteur ne peut accepter un message que si son bit est identique à celui du canal, c'est à dire de l'émetteur**

$$\forall t \in \{\alpha, \beta\}$$
$$B_6^{21} \quad \Box \left(\begin{array}{l} recep\_actif_6 = oui \wedge etat\_Canal_6 = récep\_donnée \wedge recep\_bit\_ctrl_6 = t \wedge \\ \bigcirc (\ etat\_Canal_6 = envoi\_ack\ ) \\ \Rightarrow canal\_bit\_ctrl_6 = t \end{array} \right)$$

La propriété précédente peut être reformulée comme suit :

$$B_6^{21'} \quad \square \left(\begin{array}{l} \forall t \in \{\alpha, \beta\} \\ recep\_actif_6 = oui \wedge etat\_Canal_6 = récep\_donnée \wedge \\ recep\_bit\_ctrl_6 = t \wedge canal\_bit\_ctrl_6 \neq t \\ \Rightarrow \bigcirc (\ etat\_Canal_6 \neq envoi\_ack \ ) \end{array} \right)$$

En effet, la formule $\square (\ a \wedge \bigcirc b \Rightarrow c \ )$ est équivalente à $\square (\ a \wedge \neg c \Rightarrow \bigcirc \neg b \ )$ car $a \Rightarrow b \Leftrightarrow \neg b \Rightarrow \neg a$ et $\neg \bigcirc a \Leftrightarrow \bigcirc \neg a$ et enfin $(c \wedge a) \Rightarrow b \Leftrightarrow (c \wedge \neg b) \Rightarrow \neg a$. Contrairement aux deux autres exemples de propriétés que nous avons tenté de transformer, celle-ci garde une signification intuitive proche de la description informelle. La nouvelle formulation pourrait correspondre la description informelle "Si le bit de contrle du paquet arrivé est différent de celui de l'émetteur, alors il n' aura pas d'acquittement".

– – **Lorsque l'émetteur reçoit un acquittement, il change la valeur de son bit de contrôle. Cette opération est réalisée simultanément avec l'envoi d'un paquet**

$$B_6^{22} \quad \square \left(\begin{array}{l} \forall t \in \{\alpha, \beta\} \\ etat\_Canal_6 = envoi\_ack \wedge \bigcirc (\ etat\_Canal_6 = envoi\_donnée \wedge emet\_bit\_ctrl_6 = \neg t \ ) \\ \Rightarrow emet\_bit\_ctrl_6 = t \end{array} \right)$$

Evénements

Ce niveau de raffinement ne voit pas apparaitre de nouveaux événements. Ceux dont la liste suit sont identiques à leurs abstractions: Initialisation, $ArrêtTransfert_6$, $ErreurEmetteur_6$, $ErreurRécepteur_6$, $DelaiEcoulé_6$.

La complexité du graphe d'accessiblité est trop importante pour le représenter à ce niveau de raffinement.

L'initialisation du protocole est très proche de $InitTransfert_5$. $Site\_Actif_6$ est remplacée par $emet\_bit\_ctrl_6$ et $recep\_bit\_ctrl_6$. Les deux sites doivent avoir la même valeur pour leurs bits de contrôle afin de permettre une comparaison positive et ainsi la copie du premier bloc du fichier.

$InitTransfert_6 \triangleq$ SELECT
$emet\_actif_6 = non \wedge$
$recep\_actif_6 = non \wedge$
$global\_actif_6 = non$
THEN
ANY f
WHERE
$f \in \text{seq}_1(\text{S})$
THEN
$fich\_emet_6 := f$
END ||
$fich\_emet_6 := [\,]$ ||
$emet\_actif_6 := oui$ ||
$recep\_actif_6 := oui$ ||
$emet\_transfert\_ok_6 := non$ ||
$recep\_transfert\_ok_6 := non$ ||
$global\_actif_6 := oui$ ||
$NbRéémis_6 := 0$ ||
$etat\_Canal_6 := envoi\_donnée$ ||
$emet\_bit\_ctrl_6 := \alpha$ ||
$recep\_bit\_ctrl_6 := \alpha$
END

L'arrêt de l'émetteur diffère elle aussi légèrement de son abstraction. Le bit de contrôle de l'émetteur est alterné.

$ArrêtEmetteur_6 \triangleq$ SELECT
$emet\_actif_6 = oui \wedge$
tail ($fich\_emet_6$) = [ ] $\wedge$
$etat\_Canal_6 = envoi\_ack$
THEN
$emet\_actif_6 := non \ \|$
$emet\_transfert\_ok_6 := oui \ \|$
$fich\_emet_6 :=$ tail ($fich\_emet_6$) $\|$
$etat\_Canal_6 := envoi\_donnée \ \|$
$emet\_bit\_ctrl_6 := \neg emet\_bit\_ctrl_6$
END

L'arrêt du récepteur ne peut se produire que si la donnée transportée par le canal est la dernière du fichier et si les bits de contrôle de ces deux entités sont égaux. Le récepteur alterne son bit.

$ArrêtRécepteur_6 \triangleq$ SELECT
$recep\_actif_6 = oui \wedge$
$etat\_Canal_6 = récep\_donnée \wedge$
$canal\_Dern\_Paquet_6 = oui \wedge$
$canal\_bit\_ctrl_6 = recep\_bit\_ctrl_6$
THEN
$fich\_recep_6 := fich\_recep_6 \frown canal\_Donnée_6 \ \|$
$recep\_actif_6 := non \ \|$
$recep\_transfert\_ok_6 := oui \ \|$
$etat\_Canal_6 = envoi\_ack \ \|$
$recep\_bit\_ctrl_6 := \neg recep\_bit\_ctrl_6$
END

Lorsque le site émetteur reçoit un acquittement, il décrémente le fichier source et alterne son bit de contrôle. Cette dernière opération est nécessaire pour permettre au site récepteur de différencier les paquets réémis par rapport aux nouveaux.

$RéceptionAck_6 \triangleq$ SELECT
$emet\_actif_6 = oui \wedge$
tail ($fich\_emet_6$) $\neq$ [ ] $\wedge$
$etat\_Canal_6 = envoi\_ack$
THEN
$NbRéémis_6 := 0 \ \|$
$emet\_bit\_ctrl_6 := \neg emet\_bit\_ctrl_6 \ \|$
$fich\_emet_6 :=$ tail ($fich\_emet_6$) $\|$
$etat\_Canal_6 := envoi\_donnée$
END

La réception d'un paquet est soumise aux contraintes de l'événement $ArrêtRécepteur_6$ en ce qui concerne les bits de contrôle. Le récepteur accepte un paquet seulement si le bit transporté par le canal est identique au sien. Une fois le paquet copier, il alterne son bit de contrôle.

$RéceptionPaquet_6 \triangleq$ SELECT
$recep\_actif_6 = oui \wedge$
$canal\_Donnée_6 = non \wedge$
$canal\_bit\_ctrl_6 = recep\_bit\_ctrl_6 \wedge$
$etat\_Canal_6 = récep\_donnée$
THEN
$fich\_recep_6 := fich\_recep_6 \frown canal\_Donnée_6 \ \|$
$recep\_bit\_ctrl_6 := \neg recep\_bit\_ctrl_6 \ \|$
$etat\_Canal_6 := envoi\_ack$
END

L'envoi d'un paquet (dernier ou non du fichier source) diffère de son abstraction par la copie du bit de contrôle du site émetteur dans $canal\_bit\_ctrl_6$. Cette dernière variable représente le moyen de transport du bit de contrôle de l'émetteur à destination du récepteur.

$EnvoiPaquet_6 \triangleq$ SELECT
$emet\_actif_6 = oui \wedge$
$etat\_Canal_6 = envoi\_donnée \wedge$
tail ($fich\_emet_6$) $\neq$ [ ]
THEN
$etat\_Canal_6 := récep\_donnée \ \|$
$canal\_Donnée_6 :=$ first ($fich\_emet_6$) $\|$
$canal\_Dern\_Paquet_6 := non \ \|$
$canal\_bit\_ctrl_6 := emet\_bit\_ctrl_6$
END

$EnvoiDernierPaquet_6 \triangleq$ SELECT
$emet\_actif_6 = oui \wedge$
$etat\_Canal_6 = envoi\_donnée \wedge$
tail ($fich\_emet_6$) = [ ]
THEN
$etat\_Canal_6 := récep\_donnée \ \|$
$canal\_Donnée_6 :=$ first ($fich\_emet_6$) $\|$
$canal\_Dern\_Paquet_6 := oui \ \|$
$canal\_bit\_ctrl_6 := emet\_bit\_ctrl_6$
END

2.2.8 Raffinement n ° 7

But

Le but de ce dernier raffinement est de distribuer la dernière variable centralisée $etat\_Canal_7$. Pour cela on introduit trois nouvelles variables représentant les différents états d'activation du canal de transmission.

Afin de modéliser les pertes de données pouvant se produire dans le canal de transmission, on introduit deux événements :

- $Perte\_Donnée_7$: cet événement correspond à la perte de données sur le canal du même nom,
- $Perte\_Ack_7$: idem pour le canal des acquittements,

- $Autorise\_Envoi_7$: il permet d'autoriser l'envoi de données si le canal de transmission ne contient aucune information.

Données

On garde les variables du niveau précédent à l'exception de $etat\_Canal_6$. Cette dernière est remplacée par les variables suivantes :

- $Peut\_Rec\_Don_7$: variable indiquant que des données sont disponibles sur le canal, elle est égale a *oui* si le récepteur peut lire les données qui se trouvent dans le canal de transmission,

- $Peut\_Rec\_Ack_7$: cette variable indique à l'émetteur qu'un acquittement est disponible dans le canal de transmission,

- $Peut\_Env\_Don_7$: si la valeur de cette variable est *oui*, l'émetteur peut envoyer des données par l'intermédiaire du canal de transmission.

Invariant de collage

– – **Les variables suivantes sont identiques à leurs abstractions**
$fich\_emet_7 = fich\_emet_6 \wedge$
$fich\_recep_7 = fich\_recep_6 \wedge$
$emet\_actif_7 = emet\_actif_6 \wedge$
$recep\_actif_7 = recep\_actif_6 \wedge$
$emet\_transfert\_ok_7 = emet\_transfert\_ok_6 \wedge$
$recep\_transfert\_ok_7 = recep\_transfert\_ok_6 \wedge$
$global\_actif_7 = global\_actif_6 \wedge$
$NbRéémis_7 = NbRéémis_6 \wedge$
$canal\_Donnée_7 = canal\_Donnée_6 \wedge$
$canal\_Dern\_Paquet_7 = canal\_Dern\_Paquet_6 \wedge$
$emet\_bit\_ctrl_7 = emet\_bit\_ctrl_6 \wedge$
$recep\_bit\_ctrl_7 = recep\_bit\_ctrl_6 \wedge$
$canal\_bit\_ctrl_7 = canal\_bit\_ctrl_6 \wedge$

– – **Typage des nouvelles variables**
$Peut\_Rec\_Don_7 \in \{oui, non\} \wedge$
$Peut\_Rec\_Ack_7 \in \{oui, non\} \wedge$
$Peut\_Env\_Don_7 \in \{oui, non\} \wedge$

– – **Propriétés des nouveaux bits de contrôle**
$emet\_actif_7 = oui \Rightarrow (\ Peut\_Env\_Don_7 = oui \Leftrightarrow etat\_Canal_6 = envoi\_donnée \ ) \wedge$
$(\ recep\_actif_7 = oui \wedge Peut\_Rec\_Don_7 = oui \ ) \Rightarrow etat\_Canal_6 = récep\_donnée \wedge$
$(\ emet\_actif_7 = oui \wedge Peut\_Rec\_Ack_7 = oui \ ) \Rightarrow etat\_Canal_6 = envoi\_ack \wedge$

– – **Une seule des trois nouvelles variables doit avoir la valeur *oui* à la fois**
$Peut\_Rec\_Don_7 = oui \oplus Peut\_Rec\_Ack_7 = oui \oplus Peut\_Env\_Don_7 = oui$

Propriétés temporelles

Propriétés temporelles raffinées

– – **Le protocole est modélisé comme un démon. Il attend une nouvelle demande de transfert après qu'il ait terminé de copier le fichier**

$$ {}_hB_7^1 \quad \Box \left(\begin{array}{l} global\_actif_7 = non \wedge emet\_actif_7 = non \wedge recep\_actif_7 = non \\ \Rightarrow \bigcirc \left(\begin{array}{l} global\_actif_7 = oui \wedge \\ emet\_actif_7 = oui \wedge recep\_actif_7 = oui \end{array} \right) \end{array} \right) $$

– – **Les deux sites ne se désactivent fatalement**

$$ {}_cB_7^2 \quad \Box \left(\; emet\_actif_7 = oui \Rightarrow \Diamond \left(\; emet\_actif_7 = non \; \right) \right) $$

$$ {}_cB_7^3 \quad \Box \left(\; recep\_actif_7 = oui \Rightarrow \Diamond \left(\; recep\_actif_7 = non \; \right) \right) $$

– – **Lorsque le protocole est arrêté, il peut être dans l'un des trois états cités page 14**

$$ {}_{1,2,3,f,d}B_7^4 \quad \Box \left(\begin{array}{l} global\_actif_7 = oui \wedge \bigcirc \left(\; global\_actif_7 = non \; \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} emet\_transfert\_ok_7 = oui \wedge \\ recep\_transfert\_ok_7 = oui \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_7 = non \wedge \\ recep\_transfert\_ok_7 = non \end{array} \right) \vee \\ \left(\begin{array}{l} emet\_transfert\_ok_7 = non \wedge \\ recep\_transfert\_ok_7 = oui \end{array} \right) \end{array} \right) \end{array} \right) $$

– – **Lorsque les deux sites sont inactifs, le protocole est lui aussi inactivé. Le protocole devient inactif après l'arrêt du dernier site en fonctionnement.**

$$ {}_cB_7^5 \quad \Box \left(\begin{array}{l} global\_actif_7 = oui \wedge emet\_actif_7 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_7 = oui \wedge emet\_actif_7 = non \wedge \\ recep\_actif_7 = non \end{array} \right) \\ \Rightarrow \bigcirc \; \bigcirc \left(\; global\_actif_7 = non \; \right) \end{array} \right) $$

$$ {}_cB_7^6 \quad \Box \left(\begin{array}{l} global\_actif_7 = oui \wedge recep\_actif_7 = oui \wedge \\ \bigcirc \left(\begin{array}{l} global\_actif_7 = oui \wedge emet\_actif_7 = non \wedge \\ recep\_actif_7 = non \end{array} \right) \\ \Rightarrow \bigcirc \; \bigcirc \left(\; global\_actif_7 = non \; \right) \end{array} \right) $$

– – **Les deux sites se désactivent l'un après l'autre et consécutivement**

$$ {}_cB_7^7 \quad \Box \left(\begin{array}{l} emet\_actif_7 = oui \wedge recep\_actif_7 = oui \wedge \\ \bigcirc \left(\; emet\_actif_7 = non \wedge recep\_actif_7 = oui \; \right) \\ \Rightarrow \bigcirc \; \bigcirc \left(\; recep\_actif_7 = non \; \right) \end{array} \right) $$

$$ {}_cB_7^8 \quad \Box \left(\begin{array}{l} emet\_actif_7 = oui \wedge recep\_actif_7 = oui \wedge \\ \bigcirc \left(\; recep\_actif_7 = non \wedge emet\_actif_7 = oui \; \right) \\ \Rightarrow \left(\begin{array}{l} NbRéémis_7 < \textsc{Max} \wedge \\ emet\_actif_7 = oui \end{array} \right) \mathcal{U} \left(\; emet\_actif_7 = non \; \right) \end{array} \right) $$

– – **Quand l'expéditeur sait que le protocole s'est bien terminé alors le récepteur le savait déjà**

$$ {}_eB_7^9 \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} emet\_actif_7 = non \wedge emet\_transfert\_ok_7 = oui \wedge \\ global\_actif_7 = oui \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} recep\_actif_7 = non \wedge recep\_transfert\_ok_7 = oui \wedge \\ global\_actif_7 = oui \end{array} \right) \end{array} \right) $$

– – **Quand le récepteur sait que le protocole s'est mal terminé alors l'émetteur le savait déjà**

$$ {}_eB_7^{10} \quad \Box \left(\begin{array}{l} \bigcirc \left(\begin{array}{l} recep\_actif_7 = non \wedge recep\_transfert\_ok_7 = oui \wedge \\ global\_actif_7 = oui \wedge NbRéémis_7 = (\ \text{MAX} + 1\ ) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} emet\_actif_7 = non \wedge emet\_transfert\_ok_7 = oui \wedge \\ global\_actif_7 = oui \wedge NbRéémis_7 = \text{MAX} \end{array} \right) \end{array} \right) $$

– – **Lorsque le compteur du nombre de *time out* arrive à son maximum, l'émetteur est arrêté sur une erreur et le compteur est de nouveau incrémenté pour permettre l'arrêt du récepteur**

$$ {}_{c,g}B_7^{11} \quad \Box \left(\begin{array}{l} NbRéémis_7 = \text{MAX} \wedge emet\_actif_7 = oui \wedge recep\_actif_7 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} NbRéémis_7 = (\ \text{MAX} + 1\ ) \wedge \\ emet\_actif_7 = non \wedge \\ emet\_transfert\_ok_7 = non \end{array} \right) \end{array} \right) $$

– – **Lorsque le site émetteur s'est arrêté sur une erreur, le récepteur ne peut que détecter lui aussi une anomalie.**

$$ {}_eB_7^{12} \quad \Box \left(\begin{array}{l} NbRéémis_7 = (\text{MAX} + 1) \wedge recep\_actif_7 = oui \\ \Rightarrow \bigcirc \left(\begin{array}{l} recep\_actif_7 = non \wedge \\ recep\_transfert\_ok_7 = non \end{array} \right) \end{array} \right) $$

– – **Les données ajoutées au fichier destination proviennent du fichier source**

$$ \forall r \in \text{seq(S)}, \forall e \in \text{seq(S)}, \forall t \in \{oui, non\} $$

$$ {}_aB_7^{13} \quad \Box \left(\begin{array}{l} r = fich\_recep_7 \wedge e = fich\_emet_7 \wedge e \neq [\,] \wedge \\ emet\_actif_7 = oui \wedge recep\_actif_7 = oui \wedge emet\_bit\_ctrl_7 = t \wedge \\ Peut\_Env\_Don_7 = oui \wedge \\ \left(\begin{array}{l} fich\_emet_7 = e \wedge \\ canal\_Donnée_7 = \text{first}(\ e\ ) \end{array} \right) \mathcal{U} \left(\begin{array}{l} emet\_bit\_ctrl_7 = \neg t \wedge \\ fich\_emet_7 = \text{tail}(\ e\ ) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} emet\_bit\_ctrl_7 = t \wedge \\ fich\_recep_7 = r \wedge \\ fich\_emet_7 = e \wedge \\ Peut\_Env\_Don_7 = oui \end{array} \right) \mathcal{U} \left(\begin{array}{l} emet\_bit\_ctrl_7 = t \wedge \\ fich\_recep_7 = r \frown canal\_Donnée_7 \wedge \\ fich\_emet_7 = e \wedge \\ Peut\_Rec\_Don_7 = oui \end{array} \right) \end{array} \right) $$

– – **Lorsque le fichier destination est incrémenté, le contrôle est donné à l'émetteur pour qu'il puisse décrémenter le fichier source**

$$ \forall r \in \text{seq(S)}, \forall e \in \text{seq(S)}, \forall t \in \{oui, non\} $$

$$ {}_aB_7^{14} \quad \Box \left(\begin{array}{l} fich\_recep_7 = r \wedge fich\_emet_7 = e \wedge \\ emet\_bit\_ctrl_7 = t \wedge Peut\_Rec\_Don_7 = oui \wedge \\ emet\_actif_7 = oui \wedge recep\_actif_7 = oui \wedge \\ \bigcirc (\ fich\_recep_7 = r \frown \text{first}(\ e\ ) \wedge recep\_bit\_ctrl_7 = t\ ) \\ \Rightarrow \bigcirc (\ emet\_bit\_ctrl_7 = \neg t \wedge Peut\_Rec\_Ack_7 = oui\ ) \end{array} \right) $$

– – **Lorsque le fichier source est décrémenté, le contrôle est donné au récepteur pour qu'il puisse lire le prochain paquet**

$$ \forall e \in \text{seq(S)}, \forall t \in \{\alpha, \beta\} $$

$$ {}_aB_7^{15} \quad \Box \left(\begin{array}{l} fich\_emet_7 = e \wedge e \neq [\,] \wedge \\ emet\_bit\_ctrl_7 = t \wedge emet\_actif_7 = oui \wedge \\ \bigcirc \left(\begin{array}{l} fich\_emet_7 = \text{tail}(\ e\ ) \wedge emet\_actif_7 = oui \wedge \\ emet\_bit\_ctrl_7 = \neg t \end{array} \right) \\ \Rightarrow \bigcirc (\ Peut\_Env\_Don_7 = oui \wedge emet\_bit\_ctrl_7 = \neg t\ ) \end{array} \right) $$

– – **Lorsque le protocole est initialisé, le contrôle est automatiquement donné au récepteur**

$$B_7^{16} \quad \Box \left(\begin{array}{l} emet\_actif_7 = non \wedge recep\_actif_7 = non \wedge global\_actif_7 = non \wedge \\ \bigcirc \left(\; emet\_actif_7 = oui \wedge recep\_actif_7 = oui \wedge global\_actif_7 = oui \; \right) \\ \Rightarrow \bigcirc \left(\; emet\_bit\_ctrl_7 = non \wedge recep\_bit\_ctrl_7 = non \; \right) \end{array} \right)$$

– – **Si le récepteur peut lire les données du canal alors ce dernier les a forcément lu dans le fichier source. Autrement dit, si le canal possède une donnée pouvant être lue, c'est qu'il la possédait déjà ou qu'il venait de la lire dans le fichier source**

$$_aB_7^{17} \quad \Box \left(\begin{array}{l} \bigcirc \left(\; recep\_actif_7 = oui \wedge Peut\_Rec\_Don_7 = oui \; \right) \\ \Rightarrow \left(\; Peut\_Rec\_Don_7 = oui \vee Peut\_Rec\_Ack_7 = oui \; \right) \end{array} \right)$$

– – **Si l'émetteur peut recevoir un acquittement alors le récepteur a déjà reçu un paquet**

$$B_7^{18} \quad \Box \left(\begin{array}{l} \bigcirc \left(\; emet\_actif_7 = oui \wedge Peut\_Rec\_Ack_7 = oui \; \right) \\ \Rightarrow \left(\; Peut\_Rec\_Don_7 = oui \vee Peut\_Rec\_Ack_7 = oui \; \right) \end{array} \right)$$

– – **Si l'émetteur peut envoyer des données alors il a reçu un acquittement ou le protocole vient de débuter**

$$B_7^{19} \quad \Box \left(\begin{array}{l} \bigcirc \left(\; emet\_actif_7 = oui \wedge Peut\_Env\_Don_7 = oui \; \right) \\ \Rightarrow \left(\begin{array}{l} \left(\; emet\_actif_7 = oui \wedge Peut\_Rec\_Ack_7 = oui \; \right) \vee \\ \left(\begin{array}{l} emet\_actif_7 = non \wedge recep\_actif_7 = non \wedge \\ global\_actif_7 = non \end{array} \right) \end{array} \right) \end{array} \right)$$

– – **Lorsque l'émetteur envoi un message, le bit de contrôle transporté par le canal est identique à celui du site expéditeur**

$$\begin{array}{ll} & \forall t \in \{\alpha, \beta\} \\ B_7^{20} & \Box \left(\begin{array}{l} Peut\_Rec\_Don_7 = oui \wedge t = emet\_bit\_ctrl_7 \\ \Rightarrow \bigcirc \left(\; canal\_bit\_ctrl_7 = t \; \right) \end{array} \right) \end{array}$$

– – **Le récepteur ne peut accepter un message que si son bit est identique à celui du canal, c'est à dire de l'émetteur**

$$\begin{array}{ll} & \forall t \in \{\alpha, \beta\} \\ B_7^{21} & \Box \left(\begin{array}{l} recep\_actif_7 = oui \wedge Peut\_Rec\_Don_7 = oui \wedge recep\_bit\_ctrl_7 = t \wedge \\ \bigcirc \left(\; Peut\_Rec\_Ack_7 = oui \; \right) \\ \Rightarrow canal\_bit\_ctrl_7 = t \end{array} \right) \end{array}$$

– – **Lorsque l'émetteur reçoit un acquittement, il change la valeur de son bit de contrôle. Cette opération est réalisée simultanément avec l'envoi d'un paquet**

$$\begin{array}{ll} & \forall t \in \{\alpha, \beta\} \\ B_7^{22} & \Box \left(\begin{array}{l} Peut\_Rec\_Ack_7 = oui \wedge \bigcirc \left(\; Peut\_Env\_Don_7 = oui \wedge emet\_bit\_ctrl_7 = \neg t \; \right) \\ \Rightarrow emet\_bit\_ctrl_7 = t \end{array} \right) \end{array}$$

NOUVELLES PROPRIÉTÉS TEMPORELLES

– – **Lorsque le canal passe dans un état d'acceptation des envois (*Peut_Env_Don*$_7$ = *oui*), alors il vient de perdre sont état indiquant qu'un acquittement était disponible**

$$B_7^{23} \quad \Box \left(\begin{array}{l} Peut\_Env\_Don_7 = non \wedge Peut\_Rec\_Ack_7 = oui \wedge \bigcirc \left(\; Peut\_Env\_Don_7 = oui \; \right) \\ \Rightarrow \bigcirc \left(\; Peut\_Rec\_Ack_7 = non \; \right) \end{array} \right)$$

– – **Lorsque le canal change d'état pour indiquer au récepteur qu'il peut lire des données ($Peut\_Rec\_Don_7 = oui$), alors il vient de perdre létat d'acceptation des données à envoyer**

$$B_7^{24} \quad \Box \left(\begin{array}{l} Peut\_Rec\_Don_7 = non \wedge Peut\_Env\_Don_7 = oui \wedge \bigcirc \left(\ Peut\_Rec\_Don_7 = oui \ \right) \\ \Rightarrow \bigcirc \left(\ Peut\_Env\_Don_7 = non \ \right) \end{array} \right)$$

– – **Lorsque le canal de transmission passe dans l'état d'envoi d'acquittement, alors il était dans celui permettant au récepteur de lire des données**

$$B_7^{25} \quad \Box \left(\begin{array}{l} Peut\_Rec\_Ack_7 = non \wedge Peut\_Rec\_Don_7 = oui \wedge \bigcirc \left(\ Peut\_Rec\_Ack_7 = oui \ \right) \\ \Rightarrow \bigcirc \left(\ Peut\_Rec\_Don_7 = non \ \right) \end{array} \right)$$

Evénements

Ce dernier niveau de raffinement voit l'apparition de trois événements: $Autorise\_Envoi_7$, $Perte\_Donnée_7$ et $Perte\_Ack_7$. Le premier correspond à l'autorisation d'envoyer des données pour le site émetteur, les deux autres permettent de représenter les pertes de données dans le canal de transmission.

Tout comme au niveau précédent, le graphe d'accessiblité est trop important pour être représenté. Ici, le nombre d'états ne change pas car la variable $etat\_Canal_6$ est remplacée par les trois variables $Peut\_Env\_Don_7$, $Peut\_Rec\_Don_7$ et $Peut\_Rec\_Ack_7$. Hors le nombre de possiblités combinatoires quant à leurs valeurs est identique à $etat\_Canal_6$ car :

$$\left(etat\_Canal_6 = envoi\_donnée \Leftrightarrow \left(\begin{array}{l} Peut\_Env\_Don_7 = oui \wedge Peut\_Rec\_Don_7 = non \wedge \\ Peut\_Rec\_Ack_7 = non \end{array} \right) \right) \wedge$$
$$\left(etat\_Canal_6 = récep\_donnée \Leftrightarrow \left(\begin{array}{l} Peut\_Env\_Don_7 = non \wedge Peut\_Rec\_Don_7 = oui \wedge \\ Peut\_Rec\_Ack_7 = non \end{array} \right) \right) \wedge$$
$$\left(etat\_Canal_6 = envoi\_ack \Leftrightarrow \left(\begin{array}{l} Peut\_Env\_Don_7 = non \wedge Peut\_Rec\_Don_7 = non \wedge \\ Peut\_Rec\_Ack_7 = oui \end{array} \right) \right) \wedge$$
$$(Peut\_Rec\_Don_7 = oui \oplus Peut\_Rec\_Ack_7 = oui \oplus Peut\_Env\_Don_7 = oui)$$

Les événements suivants sont identiques à leurs abstractions: Initialisation, $ArrêtTransfert_7$, $ErreurEmetteur_7$, $ErreurRécepteur_7$.

L'initialisation du protocole est très proche de son abstraction. La mise à jour de la variable $etat\_Canal_6$ est remplacée par celles de $Peut\_Env\_Don_7$, $Peut\_Rec\_Don_7$ et $Peut\_Rec\_Ack_7$.

$InitTransfert_7 \triangleq$

SELECT
  $emet\_actif_7 = non \wedge$
  $recep\_actif_7 = non \wedge$
  $global\_actif_7 = non$
THEN
  ANY f
  WHERE
    $f \in \mathrm{seq}_1(\mathrm{S})$
  THEN
    $fich\_emet_7 := f$
  END ||
  $fich\_emet_7 := [\,]$ ||
  $emet\_actif_7 := oui$ ||
  $recep\_actif_7 := oui$ ||
  $emet\_transfert\_ok_7 := non$ ||
  $recep\_transfert\_ok_7 := non$ ||
  $global\_actif_7 := oui$ ||
  $NbRéémis_7 := 0$ ||
  $emet\_bit\_ctrl_7 := récepteur$ ||
  $recep\_bit\_ctrl_7 := récepteur$ ||
  $Peut\_Rec\_Don_7 := non$ ||
  $Peut\_Rec\_Ack_7 := non$ ||
  $Peut\_Env\_Don_7 := oui$
END

L'émetteur ne peut s'arrêter que s'il reçoit un acquittement pour le dernier bloc du fichier source.

$ArrêtEmetteur_7 \triangleq$

SELECT
  $emet\_actif_7 = oui \wedge$
  $\mathrm{tail}\,(\ fich\_emet_7\ ) = [\,] \wedge$
  $Peut\_Rec\_Ack_7 = oui$
THEN
  $emet\_actif_7 := non$ ||
  $emet\_transfert\_ok_7 := oui$ ||
  $fich\_emet_7 := \mathrm{tail}\,(\ fich\_emet_7\ )$ ||
  $Peut\_Rec\_Ack_7 := non$ ||
  $Peut\_Env\_Don_7 := oui$ ||
  $emet\_bit\_ctrl_7 := \neg emet\_bit\_ctrl_7$
END

L'événement d'arrêt du récepteur doit tenir compte des nouvelles variables. Il ne peut être éxécuté que s'il vient de recevoir le dernier paquet de données. Dans ce cas, il copie le bloc et consière le protocole comme correctement terminé. Avant cela, il envoi un message d'acquittement vers lémetteur.

$ArrêtRécepteur_7 \triangleq$ SELECT
$recep\_actif_7 = oui \wedge$
$Peut\_Rec\_Don_7 = oui \wedge$
$canal\_Dern\_Paquet_7 = oui \wedge$
$canal\_bit\_ctrl_7 = recep\_bit\_ctrl_7$
THEN
$fich\_recep_7 := fich\_recep_7 \frown canal\_Dern\_Paquet_7 \ \|$
$recep\_actif_7 := non \ \|$
$recep\_transfert\_ok_7 := oui \ \|$
$Peut\_Rec\_Don_7 = non \ \|$
$Peut\_Rec\_Ack_7 := oui \ \|$
$recep\_bit\_ctrl_7 := \neg recep\_bit\_ctrl_7$
END

Lorsque le site émetteur reçoit un acquittement, il décrémente le fichier source et demande au canal de transmission de recevoir le prochain bloc de données.

$RéceptionAck_7 \triangleq$ SELECT
$emet\_actif_7 = oui \wedge$
$\text{tail}\left(\ fich\_emet_7 \ \right) \neq [\,] \wedge$
$Peut\_Rec\_Ack_7 = oui$
THEN
$NbRéémis_7 := 0 \ \|$
$emet\_bit\_ctrl_7 := \neg emet\_bit\_ctrl_7 \ \|$
$fich\_emet_7 := \text{tail}\left(\ fich\_emet_7 \ \right) \ \|$
$Peut\_Rec\_Ack_7 := non$
END

$RéceptionPaquet_7$ est toujours l'événement qui représente la réception d'un paquet de données chez le récepteur. Il réagit exatement comme son abstraction, c'est à dire qu'il copie le paquet de données si le bit de contrôle est correct, et demande au canal de transmission de faire parvenir un acquittement vers le site émetteur.

$RéceptionPaquet_7 \triangleq$ SELECT
$recep\_actif_7 = oui \wedge$
$canal\_Dern\_Paquet_7 = non \wedge$
$canal\_bit\_ctrl_7 = recep\_bit\_ctrl_7 \wedge$
$Peut\_Rec\_Don_7 = oui$
THEN
$fich\_recep_7 := fich\_recep_7 \frown canal\_Donnée_7 \ \|$
$recep\_bit\_ctrl_7 := \neg recep\_bit\_ctrl_7 \ \|$
$Peut\_Rec\_Don_7 := non \ \|$
$Peut\_Rec\_Ack_7 := oui$
END

Comme son abstraction, l'envoi d'un paquet (dernier ou non) prend un bloc de données dans le fichier source et le fait parvenir au récepteur grâce au canal de transmission.

$EnvoiPaquet_7 \triangleq$ SELECT
$emet\_actif_7 = oui \wedge$
$Peut\_Env\_Don_7 = oui \wedge$
tail ($fich\_emet_7$) $\neq [\,]$
THEN
$Peut\_Env\_Don_7 := non \;\|$
$Peut\_Rec\_Don_7 := oui \;\|$
$canal\_Donnée_7 :=$ first ($fich\_emet_7$) $\|$
$canal\_Dern\_Paquet_7 := non \;\|$
$canal\_bit\_ctrl_7 := emet\_bit\_ctrl_7$
END

$EnvoiDernierPaquet_7 \triangleq$ SELECT
$emet\_actif_7 = oui \wedge$
$Peut\_Env\_Don_7 = oui \wedge$
tail ($fich\_emet_7$) $= [\,]$
THEN
$Peut\_Env\_Don_7 := non \;\|$
$Peut\_Rec\_Don_7 = oui \;\|$
$canal\_Donnée_7 :=$ first ($fich\_emet_7$) $\|$
$canal\_Dern\_Paquet_7 := oui \;\|$
$canal\_bit\_ctrl_7 := emet\_bit\_ctrl_7$
END

L'événement $DelaiEcoulé_7$ est légèrement différent de son abstraction. Il tient compte des nouvelles variables dans sa garde.

$DelaiEcoulé_7 \triangleq$ SELECT
$emet\_actif_7 = oui \wedge$
$emet\_transfert\_ok_7 <$ MAX $\wedge$
$Peut\_Env\_Don_7 = non \wedge$
$Peut\_Rec\_Ack_7 = non$
THEN
$emet\_transfert\_ok_7 := emet\_transfert\_ok_7 + 1$
END

Les deux événements suivant permettent de modéliser la perte de données dans le canal de transmission. $Perte\_Donnée_7$ peut se produire $Peut\_Rec\_Don_7 = oui$. Il consiste à annuler la valeur qui indique que des données sont disponibles dans le canal de transmission.

$Perte\_Donnée_7 \triangleq$ SELECT
$Peut\_Rec\_Don_7 = oui$
THEN
$Peut\_Rec\_Don_7 := non$
END

$Perte\_Ack_7$ ressemble énormément à l'événement précédent. Il réinitialise la valeur de la variable $Peut\_Rec\_Ack_7$, c'est à dire qu'il annule l'indicateur de présence d'un acquittement et ainsi simule sa perte.

$Perte\_Ack_7$	$\triangleq$	SELECT $Peut\_Rec\_Ack_7 = oui$ THEN $Peut\_Rec\_Ack_7 := non$ END

L'événement permettant d'autoriser l'émetteur a envoyer des données dans le canal est $Autorise\_Envoi_7$. Il permet de forcer le canal de transmission a recevoir des données s'il ne contient aucune information (donnée ou acquittement).

$Autorise\_Envoi_7$	$\triangleq$	SELECT $emet\_actif_7 = oui \wedge$ $Peut\_Env\_Don_7 = non \wedge$ $Peut\_Rec\_Don_7 = non \wedge$ $Peut\_Rec\_Ack_7 = non \wedge$ THEN $Peut\_Env\_Don_7 := oui$ END

Chapitre 3

Différentes formes de propriété en logique temporelle linéaire

Dans ce chapitre, les différentes formes de propriétés en LTL[1] rencontrées dans les exemples du protocole BRP, de l'Automatisme Industriel, de l'Ascenseur et de la Carte à circuits intégrés sont présentées. Pour illustrer les formes de propriétés nous utiliserons l'Automatisme Industriel et le protocole BRP.

Notation :

- a, b, c, d et e dénotent des propriétés sur les états exprimées en logique des prédicats du premier ordre,
- x et y dénotent des propriétés sur les états exprimées en logique temporelle,
- pour lire les formules l'ordre de priorité des opérateurs est le suivant :

$\Rightarrow$	opérateurs le moins prioritaire
$\mathcal{U}$	
$\wedge, \vee$	
$\neg, \bigcirc, \Diamond, \Box$	
$=, \neq, >, <, \leq, \geq$	opérateurs les plus prioritaires

3.1 Les formes courantes

3.1.1 Propriété de l'état initial : a

Explication: Comme le montre la figure 3.1, la propriété a est vraie uniquement sur le premier état du système.

[1] Linear Temporal Logic

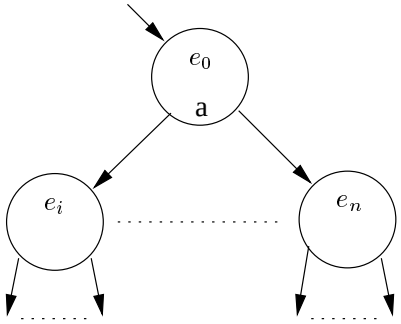


Figure 3.1: Propriété de l'état initial : a

Exemple: On retrouve dans l'AI[2] [Jul97] la propriété "le dispositif d'évacuation est initialement vide". Elle se traduit par $De = libre$.

Quelques propriétés: Cette forme apparaît notamment dans les propriétés suivantes : C_1^1, C_2^1.

3.1.2 Propriété invariante : $\Box a$

Explication: La propriété a est vraie dans tous les états du système. a est un invariant et peut figurer dans la partie du même nom d'une spécification. La figure 3.2 représente ce type de forme.

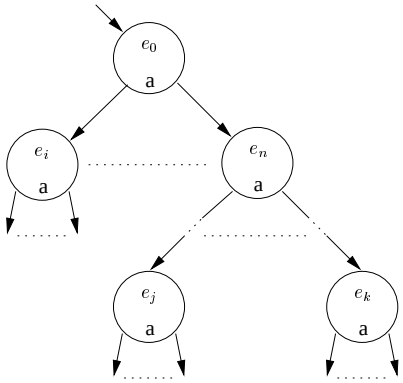


Figure 3.2: Propriété invariante : $\Box a$

Exemple: "Le dispositif de transport monte toujours plein" se traduit par la formule temporelle :

$$\Box \left(\; Mv = monte \Rightarrow Dt = occupé \; \right).$$

Quelques propriétés: A_8^3, A_8^4.

3.1.3 Réponse immédiate : $\Box \left(\; a \Rightarrow \bigcirc b \; \right)$

Explication: La figure 3.3 montre bien le comportement de cette forme qui oblige la propriété b a être vraie dans tous les états suivant e_i si la propriété a est vérifiée dans l'état e_i.

[2] Automatisme Industriel

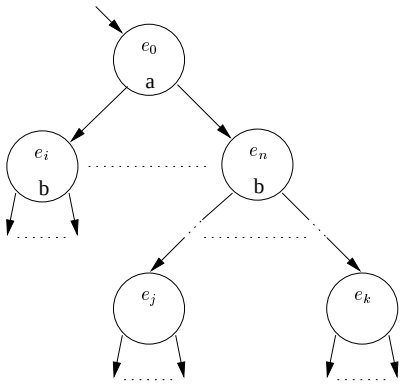


Figure 3.3: Réponse immédiate : $\Box\,(\ a \Rightarrow \bigcirc b\ )$

Exemple: "Lorsque tous les dispositifs sont occupés, une pièce est obligatoirement évacuée"

$$\Box\,(\ De = occupé \wedge Dt = occupé \wedge PosDt = haut \wedge Da = occupé \Rightarrow \bigcirc\,(\ De = libre\ )\ )$$

Quelques propriétés: C_5^1, C_{16}^3, A_4^3.

3.1.4 Réponse fatale : $\Box\,(\ a \Rightarrow \Diamond b\ )$

Explication: Comme le montre la figure 3.4, cette forme oblige la propriété b a être vérifiée dans un état du système postérieur à e_i si la propriété a est vraie durant ce dernier.

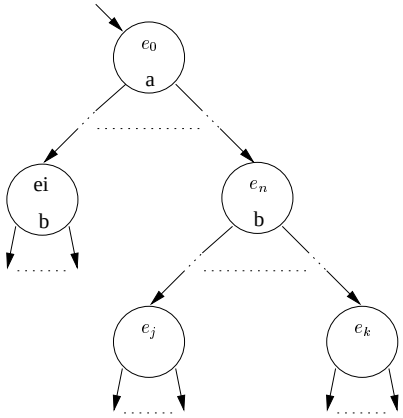


Figure 3.4: Réponse fatale : $\Box\,(\ a \Rightarrow \Diamond b\ )$

Exemple: "Le dispositif d'évacuation se libère fatalement"

$$\Box\,(\ De = occupé \Rightarrow \Diamond\,(\ De = libre\ )\ )$$

Quelques propriétés: B_2^5, A_1^2, S_3^0.

3.1.5 Obligation pour un événement : $\Box\,(\ a \wedge \bigcirc b \Rightarrow \bigcirc c\ )$

Explication: Tout événement qui fait passer d'un état satisfaisant a à un état satisfaisant b doit mener à un état où c est vraie. La figure 3.5 montre bien que les propriétés b et c sont vérifiées

dans le même état. Cette forme est équivalente à $\Box\ (\ a \Rightarrow \bigcirc\ (\ \neg b \vee c\ )\ )$ comme le montre notre exemple de reformulation de la propriété B_4^1 page 21.

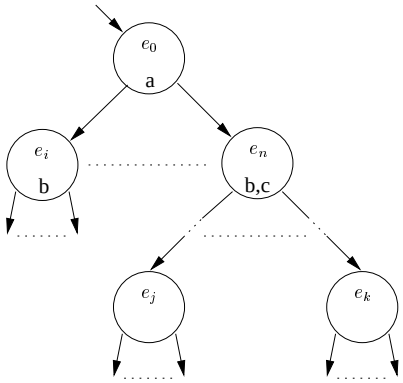


Figure 3.5: Obligation pour un événement : $\Box\ (\ a \wedge \bigcirc b \Rightarrow \bigcirc c\ )$

Exemple: "Lorsque le logiciel considère que la pince doit s'ouvrir, il doit envoyer un ordre correspondant au matériel"

$$\Box \left(\begin{array}{l} Mp = fermée \wedge \bigcirc\ (\ Mp = s'ouvre\ ) \\ \Rightarrow \bigcirc\ (\ SnMotp = oui \wedge SnMotp = s'ouvre\ ) \end{array} \right)$$

Quelques propriétés: A_{19}^3, B_{16}^7, $C_{-1}^B 40$.

3.1.6 Condition de déclenchement d'un événement : $\Box\ (\ a \wedge \bigcirc b \Rightarrow c\ )$

Explication: La figure 3.6 montre que l'événement qui fait passer de la propriété a à b est déclenchable si la propriété c est vraie. Cette forme est équivalente à $\Box\ (\ a \wedge \neg c \Rightarrow \bigcirc \neg b\ )$. Elle est illustrée par l'exemple de reformulation cité page 58.

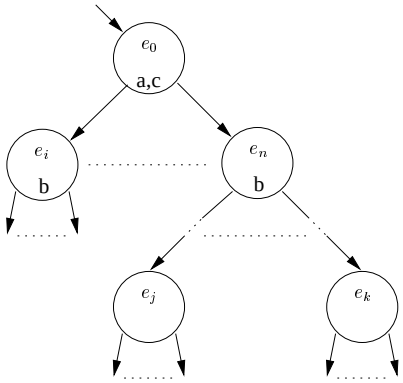


Figure 3.6: Condition de déclenchement d'un événement : $\Box\ (\ a \wedge \bigcirc b \Rightarrow c\ )$

Exemple: "Le bras doit être en arrière avant d'arriver en haut"

$$\Box\ (\ Mv = monte \wedge \bigcirc\ (\ Mv = enhaut\ ) \Rightarrow Mh = enarrière\ )$$

Quelques propriétés: A_4^9

3.1.7 Propriété d'invariant local : $\Box\left(a \Rightarrow b \,\mathcal{U}\, c \right)$

Explication: La propriété b persiste, c'est à dire qu'elle est toujours vraie, entre un état où a est vraie est un état où c est vérifiée comme le montre la figure 3.7.

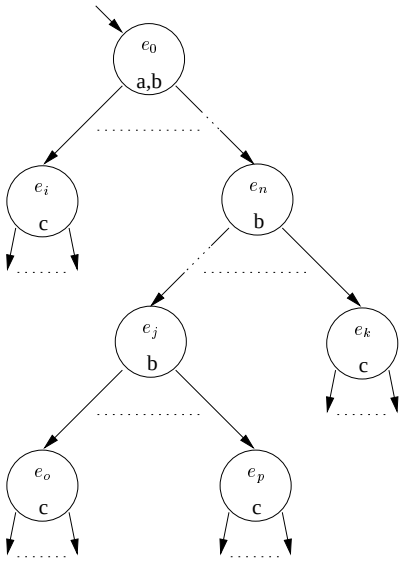


Figure 3.7: Propriété d'invariant local : $\Box\left(a \Rightarrow b \,\mathcal{U}\, c \right)$

Exemple: "La pince reste ouverte pendant tout le mouvement de descente depuis le dispositif d'évacuation jusqu'au dispositif d'arrivée"

$$\Box\left(\begin{array}{l} Mp = ouverte \wedge Mh = enavant \wedge Mv = enhaut \\ \Rightarrow \left(Mp = ouverte \right) \mathcal{U} \left(Mh = enavant \wedge Mv = enbas \right) \end{array}\right)$$

Quelques propriétés: A_9^4, A_4^4.

3.2 Les formes moins courantes

3.2.1 Obligation : $\Box\left(\bigcirc a \Rightarrow b \right)$

Explication: Une forme semblable à celle illustrée par la figure 3.8 a déjà été rencontrée. Mais contrairement à $\Box\left(a \Rightarrow \bigcirc b \right)$, la propriété b est vraie à l'état précédent celui où a est vérifiée. Cette forme est équivalente à $\Box\left(\neg b \Rightarrow \bigcirc \neg a \right)$ comme le montre le petit exemple de reformulation décrit page 22.

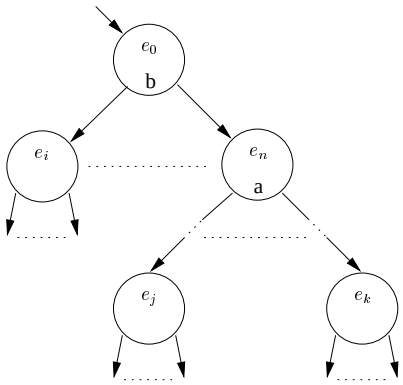


Figure 3.8: Obligation : $\Box\,(\ \bigcirc a \Rightarrow b\ )$

Exemple: Nous allons quitter l'exemple de l'AI pour passer dans celui du protocole BRP. "Si l'émetteur considère que le protocole s'est bien terminé, alors le récepteur le savait déjà"

$$\Box\left(\begin{array}{l}\bigcirc\,(\ emet\_actif = non \wedge emet\_transfert\_ok = oui\ ) \\ \Rightarrow recep\_actif = non \wedge recep\_transfert\_ok = oui\end{array}\right)$$

Quelques propriétés: B^5_{17}, B^4_9.

3.2.2 Enchaînement d'événements : $\Box\,(\ a \wedge \bigcirc b \Rightarrow \bigcirc\ \bigcirc c\ )$

Explication: Cette forme permet de contrôler l'évolution du système sur trois états. En effet, lorsque la propriété a est vraie et que b est vérifiée à l'état suivant alors la propriété est vraie durant l'état qui suit celui où b l'est.

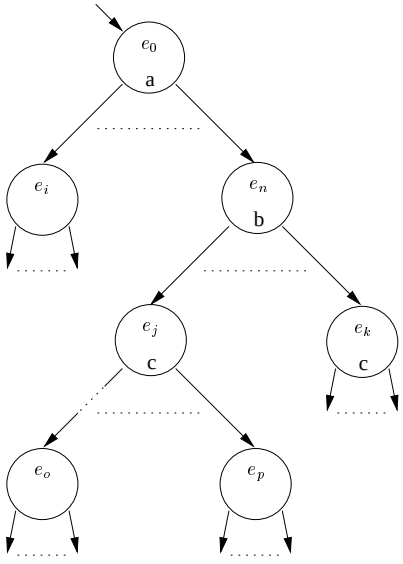


Figure 3.9: Enchaînement d'événements : $\Box\,(\ a \wedge \bigcirc b \Rightarrow \bigcirc\ \bigcirc c\ )$

Exemple: "Le récepteur se désactive juste après l'émetteur"

$$\Box \left(\begin{array}{l} emet\_actif = oui \wedge \bigcirc \left(\ emet\_actif = non \wedge recep\_actif = oui \ \right) \\ \Rightarrow \bigcirc \bigcirc \left(\ recep\_actif = non \ \right) \end{array} \right)$$

Quelques propriétés: B_5^1, B_7^3, B_6^5.

3.2.3 Réponse fatale après un événement : $\Box \left(\ a \wedge \bigcirc b \Rightarrow \Diamond c \ \right)$

Explication: Cette forme représentée par la figure 3.10 permet de contrôler l'évolution du système non plus sur trois états successifs comme le fait le forme précédente mais sur un nombre quelconque. Si la propriété a est vraie dans l'état e_i et b dans un état suivant à ce dernier alors la propriété c sera fatalement vraie dans un état postérieur à e_i.

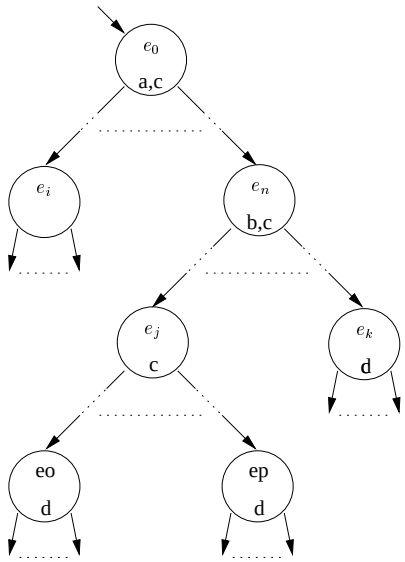


Figure 3.10: Réponse fatale après un événement : $\Box \left(\ a \wedge \bigcirc b \Rightarrow \Diamond c \ \right)$

Exemple: Dans le BRP, "Lorsque le récepteur se désactive, l'émetteur le fera tôt ou tard"

$$\Box \left(\begin{array}{l} recep\_actif = oui \wedge emet\_actif = oui \wedge \bigcirc \left(\ recep\_actif = non \wedge emet\_actif = oui \ \right) \\ \Rightarrow \Diamond \left(\ emet\_actif = non \ \right) \end{array} \right)$$

Une propriété: S_1^0.

3.2.4 Réponse par une propriété d'invariant local pour un événement : $\Box \left(\ a \wedge \bigcirc b \Rightarrow c \, \mathcal{U} \, d \ \right)$

Explication: La figure 3.11 montre le comportement de cette forme un peu particulière. Si la propriété a est vraie dans l'état e_i et que b est vérifiée à un état immédiatement ultérieur à e_i, alors la propriété c est vraie jusqu'à ce que d le soit.

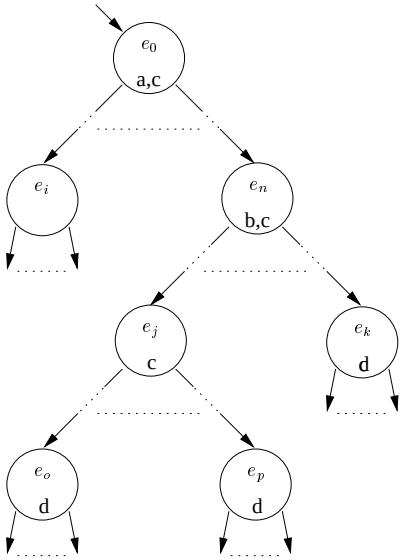


Figure 3.11: Réponse par une propriété d'invariant local pour un événement : $\Box\ (\ a \wedge \bigcirc b \Rightarrow c\ \mathcal{U}\ d\ )$

Exemple: "Les deux sites se désactivent l'un après l'autre et consécutivement"

$$\Box \left(\begin{array}{l} emet\_actif = oui \wedge recep\_actif = oui \wedge \\ \bigcirc\ (\ recep\_actif = non \wedge emet\_actif = oui\ ) \\ \Rightarrow \left(\begin{array}{l} NbRéémis < \text{MAX} \wedge \\ emet\_actif = oui \end{array} \right)\ \mathcal{U}\ (\ emet\_actif = non\ ) \end{array} \right)$$

Quelques propriétés: B_8^2, B_8^6.

3.2.5 Réponse par une propriété d'invariant local à un invariant local : $\Box\ (\ a \wedge b\ \mathcal{U}\ c \Rightarrow d\ \mathcal{U}\ e\ )$

Explication: Si la propriété a est vraie et que b est vérifiée jusqu'à ce que la propriété c le soit, alors une propriété d est vraie jusqu'à ce que e soit vérifiée. La figure 3.12 illustre cette forme de propriété.

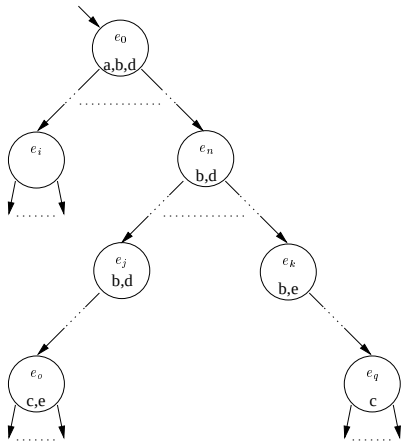


Figure 3.12: Réponse par une propriété d'invariant local à un invariant local : $\Box\,(\ a \wedge b\,\mathcal{U}\,c \Rightarrow d\,\mathcal{U}\,e\ )$

Les propriétés de la forme $\Box\,(\ a \wedge b\,\mathcal{U}\,c \Rightarrow d\,\mathcal{U}\,e\ )$ ne sont utilisées que pour raffinées les propriétés du niveau de spécification précédent. Elles paraissent compliquées mais entre bien dans le cadre de la spécification.

Exemple: "Les données ajoutées au fichier destination proviennent du fichier source" ou encore "Si l'on retire le forcément le premier paquet du fichier source alors le paquet est ajouté au fichier destination *avant* qu'il ne soit retiré"

$$\begin{array}{l}\forall r \in \mathrm{seq(S)}, \forall e \in \mathrm{seq(S)} \\ \Box \left(\begin{array}{l} r = fich\_recep \wedge e = fich\_emet \wedge e \neq [\,] \wedge \\ emet\_actif = oui \wedge recep\_actif = oui \wedge Site\_Actif = récepteur \wedge \\ (\ fich\_emet = e\ )\ \mathcal{U}\ (\ fich\_emet = \mathrm{tail}(\ e\ )\ ) \\ \Rightarrow \left(\begin{array}{l} Site\_Actif = récepteur \wedge \\ fich\_recep = r \wedge \\ fich\_emet = e \end{array} \right)\ \mathcal{U}\ \left(\begin{array}{l} Site\_Actif = émetteur \wedge \\ fich\_recep = r \frown \mathrm{first}(\ e\ ) \wedge \\ fich\_emet = e \end{array} \right) \end{array} \right) \end{array}$$

Quelques propriétés: B^4_{13}, B^6_{13}.

3.3 Liste des formes de propriétés en logique temporelle linéaire

Le tableau 3.1 récapitule toutes les formes de propriétés rencontrées.

Noms des propriétés		Formes LTL	
Formes courantes	Propriété de l'état initial	a	
	Propriété invariante	$\Box a$	
	Réponse immédiate	$\Box (a \Rightarrow \bigcirc b)$	
	Réponse fatale	$\Box (a \Rightarrow \Diamond b)$	
	Obligation pour un événement	$\Box (a \wedge \bigcirc b \Rightarrow \bigcirc c)$	$\star$
	Condition de déclenchement d'un événement	$\Box (a \wedge \bigcirc b \Rightarrow c)$	$\star$
	Propriété d'invariant local	$\Box (a \Rightarrow b \,\mathcal{U}\, c)$	
Formes moins courantes	Obligation	$\Box (\bigcirc a \Rightarrow b)$	$\star$
	Enchaînement d'événements	$\Box (a \wedge \bigcirc b \Rightarrow \bigcirc \bigcirc c)$	✓
	Réponse fatale après un événement	$\Box (a \wedge \bigcirc b \Rightarrow \Diamond c)$	✓
	Réponse par une propriété d'invariant local pour un événement	$\Box (a \wedge \bigcirc b \Rightarrow c \,\mathcal{U}\, d)$	✓
	Réponse par une propriété d'invariant local à un invariant local	$\Box (a \wedge b \,\mathcal{U}\, c \Rightarrow d \,\mathcal{U}\, e)$	✓

Les formes $\star$ se ramènent sous la forme $\Box (a \Rightarrow \bigcirc b)$

Les formes ✓ pourraient toutes êtres sous la forme $\Box (a \Rightarrow \rho)$ où ρ est une propriété temporelle

Tableau 3.1: Liste des formes de propriétés en LTL

3.4 Schémas d'équivalences

Dans cette section nous allons nous attacher à définir les équivalences existantes entre les formes de propriétés temporelles. Ces formes répertoriées dans le tableau 3.2 sont obtenues en utilisant les propriétés des logiques du premier ordre et temporelle. Nous avons développé dans l'exemple du BRP certaines reformulations possibles.

Formes initiales	Reformulations
$\Box (a \wedge \bigcirc b \Rightarrow c)$	$\Box (a \wedge \neg c \Rightarrow \bigcirc \neg b)$
$\Box (\bigcirc a \Rightarrow b)$	$\Box (\neg b \Rightarrow \bigcirc \neg a)$
$\Box (a \wedge \bigcirc b \Rightarrow \bigcirc c)$	$\Box (a \Rightarrow \bigcirc (\neg b \vee c))$
$\Box (a \wedge \bigcirc b \Rightarrow \bigcirc \bigcirc c)$	$\Box (a \Rightarrow \bigcirc \neg b \vee \bigcirc \bigcirc c)$
$\Box (a \wedge \bigcirc b \Rightarrow \Diamond c)$	$\Box (a \Rightarrow (\bigcirc \neg b \vee \Diamond c))$
$\Box (a \wedge \bigcirc b \Rightarrow c \,\mathcal{U}\, d)$	$\Box (a \Rightarrow (\bigcirc \neg b \vee (c \,\mathcal{U}\, d)))$
$\Box (a \wedge b \,\mathcal{U}\, c \Rightarrow d \,\mathcal{U}\, e)$	$\Box (a \Rightarrow \neg (b \,\mathcal{U}\, c) \vee (d \,\mathcal{U}\, e))$

Tableau 3.2: Shémas d'équivalences

Nous ne développerons qu'une seule des équivalences du tableau 3.2.

$$
\begin{aligned}
\Box (a \wedge \bigcirc b \Rightarrow c) &\Leftrightarrow \Box (\neg (a \wedge \bigcirc b) \vee c) \\
&\Leftrightarrow \Box (\neg a \vee \neg \bigcirc b \vee c) \\
&\Leftrightarrow \Box (\neg a \vee \bigcirc \neg b \vee c) \\
&\Leftrightarrow \Box (\neg a \vee c \vee \bigcirc \neg b) \\
&\Leftrightarrow \Box (\neg (a \wedge \neg c) \vee \bigcirc \neg b) \\
&\Leftrightarrow \Box (a \wedge \neg c \Rightarrow \bigcirc \neg b)
\end{aligned}
$$

3.5 Liste des formes canoniques de propriétés en logique temporelle linéaire

A partir des tableaux 3.1 et 3.2, on peut déduire une liste réduite de formes de propriétés temporelles. Le tableau 3.3 correspond à ces schémas canoniques.

Noms des propriétés	Formes LTL
Propriété de l'état initial	a
Propriété invariante	$\Box a$
Réponse immédiate	$\Box \left(\ a \Rightarrow \bigcirc b \ \right)$
Réponse fatale	$\Box \left(\ a \Rightarrow \Diamond b \ \right)$
Propriété d'invariant local	$\Box \left(\ a \Rightarrow b \,\mathcal{U}\, c \ \right)$
Réponse généralisée	$\Box \left(\ a \Rightarrow x \vee y \ \right)$

Tableau 3.3: Liste des formes canoniques de propriétés en LTL

Les schémas d'équivalence nous ont permis de réduire le nombre de formes de propriétés en logique temporelle. La liste obtenue reprśente les formes formes canoniques rencontrées durant les études de cas analysées.

Chapitre 4

Différentes formes de raffinement des propriétés temporelles

Dans ce chapitre nous allons nous intéresser aux raffinements des formes de propriétés temporelles vues au chapitre précédent. On peut distinguer deux types de raffinement: unique et multiple. Dans la première catégorie sont regroupées les propriétés ne possédant qu'un seul raffinement alors que dans la seconde plusieurs possibilités existent. D'autre part toutes les propriétés appartenant à la première catégorie ne voient pas leur structure modifiée.

Notation :

- a, b, c, d et e dénotent des propriétés sur les états exprimées en logique des prédicats du premier ordre,
- a', b', c', d', e', f' et g' dénotent des propriétés sur les états raffinés exprimées en logique des prédicats du premier ordre,
- pour lire les formules l'ordre des priorité des opérateurs est :

$\Rightarrow$	opérateur le moins prioritaire
$\mathcal{U}$	
$\wedge$, $\vee$	
$\neg$, $\bigcirc$, $\Diamond$, $\Box$	
$=, \neq, <, >, \leq, \geq$	opérateurs les plus prioritaires

4.1 Raffinements uniques sans changement de structure temporelle

Certaines formes de propriétés temporelles trouvées n'ont qu'une seule possibilité de raffinement: la même forme. Par abus de langage on peut dire qu'elles se raffinent par elles mêmes à l'invariant de collage près. La table 4.1 présente la liste des formes concernées et leurs raffinements.

Formes abstraites	Formes raffinées
a	a'
$\Box a$	$\Box a'$
$\Box\ (\ a \Rightarrow \Diamond b\ )$	$\Box\ (\ a' \Rightarrow \Diamond b'\ )$
$\Box\ (\ a \Rightarrow b\ \mathcal{U}\ c\ )$	$\Box\ (\ a' \Rightarrow b'\ \mathcal{U}\ c'\ )$
$\Box\ (\ a \wedge \bigcirc b \Rightarrow c\ )$	$\Box\ (\ a' \wedge \bigcirc b' \Rightarrow c'\ )$
$\Box\ (\ \bigcirc a \Rightarrow b\ )$	$\Box\ (\ \bigcirc a' \Rightarrow b'\ )$
$\Box\ (\ a \wedge b\ \mathcal{U}\ c \Rightarrow d\ \mathcal{U}\ e\ )$	$\Box\ (\ a' \wedge b'\ \mathcal{U}\ c' \Rightarrow d'\ \mathcal{U}\ e'\ )$
$\Box\ (\ a \wedge \bigcirc b \Rightarrow c\ \mathcal{U}\ d\ )$	$\Box\ (\ a' \wedge \bigcirc b' \Rightarrow c'\ \mathcal{U}\ d'\ )$
$\Box\ (\ a \wedge \bigcirc b \Rightarrow \Diamond c\ )$	$\Box\ (\ a' \wedge \bigcirc b' \Rightarrow \Diamond c'\ )$

Tableau 4.1: Raffinements uniques des propriétés temporelles

Notons que sur les exemples les propriétés de la forme $\Box\ (\ a \wedge \bigcirc b \Rightarrow c\ )$ se raffinent toujours par des propriétés de même forme alors que ce n'est pas le cas des propriétés de la forme $\Box\ (\ d \Rightarrow \bigcirc e\ )$. Les raisons qui déterminent le raffinement ne sont donc pas syntaxiques mais probablement liées au paradigme de reformulation utilisé qui est différent pour obtenir une forme $\Box\ (\ a \wedge \bigcirc b \Rightarrow c\ )$ et pour obtenir $\Box\ (\ d \Rightarrow \bigcirc e\ )$.

4.2 Raffinements multiples avec changement de structure temporelle

Toutes les formes temporelles citées dans cette section se raffinent en premier lieu par elles-mêmes. Par exemple $\Box\ (\ a \Rightarrow \bigcirc b\ )$ se raffine par $\Box\ (\ a' \Rightarrow \bigcirc b'\ )$. Mais elles se raffinent par d'autres formes, c'est ces autres possibilités de raffinement que nous présentons ici.

4.2.1 Raffinements de la réponse immédiate : $\Box\ (\ a \Rightarrow \bigcirc b\ )$

La figure 4.1 montre les deux possibilités de raffinement de cette forme : $\Box\ (\ a' \Rightarrow \Diamond b'\ )$ et $\Box\ (\ a' \Rightarrow c'\ \mathcal{U}\ b'\ )$. Ces deux dernières permettent de retarder le moment où la propriété b, raffinée par b', est vérifiée. La différence réside dans le maintien dans la seconde forme de la validité d'une propriété c'.

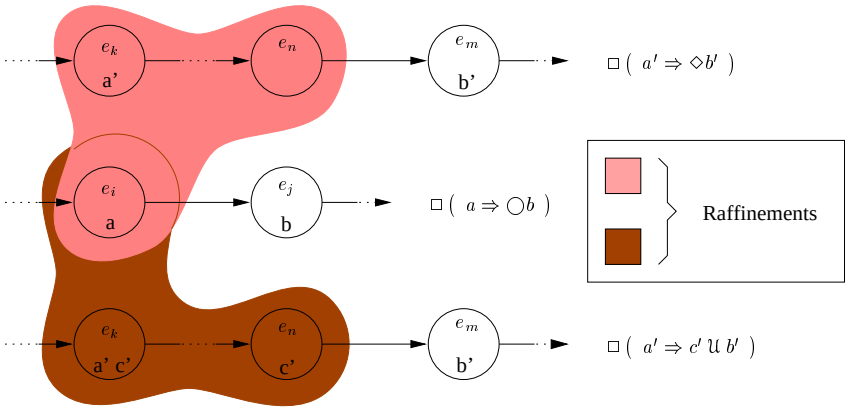


Figure 4.1: Raffinements de la réponse immédiate : $\Box\ (\ a \Rightarrow \bigcirc b\ )$

L'état abstrait e_i est raffiné par le méta-état débutant par e_k et se terminant par e_n. L'étude des divers exemples rencontrés montre que les propriétés a' et b' sont des raffinements de,

respectivement, a et b. La propriété c' parait avoir un lien étroit avec la propriété a'. En effet cette dernière est souvent de la forme $a' \Leftrightarrow (d' \wedge c')$.

De $\Box\ (\ a \Rightarrow \bigcirc b\ )$	Propriétés abstraites	Propriétés raffinées
vers $\Box\ (\ a' \Rightarrow \Diamond b'\ )$	C_2^1 C_{10}^2 B_3^0	C_2^2 C_{10}^3 B_3^1
vers $\Box\ (\ a' \Rightarrow c'\ \mathcal{U}\ b'\ )$	A_1^0	A_1^1

Tableau 4.2: Exemples de raffinements de la réponse immédiate

La table 4.2 dresse une liste non exhaustive des raffinements de ce type rencontrés dans les exemples.

Le tableau 4.3 récapitule les différentes possibilités de raffinement de la *réponse immédiate*.

Propriété abstraite	Propriétés raffinées
$\Box\ (\ a \Rightarrow \bigcirc b\ )$	$\Box\ (\ a' \Rightarrow \bigcirc a'\ )$ $\Box\ (\ a' \Rightarrow \Diamond b'\ )$ $\Box\ (\ a' \Rightarrow (\ c'\ )\ \mathcal{U}\ (\ b'\ )\ )$

Tableau 4.3: Raffinements de la réponse immédiate : $\Box\ (\ a \Rightarrow \bigcirc b\ )$

4.2.2 Raffinements de l'obligation pour un événement : $\Box\ (\ a \wedge \bigcirc b \Rightarrow \bigcirc c\ )$

Parmis les exemples étudiés, le seul raffinement particulier pour l'*obligation pour un événement* est la réponse *par un invariant local à un invariant local* $\Box\ (\ a' \wedge d'\ \mathcal{U}\ b' \Rightarrow e'\ \mathcal{U}\ c'\ )$ figurant dans le tableau 4.4.

Propriété abstraite	Propriété raffinée
B_{13}^3	B_{13}^4

Tableau 4.4: Exemple de raffinement de l'obligation pour un événement

Comme le montre la figure 4.2, cette forme de raffinement permet de retarder le moment où la propriété b' raffinement de b sera vérifiée. Il y a aussi obligation pour une propriété e' d'être vraie jusqu'à ce que c', qui raffinent la propriété c, soit valide. Comme cela vient d'être dit, les propriétés b et c sont raffinées respectivement par b' et c'. Quant à a', elle est raffinée par a. Les propriétés d' et e' semblent avoir avec les autres propriétés les relations suivantes : $a' \Leftrightarrow (\underbrace{d' \wedge f'}_{e'} \wedge g')$.

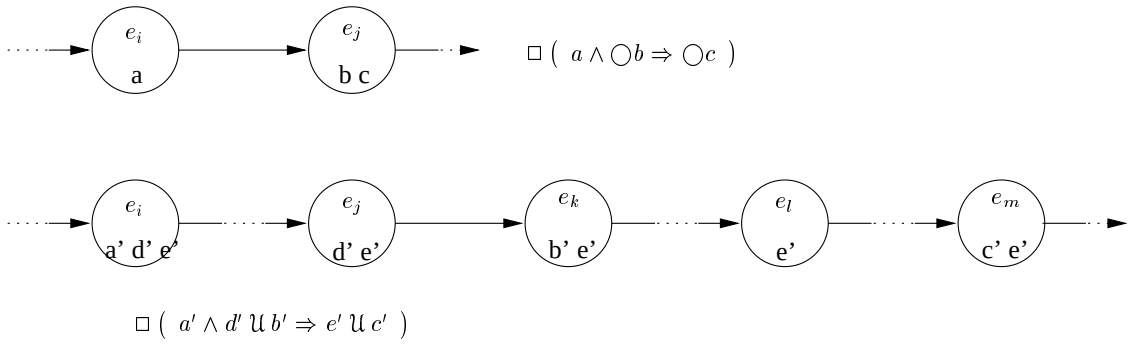


Figure 4.2: Raffinement de $\Box$ ($a \wedge \bigcirc b \Rightarrow \bigcirc c$) vers $\Box$ ($a' \wedge d' \,\mathcal{U}\, b' \Rightarrow e' \,\mathcal{U}\, c'$)

La figure 4.3 montre que d'autre possibilités de raffinement de l'*obligation pour un événement* existent. Elles permettent d'aboutir aux formes $\Box$ ($a' \wedge \bigcirc b' \Rightarrow \bigcirc \bigcirc c'$) et $\Box$ ($a' \wedge \bigcirc b' \Rightarrow d' \,\mathcal{U}\, c'$). Ces formes de raffinement ne sont pas issues des exemples présentés mais sont apparues durant les différentes versions des spécifications réalisées au cours du temps.

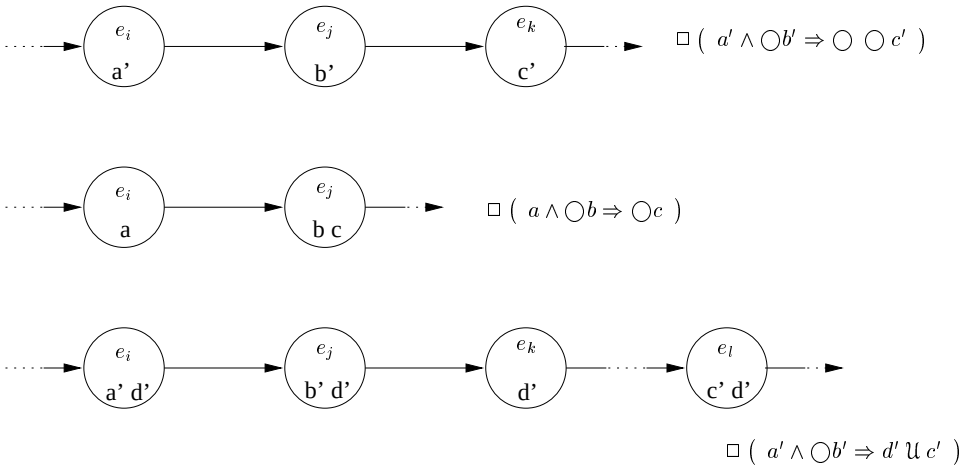


Figure 4.3: Raffinements supplémentaires de l'obligation pour un événement

La première consiste à retarder d'un état le moment où la propriété c' raffinant c est vraie. Le seconde permet non seulement de retarder c' mais aussi oblige le maintien de la propriété d'. Il semble cohérent de dire que cette dernière propriété est en relation avec la a' par $a' \Leftrightarrow (d' \wedge e')$.

Le tableau 4.5 présente les différentes possibilités de raffinement de l'*obligation pour un événement*.

Propriété abstraite	Propriétés raffinées
$\Box$ ($a \wedge \bigcirc b \Rightarrow \bigcirc c$)	$\Box$ ($a' \wedge \bigcirc b' \Rightarrow \bigcirc c'$)
	$\Box$ ($a' \wedge d' \,\mathcal{U}\, b' \Rightarrow e' \,\mathcal{U}\, c'$)
	$\Box$ ($a' \wedge \bigcirc b' \Rightarrow \bigcirc \bigcirc c'$)
	$\Box$ ($a' \wedge \bigcirc b' \Rightarrow d' \,\mathcal{U}\, c'$)

Tableau 4.5: Raffinements de l'obligation pour un événement : $\Box$ ($a \wedge \bigcirc b \Rightarrow \bigcirc c$)

4.2.3 Raffinements de l'enchaînement d'événements : $\Box\ (\ a \wedge \bigcirc b \Rightarrow \bigcirc\ \bigcirc c\ )$

La figure 4.4 représente le raffinement de la propriété d'*enchaînement des événements* trouvé dans les exemples étudiés. Il permet de transformer $\Box\ (\ a \wedge \bigcirc b \Rightarrow \bigcirc\ \bigcirc c\ )$ en $\Box\ (\ a' \wedge \bigcirc b' \Rightarrow d'\ \mathcal{U}\ c'\ )$. Grâce à cette forme de raffinement, il est possible de retarder le moment où la propriété c, raffinée par c', est vraie tout en forçant d' a être vérifiée.

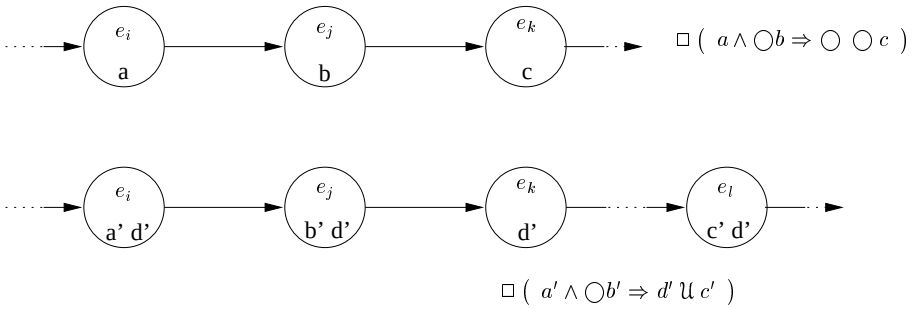


Figure 4.4: Raffinement de $\Box\ (\ a \wedge \bigcirc b \Rightarrow \bigcirc\ \bigcirc c\ )$ vers $\Box\ (\ a' \wedge \bigcirc b' \Rightarrow d'\ \mathcal{U}\ c'\ )$

L'exemple présenté dans la table 4.6 permet de dire que les propriétés a', b' et c' sont des raffinements de respectivement a, b et c. La propriété d' semble avoir une relation étroite avec a'. En effet, on peut dire que $a' \Leftrightarrow e' \wedge f'$ et que $d' \Leftrightarrow e' \wedge g'$.

Propriété abstraite	Propriété raffinée
B_8^1	B_8^2

Tableau 4.6: Exemple de raffinement de l'enchaînement d'événements

Il est possible de dégager une autre forme de raffinement en dehors des exemples. En effet, comme le montre la figure 4.5, une expression de la forme de l'enchaînement d'événements peut se raffiner par $\Box\ (\ a' \wedge d'\ \mathcal{U}\ b' \Rightarrow e'\ \mathcal{U}\ c'\ )$. Elle permet d'insérer de nouveaux états entre celui où a raffinée par a' sera vrai et celui où b raffinée par b' sera lui aussi vérifié. Cette forme de raffinement oblige aussi une condition e' a être vraie jusqu'à ce que la propriété raffinant c soit vraie. Les relations des propriétés d' et e' avec les autres sont les mmes que celles liant les propriétés de mme nom dans le raffinement de $\Box\ (\ a \wedge \bigcirc b \Rightarrow \bigcirc c\ )$ vers $\Box\ (\ a' \wedge d'\ \mathcal{U}\ b' \Rightarrow e'\ \mathcal{U}\ c'\ )$.

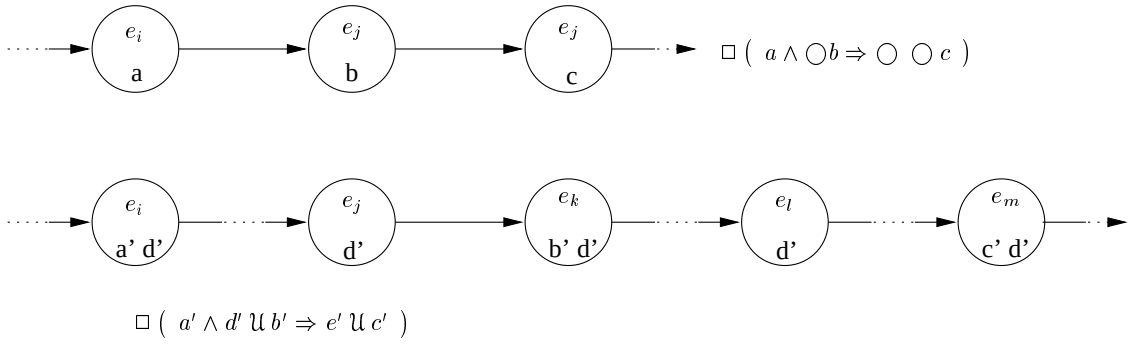


Figure 4.5: Raffinement suplémentaires de l'enchaînement d'événements

La table 4.7 récapitule les différentes possibilités de raffinement de la propriété d'enchaînement des événements.

Propriété abstraite	Propriétés raffinées
$\Box\,(\ a \wedge \bigcirc b \Rightarrow \bigcirc\ \bigcirc c\ )$	$\Box\,(\ a' \wedge \bigcirc b' \Rightarrow \bigcirc\ \bigcirc c'\ )$ $\Box\,(\ a' \wedge \bigcirc b' \Rightarrow d'\ \mathcal{U}\ c'\ )$ $\Box\,(\ a' \wedge d'\ \mathcal{U}\ b' \Rightarrow e'\ \mathcal{U}\ c'\ )$

Tableau 4.7: Raffinements de l'enchaînement d'événements: $\Box\,(\ a \wedge \bigcirc b \Rightarrow \bigcirc\ \bigcirc c\ )$

Partie II

Algorithme de détection de raffinement de graphe

Chapitre 5

Définition de l'algorithme de vérification de raffinement

Dans ce chapitre nous allons nous attacher à la définition d'un algorithme qui permet de vérifier l'existence d'un raffinement entre deux graphes.

5.1 Définition informelle de l'algorithme

Nous nous proposons de développer un algorithme qui aura pour but de vérifier que deux graphes d'accessibilité se raffinent. Intuitivement, le raffinement de graphe respecte les règles suivantes :

- on regroupe les états du graphe raffinés dans des méta-états. Chacun de ces derniers correspond à un état du graphe abstrait;
- les états appartenant à un méta-états possèdent la même valuation à l'invariant près que l'état abstrait correspondant;
- seules les transitions existant dans le système abstrait peuvent exister entre deux méta-états. Les nouvelles transitions sont internes aux méta-états;
- aucun cycle n'est permis à l'intérieur des méta-états. Ceci permet d'éviter les situations de blocage du système;
- si le système abstrait possède des transitions non-déterministes, on en retrouve au moins une dans le graphe raffiné;

En partant de deux graphes d'accessiblité, l'algorithme devra non seulement vérifier le raffinement mais aussi retourner l'ensemble des méta-états rencontrés durant cette analyse.

5.2 Définitions préalables

5.2.1 Définition d'une spécification

Une spécification est une représentation formelle de la solution apportée à l'analyse d'un problème. Une spécification ne correspond qu'à un seul niveau de raffinement. Elle contient aussi bien les variables utilisées pour modéliser le système que les contraintes s'y rapportant.

Définition 1 *Une spécification de niveau i est définie par* $SP_i =< A_i, V_i, P_{init}, P_A, I_i, Cp_i >$ *où*

- A_i *est l'ensemble des noms des événements du système,*

- V_i *est l'ensemble des variables du même système,*
- P_{init} *est l'état initial des variables,*
- P_A *est l'ensemble des événements du système,*
- I_i *est l'invariant,*
- Cp_i *est l'ensemble des propriétés temporelles.*

Exemple 1 *L'exemple suivant est issu de l'Automatisme Industriel décrit page v. Il représente la spécification formelle du robot.*
$AI_0 =< A_0, V_0, P_{init}, P_A, I_i, Cp_i >$ *avec*

$$\begin{aligned} A_0 &= \{Chargt_0, DéChargt_0, Evac_0\} \\ V_0 &= \{Dt_0, De_0\} \\ P_{init} &= (Dt_0 = libre \wedge De_0 = libre) \\ P_A &= \{Chargt_0 \stackrel{\triangle}{=} \dots \\ &\qquad DéChargt_0 \stackrel{\triangle}{=} \dots \\ &\qquad Evac_0 \stackrel{\triangle}{=} \dots\} \\ I_i &= Dt_0 \in \{libre, occupé\} \wedge De_0 \in \{libre, occupé\} \\ Cp_i &= \{A_1^0\} \end{aligned}$$

5.2.2 Définition d'un système de transitions

Un système de transitions est directement issu d'une spécification. Il correspond à une définition formelle d'un graphe d'accessibilité. Un système de transition est toujours associé à une spécification.

Définition 2 *On définit un système de transitions ST_i issu d'une spécification SP_i de la manière suivante :*
$ST_i =< A_i, V_i, E_i, e_i^0, T_i, \alpha_i, \beta_i, \delta_i >$ *où*

- A_i *est l'ensemble des noms des événements du système,*
- V_i *est l'ensemble des variables du système*
- E_i *est l'ensemble des états atteignables du système,*
- e_i^0 *est l'état initial du système de transitions,*
- T_i *est l'ensemble des transitions,*
- α_i *est une fonction permettant de récupérer l'état initial d'une transition :* $T_i \rightarrow E_i$,
- β_i *est une fonction donnant l'état final d'une transition :* $T_i \rightarrow E_i$,
- λ_i *est aussi une fonction permettant de connaitre le nom de la transition, c'est à dire le nom de l'événement correspondant :* $T_i \rightarrow A_i$.

Exemple 2 *Cet exemple présente le système de transitions correspondant au niveau formel de la spécification de l'Automatisme Industriel.*

$ST_0 =< A_0, V_0, E_0, e_0^0, T_0, \alpha_0, \beta_0, \lambda_0 >$ *avec*

$$
\begin{aligned}
A_0 &= \{Chargt_1, DéChargt_1, Evac_1\} \\
V_0 &= \{Dt_1, De_1\} \\
E_0 &= \{e_0, e_1, e_2, e_3\} \\
e_0^0 &= e_0 \\
T_0 &= \{t_0^1 = e_0 \overset{Chargt_1}{\longrightarrow} e_1, t_0^2 = e_1 \overset{DéChargt_1}{\longrightarrow} e_2, \\
&\qquad t_0^3 = e_2 \overset{Evac_1}{\longrightarrow} e_0, t_0^4 = e_2 \overset{DéChargt_1}{\longrightarrow} e_3, \\
&\qquad t_0^5 = e_3 \overset{Evac_1}{\longrightarrow} e_1\} \\
\alpha_0(t_0^1) &= e_0 \\
\beta_0(t_0^1) &= e_1 \\
\lambda_0(t_0^1) &= Chargt_1 \\
&\vdots \\
\alpha_0(t_0^3) &= e_2 \\
\beta_0(t_0^3) &= e_0 \\
\lambda_0(t_0^3) &= Evac_1 \\
&\vdots
\end{aligned}
$$

5.3 Raffinement de systèmes de transitions

Afin de pouvoir développer l'algorithme, il est nécessaire de définir des règles de raffinement de systèmes de transitions. Elles représentent une partie des propriétés intuitives énoncées au début de ce chapitre.

Définition 3 *Le système de transition ST_2 raffine le système de transition ST_1, noté $ST_1 \sqsubseteq ST_2$, si et seulement si*

- *L'ensemble des événements abstraits est inclu dans le système raffiné :*

$$A_1 \subseteq A_2 \tag{5.1}$$

- *Les états initiaux abstrait et concret possèdent la même valeur à l'invariant près :*

$$e_2^0 \wedge I_2 \Rightarrow e_1^0 \tag{5.2}$$

- *La valuation des états raffinés ne correspond qu'à celle d'un seul et unique état abstrait :*

$$
\begin{aligned}
&\textit{Pour } i \geq 0, \exists j \geq 0 \textit{ tel que } e_2^i \wedge I_2 \Rightarrow e_2^j \\
&\qquad\qquad \nexists k \geq 0 \textit{ tel que } k \neq j \textit{ et } e_2^i \wedge I_2 \Rightarrow e_1^k
\end{aligned} \tag{5.3}
$$

- *Tout chemin σ_2 de longueur finie n de ST_2 ne possédant qu'une seule transition t_1 dont le nom appartienne à ST_1 se trouvant à la fin de σ_2, raffine la transition t_1 du système abstrait :*

$\forall (t_2^1, \ldots, t_2^n) \in$ *chemins finis de longueur n de* ST_2 *tel que* $\lambda_2(t_2^i) \notin A_1$ *pour* $i \in [1, n-1]$,
$\exists t_1 \in$ *chemins de longueur 1 de* ST_1 *tel que*

$$\lambda_1(t_1) = \lambda_2(t_2^n) \tag{5.4}$$

$$\alpha_2(t_2^j) \wedge I_2 \Rightarrow \alpha_1(t_1) \text{ pour } j \in [1, n] \tag{5.5}$$

$$\beta_2(t_2^n) \wedge I_2 \Rightarrow \beta_1(t_1) \tag{5.6}$$

- *Pour tout ensemble de transitions non-déterministes*[1] *de nom* η *dans le système* ST_1*, il existe un chemin* σ_2 *de longueur n dans* ST_2 *possédant une seule transition abstraite. Cette dernière se trouve à la fin de* σ_2 *et est étiquetée par* η *:*
$\forall t_1^1, \ldots, \forall t_1^k$ *où* $t_1^j \in$ *chemins de longeur 1 de* ST_1 *pour* $j \in [1, k]$ *tel que pour* $j \in [2, k]$

$$\begin{cases} \alpha_1(t_1^j) = \alpha_1(t_1^1) \\ \lambda_1(t_1^j) = \lambda_1(t_1^1), \end{cases}$$

$\exists (t_2^1, \ldots, t_2^n) \in$ *chemins finis de longueur n de* ST_2 *tel que*

$$\lambda_1(t_1^1) = \lambda_2(t_2^n) \tag{5.7}$$

$$\alpha_2(t_2^j) \wedge I_2 \Rightarrow \alpha_1(t_1^1) \tag{5.8}$$

$$\beta_2(t_2^n) \wedge I_2 \Rightarrow \beta_1(t_1^1) \tag{5.9}$$

- *Aucun chemin de* ST_2 *composé uniquement de nouvelles transitions ne peut faire un cycle :*
$\nexists (t_2^1, \ldots, t_2^n) \in$ *chemin infini de* ST_2 *tel que*

$$\lambda_2(t_2^i) \notin A_1 \text{ pour } i \geq 1 \tag{5.10}$$

- *Tout chemin* σ_2 *de* ST_2 *débutant par une nouvelle transition doit se terminer par un transition abstraite :*
$\forall t_2 \in T_2$ *tel que* $t_2 \notin A_1$,
$\exists (t'^1_2, \ldots, t'^n_2) \in$ *chemins de longueur n de* ST_2 *tel que pour* $j \in [1, n]$ $\lambda_2(t'^j_2) \notin A_1$,
$\exists t''_2 \in T_2 \wedge \lambda_2(t''_2) \in A_1$,

$$\beta_2(t_2) = \alpha_2(t'^1_2) \wedge \beta_2(t'^n_2) = \alpha_2(t''_2) \tag{5.11}$$

5.4 Choix d'implémentation

Dans cette section nous allons nous attacher à la description des différents choix d'implémentation. Il ne faut pas perdre de vue les deux objectifs de l'algorithme : la détection du raffinement et la répartition des états raffinés dans des méta-états.

5.4.1 Parcours méta-état par méta-état

Afin de pouvoir construire facilement le métat-graphe[2] l'algorithme parcourt parallèlement les deux graphes donnés à l'aide de deux piles. La première ne contient que les transitions existant dans le graphe abstrait. La seconde pile n'est destinée qu'à stocker les transitions fesant leur apparition dans le graphe raffiné. Afin mieux représenter le parcours méta-état par méta-état, l'algorithme traite d'abord les transitions contenues dans la seconde pile puis celles de la première. Les piles

[1] Les transitions non-déterministes possèdent le même état source et la même étiquette
[2] Regroupement des états raffinés

sont remplies après chaque traitement d'une transition en ajoutant les événements qui partent de cette dernière. Le parcours des graphe est réalisé de manière asynchrone, c'est à dire que l'on change d'état abstrait uniquement si la transition abstraite et la transition raffinée possèdent la même étiquette.

5.4.2 Détection des cycles

Comme le montre la figure 5.1, seuls les cycles internes aux méta-états sont invalides. D'autre part, se limiter à la vérification que l'état n'a pas déjà été traité ne suffit pas à la détection de cycles. En effet, plusieurs chemins dans un méta-état peuvent passer par un même état sans pour cela créer de cycle.

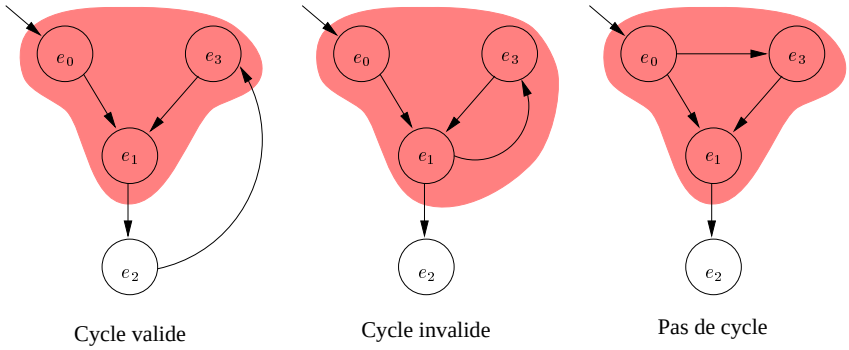


Figure 5.1: Détection des cycles dans l'algorithme de vérification du raffinement

L'algorithme mémorise le chemin parcouru à l'intérieur du méta-état en cours de traitement. A chaque fois qu'il rencontre un état déjà traité, il vérifie qu'aucun cycle ne se produit. La structure contenant le chemin parcouru est vidé à chaque fois que l'algorithme doit changer de méta-état.

5.4.3 Equivalence observationnelle pour le non-déterminisme externe

Il y a équivalence observationnelle entre un graphe abstrait et un graphe raffiné si toutes les transitions abstraites peuvent être éxécutées à partir de tous les états raffinant le point de départ des événements.

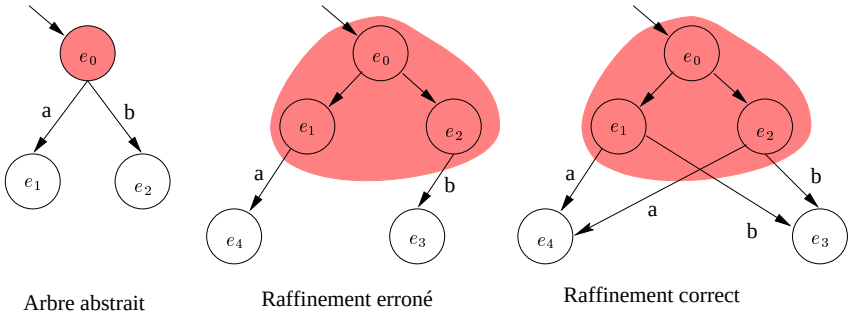


Figure 5.2: Equivalence observationnelle entre les graphes abstrait et raffiné

Plus simplement, la figure 5.2 montre un exemple d'équivalence observationnelle. Le graphe se trouvant au centre n'est pas équivalent au graphe abstrait car la transition b de ce dernier n'est plus accessible à partir de l'état raffiné e_1. Le graphe le plus à droite montre comment rendre

l'équivalence opérationnelle valide: la transition b a été ajoutée entre les états e_1 et e_4 et a entre e_2 et e_4.

5.4.4 Equivalence de traces pour le non-déterminisme interne

Les événements non-déterministes sont traités de manière particulière. Contrairement aux autres types d'événements, il n'est pas nécessaire que toutes les transitions apparaissent dans le graphe raffiné. La seule contrainte est la nécessité de la présence d'au moins une d'entre elles. Comme le montre la figure 5.3, toutes les transitions non-déterministes ne sont pas forcément accessible à partir de tous les états. En effet, la transition a menant à l'état e_2 ne peut plus être déclenchée si le système arrive dans l'état e'_2. La transition a menant à l'état e_3 disparait totalement. Par contre le choix entre les transitions a et b est toujurs possible quelque soit le chemin raffiné pris.

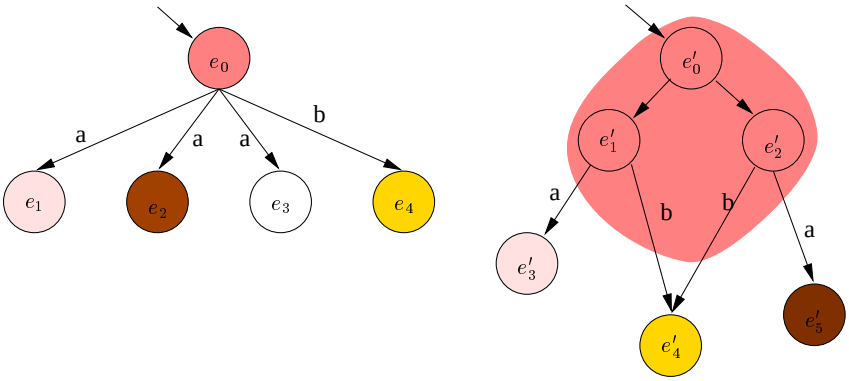


Figure 5.3: Raffinement de transitions non-déterministes

L'état e_0 est raffiné par les états e'_0, e'_1, e'_2 et les états e_1, e_2 et e_4 sont respectivement raffinés par les états e'_3, e'_5 et e'_4. e_3 ne possède pas raffinement.

L'algorithme devra donc tenir compte de cette caractéristique des événements non-déterministes. Pour cela lorsqu'il vérifie la valuation des états abstrait et concret, l'algorithme recherche l'état abstrait qui convient parmi tous ceux qui sont accessibles à partir de l'une des transitions non-déterministes abstraites.

5.4.5 Le méta-graphe

L'algorithme renvoie la répartition des états raffinés à l'aide de structures appellées méta-graphes. Comme le montre la figure 5.4, le graphe raffiné est transfomé en un tableau de méta-graphes. Ces derniers sont presque identiques à des graphes traditionnels. La différence réside dans le fait que les méta-graphes possèdent des références vers d'autres méta-graphes. Il ne s'agit pas à proprement parler de transitions mais d'informations indiquant qu'une transition lie deux méta-graphes (exemple : M_1, e_0).

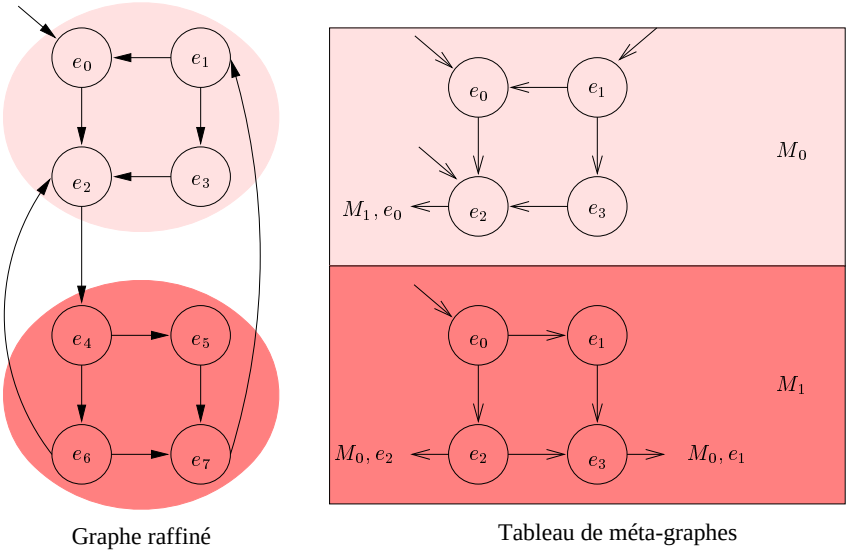


Figure 5.4: Représentation sous formes de méta-graphes

Une autre particularité des méta-graphes est l'existance de plusieurs états initiaux. Ces derniers représentent ici soit l'état inital du graphe raffiné soit l'arrivée d'une transition entre deux méta-graphes.

Chapitre 6

Implémentation algorithmique

Dans ce chapitre, nous allons nous intéresser à l'implémentation de l'algorithme de détection de raffinement de graphes.

6.1 Définition d'une structure de pile

La classe suivante est une structure généralisée représentant une pile composée d'élements de type β. Elle est utilisée par l'algorithme pour stocker les informations de parcours des deux graphes à analyser.

Classe Objet Pile de β
  Public:
    Procédure Clear { *Destruction de tous les éléments de la pile* }
    Fonction Pop : β { *Renvoie et retire l'élément en sommet de pile* }
    Fonction Exists (e : β) : **Booléen** { *Indique si e appartient à la pile* }
    Fonction IsEmpty : **Booléen** { *Renvoie* ***Vrai*** *si la pile est vide* }
Fin

6.2 Définition d'un graphe

Les graphes d'accessibilité sont stockés dans la classe suivante. Elle permet entre autre de connaître le nombre de noeuds (nodes) et de transitions ainsi que les valeurs des états. Ces dernières peuvent être d'un type quelconque β. Chaque noeud du graphe est identifié par un numéro compris dans l'intervale [0..Nombre de noeuds − 1].

Classe Objet Graphe de β
Public:
Fonction GetNodeCount : **Entier** { *Retourne le nombre de noeuds dans le graphe* }
Fonction GetDiffTransCount (e : **Entier**) : **Entier** { *Retourne le nombre de transitions distinctes partant du noeud numéro e. Les transitions non déterministes sont regroupées en une seule transition.* }
Fonction GetDiffNodeCount (**Entier**) : **Entier** { *Retourne le nombre d'états différents pouvant être atteint depuis l'état numéro e* }
Fonction GetFirstNode : **Entier** { *Renvoi le noeud initial du graphe* }
Fonction FindFirst (**Entier**; **Entier par Ref**; **Chaîne par Ref**) : **Booléen** { *Les deux fonctions suivantes permettent de parcourir l'ensemble des transitions partant d'un noeud* }
Fonction FindNext (**Entier par Ref**; **Chaîne par Ref**) : **Booléen** { *Les deux fonctions suivantes permettent de parcourir l'ensemble des transitions partant d'un noeud* }
Fonction FindFirstNoDeterminist (**Entier**; **Chaîne par Ref**; **Entier par Ref**) : **Booléen** { *Les deux fonctions suivantes permettent de parcourir les transitions non-déterministes partant d'un noeud* }
Fonction FindNextNoDeterminist (**Entier par Ref**) : **Booléen** { *Les deux fonctions suivantes permettent de parcourir les transitions non-déterministes partant d'un noeud* }
Fonction IsTrans (o : **Entier**; n : **Chaîne**; d : **Entier par Ref**) : **Booléen** { *Retourne **Vrai** si la transition de nom n partant du noeud numéro o existe. Cette fonction retourne aussi le numéro d du noeud destination* }
Fonction IsTransWithout (e : **Entier**; n : **Chaîne**) : **Booléen** { *Retourne **Vrai** si une transition possédant le nom n et ne partant pas du noeud numéro e existe* }
Fonction GetNodeValue (n : **Entier**) : β { *Retourne la valeur du noeud numéro n* }
Fin

6.3 Définition de la valeur d'un état du système

La classe BValue permet de stocker les valeurs des variables d'un système pour chacun des états de celui-ci. Une instance de cette classe est créée pour chaque noeud des graphes d'accessibilité abstrait et raffiné. Elles permettent, couplées avec l'invariant du système raffiné, de vérifier que les état raffinés possèdent les mêmes valeurs que les états abstraits à l'invariant près. Cette vérification est imposée par les formules 5.2 et 5.3 page 93. La vérification de la formule $e_2 \wedge I_2 \Rightarrow e_1$, où e_i représente l'ensemble des valuations des variables au niveau de raffinement i et I_i est l'invariant de niveau i, peut être réalisée soit par un prouveur existant comme L'Atelier B, soit grâce à la programmation d'un prouveur dans un langage informatique quelconque.

6.4 Définition d'une structure interne

L'objet suivant est une structure interne à l'algorithme. Elle permet de stocker une transition se produisant à la fois dans le graphe abstrait et dans le graphe concret.

Classe Objet GrapheTransition
Public:
Fonction Label : **Chaîne par Ref** { *Retourne le nom de la transition* }
Fonction LastAbstract : **Entier par Ref** { *Retourne le numéro de l'état abstrait de départ* }
Fonction Abstract : **Entier par Ref** { *Retourne le numéro de l'état abstrait destination* }
Fonction LastRefinement : **Entier par Ref** { *Retourne le numéro de l'état concret de départ* }
Fonction Refinement : **Entier par Ref** { *Retourne le numéro de l'état concret destination* }
Fonction NoDeterminist : **Booléen par Ref** { *Retourne une valeur indiquant si la transition est non-déterministe* }
Fonction SameAbstract : **Booléen** { *Retourne une valeur indiquant si la transition permet de rester dans le même méta-état* }
Fin

6.5 Définition d'un méta-graphe

L'algorithme étudié doit indiquer non seulement si un graphe est un raffinement d'un second, mais doit aussi retourner une structure regroupant les états du système raffiné en méta-états. Cette dernière est modélisée par l'objet suivant. Elle permet de stocker un graphe mais aussi les différentes transitions amenant vers un autre méta-graphe.

Classe Objet MetaGraphe
dérivé de Graphe de BValue
Public:
Fonction GetDiffNodeCount : **Entier** { *Retourne le nombre de méta-états distincts pouvant êtres accessibles à partir du méta-état courant* }
Fonction AddInitialNode (e : **Entier**) : **Booléen** { *Ajoute l'état numéro e dans la liste des états initiaux. e doit déjà existé dans le méta-graphe* }
Fonction AddMetaTrans (o : **Entier**, dm : **Entier**, de : **Entier**, n : **Chaîne**) : **Booléen** { *Ajoute une transition entre deux méta-etats nommée n partant de l'état o et arrivant dans l'état de du méta-état dm* }
Procédure AddNode (v : BValue) { *Ajoute un noeud de valeur v dans le méta-graphe* }
Procédure AddTrans (o, d : **Entier**; n : **Chaîne**) { *Ajoute une transition interne allant de l'état o à l'état d et d'étiquette n* }
Fin

La structure renvoyée par l'algorithme correspondra à un ensemble de méta-graphes ayant des interconnexions entre eux. Il reste à remarquer qu'un méta-état ne stocke pas les transitions vers un autre méta-état mais y fait référence.

6.6 Algorithme de vérification du raffinement de graphe

6.6.1 Algorithme

Dans cette section est présenté l'algorithme de détection du raffinement de graphe. La partie qui suit correspond à la déclaration des variables ainsi qu'aux initialisations "triviales".

```
{ GAbstait et GRaffine correspondent au graphe abstrait et au graphe raffiné. Invariant2 est
l'invariant du ssystème raffiné. ListeG est la liste des méta-graphes renvoyée par l'algorithme
et contenant une représentation du graphe raffiné sous forme de méta-états }
Fonction IsGrapheRefinement (GAbstrait, GRaffine : Graphe de BValue;  Invariant2 :
                                        Invariant : ;  ListeG : Tableau de MetaGraphe ) : Booléen
Constante:
  PasUtilise : Entier = 0
Variables:
  { Cette variable permet de stocker les numéros des états du graphe concret dans les
  méta-graphes générés. Si la valeur contenu dans ce tableau est égale à 0 alors l'état
  correspondant n'a pas encore été traité }
  IndiceSGraphe : Tableau de Entier
  { Les deux variables suivantes représentent les piles utilisées par l'algorithme pour stocker
  les différentes transitions qu'il lui reste a traiter. La première correspond aux nouvelles
  transitions, c'est à dire celles qui n'existent pas dans le graphe abstrait;  alors que la
  seconde ne contient que des transitions apparaissant dans le graphe abstrait }
  PileNveauxEvt, PileEvtAbstraits : Pile de GrapheTransition
  { La variable suivante permet à l'algorithme de stocker le chemin déjà parcouru dans le
  graphe concret. Elle permet une détection simple des cycles à l'intérieur des méta-états }
  Chemin : Pile d'Entier
  { La variable suivante correspond à la transition en cours de traitement }
  Transition : GrapheTransition
  { Les variables suivantes permettent de stocker temporairement les valeurs d'une nouvelle
  transition }
  NouvTransition : GrapheTransition
  NouvAbs, NouvRaf : Entier
  NouvNom : Chaîne
  { La variable suivante permet de compter le nombre de transitions ajoutées dans la pile
  PileNveauxEvt }
  NbAbsEvt : Entier
  { La variable suivante indique si une transition a été ajoutée dans la pile PileEvtAbstraits }
  NveauEvt : Booléen
  { La variable suivante indique si la transition en cours de traitement est la première }
  Premier : Booléen
  i : Entier
  Valeur : BValue
Début
  Si GAbstrait.GetNodeCount ≤ 0 ∨ GRaffine.GetNodeCount ≤ 0 alors
    { L'un des deux graphes est vide !!! }
    IsGrapheRefinement ← Faux
    Sortie
  Fin Si

  { Aucun état raffiné n'a été traité }
  Pour i ← 0 à GRaffine.GetNodeCount - 1  faire
    IndiceSGraphe [ i ] ← PasUtilise
  FPour

  { Initialisation du tableau des méta-états }
  ListeG.SetSize (GAbstrait.GetNodeCount)
  { L'algorithme traite la transition initiale }
  Premier ← Vrai
```

La partie suivante de l'algorithme représente l'initialisation de la pile des événements abstraits.

La transition initiale existant dans le graphe abstrait, elle doit être ajoutée dans la pile PileEvtAbstraits. Ce choix d'initialisation permet à l'algorithme de considérer qu'il est initialement en train de traité un nouvel méta-état.

```
– Suite de l'algorithme IsGrapheRefinement –
  NouvTransition.LastAbstract ← GAbstrait.GetFirstNode - 1
  NouvTransition.LastRefinement ← GRaffine.GetFirstNode - 1
  NouvTransition.Abstract ← GAbstrait.GetFirstNode
  NouvTransition.Refinement ← GRaffine.GetFirstNode
  PileEvtAbstraits.Push (NouvTransition)
```

L'algorithme est prévu pour traiter toutes les transitions rencontrées lors du parcours des graphes. Il doit donc boucler tant que l'une des deux piles possède un élément. Si ce cas se produit, l'algorithme traite en priorité les nouvelles transitions. Cette heuristique permet d'analyser les deux graphes méta-état par méta-état.

```
– Suite de l'algorithme IsGrapheRefinement –
  Tant que ¬ PileEvtAbstraits.IsEmpty ∨ ¬ PileNveauxEvt.IsEmpty faire
    Si ¬ PileNveauxEvt.IsEmpty alors
      { Traitement des nouvelles transitions en priorité }
      Transition ← PileNveauxEvt.Pop
      { Mise à jour du chemin en retirant les états inutiles }
      Chemin.Pop (Transition.LastRefinement)
    sinon
      { Traitement des transitions existant dans le graphe abstrait en second lieu }
      Transition ← PileEvtAbstraits.Pop
      { Le chemin parcouru dans le méta-état est vide car on change de méta-état }
      Chemin.Clear
    Fin Si
```

La partie suivante permet de vérifier que les états abstrait et concret possèdent les mêmes valeurs pour les variables du système à l'invariant près. Il faut toutefois différencier le cas des transitions non-déterministes de celles ne l'étant pas.

Dans le cas particulier des transitions non-déterministes, l'algorithme recherche l'état abstrait devant posséder la même valeur que l'état concret. Comme tous les états d'un graphe ont une valuation unique, il n'existe qu'une seule solution si elle existe.

```
– Suite de l'algorithme IsGrapheRefinement –
    Si Transition.NoDeterminist alors
      { Test du raffinement des états en considérant des transitions non déterministes }
      Arret ← ¬ GAbstrait.FindFirstNoDeterminist ( Transition.LastAbstract,
      Transition.Label, NouvAbs )

      Tant que ¬ Arret ∧ ¬ ( GAbstrait.GetNodeValue ( NouvAbs ) ∧Invariant2 ⇒
      GRaffine.GetNodeValue ( Transition.Refinement )) faire
        Arret ← ¬ GAbstrait.FindNextNoDeterminist ( NouvAbs )
      FTq

      Si Arret alors
        { Aucun état destination n'a été trouvé parmis les possibilités non déterministe }
        IsGrapheRefinement ← Faux
        Sortie
      Fin Si

      Transition.Abstract ← NouvAbs
```

Cette partie de l'algorithme permet de vérifié que les états abstrait et concret possèdent la

même valeur dans le cas d'une transition déterministe.

```
– Suite de l'algorithme IsGrapheRefinement –
    sinon Si ¬ ( GAbstrait.GetNodeValue ( Transition.Abstract ) ∧Invariant2 ⇒
    GRaffine.GetNodeValue ( Transition.Refinement ) ) alors
      { Les deux états ne possèdent pas la même valeur }
      IsGrapheRefinement ← Faux
      Sortie
    Fin Si
```

Ici ne sont traitées que les transitions dont l'état destination n'a jamais encore été atteint. On construit l'ensemble des méta-graphes devant être renvoyés par l'algorithme. Pour cela, l'état destination est ajouté dans le méta-état correspondant à l'état abstrait. La transition en cours de traitement est ajoutée dans l'ensemble des méta-états. Elle peut apparaitre soit entre deux états à l'interieur d'un métat-état pour les transitions portant une étiquette n'existant pas dans le graphe abstrait; soit entre deux états appartenant à des méta-états différents.

```
– Suite de l'algorithme IsGrapheRefinement –
    { Vérification que l'état destination de la transition n'a pas été traité }
    Si IndiceSGraphe [ Transition.Refinement ] = NonUtilise alors
      { On marque l'état comme utilisé en lui donnant un numéro de sommet dans
      l'ensemble des méta-états }
      IndiceSGraphe [ Transition.Refinement ] ← IndiceSGraphe [ Transition.Abstract
      ].GetNodeCount + 1
      { On ajoute la transition courante dans le chemin déjà parcouru }
      Chemin.Push (Transition.Refinement)
      { Ajout de l'état destination de la transition dans l'ensemble des méta-états }
      Valeur ← GRaffine.GetNodeValue ( Transition.Refinement )
      ListeG [ Transition.Abstract ].AddNode (Valeur)

      { Ajout de la transition en cours de traitement dans l'ensemble des méta-états }
      Si Transition.SameAbstract ∧ ¬ Premier alors
        { Ajout d'une transition à l'intérieur d'un méta-état }
        ListeG [ Transition.Abstract ].AddTrans (IndiceSGraphe [
                                                  Transition.LastRefinement ] - 1,
                                                  IndiceSGraphe [ Transition.Refinement ] -
                                                  1, Transition.Label)
      sinon
        { Indique que l'état destination est un état initial du méta-état dans lequel il se
        trouve }
        ListeG [ Transition.Abstract ].AddInitialNode (IndiceSGraphe [
                                                        Transition.Refinement ] - 1)
        Si ¬ Premier alors
          { Ajoute une transition entre deux méta-états }
          ListeG [ Transition.LastAbstract ].AddMetaTrans (IndiceSGraphe [
                                                            Transition.LastRefinement ] -
                                                            1, Transition.Abstract,
                                                            IndiceSGraphe [
                                                            Transition.Refinement ] - 1,
                                                            Transition.Label)
        Fin Si
      Fin Si
```

La boucle qui va être introduite permet de parcourir toutes les transitions partant de l'état final de celle en cours de traitement. Chacune d'entre elle sera ajoutée dans l'une des piles. Ici, nous voyons plus particulièrement le traitement des transitions existant déjà dans le graphe abstrait.

Elles sont ajoutées dans la pile des événements abstraits afin d'être traitées ultérieurement.

```
– Suite de l'algorithme IsGrapheRefinement –
        NbAbsEvt ← 0
        NveauEvt ← Faux
        Arret ← ¬ GRaffine.FindFirst ( Transition.Refinement, NouvRaf, NouvNom )

        Tant que ¬ Arret faire
          Si GAbstrait.IsTrans ( Transition.Abstract, NouvNom, NouvAbs ) alors
            { L'étiquette de la transition existe dans le graphe abstrait. Il y aura changement
            de méta-état }
            NbAbsEvt ← NbAbsEvt + 1
            NouvTransition.LastAbstract ← Transition.Abstract
            NouvTransition.LastRefinement ← Transition.Refinement
            NouvTransition.Abstract ← NouvAbs
            NouvTransition.Refinement ← NouvRaf
            NouvTransition.Label ← NouvNom
            NouvTransition.NoDeterminist ← Transition.IsNoDeterminist (
            Transition.Abstract, NouvNom )
            PileEvtAbstraits.Push (NouvTransition)
```

L'algorithme traite ici les deux autres cas concernant les nouvelles transitions. Le premier correspond à l'existance de l'événement dans l'état abstrait mais il n'est jamais déclenché à partir de l'état destination de la transition en cours de traitement. Le second autre cas est le fait que la transition trouvée dans le graphe raffiné n'existe pas dans le graphe abstrait. La nouvelle transition est alors ajoutée dans la pile PileNveauxEvt.

```
– Suite de l'algorithme IsGrapheRefinement –
          sinon Si GAbstrait.IsTransWithout ( Transition.Abstract, NouvNom ) alors
            { La transition trouvée existe dans le graphe abstrait mais ne part pas de l'état
            final de la transition en cours de traitement }
            IsGrapheRefinement ← Faux
            Sortie
          sinon
            { La transition trouvée est nouvelle. Elle n'existe pas dans le graphe abstrait }
            NveauEvt ← Vrai
            NouvTransition.LastAbstract ← Transition.Abstract
            NouvTransition.LastRefinement ← Transition.Refinement
            NouvTransition.Abstract ← Transition.Abstract
            NouvTransition.Refinement ← NouvRaf
            NouvTransition.Label ← NouvNom
            NouvTransition.NoDeterminist ← Faux
            PileNveauxEvt.Push (NouvTransition)
          Fin Si

          Arret ← ¬ GRaffine.FindNext ( NouvRaf, NouvNom )
        FTq
```

Le test suivant permet de vérifier que le nombre de transitions abstraites partant de l'état destination est cohérent avec celui de celles partant de l'état abstrait. On considère ici que si aucune nouvelle transition n'a été trouvée, toutes les transitions abstraites différentes doivent alors être accessible à partir de l'état concret. Les transitions abstraites sont considérées différentes lorsque leurs étiquettes ne possèdent pas la même valeur. Ainsi les transitions non-déterministes compteront comme une seule transition. Il reste à indiquer que lorsqu'une nouvelle transition a été trouvée, l'algorithme suppose que toutes les transitions abstraites seront toujours accessibles après avoir parcouru les nouvelles transitions.

```
– Suite de l'algorithme IsGrapheRefinement –
        { Si toutes les transitions trouvées existent dans le graphe abstrait, leur nombre doit
        être supérieur ou égal ou total des transitions abstraits différentes }
        Si ¬ NveauEvt ∧ NbAbsEvt < GAbstrait.GetDiffTransCount ( Transition.Abstract )
        alors
          IsGrapheRefinement ← Faux
          Sortie
        Fin Si
```

Ici, l'algorithme traite le cas où l'état destination de la transition a déjà été rencontré. Si il n'y a pas eu de changement de méta-état depuis la dernière rencontre, l'algorithme vérifi qu'il n'y ai pas de cycle à l'intérieur du méta-état. Si aucune boucle n'a été détectée, la transition est ajoutée dans le méta-état en cours de traitement.

```
– Suite de l'algorithme IsGrapheRefinement –
      { L'état destination de la transition a déjà été traité }
      sinon Si Transition.SameAbstract alors
        { L'état a déjà été rencontré dans le même méta-état. Il faut vérifier qu'il n'y a pas de
        cycle }
        Si Chemin.Exists ( Transition.Refinement ) alors
          IsGrapheRefinement ← Faux
          Sortie
        Fin Si

        { Il n'y a pas de cycle, on peut ajouter la transition l'intérieur du méta-état en cours
        de traitement }
        ListeG [ Transition.LastAbstract ].AddTrans (IndiceSGraphe [
                                                      Transition.LastRefinement ] - 1,
                                                      Transition.Abstract, IndiceSGraphe [
                                                      Transition.Refinement ] - 1,
                                                      Transition.Label)
```

Ici, l'algorithme traite le cas où l'état destination a déjà été rencontré mais avec un changement de méta-état depuis. La détection de cycle n'est donc pas nécessaire ici. L'état est ajouté dans la liste des états initaux du métat-état dans lequel il se trouve. La transition en cours de traitement est considérée comme une transition entre deux méta-états.

```
– Suite de l'algorithme IsGrapheRefinement –
    sinon
      { Il y a eu changement de méta-état. Pas de détection de cycle nécessaire. Ajout de la
      transition et de l'état dans le méta-graphe }
      Transition [ Transition.Abstract ].AddInitialNode (IndiceSGraphe [
                                                          Transition.Refinement ] - 1)
      ListeG [ Transition.LastAbstract ].AddMetaTrans (IndiceSGraphe [
                                                        Transition.LastRefinement ] - 1,
                                                        Transition.Abstract,
                                                        IndiceSGraphe [
                                                        Transition.Refinement ] - 1,
                                                        Transition.Label)
    Fin Si

    Premier ← Faux
  FTq

  { Aucune erreur n'a été détectée, il y a raffinement }
  IsGrapheRefinement ← Faux
  Sortie
Fin
```

6.6.2 Détection des cycles internes

La présence d'un cycle à l'intérieur des méta-états est un facteur de décision pour l'algorithme. Il y a raffinement lorsque l'on est sûr de sortir d'un méta-état. Hors les cycles internes à ces derniers ne le garantissent pas. L'algorithme considère qu'un graphe a est un raffinement d'un second graphe b s'il n'y a aucun doute quant au respect des contraintes de raffinement. Si un cycle est détecté, l'algorithme stoppe sont analyse est renvoie un résultat négatif.

L'algorithme parcours méta-état par méta-état le graphe qui devrait être le raffinement, c'est à dire qu'il traite toutes les transitions connues internes à un méta-état avant de traiter celles qui lui permettent de parcourir un autre état abstrait raffiné. D'autre part, ce parcours est réalisé en profondeur c'est à dire qu'un chemin interne au méta-état est entièrement construit avant de passer à un autre.

Ces deux principes de fonctionnement permettent de mettre en oeuvre une détection simple des cycles. L'algorithme utilise une pile dans laquelle il stockera le nom de tous les états qu'il a déjà parcouru. Si lors du traitement d'un nouvel état ce dernier existe dans la pile, l'algorithme considère qu'il y a un cycle et qu'il est nécessaire de stopper le parcours des deux graphes. Afin d'éviter tout problème de boucle externe aux méta-états, il est nécessaire de vider la pile lorsque l'algorithme change de méta-état.

6.6.3 Détection de la perte d'équivalence observationnelle

L'équivalence observationnelle impose que toutes les transitions partant d'un état α du graphe abstrait soient accessibles à partir de tous les états raffinant α. La vérification de l'existance de cette équivalence est réalisée sur tous les états. L'heuristique implémenté dégage deux cas de figure. Tout d'abord l'algorithme considère que les états ayant au moins une transition interne au méta-état partant d'eux ne sont pas incompatibles avec l'équivalence observationnelle. En effet on peut considérer que les transitions abstraites qui ne sont pas directement atteignables le seront à partir des états raffinés qui sont accessibles. Dans le second cas, les états n'ayant que des transitions abstraites et aucune transition interne font l'objet d'une vérification plus poussée. L'algorithme vérifie que toutes les transitions abstraites déterministes soient directement accessibles et que celles étant non-déterministes possèdent au moins une occurence accessible. Cette technique permet une

détection simple de la perte de l'équivalence observationnelle et ainsi celle d'un facteur interdisant le raffinement de graphe de transitions.

Chapitre 7

Exemple d'éxécution de l'algorithme

Dans ce chapitre, nous allons étudier l'exemple présenté par la figure 7.1. Il est visible qu'il n'y a pas raffinement dans le cas de ces deux graphes. En effet, il existe un cycle dans le graphe raffiné qui n'apparait pas dans l'abstraction. De plus il n'y a pas équivalence entre ces deux graphes. Nous verrons que l'algorithme détectera le cycle.

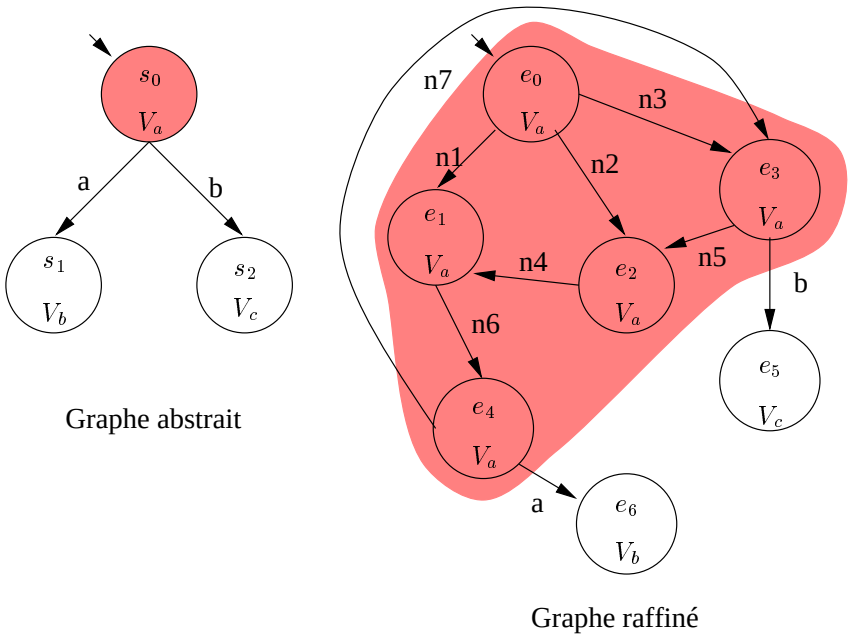
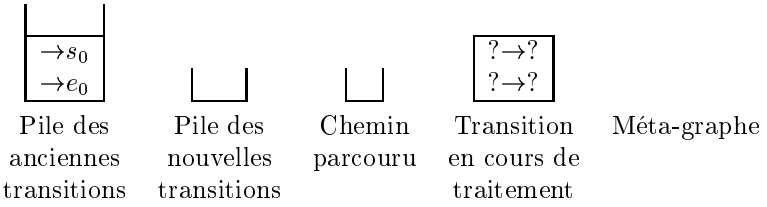


Figure 7.1: Algorithme - Exemple avec un cycle

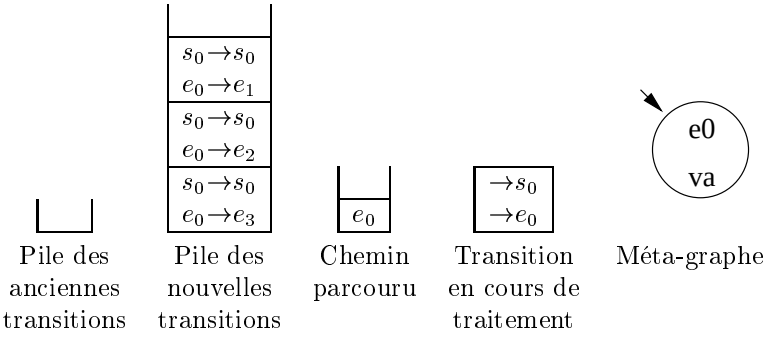
Sur la figure 7.1 e_i et s_i dénotent les états du système, V_α la valeur des états c'est à dire l'ensemble des valuations des variables pour chacun d'entre eux. Les transitions seront représentées par un quadruplet de la forme $\begin{matrix}\alpha \to \beta \\ \omega \to \delta\end{matrix}$ où α et β dénotent respectivement les états abstraits de départ et d'arrivée et où ω et δ dénotent respectivement les états concrets de départ et d'arrivée. On rappelle que l'algorithme parcourt les graphes raffinés et abstraits en parallèle.

Lors de l'initialisation de l'algorithme, la piles des événements abstraits contient la transition initiale. Celle des événements concrets est vide. Bien sûr, aucun état n'ayant été parcouru, le

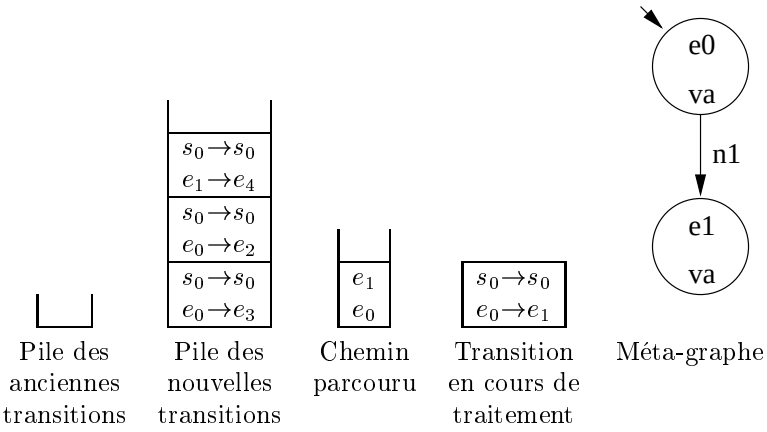
chemin est vide.



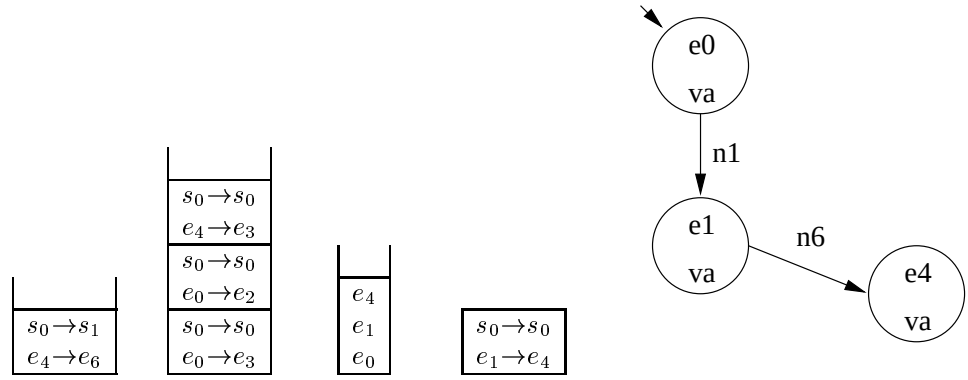
Au premier tour de la boucle principale, l'algorithme traite la transition qui se trouvait en sommet de la pile des événements abstraits car celle des transitions concrètes est vide. Comme l'état e_0 du graphe raffiné n'a jamais été traité, l'algorithme l'ajoute au chemin parcouru et empile les transitions partant de cet état. Avant de faire cela, l'algorithme a vérifié que l'état abstrait e_0 et l'état concret e_0 possède la même valeur à l'invariant de collage près. Afin de simplifier la lecture de l'exemple les valuations des états possèdent les même noms (ici V_a).



Tout comme précédemment, l'algorithme dépile la transition afin de la traitée puis l'ajoute au chemin et empile les événements qui parte du sommet destination.

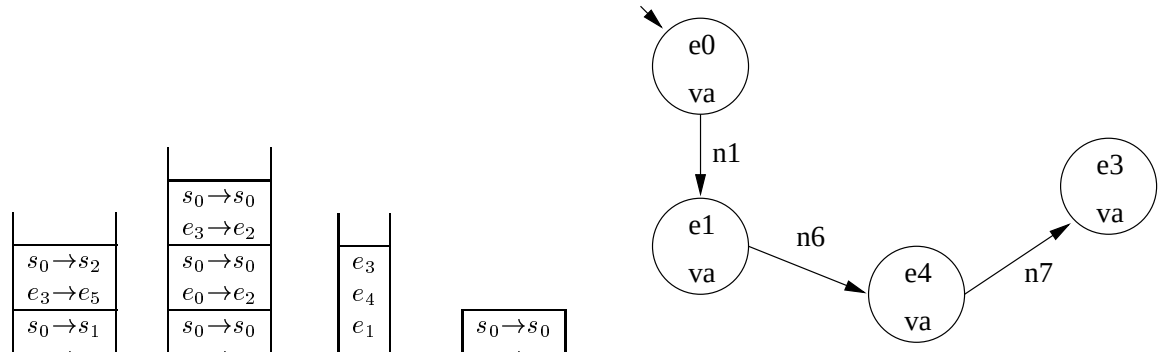


Là aussi, les mêmes actions sont réalisées à l'exception de l'empilement de la transition allant de l'état concret e_4 à e_6. En effet, cette dernière existe dans le graphe abstrait. L'algorithme place cet événement sur la pile des transitions abstraites afin de la traiter après celles contenues dans la piles des nouveaux événements.

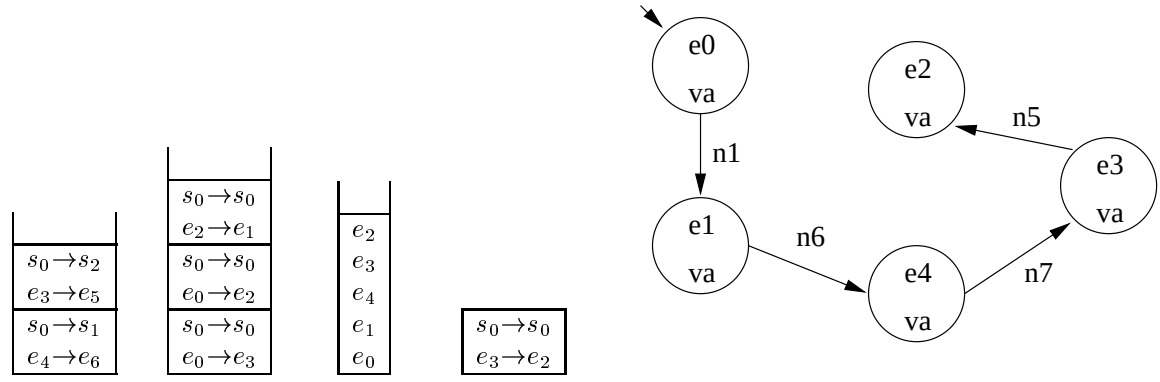


Pile des anciennes transitions | Pile des nouvelles transitions | Chemin parcouru | Transition en cours de traitement | Méta-graphe

Tout comme les itérations précédentes, les deux suivantes réalisent les mêmes actions.

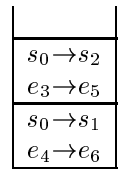


Pile des anciennes transitions | Pile des nouvelles transitions | Chemin parcouru | Transition en cours de traitement | Méta-graphe

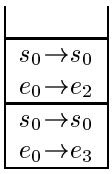


Pile des anciennes transitions | Pile des nouvelles transitions | Chemin parcouru | Transition en cours de traitement | Méta-graphe

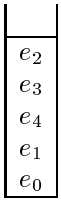
Ici la transition à traiter mène de l'état concret e_2 à e_1. Ce dernier ayant déjà été traité, l'algorithme vérifi qu'il n'existe pas dans le chemin déjà parcouru. Ce dernier test génère un cas d'erreur car un cycle partant de l'état e_1 ramène à ce dernier.



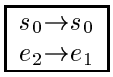
Pile des anciennes transitions



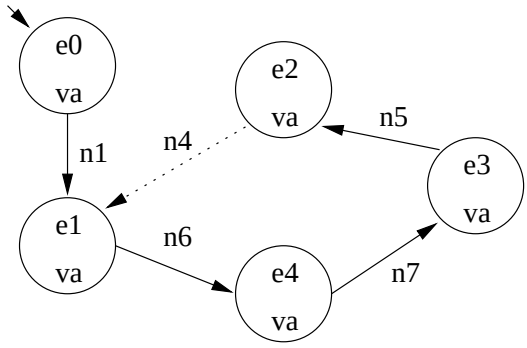
Pile des nouvelles transitions



Chemin parcouru



Transition en cours de traitement



Méta-graphe

Chapitre 8

Bilan et perspectives

Concernant le premier objectif de ce rapport qui est de définir une extension de la méthode B, le développement complet de l'exemple du protocole BRP nous a permis de mettre en oeuvre l'extension en Logique Temporelle Linéaire sur un exemple réaliste. Cette étude nous a permis d'améliorer notre perception du travail de l'analyste et ainsi de savoir si les propriétés dynamiques devait être reformulées afin de limiter encore le nombre des formes des propriétés de vivacité. Nous avons découvert que les reformulations aboutissent à des propriétés mathématiquement équivalentes mais qui ne correspondent pas toujours à une vision simple des propriétés informelles. Ces dernières peuvent être exprimées de manières différentes selon la formulation informelle utilisée mais aussi selon la vision de l'analyste vis à vis du système modélisé. Ainsi, même si les différentes reformulations des propriétés dynamiques sont équivalentes, certaines parraîssent être plus naturelles voir plus simple à comprendre. La question qui était se savoir s'il faut obliger l'analyste à reformuler ses propriétés amène une réponse négative car nous avons considéré que le pouvoir d'expression de l'analyse n'en devenait que trop limité. Toutefois il ne faut pas abandonner cette idée. En effet, elle s'avererait très utile dans le cadre d'une vérification automatique en limitant le nombre de formes utilisées par le prouveur durant son travail. Ainsi le travail de reformulation n'incomberait plus à l'analyste mais au logiciel devant prouver la validité de l'analyse. Cette vision du processus de formalisation et de preuve semble être un bon compromis entre le pouvoir d'expression donné à l'être humain et les problèmes de complexités rencontrés par le prouveur. De plus elle n'interdit pas une éventuelle reformulation des propriétés dynamiques de la part de l'analyste.

Pour le second objectif qui est de mesurer l'impact du raffinement sur l'extension LTL de B, l'analyse des diverses études de cas a permis de dégager une liste des formes de propriétés dynamiques les plus couramment utilisées. Parmis celles-ci on retrouve des clauses simples et très utilisées comme $\Box\ (\ a \Rightarrow \bigcirc b\ )$ et $\Box\ (\ a \wedge \bigcirc b \Rightarrow c\ )$ portant uniquement sur deux états. Il apparaît aussi des formes un peu plus complexes qui concernent le comportement du système sur quelques états : $\Box\ (\ a \wedge \bigcirc b \Rightarrow \bigcirc\ \bigcirc c\ )$ par exemple. Différents raffinements de ces formes ont été trouvés. Ils permettent d'avoir un idée de l'évolution des propriétés de sureté au cours des raffinements d'une spécification. Ces derniers apparaîssent sous deux formes. La première, nommée raffinement d'état, correspond à l'existance d'une équivalence observationnelle entre les états abstraits et les états raffinés. La seconde forme de raffinement est celle de chemin. Contrairement au premier type de raffinement, l'équivalence observationnelle n'existe pas toujours. Ce phénomène est dû au raffinement d'événements non-déterministes. Ainsi tous les événements existant au niveau abstrait ne sont pas forcément accessibles dans tous les états concrets correspondant. Un autre aspect de cette influence est le raffinement de propriétés invariantes en propriétés dynamiques. Ceci est très intéressant par le fait que des propriétés qui devaient être vérifiées sur tous les états du système ne doivent plus l'être que sur un nombre limité. Cette transformation de la portée des propriétés est dûe au fait que ces dernières sont liées au niveau de raffinement. Outre les différents raffinements dégagés et la liste des formes de propriétés dynamiques, ce document

nous permet de réaliser une classification de ces dernières en deux classes. La première classe correspond aux propriété de sûreté. Elles représentent les propriétés indiquant que *rien de mauvais ne doit arriver*. On retrouve dans ce groupe les formes $\Box a$, $\Box\,(\ a \Rightarrow \bigcirc b\ )$, $\Box\,(\ a \wedge \bigcirc b \Rightarrow c\ )$ et $\Box\,(\ a \wedge \bigcirc b \Rightarrow \bigcirc c\ )$ et plus rarement les propriétés de la forme $\Box\,(\ a \wedge \bigcirc b \Rightarrow \bigcirc\ \bigcirc c\ )$ et $\Box\,(\ \bigcirc a \Rightarrow b\ )$. La seconde classe est celle des propriétés de vivacité. Elles représentent le fait que *quelque chose de bon doit se produire*. Tout comme pour les propriétés de sûreté, on y retrouve des schéma s courants et des plus rares. Les formes $\Box\,(\ a \Rightarrow \Diamond b\ )$ et $\Box\,(\ a \Rightarrow b\ \mathcal{U}\ c\ )$ font partie de la première catégorie alors que $\Box\,(\ a \wedge \bigcirc b \Rightarrow c\ \mathcal{U}\ d\ )$ ou encore $\Box\,(\ a \wedge \bigcirc b \Rightarrow \Diamond c\ )$ sont des propriétés utilisées moins couramment. Il nous reste a dégager le rapport entre les deux classes et les raffinements. Ainsi les propriétés de sûreté peuvent se raffiner soit en propriétés de sûreté dans le cas des formes $\Box\,(\ a \wedge \bigcirc b \Rightarrow c\ )$ et $\Box\,(\ a \wedge \bigcirc b \Rightarrow \bigcirc c\ )$, soit en propriétés de vivacité. Ce dernier raffinement est illustré par exemple par $\Box\,(\ a \Rightarrow \bigcirc b\ )$ qui se raffine en $\Box\,(\ a \Rightarrow \Diamond b\ )$ ou $\Box\,(\ a \Rightarrow b\ \mathcal{U}\ c\ )$. Durant les raffinements, les propriétés de vivacité restent des propriétés de vivacité.

L'algorithme développé dans ce mémoire permet de vérifier qu'un graphe est un raffinement possible d'un second. Il prend en compte des valeurs des états abstraits et concrets ($e_2 \wedge I_2 \Rightarrow e_1$), du raffinement des transitions non déterministes ainsi que des notions de raffinement d'état et de chemin. Le principal défaut de l'algorithme est le fait que les graphes raffinés et abstraits doivent lui être donnés. L'algorithme se contente de vérifier le raffinement et de construire un méta-graphe correspond au graphe raffiné. Le méta-graphe généré par l'algorithme paraît utile pour une vérification modulaire par "model checking". Chaque méta-état composant le graphe généré peut faire l'objet d'une vérification autonome, ce qui permet de diminuer la complexité de la vérification. Une amélioration possible est le développement d'un nouvel algorithme ne demandant qu'un seul graphe. Il construirait alors le graphe raffiné en fonctions de critères donnés par l'analyste. Il reste toutefois a définir si ce type d'approche reste possible et dans quelle mesure l'algorithme pourrait créer seul le graphe raffiné.

Liste des figures

Liste des tableaux

Bibliographie

[195] Technical Commitee ISO/IEC JTC 1. Information technology - identification cards - integrated circuit(s) cards with contacts. electronic signals and transmission protocols. Technical report, the International Organization dor Standardization and the International Electrotechnical Comission, September 1995.

[Abr96a] J.R. Abrial. Extending B without changing it (for developping distributed systems). In 1^{st} *Conference on the* B *Method*, pages 169–190, Nantes, November 1996.

[Abr96b] J.R. Abrial. *The* B *Book*. Cambridge University Press, 1996.

[Abr97] J.R. Abrial. Constructions d'Automatismes industriels avec B. Congrès AFADL - ONERA-CERT, May 1997.

[AM97] J.R. Abrial and L. Mussat. Specification and design of a transmission protocol by successive refinements using B. *LNCS*, 1997.

[AM98] J.R. Abrial and L. Mussat. Introducing dynamic constraints in B. In 2^{nd} *Conference on the* B *Method*, Montpellier, April 1998.

[Arn92] A. Arnold. *Systèmes de Transitions Finis et Sémantique Des Processus Communicants*. 1992.

[Arn95] A. Arnold. Automatic verification of properties in transition systems. In *Software - Practice and Experience*, volume vol n ° 25, n ° 6, pages 579–596. 1995.

[Bay95] Benjamin Bayard. *Joli manuel LaTeX 2ε*. Ecole Supérieure d'Ingémieurs en Electronique et Electrotechnique, 1995.

[Bos95] J.C. Bossy. *Le grafcet. Sa pratique et ses applications*. 1995.

[CWB94] J. Cuéllar, I. Wildgruber, and D. Barnard. Combining the design of industrial systems with effective verification techniques. In *FME'94, LNCS n ° 873*, pages 639–658, 1994.

[DAC98] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. 1998.

[EC82] E.A. Emerson and E.M. Clarke. Using branching time temporal logic to synthesize synchronisation skeletons. *Sci. of Computer Progr.*, (2): 241–266, 1982.

[GH93] P. Godefroid and G.J. Holzmann. On the verification of temporal properties. In *Proc. IFIP/WG6.1 Symp. On Protocols Specification, Testing and Verification*, Liege, Belgium, June 1993. PSTV93.

[GMS97] Michel Goosens, Frank Mittelbach, and Alexander Samarin. *The LaTeXCompanion*. Addison-Wesley Publishing Company, 1997.

[GPVW95] R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. PSTV95 Conference*, Warsaw, Poland, June 1995.

[Hola] G.J. Holzmann. *Basic Spin manual.* Number NJ 07974. Murray hill.

[Holb] G.J. Holzmann. Overview of the spin model checker. Technical report, Computing Principles Research Department, Bell laboratories.

[JLM+97] J. Julliand, B. Legeard, T. Machicoane, B. Parreaux, and B. Tatibouët. Specification of an integrated circuit card protocol application using B method and linear temporal logic. In *2nd Conference on* B *Method*, 1997.

[Jul97] J. Julliand. Spécification formelle et vérification de logiciels - cours de DEA, 1997.

[Lam91] L. Lamport. The temporal logic of actions. Technical report, Digital Systems Research center, Palo Alto, California, December 1991.

[Man92] Z. Manna. *The temporal Logic of Reactive and Concurrent Systems : specification.* 1992.

[Mer96] D. Mery. Machines abstraites temporelles. analyse comparative de B et de TLA+. In 1st *Conference on the B method*, November 1996.

[MP96] Z. Manna and A. Pnueli. STeP : Stanford temporal prover. Technical report, University of Stanford, 1996.

[oE98] University of Essen. Linear time temporal logic (LTL), http: //www.cs.uni-essen.de/Fachgebiete/SysMod/Forschung/QUAFOS/TechRep/Q1/, March 1998.

[WG96] C. Willems and F. Geraerds. *Aide mémoire pour LaTeX.* Université de Liège, Belgique, 1996.

Numéro d'ordre 2083
Année 1998

Mémoire de D.E.A.
d'Informatique, Automatique et Productique

Classification de propriétés dynamiques et de leurs raffinements à partir de quelques études de cas

Partie Annexes

Par Stéphane Galland

Stage encadré par le Professeur Jacques Julliand

Présenté le	17/07/98
Jury :	
Prof. J. Julliand	Laboratoire d'Informatique de Besançon
Prof. F. Bellegarde	Laboratoire d'Informatique de Besançon
Prof. B. Legeard	Laboratoire d'Informatique de Besançon
Mdc H. Mountassir	Laboratoire d'Informatique de Besançon
Mdc C. Varnier	Laboratoire d'Automatique de Besançon

Partie III

Annexes

Annexe A

L'exemple de l'Automatisme Industriel : Spécification par raffinement

A.1 Cahier des charges du robot

Cet exemple, emprunté à Abrial, est présenté dans [Abr97] et [Bos95]. Le système physique à piloter est schématisé figure A.1. Il est composé de trois dispositifs :

- un dispositif d'arrivée des pièces appelé Da,
- un dispostif d'évacuation des picèces appelé De sur lequel le dispositif de transport dépose les pièces prises sur le dispositif d'arrivée,
- et un dispositif de transport des pièces, appelé Dt, qui est un bras muni d'une pince; le bras dispose de deux mouvements, vertical et horizontal; la pince a un mouvement d'ouverture-fermeture.

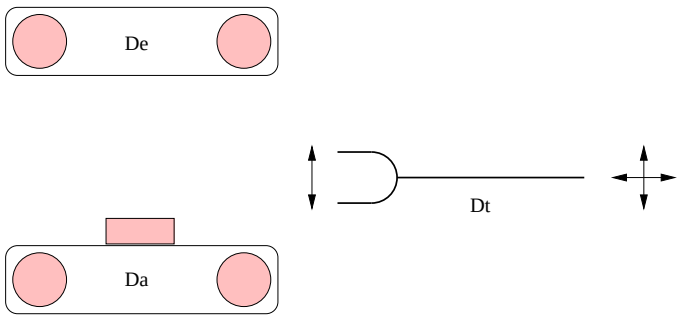


Figure A.1: Robot de transport de pièces

L'analyse des besoins pour spécifier ce dispositif est abordée en cinq niveaux successifs de complexité :

- niveau 0: Nous observons uniquement de dispositif d'évacuation et le dispositif de transport sans mouvement autre que le départ des pièces sur le dispositif d'évacuation,
- niveau 1: Nous observons maintenant le dispositif d'arrivée des pièces et les positions verticales du dispositif de transport,

- niveau 2: Nous observons maintenant les mouvements verticaux du dispositif de transport,
- niveau 3: nous observons à ce niveau les mouvements horizontaux du dispositif de transport ainsi que la fermeture et l'ouverture de la pince,
- niveau 4: Nous observons enfin le dialogue avec les dispositifs physiques du système (moteurs et capteurs).

Le problème du robot est soumis aux principes de fonctionnement suivants illustrés par la figure A.2 et classés par niveau de raffinement:

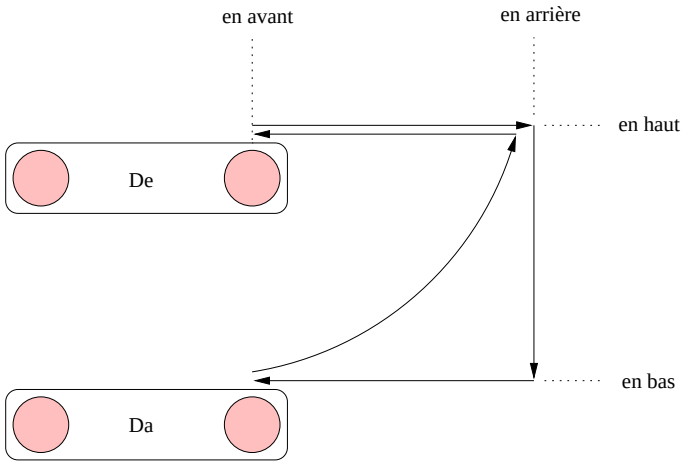


Figure A.2: Mouvements tolérés de l'automatisme

- Niveau 0: propriété du déchargement
  - Dt ne peut décharger une pièce sur De que si De est libre $[C_1]$[1],
- Niveau 1: propriétés liées à la position verticale
  - Dt ne peut charger une pièce depuis Da que si Dt est libre $[C_2]$,
  - Dt décharge en position haute $[C_3]$,
  - Dt charge en position basse $[C_4]$,
- Niveau 1 et 2: propriétés du mouvement vertical
  - Dt doit tenir une pièce pour monter $[C_5]$,
  - Dt doit être vide pour descendre $[C_6]$,
- Niveau 3: propriétés des mouvements horizontaux et de la pince
  - Dt descend en position arrière $[C_7]$,
  - le bras avance en haut horizontalement $[C_8]$,
  - le bras avance en bas horizontalement $[C_9]$,
  - le bras recule en haut horizontalement $[C_{10}]$,
  - le bras peut combiner un mouvement de montée avec un mouvement de recul mais il doit arriver en arrière avant d'être en haut $[C_{11}]$,
  - la pince se ferme en bas en avant pour charger une pièce $[C_{12}]$,

[1] $[C_n]$ est une référence à une expression informelle de propriété.

- la pince s'ouvre en haut en avant pour décharger une pièce [C_{13}],
- la pince recule en bas fermée [C_{14}],
- la pince monte fermée [C_{15}],
- la pince avance en haut fermée [C_{16}],
- la pince recule en haut ouverte [C_{17}],
- la pince descend ouverte [C_{18}],
- la pince descend en arrière [C_{19}],
- la pince avance en bas ouverte [C_{20}].

• Niveau 4: propriétés des dispositifs physiques. Nous présentons celles-ci section A.6

Les propriétés [C_8], [C_9], [C_{10}] et [C_{11}] permettent de garantir que le bras ne heurte pas De en montant ou en descendant. Nous décrivons cinq spécifications de plus en plus concrètes de cet automatisme. La première spécification décrite section A.2 consiste à modéliser le transport des pièces du dispositif de transport sur le dispositif d'évacuation, sans préciser ni comment les pièces arrivent dans le dispositif de transport ni comment celui-ci se déplace. La seconde spécification décrite section A.3 consiste à introduire l'observation de l'arrivée des pièces sur le dispositif d'arrivée et à observer les positions haute et basse du dispositif de transport. La troisième spécification décrite section A.4 consiste à introduire le mouvement vertical du bras. La quatrième décrite section A.5 introduit le mouvement horizontal du bras et le mouvement de la pince. La cinquième spécification consiste à introduire les dispositifs physiques tels que les moteurs. Elle permet de séparer matériel et logiciel de pilotage.

A.2 Première spécification du Robot

La première spécification consiste à observer le système en ignorant le dispositif d'arrivée des pièces Da_0 et en simplifiant le dispositif de transport, appelé Dt_0, et le dispositif d'évacuation, nommé De_0, pour lesquels on observe que leurs états *occupé* et *libre*, ce qui est schématisé dans la figure A.3.

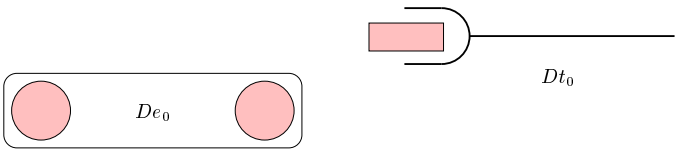


Figure A.3: Vue simplifiée du robot

La figure A.4 décrit la spécification formelle "à la B" complétée par le paragraphe intitulé DYNAMIC PROPERTY. Nous commentons ce qui concerne la partie descriptive (variables, invariant et propriétés dynamiques) section A.2.1 et la partie opérationnelle (initialisation et événements) section A.2.2. Enfin définissons sa sémantique sous forme d'un système de transitions section A.2.3.

A.2.1 Spécification descriptive

Le système est modélisé par deux variables d'état Dt_0 et De_0 qui représentent respectivement l'état des dispositifs de transport et d'évacuation par deux valeurs: *libre* si les dispositifs ne contiennent pas de pièce, *occupé* si ils en contiennent.

L'invariant (voir paragraphe INVARIANT figure A.4) indique que les quatres états montrés par la figure A.5 sont admis.

A ce niveau nous décrivons uniquement la propriété dynamique $[C_1]$ appelée C_0^1 au niveau 0 (voir section A.1 et paragraphe DYNAMIC PROPERTY figure A.4) qui s'exprime intuitivement ainsi : "si le dispositif De_0 et le dispositif de transport Dt_0 sont occupés alors le dispositif d'évacuation De_0 doit se libérer à l'état suivant". Cette propriété aurait pu s'exprimer également ainsi : $\Box\,(\,(\,Dt_0 = occupé \wedge \bigcirc\,(\,Dt_0 = libre\,)\,) \Rightarrow De_0 = libre\,)$ qui s'interprète intuitivement : "si Dt_0 décharge alors De_0 doit être libre". Notons que dans le premier cas on se place dans l'état critique où Dt_0 risque de décharger sur De_0 occupé et on indique ce qui doit se passer de bon dans le futur. Ici étant donné le niveau d'abstraction, De_0 doit se libérer à l'état suivant. Dans le second cas on se place après l'événement critique de déchargement et on regarde quel devrait être le passé pour que rien de mauvais ne se soit produit. La première approche est une approche vivacité alors que la seconde est une approche de sécurité. On notera que l'approche de vivacité tient compte du niveau d'abstraction et qu'on aurait pu s'en affranchir en décrivant la propriété ainsi $\Box\,(\,Dt_0 = occupé \wedge De_0 = occupé \Rightarrow\,(\,Dt_0 = occupé\,)\,\mathcal{U}\,(\,De_0 = libre \wedge Dt_0 = occupé\,)\,)$.

ABSTRACT SYSTEM Robot
SETS
ETAT_DISPOSITIF $= \{\, libre, occupé \,\}$
VARIABLES
De_0, Dt_0
INVARIANT
$De_0 \in$ ETAT_DISPOSITIF $\wedge\ Dt_0 \in$ ETAT_DISPOSITIF
DYNAMIC PROPERTY
$C_0^1 \triangleq \Box\,(\,(\,De_0 = occupé \wedge Dt_0 = occupé\,) \Rightarrow \bigcirc\,(\,De_0 = libre\,)\,)$
INITIALISATION
$Initialisation \triangleq De_0, Dt_0 := libre, libre$
EVENTS
$Chargt_0 \triangleq SELECT\ Dt_0 = libre\ THEN\ Dt_0 := occupé\ END$
$DéChargt_0 \triangleq SELECT\ De_0 = libre \wedge Dt_0 = occupé\ THEN\ De_0, Dt_0 := occupé, libre\ END$
$Evac_0 \triangleq SELECT\ De_0 = occupé\ THEN\ De_0 := libre\ END$
END Robot

Figure A.4: Spécification abstraite du Robot

A.2.2 Spécification opérationnelle

La spécification opérationelle consiste à décrire l'état inital (voir paragraphe INITIALISATION figure A.4) du système qui doit être unique et les événements qui font évoluer le système. Sur l'exemple nous décrivons trois événements (voir paragraphe EVENTS figure A.4) :

- le chargement d'une pièce dans le dispositif de transport Dt_0 appelé $Chargt_0$,
- son déchargement sur le dispositif d'évacuation De_0 appelé $DéChargt_0$,
- l'évacuation d'une pièce qui est sur le dispositif d'évacuation De_0 appelé $Evac_0$.

Chaque événement est défini par sa condition de déclenchement décrite dans la partie SELECT et par l'action effectuée décrite dans la partie THEN. L'action modifie les variables d'états définies par une substitution généralisée du langage B [Abr96]. Si la condition de déclenchement est vraie, on dit que l'événement est activable. Le système est non déterministe car étant donné un état tous les événements activables sont activés. C'est la sémantique que nous donnons en associant un système de transition à cette spécification.

A.2.3 Sémantique de la spécification

La sémantique d'une telle spécification est l'ensemble des chemins d'exécution finis et infinis d'un système de transition [Arn92] qui lui est associé. Par exemple la sémantique de la spécification figure A.4 est l'ensemble des chemins du système de transition représenté par le graphe figure A.5. Ce graphe a pour racine l'état initial résultat de l'action *Initialisation* décrite figure A.4. De chaque état partent des transitions pour chaque événement activable sur cet état. L'état cible des transitions est obtenu en appliquant la substitution B (partie action) de l'événement activé sur l'état source. Le nom des événements activés figurent sous forme d'étiquettes sur les flèches reliant un état source à un état cible.

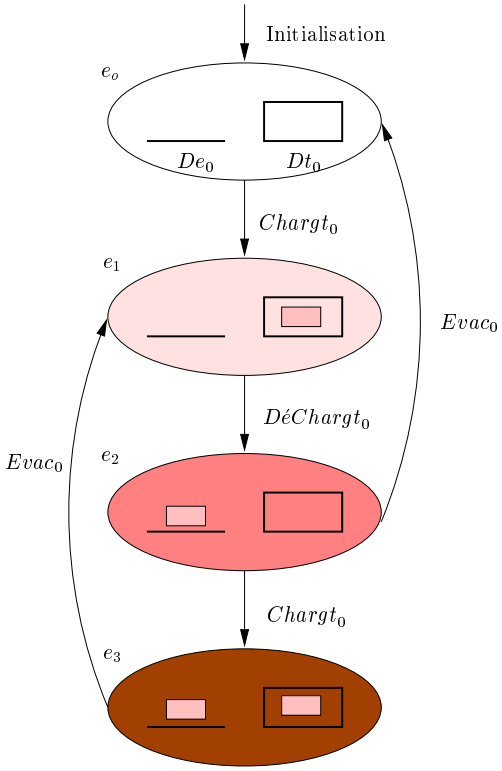


Figure A.5: Système de transition du robot de transport abstrait

A.3 Premier raffinement

Le raffinement d'une spécification est un énoncé de même forme que la spécification abstraite. Pour la distinguer nous l'appelons spécification concrète. Sa forme doit respecter les contraintres suivantes par rapport à la spécification abstraite :

- les deux ensembles de variables sont disjoints,
- l'invariant concret, appelé invariant de collage, exprime les relations entre les variables concrètes et les variables abstraites,
- les propriétés dynamiques sont réexprimées sur les nouvelles variables et de nouvelles propriétés sont ajoutées,
- les anciens événements sont raffinés et de nouveaux sont ajoutés.

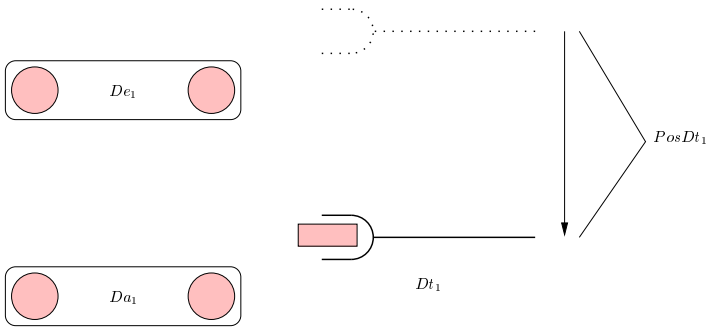


Figure A.6: Vue des positions verticales du robot

La figure A.6 schématise la vue détaillée du robot de transport. Cette fois, nous observons l'arrivée des pièces sur Da_1 et les positions verticales du dispositif de transport Dt_1.

SYSTEM $Robot_1$

VARIABLES

$De_1, Dt_1, Da_1, PosDt_1$

INVARIANT

$De_1 = De_0 \wedge Dt_1 = Dt_0$

$Da_1 \in$ ETAT_DISPOSITIF $\wedge\ PosDt_1 \in \{\ enhaut, enbas\ \}$

DYNAMIC PROPERTIES

$$C_1^1 \triangleq \Box\left((De_1 = occupé \wedge Dt_1 = occupé) \Rightarrow \left(Dt_1 = occupé \right) \mathcal{U} \left(\begin{array}{l} De_1 = libre \wedge \\ Dt_1 = occupé \end{array} \right) \right)$$

$$C_1^2 \triangleq \Box\left((Da_1 = occupé \wedge \bigcirc Da_1 = libre) \Rightarrow Dt_1 = libre \right)$$

$$C_1^3 \triangleq \Box\left((Dt_1 = occupé \wedge \bigcirc Dt_1 = libre) \Rightarrow PosDt_1 = enhaut \right)$$

$$C_1^4 \triangleq \Box\left((Dt_1 = libre \wedge \bigcirc Dt_1 = occupé) \Rightarrow PosDt_1 = enbas \right)$$

$$C_1^5 \triangleq \Box\left((PosDt_1 = enbas \wedge \bigcirc PosDt_1 = enhaut) \Rightarrow Dt_1 = occupé \right)$$

$$C_1^6 \triangleq \Box\left((PosDt_1 = enhaut \wedge \bigcirc PosDt_1 = enbas) \Rightarrow Dt_1 = libre \right)$$

INITIALISATION

$Initialisation \triangleq De_1, Dt_1, Da_1, PosDt_1 := libre, libre, libre, enbas$

EVENTS

$Chargt_1 \triangleq$ $SELECT\ Dt_1 = libre \wedge \mathbf{Da_1 = occupé} \wedge \mathbf{PosDt_1 = enbas}$
$THEN\ Dt_1, \mathbf{Da_1} := occupé, \mathbf{libre}\ END$

$DéChargt_1 \triangleq$ $SELECT\ De_1 = libre \wedge Dt_1 = occupé \wedge \mathbf{PosDt_1 = enhaut}$
$THEN\ De_1, Dt_1 := occupé, libre\ END$

$Evac_1 \triangleq SELECT\ De_1 = occupé\ THEN\ De_1 := libre\ END$

$ArMontée_1 \triangleq$ $SELECT\ PosDt_1 = enbas \wedge Dt_1 = occupé$
$THEN\ PosDt_1 = enhaut\ END$

$ArDescente_1 \triangleq$ $SELECT\ PosDt_1 = enhaut \wedge Dt_1 = libre$
$THEN\ PosDt_1 = enbas\ END$

$ArPièce_1 \triangleq SELECT\ Da_1 = libre\ THEN\ Da_1 := occupé\ END$

END $Robot_1$

Figure A.7: Spécification raffinée (concrète) du robot

A.3.1 Spécification descriptive

Comme le montre la figure A.7, le système est modélisé par quatre variables d'états :

- Dt_1 et De_1 qui représentent respectivement les dispositifs abstraits Dt_0 et De_0,
- Da_1 qui représente l'état du dispositif d'arrivée des pièces,
- $PosDt_1$ qui représente la position de Dt_1 qui est *enhaut* ou *enbas*.

L'invariant de collage figure A.7 indique que le dispositif d'évacuation concret De_1 est est identique au dispositif abstrait De_0 et le dispositif de transport concret Dt_1 est identique au dispositif abstrait Dt_0. La propriété dynamique abstraite C_0^1 de la forme $\Box\ (\ p \wedge p' \Rightarrow \bigcirc \neg p\ )$ est raffinée sous la forme $\Box\ (\ p \wedge p' \Rightarrow p'\ \mathcal{U}\ \neg p\ )$. Elle signifie intuitivement que le passage d'un état où $p \wedge p'$ est vrai à un état où $\neg p$ est vrai qui s'effectuait en une transition s'effectue maintenant par un chemin sur lequel p' doit rester invariant. Sur l'exemple, lorsque les dispositifs abstraits d'évacuation et de transport étaient occupés, l'évacuation de la pièce devait être la seule action possible. Au niveau concret, les actions d'arrivée de pièce et de montée du dispositif de transport sont possibles, mais le dispositif de transport doit rester occupé tant que la pièce n'est pas évacuée du dispositif d'évacuation. C'est ce qu'indique la propriété C_1^1 figure A.7. Notons que la propriété C_0^1 aurait déjà pu être exprimée sous la forme $\Box\ (\ p \wedge p' \Rightarrow p'\ \mathcal{U}\ \neg p\ )$ au niveau 0. Elle serait alors raffinée par le même schéma au renommage de variables près. Notons également que si la propriété C_0^1 avait été exprimée par $\Box\ (\ Dt_0 = \textit{occupé} \wedge \bigcirc\ (\ Dt_0 = \textit{libre}\ ) \Rightarrow De_0 = \textit{libre}\ )$, elle serait raffinée par le même schéma $\Box\ (\ (\ Dt_1 = \textit{occupé} \wedge \bigcirc\ (\ Dt_1 = \textit{libre}\ )\ ) \Rightarrow De_1 = \textit{libre}\ )$.

La propriété C_0^2 de la forme $\Box\ (\ p \Rightarrow \bigcirc p'\ )$ est raffinée simplement en transformant l'opérateur *Suivant* en un opérateur *Until*. Ceci signifie intuitivement que le passage d'un état satisfaisant p à un état satisfaisant p' qui se faisait en une transition, s'effectue maintenant par un ensemble de chemins formés d'une succession de nouveaux événements terminée par une transition appliquant le même événement qu'au niveau abstrait. D'autre part une condition p'' est vraie tout au long du chemin. Sur l'exemple ce raffinement permet aux nouveaux événements $ArPièce_1$ et $Descente_1$ d'avoir lieu entre un état où De et Dt sont libres et un chargement.

On a ajouté les propriétés C_1^2, C_1^3, C_1^4, C_1^5 et C_1^6 sous la forme $\Box\ (\ p \wedge \bigcirc p' \Rightarrow p''\ )$. Cette forme est utilisée pour exprimer que si un événement t s'applique (spécifiée par $p \wedge \bigcirc p'$ qui est une transition d'état) alors p'' doit être vraie.

A.3.2 Spécification opérationelle

Celle-ci raffine les anciens événements et y ajoute la description de trois nouveaux événements :

- l'arrêt de la montée du dispostif de transport Dt appelée $ArMontée_1$,
- l'arrêt de la descente appelée $ArDescente_1$,
- l'arrivée d'une pièce appelée $ArPièce_1$.

On note que pour réaliser les propriétés $[C_6]$ et $[C_7]$ les événements $ArDescente_1$ (resp. $ArMontée_1$) possèdent comme condition d'activation $Dt_1 = \textit{libre}$ (resp. $Dt_1 = \textit{occupé}$).

On note également que les gardes des anciens événements sont renforcées. Dans l'exemple les gardes des événements sont renforcées par les conditions en **gras** (ex: $Chargt_1$ et $DéChargt_1$ figure A.7) portant sur les nouvelles variables. Les conditions $PosDt_1 = \textit{enhaut}$ (resp. $PosDt_1 = \textit{enbas}$) permettent de réaliser les propriétés $[C_2]$ (resp. $[C_3]$) des événements $Chargt_1$ (resp. $DéChargt_1$). Les actions peuvent également être détaillées pour agir sur les nouvelles variables (ex: $Chargt_1$ qui agit sur Da_1).

A.3.3 Sémantique du robot raffiné

La sémantique de cette spécification est l'ensemble des chemins infinis représenté par le graphe du système de transition figure A.8. On note que les états du graphe raffiné de la figure A.8 se regroupent pour former la structure du graphe abstrait de la figure A.5. Une transition abstraite est raffinée par un ensemble de chemins concrets qui se terminent tous par le déclenchement du même événement que celui qui étiquette la transition abstraite. Par exemple la transition $e_0 \xrightarrow{Chargt_0} e_1$ figure A.5 est raffinée par les cinq chemins suivants figure A.8: $e_1 \xrightarrow{ArDescente_1} e_0 \xrightarrow{ArPièce_1} e_2 \xrightarrow{Chargt_1} e_4$, $e_1 \xrightarrow{ArPièce_1} e_3 \xrightarrow{ArDescente_1} e_2 \xrightarrow{Chargt_1} e_4$, $e_0 \xrightarrow{ArPièce_1} e_2 \xrightarrow{Chargt_1} e_4$, $e_2 \xrightarrow{Chargt_1} e_4$ et $e_3 \xrightarrow{ArDescente_1} e_2 \xrightarrow{Chargt_1} e_4$.

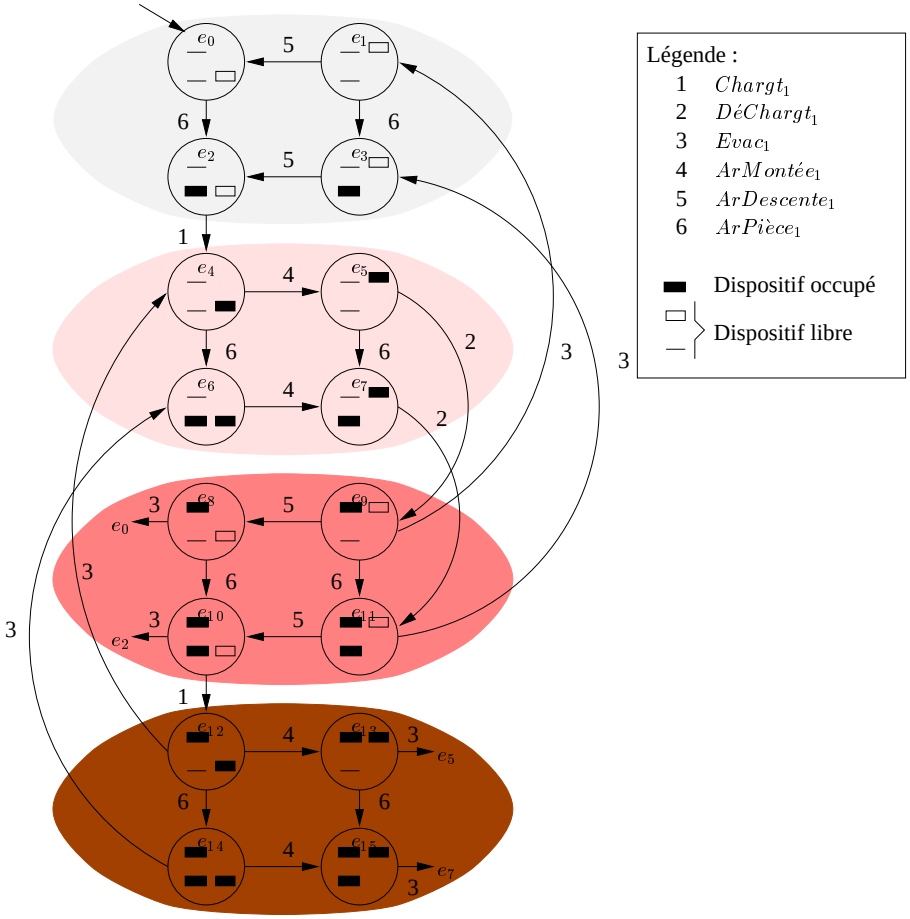


Figure A.8: Système de transition du robot de transport lors de la première spécification

A.4 Deuxième raffinement du robot

Nous introduisons le mouvement effectif (montant et descendant) du dispositif de transport *Dt.* nous décomposons les événements $ArMontée_2$ et $ArDescente_2$ en deux étapes : $Montée_2$, $ArMontée_2$ et $Descente_2$, $ArDescente_2$. La figure A.9 schématise cette nouvelle vue du robot.

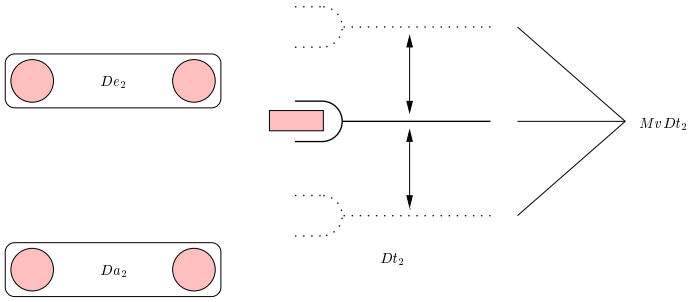


Figure A.9: Vue du mouvement vertical du robot

A.4.1 Spécification descriptive

A partir de ce niveau de raffinement, nous allons abandonner la représentation de la spécification descriptive sous forme de figure. Pour des raisons de commodité de lecture chacun des paragraphes des figures des niveaux précédents feront l'objet d'une section spécifique.

Variables

Les variables De_2, Dt_2 et Da_2 restent identiques aux variables du niveau précédent De_1, Dt_1 et Da_1. La variable $PosDt_1$ est remplacée par la variable $MvDt_2$ qui peut prendre quatres valeurs : *enbas*, *enhaut*, *monte* et *descend*.

Invariant

– – **Invariant de collage**
$De_2 = De_1 \wedge Dt_2 = Dt_1 \wedge \quad [I_1]$
$Da_2 = Da_1 \wedge MvDt_2 \in \{\ enbas, enhaut, monte, descend\ \} \wedge \quad [I_2]$

– – **Lorsque Dt_2 est en bas ou qu'il monte, la variable $PosDt_1$ avait la valeur *enbas*, ce qui signifie que l'appareil abstrait du premier raffinement était en bas**
$(\ MvDt_2 \in \{\ enbas, monte\ \} \Leftrightarrow PosDt_1 = enbas\ ) \wedge \quad [I_3]$

– – **idem pour la position haut**
$(\ MvDt_2 \in \{\ enhaut, descend\ \} \Leftrightarrow PosDt_1 = enhaut\ ) \wedge \quad [I_4]$

– – **Le dispositif de transport contient une pièce quand il monte**
$(\ MvDt_2 = monte \Rightarrow Dt_2 = occupé\ ) \wedge \quad [I_5]$

– – **Le dispositif de transport est vide quand il monte**
$(\ MvDt_2 = descend \Rightarrow Dt_2 = libre\ ) \quad [I_6]$

Par rapport aux deux énoncés précédents, en plus du typage des variables, on a exprimé des contraintes sur l'état du système. Sur les 32 états potentiels : $2 \times 2 \times 2 \times 4$, 24 sont possibles : les états où $MvDt_2 = monte \wedge Dt_2 = libre$ et $MvDt_2 = descend \wedge Dt_2 = occupé$ ne peuvent exister d'après les propriétés $[I_5]$ et $[I_6]$.

On note que les propriétés dynamiques C_1^5 et C_1^6 semblent raffinéees par les invariants $[I_5]$ et $[I_6]$. Mais ces derniers sont insuffisants pour exprimer les propriétés C_1^5 et C_1^6. Par exemple, il ne faut pas interdire un état où $MvDt_2 = enhaut \wedge Dt_2 = libre$ suit un état où $MvDt_2 = monte \wedge Dt_2 = occupé$. Seule une propriété dynamique peut exprimer ce fait. Par conséquent on

peut dire que les invariants $[I_5]$ et $[I_6]$ sont des sous spécifications des propriétés dynamiques C_1^5 et C_1^6. Nous spécifions donc C_2^5 (resp. C_2^6) comme une propriété dynamique qui indique que Dt_2 reste occupé pendant tout le mouvement de montée (resp. descente).

Propriétés dynamiques

$$C_2^1 \triangleq \Box\left(\left(De_2 = occupé \wedge Dt_2 = occupé \Rightarrow \left(Dt_2 = occupé\right) \mathcal{U} \left(\begin{array}{l} De_2 = libre \wedge \\ Dt_2 = occupé \end{array}\right)\right)\right)$$

$$C_2^2 \triangleq \Box\left(\left(Da_2 = occupé \wedge \bigcirc\left(Da_2 = libre\right)\right) \Rightarrow Dt_2 = libre\right)$$

$$C_2^3 \triangleq \Box\left(\left(Dt_2 = occupé \wedge \bigcirc\left(Dt_2 = libre\right)\right) \Rightarrow MvDt_2 = enhaut\right)$$

$$C_2^4 \triangleq \Box\left(\left(Dt_2 = libre \wedge \bigcirc\left(Dt_2 = occupé\right)\right) \Rightarrow MvDt_2 = enbas\right)$$

$$C_2^5 \triangleq \Box\left(\begin{array}{l} \left(MvDt_2 = enbas \wedge \bigcirc MvDt_2 = monte\right) \\ \Rightarrow \left(Dt_2 = occupé\right) \mathcal{U} \left(\begin{array}{l} MvDt_2 = enhaut \wedge \\ Dt_2 = occupé \end{array}\right) \end{array}\right)$$

$$C_2^6 \triangleq \Box\left(\begin{array}{l} \left(MvDt_2 = enhaut \wedge \bigcirc\left(MvDt_2 = descend\right)\right) \\ \Rightarrow \left(Dt_2 = libre\right) \mathcal{U} \left(\begin{array}{l} MvDt_2 = enbas \wedge \\ Dt_2 = libre \end{array}\right) \end{array}\right)$$

Les propriétés C_2^1 à C_2^6 raffinent les propriétés C_1^1 à C_1^6. Nous ne spécifions pas de nouvelles propriétés à ce niveau. Nous remarquons que la forme des propriétés temporelles C_2^1 à C_2^4 est inchangée à ce niveau. Ceci est dû d'une part à la forme des propriétés et d'autre part au "faible" renforcement de spécification effectué par ce raffinement.

A.4.2 Spécification opérationnelle

A ce niveau on observe huit événements, les six anciens ($Chargt_2$, $DéChargt_2$, $Evac_2$, $ArMontée_2$, $ArDescente_2$, $ArPièce_2$) et deux nouveaux ($Montée_2$, $Descente_2$).

Initialisation $\triangleq$ $Dt_2 := libre \parallel De_2 := libre \parallel Da_2 := libre \parallel MvDt_2 := enbas$

$Chargt_2 \triangleq$
SELECT
$Dt_2 = libre \wedge Da_2 = occupé \wedge \mathbf{MvDt_2 = enbas}$
THEN
$Dt_2, Da_2 := occupé, libre$
END

$DéChargt_2 \triangleq$
SELECT
$Dt_2 = occupé \wedge De_2 = libre \wedge \mathbf{MvDt_2 = enhaut}$
THEN
$Dt_2, De_2 := libre, occupé$
END

$Evac_2 \triangleq$ idem $Evac_1$ figure A.7 page x avec indice 2

$ArPièce_2 \triangleq$ idem $ArPièce_1$ figure A.7 page x avec indice 2

$ArMontée_2 \triangleq$ SELECT
$Dt_2 = occupé \wedge \mathbf{MvDt_2 = monte}$
THEN
$\mathbf{MvDt_2 := enhaut}$
END

$ArDescente_2 \triangleq$ SELECT
$Dt_2 = libre \wedge \mathbf{MvDt_2 = descend}$
THEN
$\mathbf{MvDt_2 := enbas}$
END

$Montée_2 \triangleq$ SELECT
$Dt_2 = occupé \wedge MvDt_2 = enbas$
THEN
$MvDt_2 = monte$
END

$Descente_2 \triangleq$ SELECT
$Dt_2 = libre \wedge MvDt_2 = enhaut$
THEN
$MvDt_2 = descend$
END

A.4.3 Sémantique

Sur le graphe d'accessibilité de la figure A.10 on voit que l'événement $ArDescente_1$ (respectivement $ArMontée_1$) du niveau 1 est décomposé en $Descente_2$ suivi de $ArDescente_2$ (respectivement $Montée_2$ suivi de $ArMontée_2$) au niveau 2.

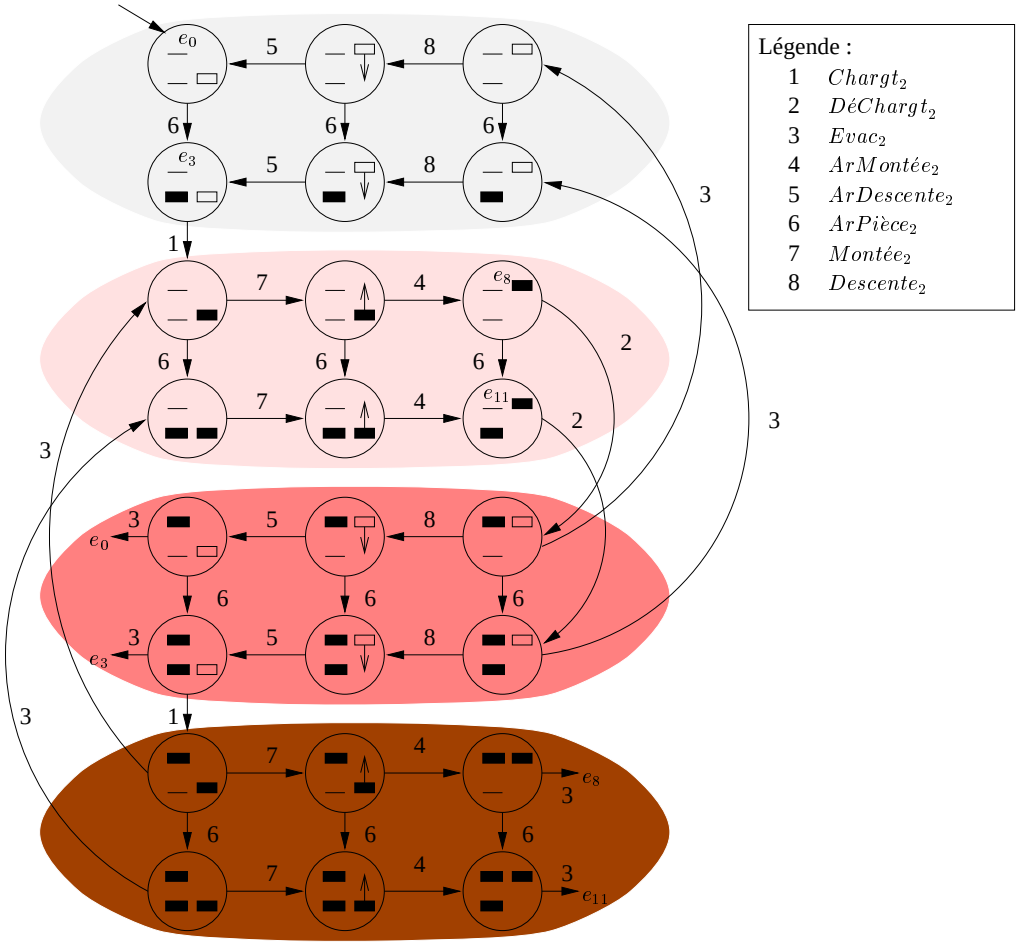


Figure A.10: Système de transition du robot au deuxième raffinement

A.5 Troisième raffinement du robot

Après avoir introduit le mouvement vertical, nous introduisons le mouvement horizontal du bras et celui de la pince. Nous introduisons des événements duaux de $ArMontée_3$, $ArDescente_3$, $Montée_3$ et $Descente_3$ pour le mouvement horizontal du bras: $ArRecule_3$, $ArAvance_3$, $ReculeEnHaut_3$, $ReculeEnBas_3$, $AvanceEnHaut_3$, $AvanceEnBas_3$ et pour la pince: $ArFermeture_3$, $ArOuverture_3$, $Fermeture_3$ et $Ouverture_3$.

A.5.1 Spécification descriptive

Variables

Les variables De_3, Da_3 et $MvDt_3$ sont identiques aux variables De_2, Da_2 et $MvDt_2$. On introduit les nouvelles variables suivantes:

- $MhDt_3$: mouvement horizontal de Dt,
- Mp_3: mouvement de la pince.

La variable Dt_2 est raffinée par Mp_3 qui représente l'état de Dt ainsi:

- $Dt_2 = occupé$ si Mp_3 se ferme ou est fermée,
- $Dt_2 = libre$ si Mp_3 s'ouvre ou est ouverte.

Invariant

– – **Invariant de collage**
$De_3 = De_2 \wedge Da_3 = Da_2 \wedge MvDt_3 = MvDt_2 \wedge$
$MhDt_3 \in \{\ enavant, enarrière, avance, recule\ \} \wedge$
$Mp_3 \in \{\ ouverte, fermée, s'ouvre, seferme\ \} \wedge$

– – **Relation entre Mp_3 et Dt_2.**
$Mp_3 \in \{\ fermée, seferme\ \} \Rightarrow Dt_2 = occupé \wedge$
$Mp_3 \in \{\ ouverte, s'ouvre\ \} \Rightarrow Dt_2 = libre \wedge$

Ces propriétés expriment :

- que la pince se ferme sur une pièce,
- et que dès que la pince s'ouvre, la pièce est libre, autrement dit :
  - Dt contient une pièce est exprimé par une pince fermée ou en train de se fermer,
  - Dt libre est exprimé par la pince s'ouvre ou est ouverte.

– – **La pince descend toujours en position arrière**
$MvDt_3 = descend \Rightarrow MhDt_3 = enarrière \wedge \quad [I_7]$

– – **Le bras ne peut pas avancer en montant ou en descendant**
$MhDt_3 = avance \Rightarrow MvDt_3 \in \{\ enhaut, enbas\ \} \wedge \quad [I_8]$ et $[I_9]$

– – **Le bras ne peut pas reculer en descendant**
$MhDt_3 = recule \Rightarrow MvDt_3 \neq descend \wedge \quad [I_{10}]$ et $[I_{11}]$

– – **La pince se ferme en bas et en avant**
$Mp_3 = seferme \Rightarrow (\ MhDt_3 = enavant \wedge MvDt_3 = enbas\ ) \wedge \quad [I_{12}]$

– – **La pince s'ouvre en haut et en avant**
$Mp_3 = s'ouvre \Rightarrow (\ MhDt_3 = enavant \wedge MvDt_3 = enhaut\ ) \wedge \quad [I_{13}]$

– – **Si la pince recule en bas ou avance en haut alors elle tient une pièce**
$\left(\begin{array}{l}(\ MhDt_3 = recule \wedge MhDt_3 = enbas\ ) \vee \\ (\ MhDt_3 = avance \wedge MhDt_3 = enhaut\ )\end{array}\right) \Rightarrow Mp_3 = fermée \wedge \quad [I_{14}]$ et $[I_{16}]$

– – **La pince monte fermée et descend ouverte**
$MvDt_3 = monte \Rightarrow Mp_3 = fermée \wedge \quad [I_{15}]$
$MvDt_3 = descend \Rightarrow Mp_3 = ouverte \wedge \quad [I_{18}]$

– – **Si la pince recule en haut ou avance en bas alors elle est ouverte**
$\left(\begin{array}{l}(\ MhDt_3 = recule \wedge MhDt_3 = enhaut\ ) \vee \\ (\ MhDt_3 = avance \wedge MhDt_3 = enbas\ )\end{array}\right) \Rightarrow Mp_3 = ouverte \wedge \quad [I_{17}]$ et $[I_{20}]$

– – **Le bras descend en position arrière, par contre il peut monter sans être en position arrière mais en reculant**

$MvDt_3 = descend \Rightarrow Mp_3 = enarrière \wedge \quad [I_{19}]$

– – **On pourrait interdire l'arrivée d'une pièce tant que la pince n'a pas débuté un mouvement de recul**

$(\ Mp_3 = fermée \wedge MhDt_3 = enavant \wedge MvDt_3 \in \{ \ enbas, monte \ \} \ ) \Rightarrow Da_3 = libre$

$MhDt_3 \in \{ \ enbas, monte \ \}$ impose un mouvement de recul pour libérer l'arrivée des pièces.

Les propriétés $[I_1]$, $[I_2]$, $[I_3]$, $[I_4]$, $[I_5]$, $[I_6]$, $[I_{11}]$, sont partiellement exprimées dans l'invariant. Par exemple la propriété $[I_1]$ est impossible à exprimer dans l'invariant car elle ne correspond pas à une restriction sur les états atteignables mais à une restriction sur les séquences d'état.

Propriétés dynamiques

$$C_3^1 \triangleq \Box \left(\begin{array}{l} (\ De_3 = occupé \wedge Mp_3 \in \{fermée, seferme\} \ ) \\ \Rightarrow \ (\ Mp_3 \in \{fermée, seferme\} \ ) \ \mathcal{U} \ \left(\begin{array}{l} De_3 = libre \wedge \\ Mp_3 \in \{fermée, seferme\} \end{array} \right) \end{array} \right)$$

$$C_3^2 \triangleq \Box \ (\ (\ Da_3 = occupé \wedge \bigcirc (\ Da_3 = libre \ ) \ ) \Rightarrow Mp_3 \in \{ouverte, s'ouvre\} \ )$$

$$C_3^3 \triangleq \Box \ (\ (\ Mp_3 \in \{fermée, seferme\} \wedge \bigcirc (\ Mp_3 \in \{ouverte, s'ouvre\} \ ) \ ) \Rightarrow MvDt_3 = enhaut \ )$$

$$C_3^4 \triangleq \Box \ (\ (\ Mp_3 \in \{ouverte, s'ouvre\} \wedge \bigcirc (\ Mp_3 \in \{fermée, seferme\} \ ) \ ) \Rightarrow MvDt_3 = enbas \ )$$

$$C_3^5 \triangleq \Box \left(\begin{array}{l} (\ MvDt_3 = enbas \wedge \bigcirc (\ MvDt_3 = monte \ ) \ ) \\ \Rightarrow \ (\ Mp_3 \in \{fermée, seferme\} \ ) \ \mathcal{U} \ \left(\begin{array}{l} MvDt_3 = enhaut \wedge \\ Mp_3 \in \{fermée, seferme\} \end{array} \right) \end{array} \right)$$

$$C_3^6 \triangleq \Box \left(\begin{array}{l} (\ MvDt_3 = enhaut \wedge \bigcirc (\ MvDt_3 = descend \ ) \ ) \\ \Rightarrow \ (\ Mp_3 \in \{ouverte, s'ouvre\} \ ) \ \mathcal{U} \ (\ MvDt_3 = enbas \wedge Mp_3 \in \{ouverte, s'ouvre\} \ ) \end{array} \right)$$

$$C_3^7 \triangleq \Box \left(\begin{array}{l} (\ MvDt_3 = enhaut \wedge \bigcirc (\ MvDt_3 = descend \ ) \ ) \\ \Rightarrow \ (\ MhDt_3 = enarrière \ ) \ \mathcal{U} \ (\ MvDt_3 = enbas \wedge MhDt_3 = enarrière \ ) \end{array} \right)$$

$$C_3^{8,9} \triangleq \Box \left(\begin{array}{l} (\ MhDt_3 = enarrière \wedge \bigcirc (\ MhDt_3 = avance \ ) \ ) \\ \Rightarrow \left(\left(\begin{array}{l} MvDt_3 = enhaut \vee \\ MvDt_3 = enbas \end{array} \right) \right) \mathcal{U} \left(MhDt_3 = enavant \wedge \left(\begin{array}{l} MvDt_3 = enhaut \vee \\ MvDt_3 = enbas \end{array} \right) \right) \end{array} \right)$$

$$C_3^{10} \triangleq \Box \left(\begin{array}{l} MvDt_3 = enavant \wedge MhDt_3 = enhaut \wedge Mp_3 \in \{ouverte, s'ouvre\} \\ \Rightarrow \ (\ MvDt_3 = enhaut \ ) \ \mathcal{U} \ (\ MhDt_3 = enarrière \ ) \end{array} \right)$$

$$C_3^{11} \triangleq \Box \left(\begin{array}{l} MvDt_3 = monte \wedge \bigcirc MvDt_3 = enhaut \\ \Rightarrow \ (\ MhDt_3 = enarrière \ ) \ \mathcal{U} \ (\ MvDt_3 = enhaut \ ) \end{array} \right)$$

$$C_3^{14} \triangleq \Box \left(\begin{array}{l} MhDt_3 = enavant \wedge MvDt_3 = enbas \wedge \bigcirc MhDt_3 = recule \\ \Rightarrow \ (\ Mp_3 = fermée \ ) \ \mathcal{U} \ (\ MhDt_3 = enarriere \wedge Mp_3 = fermée \ ) \end{array} \right)$$

$$C_3^{15} \triangleq \Box \left(\begin{array}{l} MhDt_3 \neq avance \wedge MvDt_3 = enbas \wedge \bigcirc MvDt_3 = monte \\ \Rightarrow \ (\ Mp_3 = fermée \ ) \ \mathcal{U} \ (\ MvDt_3 = enhaut \wedge Mp_3 = fermée \ ) \end{array} \right)$$

$$C_3^{16} \triangleq \Box \left(\begin{array}{l} MvDt_3 = enhaut \wedge MhDt_3 = enarrière \wedge \bigcirc MhDt_3 = avance \\ \Rightarrow \ (\ Mp_3 = fermée \ ) \ \mathcal{U} \ (\ MhDt_3 = enavant \wedge Mp_3 = fermée \ ) \end{array} \right)$$

$$C_3^{17} \triangleq \Box \left(\begin{array}{l} MvDt_3 = enhaut \wedge MhDt_3 = enavant \wedge \bigcirc MhDt_3 = recule \\ \Rightarrow \ (\ Mp_3 = ouverte \ ) \ \mathcal{U} \ (\ MhDt_3 = enarrière \wedge Mp_3 = ouverte \ ) \end{array} \right)$$

$$C_3^{18} \triangleq \Box \left(\begin{array}{l} MvDt_3 = enhaut \wedge MhDt_3 = enarrière \wedge \bigcirc aimvdt = descend \\ \Rightarrow \ (\ Mp_3 = ouverte \ ) \ \mathcal{U} \ (\ MvDt_3 = enbas \wedge Mp_3 = ouverte \ ) \end{array} \right)$$

$$C_3^{19} \triangleq \Box \left(\begin{array}{l} MvDt_3 = enhaut \wedge Mp_3 = ouverte \wedge \bigcirc aimvdt = descend \\ \Rightarrow \ (\ MhDt_3 = enarrière \ ) \ \mathcal{U} \ (\ MvDt_3 = enbas \wedge Mp_3 = ouverte \ ) \end{array} \right)$$

$$C_3^{20} \triangleq \Box \left(\begin{array}{l} MvDt_3 = enbas \wedge MhDt_3 = enarrière \wedge \bigcirc MhDt_3 = avance \\ \Rightarrow \ (\ Mp_3 = ouverte \ ) \ \mathcal{U} \ (\ MhDt_3 = enavant \wedge Mp_3 = ouverte \ ) \end{array} \right)$$

Les six premières propriétés raffinent C_2^1 à C_2^6. Les propriétés informelles de $[I_7]$ à $[I_{11}]$ et de $[I_{14}]$ à $[I_{20}]$ sont sous spécifiées dans l'invariant. Il est nécessaire de les exprimer sous la forme des

propriétés dynamiques C_3^7 à C_3^{11} et C_3^{14} à C_3^{20}.

A.5.2 Spécification opérationelle

Liste des événements : 8 anciens et 8 nouveaux,

- $Chargt_3$, $DéChargt_3$, $Evac_3$, $ArPièce_3$, $ArDescente_3$, $ArMontée_3$, $Descente_3$, $Montée_3$ sont les anciens événements,
- $ArRecule_3$, $ArAvance_3$ représentent l'arrêt du mouvement horizontal dans l'une ou l'autre direction,
- $AvanceEnHaut_3$, $ReculeEnHaut_3$ représente le mouvement horizontal lorsque le dispositif de transport se trouve en haut,
- $AvanceEnBas_3$, $ReculeEnBas_3$ correspond aux mouvements de Dt lorsqu'il se tourve en bas,
- $ArOuverture_3$, $ArFermeture_3$ sont les événements modélisant respectivement l'arrêt de l'ouverture et celui de la fermeture de la pince,
- l'ouverture de la pince est représentée par l'événement $DéChargt_3$,
- la fermeture de la pince est représentée par l'événement $Chargt_3$.

Remarque: Il n'y a pas d'ouverture en bas et de fermeture en haut.

Initialisation $\triangleq$ $De_3 := libre \ \| \ Da_3 := libre \ \| \ MvDt_3 := enbas \ \| \ MhDt_3 = enavant \ \| \ Mp_3 = ouverte$

$Chargt_3 \triangleq$ SELECT
$Da_3 = occupé \wedge MvDt_3 = enbas \wedge$
$\mathbf{Mp_3 = ouverte \wedge MhDt_3 = enavant}$
THEN
$Mp_3, Da_3 := seferme, libre$
END

$DéChargt_3 \triangleq$ SELECT
$De_3 = libre \wedge MvDt_3 = enhaut \wedge$
$\mathbf{Mp_3 = fermée \wedge MhDt_3 = enavant}$
THEN
$Mp_3, De_3 := s'ouvre, occupé$
END

$Evac_3 \triangleq$ idem $Evac_2$ figure A.7 page x avec indice 3

$ArPièce_3 \triangleq$ idem $ArPièce_2$ figure A.7 page x avec indice 3

$ArMontée_3 \triangleq$ SELECT
$MvDt_3 = monte \wedge \mathbf{Mp_3} = \mathbf{fermée} \wedge \mathbf{MhDt_3} = \mathbf{enarrière}$
THEN
$MvDt_3 := enhaut$
END

$ArDescente_3 \triangleq$ SELECT
$MvDt_3 = descend \wedge \mathbf{Mp_3} = \mathbf{ouverte} \wedge \mathbf{MhDt_3} = \mathbf{enarrière}$
THEN
$MvDt_3 := enbas$
END

$Montée_3 \triangleq$ SELECT
$MvDt_3 = enbas \wedge \mathbf{Mp_3} = \mathbf{fermée} \wedge \mathbf{MhDt_3} \in \{\ recule, enarrière\ \}$
THEN
$MvDt_3 := monte$
END

$Descente_3 \triangleq$ SELECT
$MvDt_3 = enhaut \wedge \mathbf{Mp_3} = \mathbf{ouverte} \wedge \mathbf{MhDt_3} = \mathbf{enarrière}$
THEN
$MvDt_3 := descend$
END

$ArRecule_3 \triangleq$ SELECT
$MhDt_3 = recule \wedge \left(\begin{array}{l} (\ MvDt_3 = enhaut \wedge Mp_3 = ouverte\ ) \vee \\ (\ MvDt_3 \in \{\ enbas, monte\ \} \wedge Mp_3 = fermée\ ) \end{array} \right)$
THEN
$MhDt_3 := enarrière$
END

$ArAvance_3 \triangleq$ SELECT
$MhDt_3 = avance \wedge \left(\begin{array}{l} (\ MvDt_3 = enhaut \wedge Mp_3 = fermée\ ) \vee \\ (\ MvDt_3 = enbas \wedge Mp_3 = ouverte\ ) \end{array} \right)$
THEN
$MhDt_3 := enavant$
END

$AvanceEnHaut_3 \triangleq$ SELECT
$MhDt_3 = enarriere \wedge MvDt_3 = enhaut \wedge Mp_3 = fermée$
THEN
$MhDt_3 := avance$
END

$AvanceEnBas_3 \triangleq$ SELECT
$MhDt_3 = enarriere \wedge MvDt_3 = enbas \wedge Mp_3 = ouverte$
THEN
$MhDt_3 := avance$
END

$ReculeEnHaut_3 \triangleq$ SELECT
$MhDt_3 = enavant \wedge MvDt_3 = enhaut \wedge Mp_3 = ouverte$
THEN
$MhDt_3 := recule$
END

$ReculeEnBas_3 \triangleq$ SELECT
$MhDt_3 = enavant \wedge MvDt_3 = enbas \wedge Mp_3 = fermée$
THEN
$MhDt_3 := recule$
END

$ArOuverture_3 \triangleq$ SELECT
$Mp_3 = s'ouvre \wedge \texttt{MhDt}_3 = \texttt{enavant} \wedge \texttt{MvDt}_3 = \texttt{enhaut}$
THEN
$Mp_3 := ouverte$
END

$ArFermeture_3 \triangleq$ SELECT
$Mp_3 = seferme \wedge \texttt{MhDt}_3 = \texttt{enavant} \wedge \texttt{MvDt}_3 = \texttt{enbas}$
THEN
$Mp_3 := fermée$
END

La garde de l'événement $ArOuverture_3$ (resp. $ArFermeture_3$) peut être simplifiée. En effet la partie figurant en caractères `modernes` n'est pas nécessaire. Cet événement n'est déclenchable que si $DéChargt_3$ (resp. $Chargt_3$) à été éxécuté. Hors ce dernier ne se produit que lorsque le pince se trouve en position de déchargement (resp. chargement). De plus enlever la partie superflue de la garde peut être réaliser sans risque car $ReculeEnHaut_3$ (resp. $ReculeEnBas_3$) ne peut arriver que si $Mp_3 = ouverte$ (resp. $Mp_3 = fermée$).

A.5.3 Sémantique

La représentation du graphe d'accessibilité devient impossible à partir de ce niveau. La complexité et la taille du nouveau graphe de transition ne permet pas de le représenter.

A.6 Quatrième raffinement

Durant ce raffinement, nous allons réaliser la séparation entre la partie logicielle et la partie matérielle de l'automatisme. Nous allons introduire des variables représentant l'état physique des

différents moteurs ainsi que celles permettant de modéliser la communication entre le logiciel et le matériel (sous forme d'interruptions). On constate tout au long de ce niveau de raffinement qu'il y a un décalage entre la partie matérielle et la partie logiciel. En effet, il est necessaire de modéliser des délais de réponse des différentes parties comme par exemple le fait que lorsque le logiciel donne l'ordre de démarrer un moteur, le moteur physique n'est pas encore en mouvement même si la variable du logiciel l'indique. Un autre exemple de décalage se situ lors de la génération de l'interruption. Le moteur est déjà arrêté lorsque cette dernière se produit alors que le logiciel, informé par l'interruption, n'a pas encore pris en compte l'arrêt du déplacement du moteur.

A.6.1 Spécification descriptive

Variables

On conserve les variables du raffinement précédent en se contentant de changer leurs indices. On introduit de nouvelles variables permettant de modéliser la gestion matérielle de l'automatisme. Elles peuvent se répartirent dans trois catégories : les variables locales au matériel, les variables de communication du logiciel vers le matériel et les variables de communication dans l'autre sens. Il existe une quatrième catégorie où se trouvent toutes les variables existant du niveau précédent. Elles sont considérés locales au logiciel.

Les variables locales au matériel

- $EtMotv_4$: cette variable représente l'état du moteur réalisant les mouvements verticaux du bras,
- $EtMoth_4$: idem pour le moteur permettant de déplacer la pince horizontalement,
- $EtMotp_4$: idem pour le moteur permettant d'ouvrir et de fermer la pince.

Variables de communication du logiciel vers le matériel

- $DmMotv_4$: cette variable permet au logiciel de dire au moteur vertical de démarrer. On ne fait pas allusion au sens du mouvement (monter ou descendre),
- $DmMoth_4$: idem pour le moteur horizontal,
- $SnMotp_4$: idem pour le moteur de la pince,
- $SnMotv_4$: cette variable représente le sens du mouvement vertical désiré par le logiciel. Elle doit être utilisée conjointement avec $DmMotv_4$,
- $SnMoth_4$: idem pour le mouvement horizontal et $DmMoth_4$,
- $SnMotp_4$: idem pour le mouvement de la pince et $SnMotp_4$.

Variables de communication du matériel vers le logiciel

Lorsqu'un moteur désire envoyer une information au logiciel, il déclenche une interruption. Les variables suivantes ont pour valeur *oui* si l'interruption est générée et *non* sinon. Les interruptions reconnus dans ce dernier niveau de raffinement sont :

- $ItDa_4$: une pièce est arrivée sur Da_4.
- $ItDe_4$: une pièce est évacuée de De_4.
- $ItMotv_4$: le moteur vertical s'est arrêté.
- $ItMoth_4$: le moteur horizontal a stoppé sont mouvement.

- $ItMotp_4$: le moteur de la pince s'est arrêté.

INVARIANT

$De_4 = De_3 \wedge Da_4 = Da_3 \wedge MvDt_4 = MvDt_3 \wedge$
$MhDt_4 = MhDt_3 \wedge Mp_4 = Mp_3 \wedge$
$EtMotv_4 \in \{ monte, descend, arrêt \} \wedge EtMoth_4 \in \{ avance, recule, arrêt \} \wedge$
$EtMotp_4 \in \{ ouverture, fermeture, arrêt \} \wedge DmMotv_4 \in \{oui, non\} \wedge$
$DmMoth_4 \in \{oui, non\} \wedge SnMotp_4 \in \{oui, non\} \wedge$
$SnMotv_4 \in \{ monte, descend \} \wedge SnMoth_4 \in \{ avance, recule \} \wedge$
$SnMotp_4 \in \{ ouverture, fermeture \} \wedge ItMotv_4 \in \{oui, non\} \wedge$
$ItMoth_4 \in \{oui, non\} \wedge ItMotp_4 \in \{oui, non\} \wedge$

– – **Si un ordre de mouvementest donné par le logiciel à un moteur, l'état de ce dernier indique aussi qu'il y a mouvement**
$\forall v \in \{ monte, descend \} (DmMotv_4 = oui \wedge SnMotv_4 = v) \Rightarrow MvDt_4 = v \wedge$
$\forall h \in \{ avance, recule \} (DmMoth_4 = oui \wedge SnMoth_4 = h) \Rightarrow MhDt_4 = h \wedge$
$(SnMotp_4 = oui \wedge SnMotp_4 = ouverture) \Rightarrow Mp_4 = s'ouvre \wedge$
$(SnMotp_4 = oui \wedge SnMotp_4 = fermeture) \Rightarrow Mp_4 = seferme \wedge$

– – **Un moteur ne peut démarrer que s'il est arrêt**
$DmMotv_4 = oui \Rightarrow EtMotv_4 = arrêt \wedge$
$DmMoth_4 = oui \Rightarrow DmMoth_4 = arrêt \wedge$
$SnMotp_4 = oui \Rightarrow SnMotp_4 = arrêt \wedge$

– – **Lorsqu'une interruption est déclenchée, les moteurs sont arrêtées mais les variables correspondantes du logiciel indiquent toujours un mouvement. Ceci est dû au fait que l'interruption n'a pas encore été traitée par le logiciel**
$ItMotv_4 = oui \Rightarrow (EtMotv_4 = arrêt \wedge MvDt_4 \in \{ monte, descend \}) \wedge$
$ItMoth_4 = oui \Rightarrow (EtMoth_4 = arrêt \wedge MhDt_4 \in \{ avance, recule \}) \wedge$
$ItMotp_4 = oui \Rightarrow (EtMotp_4 = arrêt \wedge Mp_4 \in \{ fermeture, ouverture \}) \wedge$

– – **Si le logiciel considère que les dispositifs sont stoppés alors les moteurs correspondant le sont**
$MvDt_4 \in \{ enhaut, enbas \} \Rightarrow EtMotv_4 = arrêt \wedge$
$MhDt_4 \in \{ enavant, enarrière \} \Rightarrow EtMoth_4 = arrêt \wedge$
$Mp_4 \in \{ fermée, ouverte \} \Rightarrow EtMotp_4 = arrêt$

– – **Si un des moteurs est en mouvement alors le logiciel sait qu'il est en mouvement**
$\forall v \in \{ monte, descend \} EtMotv_4 = v \Rightarrow MvDt_4 = v \wedge$
$\forall h \in \{ avance, recule \} EtMoth_4 = h \Rightarrow MhDt_4 = h \wedge$
$EtMotp_4 = ouverture \Rightarrow Mp_4 = s'ouvre \wedge$
$EtMotp_4 = fermeture \Rightarrow Mp_4 = seferme \wedge$

PROPRIÉTÉS DYNAMIQUES

$$C_4^1 \triangleq \Box \left(\begin{array}{l} (\ De_4 = occupé \wedge Mp_4 \in \{fermée, seferme\} \ ) \\ \Rightarrow \ (\ Mp_4 \in \{fermée, seferme\} \ ) \ \mathcal{U} \ \left(\begin{array}{l} De_4 = libre \wedge \\ Mp_4 \in \{fermée, seferme\} \end{array} \right) \end{array} \right)$$

$$C_4^2 \triangleq \Box (\ (\ Da_4 = occupé \wedge \bigcirc (\ Da_4 = libre \ ) \ ) \Rightarrow Mp_4 \in \{ouverte, s'ouvre\} \ )$$

$$C_4^3 \triangleq \Box (\ (\ Mp_4 \in \{fermée, seferme\} \wedge \bigcirc (\ Mp_4 \in \{ouverte, s'ouvre\} \ ) \ ) \Rightarrow MvDt_4 = enhaut \ )$$

$$C_4^4 \triangleq \Box (\ (\ Mp_4 \in \{ouverte, s'ouvre\} \wedge \bigcirc (\ Mp_4 \in \{fermée, seferme\} \ ) \ ) \Rightarrow MvDt_4 = enbas \ )$$

$$C_4^5 \triangleq \Box \left(\begin{array}{l} (\ MvDt_4 = enbas \wedge \bigcirc (\ MvDt_4 = monte \ ) \ ) \\ \Rightarrow \ (\ Mp_4 \in \{fermée, seferme\} \ ) \ \mathcal{U} \ \left(\begin{array}{l} MvDt_4 = enhaut \wedge \\ Mp_4 \in \{fermée, seferme\} \end{array} \right) \end{array} \right)$$

$$C_4^6 \triangleq \Box \left(\begin{array}{l} (\ MvDt_4 = enhaut \wedge \bigcirc (\ MvDt_4 = descend \ ) \ ) \\ \Rightarrow \ (\ Mp_4 \in \{ouverte, s'ouvre\} \ ) \ \mathcal{U} \ (\ MvDt_4 = enbas \wedge Mp_4 \in \{ouverte, s'ouvre\} \ ) \end{array} \right)$$

$$C_4^7 \triangleq \Box \left(\begin{array}{l} (\ MvDt_4 = enhaut \wedge \bigcirc (\ MvDt_4 = descend \ ) \ ) \\ \Rightarrow \ (\ MhDt_4 = enarrière \ ) \ \mathcal{U} \ (\ MvDt_4 = enbas \wedge MhDt_4 = enarrière \ ) \end{array} \right)$$

$$C_4^{8,9} \triangleq \Box \left(\begin{array}{l} (\ MhDt_4 = enarrière \wedge \bigcirc (\ MhDt_4 = avance \ ) \ ) \\ \Rightarrow \left(\left(\begin{array}{l} MvDt_4 = enhaut \vee \\ MvDt_4 = enbas \end{array} \right) \right) \mathcal{U} \left(MhDt_4 = enavant \wedge \left(\begin{array}{l} MvDt_4 = enhaut \vee \\ MvDt_4 = enbas \end{array} \right) \right) \end{array} \right)$$

$$C_4^{10} \triangleq \Box \left(\begin{array}{l} MvDt_4 = enavant \wedge MhDt_4 = enhaut \wedge Mp_4 \in \{ouverte, s'ouvre\} \\ \Rightarrow \ (\ MvDt_4 = enhaut \ ) \ \mathcal{U} \ (\ MhDt_4 = enarrière \ ) \end{array} \right)$$

$$C_4^{11} \triangleq \Box \left(\begin{array}{l} MvDt_4 = monte \wedge \bigcirc MvDt_4 = enhaut \\ \Rightarrow \ (\ MhDt_4 = enarrière \ ) \ \mathcal{U} \ (\ MvDt_4 = enhaut \ ) \end{array} \right)$$

$$C_4^{14} \triangleq \Box \left(\begin{array}{l} MhDt_4 = enavant \wedge MvDt_4 = enbas \wedge \bigcirc MhDt_4 = recule \\ \Rightarrow \ (\ Mp_4 = fermée \ ) \ \mathcal{U} \ (\ MhDt_4 = enarriere \wedge Mp_4 = fermée \ ) \end{array} \right)$$

$$C_4^{15} \triangleq \Box \left(\begin{array}{l} MhDt_4 \neq avance \wedge MvDt_4 = enbas \wedge \bigcirc MvDt_4 = monte \\ \Rightarrow \ (\ Mp_4 = fermée \ ) \ \mathcal{U} \ (\ MvDt_4 = enhaut \wedge Mp_4 = fermée \ ) \end{array} \right)$$

$$C_4^{16} \triangleq \Box \left(\begin{array}{l} MvDt_4 = enhaut \wedge MhDt_4 = enarrière \wedge \bigcirc MhDt_4 = avance \\ \Rightarrow \ (\ Mp_4 = fermée \ ) \ \mathcal{U} \ (\ MhDt_4 = enavant \wedge Mp_4 = fermée \ ) \end{array} \right)$$

$$C_4^{17} \triangleq \Box \left(\begin{array}{l} MvDt_4 = enhaut \wedge MhDt_4 = enavant \wedge \bigcirc MhDt_4 = recule \\ \Rightarrow \ (\ Mp_4 = ouverte \ ) \ \mathcal{U} \ (\ MhDt_4 = enarrière \wedge Mp_4 = ouverte \ ) \end{array} \right)$$

$$C_4^{18} \triangleq \Box \left(\begin{array}{l} MvDt_4 = enhaut \wedge MhDt_4 = enarrière \wedge \bigcirc aimvdt = descend \\ \Rightarrow \ (\ Mp_4 = ouverte \ ) \ \mathcal{U} \ (\ MvDt_4 = enbas \wedge Mp_4 = ouverte \ ) \end{array} \right)$$

$$C_4^{19} \triangleq \Box \left(\begin{array}{l} MvDt_4 = enhaut \wedge Mp_4 = ouverte \wedge \bigcirc aimvdt = descend \\ \Rightarrow \ (\ MhDt_4 = enarrière \ ) \ \mathcal{U} \ (\ MvDt_4 = enbas \wedge Mp_4 = ouverte \ ) \end{array} \right)$$

$$C_4^{20} \triangleq \Box \left(\begin{array}{l} MvDt_4 = enbas \wedge MhDt_4 = enarrière \wedge \bigcirc MhDt_4 = avance \\ \Rightarrow \ (\ Mp_4 = ouverte \ ) \ \mathcal{U} \ (\ MhDt_4 = enavant \wedge Mp_4 = ouverte \ ) \end{array} \right)$$

Les 18 premières propriétés raffinents C_3^1 à C_3^{11} et C_3^{14} à C_3^{20}.

– – **Les moteurs s'arrêtent fatalement**

$C_4^{21} \quad \Box (\ EtMotv_4 \in \{ \ monte, descend \ \} \Rightarrow \Diamond (\ ItMotv_4 = oui \ ) \ )$

$C_4^{22} \quad \Box (\ EtMoth_4 \in \{ \ avance, recule \ \} \Rightarrow \Diamond (\ ItMoth_4 = oui \ ) \ )$

$C_4^{23} \quad \Box (\ EtMotv_4 \in \{ \ ouverture, fermeture \ \} \Rightarrow \Diamond (\ ItMotp_4 = oui \ ) \ )$

– – **Quand le matériel reçoit un ordre, il est éxécuté dans l'instant qui suit compte tenu du délai de réponse du moteur**

$\forall s \in \{ \ monte, descend \ \}$

$$C_4^{24} \quad \Box \left(\begin{array}{l} DmMotv_4 = non \wedge \bigcirc (\ DmMotv_4 = oui \wedge SnMotv_4 = s \ ) \\ \Rightarrow \bigcirc \bigcirc (\ DmMotv_4 = non \wedge EtMotv_4 = s \ ) \end{array} \right)$$

$\forall s \in \{ \ avance, recule \ \}$

$$C_4^{25} \quad \Box \left(\begin{array}{l} DmMoth_4 = non \wedge \bigcirc (\ DmMoth_4 = oui \wedge SnMoth_4 = s \ ) \\ \Rightarrow \bigcirc \bigcirc (\ DmMoth_4 = non \wedge EtMoth_4 = s \ ) \end{array} \right)$$

$\forall s \in \{ \ ouverture, fermeture \ \}$

$$C_4^{26} \quad \Box \left(\begin{array}{l} SnMotp_4 = non \wedge \bigcirc (\ SnMotp_4 = oui \wedge SnMotp_4 = s \ ) \\ \Rightarrow \bigcirc \bigcirc (\ SnMotp_4 = non \wedge EtMotp_4 = s \ ) \end{array} \right)$$

– – **Lorsque l'état logiciel d'un moteur change, l'ordre de mouvement correspondant est immédiatement donné**

$$C_4^{27} \quad \Box \left(\begin{array}{l} MvDt_4 = enbas \wedge \bigcirc (\ MvDt_4 = monte \ ) \\ \Rightarrow \bigcirc (\ DmMotv_4 = oui \wedge SnMotv_4 = monte \ ) \end{array} \right)$$

$$C_4^{28} \quad \Box \left(\begin{array}{l} MvDt_4 = enhaut \wedge \bigcirc (\ MvDt_4 = descend \ ) \\ \Rightarrow \bigcirc (\ DmMotv_4 = oui \wedge SnMotv_4 = descend \ ) \end{array} \right)$$

$$C_4^{29} \quad \Box \left(\begin{array}{l} MhDt_4 = enavant \wedge \bigcirc (\ MhDt_4 = recule \ ) \\ \Rightarrow \bigcirc (\ DmMoth_4 = oui \wedge SnMoth_4 = recule \ ) \end{array} \right)$$

$$C_4^{30} \quad \Box \left(\begin{array}{l} MhDt_4 = enarrière \wedge \bigcirc (\ MhDt_4 = avance \ ) \\ \Rightarrow \bigcirc (\ DmMoth_4 = oui \wedge SnMoth_4 = avance \ ) \end{array} \right)$$

$$C_4^{31} \quad \Box \left(\begin{array}{l} Mp_4 = fermée \wedge \bigcirc (\ Mp_4 = ouverture \ ) \\ \Rightarrow \bigcirc (\ SnMotp_4 = oui \wedge SnMotp_4 = ouverture \ ) \end{array} \right)$$

$$C_4^{32} \quad \Box \left(\begin{array}{l} Mp_4 = ouverte \wedge \bigcirc (\ Mp_4 = fermeture \ ) \\ \Rightarrow \bigcirc (\ SnMotp_4 = oui \wedge SnMotp_4 = fermeture \ ) \end{array} \right)$$

– – **Lorsque le logiciel reçoit une interruption, l'état des dispositifs est modifié compte tenu là aussi d'un délai de réponse d logiciel vis à vis de la génération de l'interruption**

$$C_4^{33} \quad \Box \left(\begin{array}{l} ItMotv_4 = non \bigcirc (\ ItMotv_4 = oui \wedge MvDt_4 = monte \ ) \\ \Rightarrow \bigcirc \bigcirc (\ ItMotv_4 = non \wedge MvDt_4 = enhaut \ ) \end{array} \right)$$

$$C_4^{34} \quad \Box \left(\begin{array}{l} ItMotv_4 = non \bigcirc (\ ItMotv_4 = oui \wedge MvDt_4 = descend \ ) \\ \Rightarrow \bigcirc \bigcirc (\ ItMotv_4 = non \wedge MvDt_4 = enbas \ ) \end{array} \right)$$

$$C_4^{35} \quad \Box \left(\begin{array}{l} ItMoth_4 = non \bigcirc (\ ItMoth_4 = oui \wedge MhDt_4 = avance \ ) \\ \Rightarrow \bigcirc \bigcirc (\ ItMoth_4 = non \wedge MhDt_4 = enavant \ ) \end{array} \right)$$

$$C_4^{36} \quad \Box \left(\begin{array}{l} ItMoth_4 = non \bigcirc (\ ItMoth_4 = oui \wedge MhDt_4 = recule \ ) \\ \Rightarrow \bigcirc \bigcirc (\ ItMoth_4 = non \wedge MhDt_4 = enarrière \ ) \end{array} \right)$$

$$C_4^{37} \quad \Box \left(\begin{array}{l} ItMotp_4 = non \bigcirc (\ ItMotp_4 = oui \wedge Mp_4 = ouverture \ ) \\ \Rightarrow \bigcirc \bigcirc (\ ItMotp_4 = non \wedge Mp_4 = ouverte \ ) \end{array} \right)$$

$$C_4^{38} \quad \Box \left(\begin{array}{l} ItMotp_4 = non \bigcirc (\ ItMotp_4 = oui \wedge Mp_4 = fermeture \ ) \\ \Rightarrow \bigcirc \bigcirc (\ ItMotp_4 = non \wedge Mp_4 = fermée \ ) \end{array} \right)$$

– – **Le logiciel croit que les moteurs font mouvement tant qu'il n'a pas reçu d'interruption**

$$C_4^{39} \quad \Box \left(\begin{array}{l} MvDt_4 \in \{ \ enhaut, enbas \ \} \wedge \bigcirc (\ MvDt_4 \in \{ \ monte, descend \ \} \ ) \\ \Rightarrow \ (\ MvDt_4 \in \{ \ monte, descend \ \} \ ) \ \mathcal{U} \ (\ ItMotv_4 = oui \ ) \end{array} \right)$$

$$C_4^{40} \quad \Box \left(\begin{array}{l} MhDt_4 \in \{ \ enavant, enarrière \ \} \wedge \bigcirc (\ MhDt_4 \in \{ \ avance, recule \ \} \ ) \\ \Rightarrow \ (\ MhDt_4 \in \{ \ avance, recule \ \} \ ) \ \mathcal{U} \ (\ ItMoth_4 = oui \ ) \end{array} \right)$$

$$C_4^{41} \quad \Box \left(\begin{array}{l} Mp_4 \in \{ \ ouverte, fermée \ \} \wedge \bigcirc (\ Mp_4 \in \{ \ ouverture, fermeture \ \} \ ) \\ \Rightarrow \ (\ Mp_4 \in \{ \ ouverture, fermeture \ \} \ ) \ \mathcal{U} \ (\ ItMotp_4 = oui \ ) \end{array} \right)$$

Spécification opérationelle

Huit nouveaux événements sont introduits dans ce niveau de raffinement en plus des seize déjà existants :

- $Chargt_4$, $DéChargt_4$, $Evac_4$, $ArPièce_4$, $ArDescente_4$, $ArMontée_4$, $Montée_4$, $Descente_4$, $ArRecule_4$, $ArAvance_4$, $AvanceEnHaut_4$, $AvanceEnBas_4$, $ReculeEnHaut_4$, $ReculeEnBas_4$, $ArOuverture_4$, $ArFermeture_4$ sont les raffinements des événements déjà existants,

- $DemMotv_4$, $DemMoth_4$, $DemMotp_4$ correspondent à l'événement de de démarrage du moteur vertical, horizontal et de la pince,
- $ItArPièce_4$ et $ItDpPièce_4$ sont les événements représentant respectivement l'arrivée d'une pièce sur Da et le départ de la pièce se trouvant sur De,
- $ItArMotv_4$, $ItArMoth_4$, $ItArMotp_4$ sont les événements générant les interruptions indiquant au logiciel que les moteurs se sont arrêtés.

$$\begin{aligned}\text{Initialisation} \triangleq\ & De_4 := libre \parallel Da_4 := libre \parallel MvDt_4 := enbas \parallel \\ & MhDt_4 = enavant \parallel Mp_4 = ouverte \parallel \\ & EtMotv_4 = arrêt \parallel EtMoth_4 = arrêt \parallel EtMotp_4 = arrêt \parallel \\ & DemMotv_4 = non \parallel DemMoth_4 = non \parallel DemMotp_4 = non \parallel \\ & ItArPièce_4 = non \parallel ItDpPièce_4 = non \parallel ItArMotv_4 = non \parallel \\ & ItArMoth_4 = non \parallel ItArMotp_4\end{aligned}$$

Evénements logiciels

Les événements suivant ne correspondent qu'à la partie logiciel de l'automatisme.

$$\begin{aligned}Chargt_4 \triangleq\ & \underline{\text{SELECT}} \\ & \quad Mp_4 = ouverte \wedge Da_4 = occupé \wedge \\ & \quad MvDt_4 = enbas \wedge MhDt_4 = enavant \\ & \underline{\text{THEN}} \\ & \quad Mp_4, Da_4, SnMotp_4, SnMotp_4 := seferme, libre, oui, fermeture \\ & \underline{\text{END}}\end{aligned}$$

$$\begin{aligned}DéChargt_4 \triangleq\ & \underline{\text{SELECT}} \\ & \quad Mp_4 = fermée \wedge De_4 = libre \wedge \\ & \quad MvDt_4 = enhaut \wedge MhDt_4 = enavant \\ & \underline{\text{THEN}} \\ & \quad Mp_4, De_4, SnMotp_4, SnMotp_4 := s'ouvre, occupé, oui, ouverture \\ & \underline{\text{END}}\end{aligned}$$

$$\begin{aligned}Evac_4 \triangleq\ & \underline{\text{SELECT}} \\ & \quad De_4 = occupé \wedge \mathbf{ItDpPièce_4 = oui} \\ & \underline{\text{THEN}} \\ & \quad De_4, ItDpPièce_4 := libre, non \\ & \underline{\text{END}}\end{aligned}$$

$$\begin{aligned}ArPièce_4 \triangleq\ & \underline{\text{SELECT}} \\ & \quad Da_4 = libre \wedge \mathbf{ItArPièce_4 = oui} \\ & \underline{\text{THEN}} \\ & \quad Da_4, ItArPièce_4 := occupé, non \\ & \underline{\text{END}}\end{aligned}$$

$ArMontée_4 \triangleq$ SELECT
$MvDt_4 = monte \wedge Mp_4 = fermée \wedge$
$MhDt_4 = enarrière \wedge \mathbf{ItArMotv_4 = oui}$
THEN
$MvDt_4, ItArMotv_4 := enhaut, non$
END

$ArDescente_4 \triangleq$ SELECT
$MvDt_4 = descend \wedge Mp_4 = ouverte \wedge$
$MhDt_4 = enarrière \wedge \mathbf{ItMotv_4 = oui}$
THEN
$MvDt_4, ItMotv_4 := enbas, non$
END

$Montée_4 \triangleq$ SELECT
$Mp_4 = fermée \wedge MvDt_4 = enbas \wedge MhDt_4 \in \{\ recule, enarrière\ \}$
THEN
$MvDt_4, DmMotv_4, SnMotv_4 := monte, oui, monte$
END

$Descente_4 \triangleq$ SELECT
$Mp_4 = ouverte \wedge MvDt_4 = enhaut \wedge MhDt_4 = enarrière$
THEN
$MvDt_4, DmMotv_4, SnMotv_4 := descend, oui, descend$
END

$ArRecule_4 \triangleq$ SELECT
$MhDt_4 = recule \wedge \left(\begin{array}{l} (\ MvDt_4 = enhaut \wedge Mp_4 = ouverte\ ) \vee \\ (\ MvDt_4 \in \{\ enbas, monte\ \} \wedge Mp_4 = fermée\ ) \end{array} \right) \wedge$
$ItMoth_4 = oui$
THEN
$MhDt_4, ItMoth_4 := enarrière, non$
END

$ArAvance_4 \triangleq$ SELECT
$MhDt_4 = avance \wedge \left(\begin{array}{l} (\ MvDt_4 = enhaut \wedge Mp_4 = fermée\ ) \vee \\ (\ MvDt_4 = enbas \wedge Mp_4 = ouverte\ ) \end{array} \right) \wedge$
$ItMoth_4 = oui$
THEN
$MhDt_4, ItMoth_4 := enavant, non$
END

$AvanceEnHaut_4 \triangleq$ SELECT
$MhDt_4 = enarriere \wedge MvDt_4 = enhaut \wedge Mp_4 = fermée$
THEN
$MhDt_4, DmMoth_4, SnMoth_4 := avance, oui, avance$
END

$AvanceEnBas_4 \triangleq$ SELECT
$MhDt_4 = enarriere \wedge MvDt_4 = enbas \wedge Mp_4 = ouverte$
THEN
$MhDt_4, DmMoth_4, SnMoth_4 := avance, oui, avance$
END

$ReculeEnHaut_4 \triangleq$ SELECT
$MhDt_4 = enavant \wedge MvDt_4 = enhaut \wedge Mp_4 = ouverte$
THEN
$MhDt_4, DmMoth_4, SnMoth_4 := recule, oui, recule$
END

$ReculeEnBas_4 \triangleq$ SELECT
$MhDt_4 = enavant \wedge MvDt_4 = enbas \wedge Mp_4 = fermée$
THEN
$MhDt_4, DmMoth_4, SnMoth_4 := recule, oui, recule$
END

$ArOuverture_4 \triangleq$ SELECT
$Mp_4 = s'ouvre \wedge MhDt_4 = enavant \wedge$
$MvDt_4 = enhaut \wedge \mathbf{ItMotp_4 = oui}$
THEN
$Mp_4, ItMotp_4 := ouverte, non$
END

$ArFermeture_4 \triangleq$ SELECT
$Mp_4 = seferme \wedge MhDt_4 = enavant \wedge$
$MvDt_4 = enbas \wedge \mathbf{ItMotp_4 = oui}$
THEN
$Mp_4, ItMotp_4 := fermée, non$
END

$DemMotv_4 \triangleq$ SELECT
$EtMotv_4 = arrêt \wedge DemMotv_4 = oui$
THEN
$DemMotv_4, EtMotv_4 := non, SnMotv_4$
END

$DemMoth_4 \triangleq$ SELECT
$EtMoth_4 = arrêt \wedge DemMoth_4 = oui$
THEN
$DemMoth_4, EtMoth_4 := non, SnMoth_4$
END

$DemMotp_4 \triangleq$ SELECT
$EtMotp_4 = arrêt \wedge DemMotp_4 = oui$
THEN
$DemMotp_4, EtMotp_4 := non, SnMotp_4$
END

$ItArPièce_4 \triangleq$ SELECT
$Da_4 = libre \wedge ItArPièce_4 = non$
THEN
$ItArPièce_4 = oui$
END

$ItDpPièce_4 \triangleq$ SELECT
$De_4 = occupé \wedge ItDpPièce_4 = non$
THEN
$ItDpPièce_4 = oui$
END

$ItArMotv_4 \triangleq$ SELECT
$EtMotv_4 \neq arrêt$
THEN
$EtMotv_4, ItArMotv_4 := arrêt, oui$
END

$ItArMoth_4 \triangleq$ SELECT
$EtMoth_4 \neq arrêt$
THEN
$EtMoth_4, ItArMoth_4 := arrêt, oui$
END

$ItArMotp_4 \triangleq$ SELECT
$EtMotp_4 \neq arrêt$
THEN
$EtMotp_4, ItArMotp_4 := arrêt, oui$
END

Bibliographie

[Abr96] J.R. Abrial. *The* B *Book*. Cambridge University Press, 1996.

[Abr97] J.R. Abrial. Constructions d'Automatismes industriels avec B. Congrès AFADL - ONERA-CERT, May 1997.

[Arn92] A. Arnold. *Systèmes de Transitions Finis et Sémantique Des Processus Communicants*. 1992.

[Bos95] J.C. Bossy. *Le grafcet. Sa pratique et ses applications*. 1995.

Annexe B

L'exemple d'un ascenseur

B.1 Spécification formelle

B.1.1 But

Nous voulons modéliser l'exemple d'un ascenseur. Nous nous concentrerons sur la stratégie qui garanti que tous les appels des usagers sont satisfaits. Le principe est celui du déplacement séquentiel de la cabine qui ne change de sens que lorsqu'il n'y a plus d'appel dans sa direction de déplacement courante. Ici nous ne gérerons pas les panneaux lumineux et les portes.

Les trois propriétés a satisfaire sont :

1. Il faut satisfaire toutes les requêtes.
2. Le déplacement est réalisé séquentiellement.
3. La cabine doit s'arrêtée au dernier étage déservi s'il n'y a plus d'appel.

B.1.2 Données

Nous définisions les variables suivantes :

- $Etage\_Courant_0$: numéro de l'étage où est située la cabine.
- $Direction_0$: direction de la cabine.
- $Appel\_Etage_0$: ensemble des appels provenant des étages.
- $Appel\_Cabine_0$: ensemble des appels provenant de la cabine.

Nous définissons aussi quelques ensembles qui permettrons de typer les variables ci-dessus :

- ETAGES$= 0..(\text{NBREET} - 1)$ où NBREET est un paramètre du système et représente le nombre d'étages pouvant êtres déservis par l'ascenseur.
- ETATAP$= \{aucun, appel\}$. Cet ensemble repésente l'état d'un appel.
- DIR$= \{bas, haut\}$ représente les directions pouvant êtres prises par l'ascenseur.
- APPELET$=$ ETAGES $\times$ DIR $- \{(0, bas), (\text{NBREET} - 1, haut)\}$ est l'ensemble des possibilités d'appel depuis les étages.

B.1.3 Invariant

– – **Type des variables**
$Etage\_Courant_0 \in \text{ETAGES} \wedge$
$Direction_0 \in \text{DIR} \cup \{aucune\} \wedge$
$Appel\_Etage_0 \subseteq \text{APPELET} \rightarrow \text{ETATAP} \wedge$
$Appel\_Cabine_0 \subseteq \text{ETAGES} \rightarrow \text{ETATAP} \wedge$

Afin de simplifier la lecture des propriétés suivantes, le prédicat $Pas\_d'Appel_0$ est introduit.

$$Pas\_d'Appel_0 \triangleq \forall n \in \text{ETAGES}, \forall a \in \text{APPELET} \left(\begin{array}{l} ((n \mapsto appel) \in Appel\_Cabine_0) \vee \\ ((a \mapsto appel) \in Appel\_Etage_0) \end{array} \right)$$

– – **Si la cabine est arrêtée alors il n'y a pas d'appel aux étages**
$(Direction_0 = aucune) \Rightarrow Pas\_d'Appel_0 \wedge$

– – **Si il n'y a pas d'appel aux étages, la cabine est arrêtée**
$Pas\_d'Appel_0 \Rightarrow (Direction_0 = aucune)$

B.1.4 Propriétés temporelles

Certains prédicats vont être introduis afin que les expression temporelles puisse être facilement lues.

Il y a au moins un appel :

$$Un\_Appel_0 \triangleq \exists n \in \text{ETAGES}, \exists a \in \text{APPELET} \quad n \neq Etage\_Courant_0 \wedge \left(\begin{array}{l} ((n \mapsto appel) \in Appel\_Cabine_0) \vee \\ ((a \mapsto appel) \in Appel\_Etage_0) \end{array} \right)$$

Il y a au moins un appel dans la cabine pour monter :

$$Un\_Appel\_Cab\_Pour\_Monter_0 \triangleq \exists n \in \text{ETAGES} \quad n > Etage\_Courant_0 \wedge \quad (n \mapsto appel) \in Appel\_Cabine_0$$

Il y a au moins un appel dans la cabine pour descendre :

$$Un\_Appel\_Cab\_Pour\_Descendre_0 \triangleq \exists n \in \text{ETAGES} \quad n < Etage\_Courant_0 \wedge \quad (n \mapsto appel) \in Appel\_Cabine_0$$

Il y a au moins un appel à un étage pour monter :

$$Un\_Appel\_Etg\_Pour\_Monter_0 \triangleq \forall d \in \text{DIR}, \exists n \in \text{ETAGES} \quad n > Etage\_Courant_0 \wedge \quad ((n, d) \mapsto appel) \in Appel\_Etage_0$$

Il y a au moins un appel à un étage pour descendre :

$$Un\_Appel\_Etg\_Pour\_Descendre_0 \triangleq \forall d \in \text{DIR}, \exists n \in \text{ETAGES}\ n < Etage\_Courant_0 \wedge ((n,d) \mapsto appel) \in Appel\_Etage_0$$

Il n'y a pas d'appel à l'étage courant :

$$Pas\_Appel\_Etg\_Courant_0 \triangleq \forall d \in \text{DIR} \left(\begin{array}{l} (Etage\_Courant_0 \mapsto appel) \notin Appel\_Cabine_0 \wedge \\ ((Etage\_Courant_0, d) \mapsto appel) \notin Appel\_Etage_0 \end{array} \right)$$

– – Si la cabine est arrêtée et s'il arrive des appels, alors l'ascenseur ne peut que redémarrer

$${}_1S_0^1 \quad \Box \left(\begin{array}{l} Direction_0 = aucune \wedge \bigcirc (\ Un\_Appel_0 \ ) \\ \Rightarrow \Diamond (\ Direction_0 \in \{ \ haut, bas \ \} \ ) \end{array} \right)$$

– – S'il n'y a pas d'appel alors la cabine ne peut que s'arrêter jusqu'à ce qu'il y en ait

$$\forall n \in \text{ETAGES}$$
$${}_3S_0^2 \quad \Box \left(\begin{array}{l} Pas\_d'Appel_0 \wedge Etage\_Courant_0 = n \\ \Rightarrow \left(\begin{array}{l} Direction_0 = aucune \wedge \\ Etage\_Courant_0 = n \end{array} \right) \mathcal{U} (\ Un\_Appel_0 \ ) \end{array} \right)$$

– – S'il y a un appel alors la cabine déservira fatalement l'étage concerné

$$\forall n \in \text{ETAGES}, \forall d \in \text{DIR}$$
$${}_1S_0^3 \quad \Box \left(\begin{array}{l} ((n,d) \mapsto appel) \in Appel\_Etage_0 \\ \Rightarrow \Diamond (\ Etage\_Courant_0 = n \wedge Direction_0 = d \ ) \end{array} \right)$$

$$\forall n \in \text{ETAGES}$$
$${}_1S_0^4 \quad \Box \left(\begin{array}{l} (n \mapsto appel) \in Appel\_Cabine_0 \\ \Rightarrow \Diamond (\ Etage\_Courant_0 = n \ ) \end{array} \right)$$

– – **Deux appels consécutifs dans la même direction sont satisfait dans le même ordre**

$$
{}_2S_0^5 \quad \Box \quad
\begin{array}{l}
\forall n \in \text{ETAGES}, \forall n_1 \in \text{ETAGES}, \forall n_2 \in \text{ETAGES} \\
\left(\begin{array}{l}
\left(\begin{array}{l} \left(\begin{array}{l} ((n_1, bas) \mapsto appel) \in Appel\_Etage_0 \end{array} \right) \vee \\ \left(\begin{array}{l} (n_1 \mapsto appel) \in Appel\_Cabine_0 \end{array} \right) \end{array} \right) \wedge \\
\left(\begin{array}{l} \left(\begin{array}{l} ((n_2, bas) \mapsto appel) \in Appel\_Etage_0 \end{array} \right) \vee \\ \left(\begin{array}{l} (n_2 \mapsto appel) \in Appel\_Cabine_0 \end{array} \right) \end{array} \right) \wedge \\
n_1 > n_2 \wedge \\
\left(\begin{array}{l} \left(\begin{array}{l} n_2 < n \wedge n < n_1 \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} \left(\begin{array}{l} ((n, bas) \mapsto aucun) \in Appel\_Etage_0 \end{array} \right) \wedge \\ \left(\begin{array}{l} (n \mapsto aucun) \in Appel\_Cabine_0 \end{array} \right) \end{array} \right) \end{array} \right) \\
\Rightarrow \Diamond \left(\begin{array}{l} Etage\_Courant_0 = n_1 \Rightarrow \Diamond \left(\begin{array}{l} Etage\_Courant_0 = n_2 \end{array} \right) \end{array} \right)
\end{array} \right)
\end{array}
$$

$$
{}_2S_0^6 \quad \Box \quad
\begin{array}{l}
\forall n \in \text{ETAGES}, \forall n_1 \in \text{ETAGES}, \forall n_2 \in \text{ETAGES} \\
\left(\begin{array}{l}
\left(\begin{array}{l} \left(\begin{array}{l} ((n_1, haut) \mapsto appel) \in Appel\_Etage_0 \end{array} \right) \vee \\ \left(\begin{array}{l} (n_1 \mapsto appel) \in Appel\_Cabine_0 \end{array} \right) \end{array} \right) \wedge \\
\left(\begin{array}{l} \left(\begin{array}{l} ((n_2, haut) \mapsto appel) \in Appel\_Etage_0 \end{array} \right) \vee \\ \left(\begin{array}{l} (n_2 \mapsto appel) \in Appel\_Cabine_0 \end{array} \right) \end{array} \right) \wedge \\
n_1 < n_2 \wedge \\
\left(\begin{array}{l} \left(\begin{array}{l} n_1 < n \wedge n < n_2 \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} \left(\begin{array}{l} ((n, haut) \mapsto aucun) \in Appel\_Etage_0 \end{array} \right) \wedge \\ \left(\begin{array}{l} (n \mapsto aucun) \in Appel\_Cabine_0 \end{array} \right) \end{array} \right) \end{array} \right) \\
\Rightarrow \Diamond \left(\begin{array}{l} Etage\_Courant_0 = n_1 \Rightarrow \Diamond \left(\begin{array}{l} Etage\_Courant_0 = n_2 \end{array} \right) \end{array} \right)
\end{array} \right)
\end{array}
$$

B.1.5 Evénements

A l'initialisation du système, l'ascenseur se trouve immobilisé au premier étage. D'autre part, il n'y a pas appel à partir de la cabine ou d'un étage.

Initialisation $\triangleq$

$Etage\_Courant_0 := 0 \ \|$
$Direction_0 := aucune \ \|$
ANY n, d
WHERE
$n \in \text{ETAGES} \wedge d \in \text{DIR}$
THEN
$Appel\_Cabine_0 := \{ \ (i \mapsto aucun) \ \} \ \|$
$Appel\_Etage_0 := \{ \ ((i, d) \mapsto aucun) \ \}$
END

L'évévenement suivant représente l'arrivée d'un appel en provenance d'un étage. Il y a enregistrement de la demande uniquement si personne n'avait encore demandé l'ascenseur pour l'étage concerné. $Appel\_Etage_0$ est paramètré par le numéro de l'étage où l'appel a été réalisé (NVE) et la direction désirée par l'usagé (NVD).

$$
Appel\_Etage_0 \left(\begin{array}{l} \text{NVE} \in \text{ETAGES}, \\ \text{NVD} \in \text{DIR} \end{array} \right) \triangleq
$$

SELECT
$((\text{NVE}, \text{NVD}) \mapsto appel) \notin Appel\_Etage_0$
THEN
$Appel\_Etage_0 :=$
$\left(\begin{array}{l} Appel\_Etage_0 - \\ \{ \ (\text{NVE}, \text{NVD}) \mapsto aucun \ \} \cup \\ \{ \ (\text{NVE}, \text{NVD}) \mapsto appel \ \} \end{array} \right)$
END

Tout comme l'événement précédent, $Appel\_Cabine_0$ permet d'enregistrer les appels. Seuls ceux réalisés à partir de la cabine sont pris en compte ici. Cet événement est lui aussi paramètré par le numéro de l'étage où l'usagé veut se rendre.

$$
\begin{array}{l}
Appel\_Cabine_0 \ (\ \text{NvE} \in \text{Etages}\ ) \ \triangleq \\
\quad \underline{\text{SELECT}} \\
\quad\quad (\text{NvE} \mapsto appel) \notin Appel\_Cabine_0 \\
\quad \underline{\text{THEN}} \\
\quad\quad Appel\_Cabine_0 := \\
\quad\quad\quad \left(\begin{array}{l} Appel\_Cabine_0 - \\ \{\ \text{NvE} \mapsto aucun\ \} \cup \\ \{\ \text{NvE} \mapsto appel\ \} \end{array} \right) \\
\quad \underline{\text{END}}
\end{array}
$$

$Suppression\_Appel_0$ n'est pas un événement en soit mais plutôt un prédicat réutilisable par d'autres événements. Le paramètre NvD correspond à la direction pour laquelle les appels doivent être supprimés.

$$
\begin{array}{l}
Suppression\_Appel_0 \ (\ \text{NvD} \in \text{Dir}\ ) \ \triangleq \\
\quad \underline{\text{BEGIN}} \\
\quad\quad Appel\_Cabine_0 := \\
\quad\quad\quad \left(\begin{array}{l} Appel\_Cabine_0 - \\ \{\ Etage\_Courant_0 \mapsto appel\ \} \cup \\ \{\ Etage\_Courant_0 \mapsto aucun\ \} \end{array} \right) \quad \| \\
\quad\quad Appel\_Etage_0 := \\
\quad\quad\quad \left(\begin{array}{l} Appel\_Etage_0 - \\ \{\ (Etage\_Courant_0, \text{NvD}) \mapsto appel\ \} \cup \\ \{\ (Etage\_Courant_0, \text{NvD}) \mapsto aucun\ \} \end{array} \right) \\
\quad \underline{\text{END}}
\end{array}
$$

L'événement suivant représente le mouvement ascendant de l'ascenseur pour déservir un étage supérieur. Le cahier des charges obligant le traitement séquentiel des étages, l'ascenseur s'arrête au premier étage demandeur dans le sens de la montée. Une fois l'ascenseur arrivé, toutes les demandes concernant l'étage courant sont retirées.

$Monte\_même\_Dir_0 \triangleq$

SELECT

$\left(\begin{array}{l} Un\_Appel\_Cab\_Pour\_Monter_0 \vee \\ Un\_Appel\_Etg\_Pour\_Monter_0 \end{array} \right) \wedge$

$Direction_0 = haut \wedge$

$Pas\_Appel\_Etg\_Courant_0$

THEN

ANY n, n_1

WHERE

$n \in \text{ETAGES} \wedge n_1 \in \text{ETAGES} \wedge$

$n > Etage\_Courant_0 \wedge$

$\left(\begin{array}{l} (n \mapsto appel) \in Appel\_Cabine_0 \vee \\ ((n, haut) \mapsto appel) \in Appel\_Etage_0 \end{array} \right) \wedge$

$n_1 \geq Etage\_Courant_0 \wedge n > n_1 \wedge$

$(n_1 \mapsto appel) \in Appel\_Cabine_0 \wedge$

$((n_1, haut) \mapsto appel) \in Appel\_Etage_0$

THEN

$Etage\_Courant_0 := n$

END $\|$

$Suppression\_Appel_0 \ (\ haut \ )$

END

$Descend\_même\_Dir_0$ est un événement similaire au précédent. Il représente le mouvement descendant de la cabine. Cet événement possèdent les mêmes effets que $Monte\_même\_Dir_0$, c'est à dire qu'il change la cabine d'étage.

$Descend\_même\_Dir_0 \triangleq$

SELECT

$\left(\begin{array}{l} Un\_Appel\_Cab\_Pour\_Descendre_0 \vee \\ Un\_Appel\_Etg\_Pour\_Descendre_0 \end{array} \right) \wedge$

$Direction_0 = bas \wedge$

$Pas\_Appel\_Etg\_Courant_0$

THEN

ANY n, n_1

WHERE

$n \in \text{ETAGES} \wedge n_1 \in \text{ETAGES} \wedge$

$n < Etage\_Courant_0 \wedge$

$\left(\begin{array}{l} (n \mapsto appel) \in Appel\_Cabine_0 \vee \\ ((n, bas) \mapsto appel) \in Appel\_Etage_0 \end{array} \right) \wedge$

$n_1 \leq Etage\_Courant_0 \wedge n < n_1 \wedge$

$(n_1 \mapsto appel) \in Appel\_Cabine_0 \wedge$

$((n_1, bas) \mapsto appel) \in Appel\_Etage_0$

THEN

$Etage\_Courant_0 := n$

END $\|$

$Suppression\_Appel_0 \ (\ bas \ )$

END

L'événement suivant permet de gérer les changements de direction de l'ascenseur. Ils se produisent lorsque la cabine arrive au dernier ou au premier étage sans s'arrêter ou encore lorsqu'il n'y a plus d'appel dans la direction courante de l'ascenseur. Si un de ces cas se produit, l'ascenseur change de direction.

$$
\begin{array}{l}
Change\_Dir_0 \triangleq \underline{\text{SELECT}} \\
\quad \left(\begin{array}{l} Direction_0 = haut \wedge \\ Etage\_Courant_0 = \text{NBREET} - 1 \end{array} \right) \vee \\
\quad \left(\begin{array}{l} Direction_0 = bas \wedge \\ Etage\_Courant_0 = 0 \end{array} \right) \vee \\
\quad \left(\begin{array}{l} Direction_0 = haut \wedge \\ \neg Un\_Appel\_Etg\_Pour\_Monter_0 \wedge \\ \neg Un\_Appel\_Cab\_Pour\_Monter_0 \wedge \\ \left(\begin{array}{l} Un\_Appel\_Etg\_Pour\_Descendre_0 \vee \\ Un\_Appel\_Cab\_Pour\_Descendre_0 \end{array} \right) \end{array} \right) \vee \\
\quad \left(\begin{array}{l} Direction_0 = bas \wedge \\ \neg Un\_Appel\_Etg\_Pour\_Descendre_0 \wedge \\ \neg Un\_Appel\_Cab\_Pour\_Descendre_0 \wedge \\ \left(\begin{array}{l} Un\_Appel\_Etg\_Pour\_Monter_0 \vee \\ Un\_Appel\_Cab\_Pour\_Monter_0 \end{array} \right) \end{array} \right) \\
\quad \underline{\text{THEN}} \\
\quad\quad \underline{\text{IF}} \;\; Direction_0 = bas \\
\quad\quad \underline{\text{THEN}} \\
\quad\quad\quad Direction_0 := haut \\
\quad\quad \underline{\text{ELSE}} \\
\quad\quad\quad Direction_0 := bas \\
\quad\quad \underline{\text{END}} \\
\quad \underline{\text{END}}
\end{array}
$$

La cabine de l'ascenseur ne s'arrête que s'il n'y plus d'appel.

$$
\begin{array}{l}
S'arrête_0 \triangleq \underline{\text{SELECT}} \\
\quad\quad Pas\_d'Appel_0 \\
\quad \underline{\text{THEN}} \\
\quad\quad Direction_0 := aucune \\
\quad \underline{\text{END}}
\end{array}
$$

L'événement suivant permet à l'ascenseur de repartir dans la direction d'un appel. Si le système est confronté à deux appels provenant de directions différentes, il effectura un choix non déterministe pour choisir le sens de son mouvement.

```
Repart_0 ≙ SELECT
             Direction_0 = aucune∧
             Un_Appel_0
           THEN
             IF Un_Appel_Etg_Pour_Monter_0∨
                Un_Appel_Cab_Pour_Monter_0
             THEN
               IF Un_Appel_Etg_Pour_Descendre_0∨
                  Un_Appel_Cab_Pour_Descendre_0
               THEN
                 CHOICE
                   Direction_0 := haut
                 OR
                   Direction_0 := bas
                 END
               ELSE
                 Direction_0 := haut
               END
             ELSE
               Direction_0 := bas
             END
           END
```

Annexe C

Résumé de l'exemple de la Carte à Circuits Intégrés

Cette annexe résume les notions et les propriétés décrites dans [JLM+97] et nécessaires pour l'élaboration de ce document.

C.1 Présentation du problème

C.1.1 Description informelle

Le problème de la Carte à Circuit Intrégrés est issu de la demande d'une entreprise afin de pouvoir vérifier que ses lecteurs de cartes soient conformes à la norme européenne EN 27816 d'avril 1992[195]. Ce dernier permet d'avoir un standard de communication entre une carte à circuit intégrés et un lecteur de carte. Le principe de base peut se divisé en cinq phases :

- connexion et mise sous tension des contacts par le *lecteur*,
- remise à zéro de la *carte*,
- réponse de la carte à la remise à zéro,
- échange ultérieur d'informations entre la *carte* et le *lecteur*,
- séquence de mise hors tension des contatcs du *lecteur*.

Ici, nous ne nous interesserons qu'à l'échange d'informations entre les deux entités. Les mises sous et hors tensions sont réalisées convenablement. Le protocole utilisé pour le transfert des informations est T=1, c'est à dire un protocole permettant l'échange de messages en mode asynchrone "halt duplex".

C.1.2 Le procotole T=1

Le protocole T=1 défini un environnement de transmission de messages par blocs entre un *lecteur* et une *carte à puce*. Les différents messages ou trames transitant entre ces deux derniers sont :

- une trame d'information I-Trame, représentée par $I(identificateur, \hat{chaînage})$.Ce message est utilisé pour transmettre des informations. Elle est composée d'un identificateur et une information de chaînage indiquant si la trame correspond au dernier bloc de l'information ou non;

- une trame d'acquittement R-Trame, représentée par $R(identificateur)$. Elle permet d'indiquer qu'un bloc chaîné à été correctement reçu. L'*identificateur* correspond à celui du message d'information acquitté;
- une trame de supervision S-Trame, notée $S(typerequête)$. Les différentes commandes utilisables sont :
  - $WtxReq$ et $WtxRep$ sont la demande et la réponse à un ajustement des délais de traitement,
  - $IfsReq$ et $IfsRep$ sont la demande et la réponse à un changement de la taille des blocs de données,
  - $ResynchReq$ et $ResynchRep$ sont la demande et la réponse à une resynchronisation de la transmission,
  - $AbortReq$ et $AbortRep$ sont la demande et le réponse à une annulation d'un eéhange chaîné.

Il existe trois possibilités d'erreur dans ce protocole : l'erreur de transmission, perte de la synchronisation et la perte totale d'un message.

C.2 Spécification descriptive

C.2.1 Spécification formelle

But

Dans ce premier niveau d'analyse, nous nous interesserons à l'échange de messages sans toutefois prendre en compte le découpage en bloc de ces derniers ainsi que les divers possibilités d'erreur pouvant êtres rencontrées.

Seules les règles 1 et 2.1 sont concernées par ce niveau de raffinement.

Données

Les ensembles suivants sont définis afin de simplifier la lecture des propriétés :
$\textsc{Ident} = \{0, 1\}$
$\textsc{Dialogue} = \{EnCours, Interrompu\}$
$\textsc{Trame} = \textsc{Ident}$

Les variables suivantes sont nécessaires :

- TC_0: Dernière trame émise par la *carte*.
- TD_0: idem pour le *lecteur*.
- TC_tr_0: cette variable indique si la dernière trame reçue par la *carte* à été acquittée.
- TD_tr_0: idem pour le *lecteur*.
- $Dialogue_0$: état du dialogue entre les deux entités.

Invariant

– – **Typage des variables**
$TC_0 \in \textsc{Trame} \wedge$
$TD_0 \in \textsc{Trame} \wedge$
$TC\_tr_0 \in \{oui, non\} \wedge$
$TD\_tr_0 \in \{oui, non\} \wedge$
$Dialogue_0 \in \textsc{Dialogue} \wedge$

– – **La *carte* et le *lecteur* sont alternativement en émission et réception**
$TD\_tr_0 = \neg TC\_tr_0$

Propriétés temporelles

– – **Règle 1: La première trame doit être une I-trame émise par le *lecteur***
${}_1C_0^1 \quad TD_0 = 0 \wedge TD\_tr_0 = non \wedge TC\_tr_0 = oui \wedge$
$Dialogue_0 = EnCours \wedge TC_0 = 1$

– – **Règle 2.1: à une I-trame envoyée par le dispositif A, le dispositif B répond par une I-trame pour transmettre des données et pour indiquer qu'il peut recevoir la trame suivante de A**

$$ {}_{2.1}C_0^2 \quad \Box \left(\begin{array}{l} \forall w \in \textsc{Ident} \\ TD_0 = w \wedge TC\_tr_0 = non \wedge \\ TD\_tr_0 = oui \wedge Dialogue_0 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} (\ TD_0 = \overline{w} \wedge TD\_tr_0 = non \wedge TC\_tr_0 = oui\ ) \vee \\ Dialogue_0 = Interrompu \end{array} \right) \end{array} \right) $$

$$ {}_{2.1}C_0^3 \quad \Box \left(\begin{array}{l} \forall w \in \textsc{Ident} \\ TC_0 = w \wedge TD\_tr_0 = non \wedge \\ TC\_tr_0 = oui \wedge Dialogue_0 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} (\ TC_0 = \overline{w} \wedge TC\_tr_0 = non \wedge TD\_tr_0 = oui\ ) \vee \\ Dialogue_0 = Interrompu \end{array} \right) \end{array} \right) $$

C.2.2 Raffinement n ° 1

But

Durant ce raffinement, nous allons introduire la possibilité de transférer les informations par paquet. De plus, les avorttements seront eux aussi pris en compte. Ceux-ci ne seront concidérés que pour le cas où un chaînage ne peut se poursuivre totalement.

Les nouvelles règles concernées sont 2.2 et 5.

Données

On introduit de nouveaux ensembles :
$\textsc{TypeTrame} = \{I, R\}$
$\textsc{Chainage} = \{\hat{Chaîné}, \hat{NonChaîné}, InDef\}$
$\textsc{Trame} = \textsc{TypeTrame} \times \textsc{Ident} \times \textsc{Chainage}$

On conserve les variables du niveau précédent aux indices près. Les variables suivantes sont nécessaires :

- $\hat{EchangeChaîné}_1$: cette variable permet de connaitre l'état d'un échange chaîné.
- $\#\hat{BlocChaînéL}_1$: indique le nombre de blocs chaînés émis par le *lecteur* depuis le début des échanges.
- $\#\hat{BlocChaînéC}_1$: idem pour la *carte*.
- $NoTD_1$: identification de la dernière trame émise pas le *lecteur*.
- $NoTC_1$: idem pour la *carte*.

Invariant de collage

– – Les variables suivantes sont identiques à leurs abstractions

$TC\_tr_1 = TC\_tr_0 \wedge$
$TD\_tr_1 = TD\_tr_0 \wedge$
$Dialogue_1 = Dialogue_0 \wedge$

– – Typage des nouvelles variables

$TC_1 \in \text{TRAME} \wedge$
$TD_1 \in \text{TRAME} \wedge$
$EchangeCha\hat{i}n\acute{e}_1 \in \text{DIALOGUE} \wedge$
$\#BlocCha\hat{i}n\acute{e}L_1 \in \mathbb{N} \wedge$
$\#BlocCha\hat{i}n\acute{e}C_1 \in \mathbb{N} \wedge$
$NoTD_1 \in \text{IDENT} \wedge$
$NoTC_1 \in \text{IDENT} \wedge$

– – Restrictions sur les différents types de trames

$\forall w \in \text{IDENT}, \forall x \in \text{CHAINAGE}$

$$\left(\begin{array}{l} \left(\begin{array}{l} (\ TD_1 = (I,w,x) \vee TC_1 = (I,w,x)\ ) \\ \Rightarrow x \neq InDef \end{array} \right) \vee \\ \left(\begin{array}{l} (\ TD_1 = (R,w,x) \vee TC_1 = (R,w,x)\ ) \\ \Rightarrow x = InDef \end{array} \right) \end{array} \right) \wedge$$

– – Les trames non chaînées correspondent aux trames du niveau abstrait. La dernière trame d'un bloc correspond à une trame abstraite

$\forall w \in \text{IDENT},$

$$\left(\begin{array}{l} TC_1 = (I,w,NonCha\hat{i}n\acute{e}) \wedge \\ \#BlocCha\hat{i}n\acute{e}C_1 \bmod 2 \neq 0 \end{array} \right) \Rightarrow TC_0 = \neg w \ \Big) \wedge$$

$$\left(\begin{array}{l} TC_1 = (I,w,NonCha\hat{i}n\acute{e}) \wedge \\ \#BlocCha\hat{i}n\acute{e}C_1 \bmod 2 = 0 \end{array} \right) \Rightarrow TC_0 = w \ \Big) \wedge$$

$$\left(\begin{array}{l} TD_1 = (I,w,NonCha\hat{i}n\acute{e}) \wedge \\ \#BlocCha\hat{i}n\acute{e}L_1 \bmod 2 \neq 0 \end{array} \right) \Rightarrow TD_0 = \neg w \ \Big) \wedge$$

$$\left(\begin{array}{l} TD_1 = (I,w,NonCha\hat{i}n\acute{e}) \wedge \\ \#BlocCha\hat{i}n\acute{e}L_1 \bmod 2 = 0 \end{array} \right) \Rightarrow TD_0 = w \ \Big)$$

Propriétés temporelles

PROPRIÉTÉS TEMPORELLES RAFFINÉES

– – Règle 1: La première trame doit être une I-trame émise par le *lecteur*

$$_1C_1^1 \quad \begin{array}{l} (\ TD_1 = (I,0,NonCha\hat{i}n\acute{e}) \vee TD_1 = (I,0,Cha\hat{i}n\acute{e})\ ) \wedge \\ NoTD_1 = 0 \wedge NoTC_1 = 1 \wedge \\ TC\_tr_1 = oui \wedge TD\_tr_1 = non \wedge \\ Dialogue_1 = EnCours \wedge \#BlocCha\hat{i}n\acute{e}L_1 = 0 \wedge \\ \#BlocCha\hat{i}n\acute{e}C_1 = 0 \wedge EchangeCha\hat{i}n\acute{e}_1 = EnCours \end{array}$$

– – **Règle 2.1: à une I-trame envoyée par le dispositif A, le dispositif B répond par une I-trame pour transmettre des données et pour indiquer qu'il peut recevoir la trame suivante de A**

$$\forall w, w' \in \text{IDENT}, \exists x \in \text{CHAINAGE}$$

$$_{2.1}C_1^2 \quad \Box \left(\begin{array}{l} TC_1 = (I, w, \hat{NonChaîné}) \wedge \\ TC\_tr_1 = non \wedge TD\_tr_1 = oui \wedge \\ Dialogue_1 = EnCours \wedge NoTD_1 = w' \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} TD_1 = (I, \neg w', x) \wedge \\ TD\_tr_1 = non \wedge TC\_tr_1 = oui \end{array} \right) \vee \\ Dialogue_1 = Interrompu \end{array} \right) \end{array} \right)$$

$$\forall w, w' \in \text{IDENT}, \exists x \in \text{CHAINAGE}$$

$$_{2.1}C_1^3 \quad \Box \left(\begin{array}{l} TD_1 = (I, w, \hat{NonChaîné}) \wedge \\ TD\_tr_1 = non \wedge TC\_tr_1 = oui \wedge \\ Dialogue_1 = EnCours \wedge NoTC_1 = w' \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} TC_1 = (I, \neg w', x) \wedge \\ TC\_tr_1 = non \wedge TD\_tr_1 = oui \end{array} \right) \vee \\ Dialogue_1 = Interrompu \end{array} \right) \end{array} \right)$$

NOUVELLES PROPRIÉTÉS TEMPORELLES

– – **Règle 2.2: à une I-trame envoyée par A, B répond par une R-trame pour indiquer que le bloc reçu est correct et qu'un autre bloc peut être envoyé**

$$\forall w \in \text{IDENT}$$

$$_{2.2}C_1^4 \quad \Box \left(\begin{array}{l} TC_1 = (I, w, \hat{Chaîné}) \wedge TC\_tr_1 = non \wedge \\ TD\_tr_1 = oui \wedge Dialogue_1 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} TD_1 = (R, \neg w, InDef) \wedge \\ TD\_tr_1 = non \wedge TC\_tr_1 = oui \end{array} \right) \vee \\ Dialogue_1 = Interrompu \vee \hat{EchangeChaîné}_1 = Interrompu \end{array} \right) \end{array} \right)$$

$$\forall w \in \text{IDENT}$$

$$_{2.2}C_1^5 \quad \Box \left(\begin{array}{l} TD_1 = (I, w, \hat{Chaîné}) \wedge TD\_tr_1 = non \wedge \\ TC\_tr_1 = oui \wedge Dialogue_1 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} TC_1 = (R, \neg w, InDef) \wedge \\ TC\_tr_1 = non \wedge TD\_tr_1 = oui \end{array} \right) \vee \\ Dialogue_1 = Interrompu \vee \hat{EchangeChaîné}_1 = Interrompu \end{array} \right) \end{array} \right)$$

– – **Règle 5: Une I-trame non chaînée est un message non chaîné ou le dernier bloc d'un message chaîné. Une I-trame chaînée doit être suivie d'un autre bloc. Une R-trame accuse réception du dernier bloc envoyé et demande la transmission du suivant**

$$
{}_5C_1^6 \quad \Box \left(\begin{array}{l} \forall w \in \textsc{Ident}, \exists w' \in \textsc{Ident} \\ TD_1 = (I, w, \hat{Chaîné}) \wedge \\ TD\_tr_1 = non \wedge Dialogue_1 = EnCours \\ \Rightarrow \Diamond \left(\begin{array}{l} (\ TD_1 = (I, w', \hat{NonChaîné}) \wedge TD_1 = non\ ) \vee \\ \hat{EchangeChaîné}_1 = Interrompu \vee \\ Dialogue_1 = Interrompu \end{array} \right) \end{array} \right)
$$

$$
{}_5C_1^7 \quad \Box \left(\begin{array}{l} \forall w \in \textsc{Ident}, \exists w' \in \textsc{Ident} \\ TC_1 = (I, w, \hat{Chaîné}) \wedge \\ TC\_tr_1 = non \wedge Dialogue_1 = EnCours \\ \Rightarrow \Diamond \left(\begin{array}{l} (\ TC_1 = (I, w', \hat{NonChaîné}) \wedge TC_1 = non\ ) \vee \\ \hat{EchangeChaîné}_1 = Interrompu \vee \\ Dialogue_1 = Interrompu \end{array} \right) \end{array} \right)
$$

$$
{}_5C_1^8 \quad \Box \left(\begin{array}{l} \forall w \in \textsc{Ident}, \exists x \in \textsc{Chainage} \\ TC_1 = (R, w, InDef) \wedge TC\_tr_1 = non \wedge \\ Dialogue_1 = EnCours \wedge TD_1 = (I, \neg w, \hat{Chaîné}) \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} TD_1 = (I, w, x) \wedge \\ TC\_tr_1 = non \wedge TD\_tr_1 = oui \end{array} \right) \vee \\ \hat{EchangeChaîné}_1 = Interrompu \vee \\ Dialogue_1 = Interrompu \end{array} \right) \end{array} \right)
$$

$$
{}_5C_1^9 \quad \Box \left(\begin{array}{l} \forall w \in \textsc{Ident}, \exists x \in \textsc{Chainage} \\ TD_1 = (R, w, InDef) \wedge TD\_tr_1 = non \wedge \\ Dialogue_1 = EnCours \wedge TC_1 = (I, \neg w, \hat{Chaîné}) \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} TC_1 = (I, w, x) \wedge \\ TD\_tr_1 = non \wedge TC\_tr_1 = oui \end{array} \right) \vee \\ \hat{EchangeChaîné}_1 = Interrompu \vee \\ Dialogue_1 = Interrompu \end{array} \right) \end{array} \right)
$$

C.2.3 Raffinement n ° 2

But

Dans ce second raffinement, nous allons introduire la possibilité de changer les paramètres de la communication: longueur des messages, temps de réponse, paramètres de synchronisation, avortement de transmission.

Les nouvelles règles concernées sont 3, 4, 6, 6.2, 6.3 et 9.

Données

On introduit les ensembles:

$\textsc{Ident} = \{0, 1, InDef\}$

$\textsc{TypeTrame} = \{I, R, S\}$

$$
\textsc{Sup} = \left\{ \begin{array}{l} ResynchReq, ResynchRep, IfsReq, \\ IfsRep, AbortReq, AbortRep, \\ WtxReq, WtxRep, InDef \end{array} \right\}
$$

$\textsc{Trame} = \textsc{TypeTrame} \times \textsc{Ident} \times \textsc{Chainage} \times \textsc{Sup}$

On conserve les variables du niveau précédent aux indices près à l'exception de $\#BlocChaîné L_2$, $\#BlocChaîné C_2$, $EchangeChaîné_2$, $NoTD_2$ et $NoTC_2$. Les variables suivantes sont nécessaires :

- $AckTC_2$: I-trame en attente d'accusé de réception pour la *carte*.
- $AckTD_2$: idem pour le *lecteur*.
- $AckTC\_tr_2$: Indique si la *carte* à reçu un accusé de réception pour la dernière trame.
- $AckTD\_tr_2$: idem pour le *lecteur*.

Invariant de collage

– – Les variables suivantes sont identiques à leurs abstractions
$TC\_tr_2 = TC\_tr_1 \wedge$
$TD\_tr_2 = TD\_tr_1 \wedge$
$Dialogue_2 = Dialogue_1 \wedge$

– – Définition des nouvelles variables
$TC_2 \in \textsc{Trame} \wedge$
$TD_2 \in \textsc{Trame} \wedge$
$AckTC_2 \in \textsc{Trame} \wedge$
$AckTD_2 \in \textsc{Trame} \wedge$
$AckTC\_tr_2 \in \{oui, non\} \wedge$
$AckTD\_tr_2 \in \{oui, non\}$

– – Restrictions sur le typage
$\forall w \in \textsc{Ident}, \forall x \in \textsc{Chainage}, \forall y \in \textsc{Sup}$
$$\left(\begin{array}{l} \left(\begin{array}{l} TD_2 = (I,w,x,y) \vee TC_2 = (I,w,x,y) \vee \\ AckTC_2 = (I,w,x,y) \vee AckTD_2 = (I,w,x,y) \end{array} \right) \\ \Rightarrow \left(\; w \neq InDef \wedge x \neq InDef \wedge y = InDef \; \right) \end{array} \right) \wedge$$
$$\left(\begin{array}{l} \left(\begin{array}{l} TD_2 = (S,w,x,y) \vee TC_2 = (S,w,x,y) \vee \\ AckTC_2 = (S,w,x,y) \vee AckTD_2 = (S,w,x,y) \end{array} \right) \\ \Rightarrow \left(\; w = InDef \wedge x = InDef \wedge y \neq InDef \; \right) \end{array} \right) \wedge$$
$$\left(\begin{array}{l} \left(\begin{array}{l} TD_2 = (R,w,x,y) \vee TC_2 = (R,w,x,y) \vee \\ AckTC_2 = (R,w,x,y) \vee AckTD_2 = (R,w,x,y) \end{array} \right) \\ \Rightarrow \left(\; w \neq InDef \wedge x = InDef \wedge y = InDef \; \right) \end{array} \right) \wedge$$

– – Les trames en attente sont toujours des I-tranes
$\forall w \in \textsc{Ident}, \exists x \in \textsc{Chainage}$
$\left(\; AckTD_2 = (I,w,x,InDef) \wedge AckTC_2 = (I,w,x,InDef) \; \right) \wedge$

– – Règle 6: Une demande de resynchronisation (*Resynch*) ne peut pas être émise par la carte
$TC_2 \neq (S, InDef, InDef, ResynchReq) \wedge$
$TD_2 \neq (S, InDef, InDef, ResynchRep) \wedge$

– – Règle 3: Une demande de temps supplémentaire ne peut pas être émise par le *lecteur*. On n'exprime que partiellement la règle 3.
$TD_2 \neq (S, InDef, InDef, WtxReq) \wedge$
$TC_2 \neq (S, InDef, InDef, WtxRep) \wedge$

– – **La trame en attente d'accusé de réception est identique à la dernière trame émise si celle-ci est une trame d'information**

$\forall w \in \text{IDENT}, \forall x \in \text{CHAINAGE}$

$$\left(\begin{array}{l} TC_2 = (I, w, x, InDef) \Rightarrow AckTC_2 = TC_2 \wedge \\ TD_2 = (I, w, x, InDef) \Rightarrow AckTD_2 = TD_2 \end{array} \right) \wedge$$

– – **Il y a au plus une I-trame en attente de réception émise soit par le *lecteur*, soit par la *carte*.**

$$\left(\; AckTC\_tr_2 \Rightarrow \neg AckTD\_tr_2 \; \right) \wedge$$
$$\left(\; AckTD\_tr_2 \Rightarrow \neg AckTC\_tr_2 \; \right) \wedge$$

– – **Les R-trames et les I-trames de ce niveau sont identiques à celles du niveau précédent à la projection Sup près qui est indéfinie**

$\forall w \in \text{IDENT}, \forall x \in \text{CHAINAGE}$

$$\left(\; TC_2 = (I, w, x, InDef) \Rightarrow TC_1 = (I, w, x) \; \right) \wedge$$
$$\left(\; TD_2 = (I, w, x, InDef) \Rightarrow TD_1 = (I, w, x) \; \right) \wedge$$
$$\left(\; TC_2 = (R, w, InDef, InDef) \Rightarrow TC_1 = R, w, InDef \; \right) \wedge$$
$$\left(\; TD_2 = (R, w, InDef, InDef) \Rightarrow TD_1 = (R, w, InDef) \; \right) \wedge$$

– – **Les identificateurs des I-tranes sont identiques à leurs abstractions**

$\forall w \in \text{IDENT}, \forall x \in \text{CHAINAGE}, \forall y \in \text{SUP}$

$$\left(\; TD_2 = (I, w, x, y) \wedge w = NoTD_1 \; \right) \wedge$$
$$\left(\; TC_2 = (I, w, x, y) \wedge w = NoTC_1 \; \right)$$

Propriétés temporelles

PROPRIÉTÉS TEMPORELLES RAFFINÉES

– – **Règle 1: La première trame doit être une I-trame émise par le *lecteur***

$\exists y \in \{ResynchReq, IfsReq\}, \exists x \in \text{CHAINAGE}$

$$_1C_2^1 \quad \left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} TD_2 = (I, 0, x, InDef) \wedge \\ AckTD_2 = TD_2 \wedge AckTD\_tr_2 = non \end{array} \right) \vee \\ \left(\begin{array}{l} TD_2 = (S, InDef, InDef, y) \wedge \\ AckTD_2 = (I, 1, NonChaîné, InDef) \wedge \\ AckTD\_tr_2 = oui \end{array} \right) \end{array} \right) \wedge \\ AckTC\_tr_2 = oui \wedge AckTC_2 = (I, 1, NonChaîné, InDef) \wedge \\ TD\_tr_2 = non \wedge Dialogue_2 = EnCours \end{array} \right)$$

– – **Règle** 2.1**: à une I-trame envoyée par le dispositif A, le dispositif B répond par une I-trame pour transmettre des données et pour indiquer qu'il peut recevoir la trame suivante de A**

$$
\begin{array}{l}
\forall w, w' \in \textsc{Ident}, \forall x' \in \textsc{Chainage}, \exists x \in \textsc{Chainage} \\
{}_{2.1}C_2^2 \quad \Box \left(\begin{array}{l} AckTC_2 = (I, w, \hat{NonChaîné}, InDef) \wedge AckTC\_tr_2 = non \wedge \\ Dialogue_2 = EnCours \wedge AckTD_2 = (I, w', x', InDef) \\ \Rightarrow \Diamond \left(\begin{array}{l} \left(\begin{array}{l} TD_2 = (I, \neg w', x, InDef) \wedge \\ TD\_tr_2 = non \wedge AckTC\_tr_2 = oui \end{array} \right) \vee \\ Dialogue_2 = Interrompu \end{array} \right) \end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\forall w, w' \in \textsc{Ident}, \forall x' \in \textsc{Chainage}, \exists x \in \textsc{Chainage} \\
{}_{2.1}C_2^3 \quad \Box \left(\begin{array}{l} AckTD_2 = (I, w, \hat{NonChaîné}, InDef) \wedge AckTD\_tr_2 = non \wedge \\ Dialogue_2 = EnCours \wedge AckTC_2 = (I, w', x', InDef) \\ \Rightarrow \Diamond \left(\begin{array}{l} \left(\begin{array}{l} TC_2 = (I, \neg w', x, InDef) \wedge \\ TC\_tr_2 = non \wedge AckTD\_tr_2 = oui \end{array} \right) \vee \\ Dialogue_2 = Interrompu \end{array} \right) \end{array} \right)
\end{array}
$$

– – **Règle** 2.2**: à une I-trame envoyée par A, B répond par une R-trame pour indiquer que le bloc reçu est correct et qu'un autre bloc peut être envoyé**

$$
\begin{array}{l}
\forall w \in \textsc{Ident} \\
{}_{2.2}C_2^4 \quad \Box \left(\begin{array}{l} AckTC_2 = (I, w, \hat{Chaîné}, InDef) \wedge AckTC\_tr_2 = non \\ \Rightarrow \Diamond \left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} TD_2 = (R, \neg w, InDef, InDef) \vee \\ TD_2 = (S, InDef, InDef, AbortReq) \vee \\ TD_2 = (S, InDef, InDef, AbortRep) \end{array} \right) \wedge \\ TD\_tr_2 = non \wedge AckTC\_tr_2 = oui \end{array} \right) \vee \\ Dialogue_2 = Interrompu \end{array} \right) \end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\forall w \in \textsc{Ident} \\
{}_{2.2}C_2^5 \quad \Box \left(\begin{array}{l} AckTD_2 = (I, w, \hat{Chaîné}, InDef) \wedge AckTD\_tr_2 = non \\ \Rightarrow \Diamond \left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} TC_2 = (R, \neg w, InDef, InDef) \vee \\ TC_2 = (S, InDef, InDef, AbortReq) \vee \\ TC_2 = (S, InDef, InDef, AbortRep) \end{array} \right) \wedge \\ TC\_tr_2 = non \wedge AckTD\_tr_2 = oui \end{array} \right) \vee \\ Dialogue_2 = Interrompu \end{array} \right) \end{array} \right)
\end{array}
$$

– – **Règle 5: Une I-trame non chaînée est un message non chaîné ou le dernier bloc d'un message chaîné. Une I-trame chaînée doit être suivie d'un autre bloc. Une R-trame accuse réception du dernier bloc envoyé et demande la transmission du suivant**

$$
\begin{array}{l}
\forall w \in \textsc{Ident}, \exists w' \in \textsc{Ident} \\
{}_5C_2^6 \quad \Box \left(\begin{array}{l}
AckTD_2 = (I, w, \hat{Chaîné}, InDef) \wedge \\
AckTD\_tr_2 = non \wedge Dialogue_2 = EnCours \\
\Rightarrow \Diamond \left(\begin{array}{l}
\left(\begin{array}{l} AckTD_2 = (I, w', \hat{NonChaîné}, InDef) \wedge \\ AckTD\_tr_2 = non \end{array} \right) \vee \\
\left(\begin{array}{l} TC_2 = (S, InDef, InDef, AbortRep) \wedge \\ TC\_tr_2 = non \end{array} \right) \vee \\
\left(\begin{array}{l} TD_2 = (S, InDef, InDef, AbortReq) \wedge \\ TD\_tr_2 = non \end{array} \right) \vee \\
Dialogue_2 = Interrompu
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\forall w \in \textsc{Ident}, \exists w' \in \textsc{Ident} \\
{}_5C_2^7 \quad \Box \left(\begin{array}{l}
AckTC_2 = (I, w, \hat{Chaîné}, InDef) \wedge \\
AckTC\_tr_2 = non \wedge Dialogue_2 = EnCours \\
\Rightarrow \Diamond \left(\begin{array}{l}
\left(\begin{array}{l} AckTC_2 = (I, w', \hat{NonChaîné}, InDef) \wedge \\ AckTC\_tr_2 = non \end{array} \right) \vee \\
\left(\begin{array}{l} TD_2 = (S, InDef, InDef, AbortRep) \wedge \\ TD\_tr_2 = non \end{array} \right) \vee \\
\left(\begin{array}{l} TC_2 = (S, InDef, InDef, AbortReq) \wedge \\ TC\_tr_2 = non \end{array} \right) \vee \\
Dialogue_2 = Interrompu
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\forall w \in \textsc{Ident}, \exists x \in \textsc{Chainage} \\
{}_5C_2^8 \quad \Box \left(\begin{array}{l}
TC_2 = (R, w, InDef, InDef) \wedge TC\_tr_2 = non \wedge \\
Dialogue_2 = EnCours \wedge TD_2 = (I, \neg w, \hat{Chaîné}, InDef) \\
\Rightarrow \Diamond \left(\begin{array}{l}
AckTD\_tr_2 = oui \vee Dialogue_2 = Interrompu \vee \\
\left(\begin{array}{l} TD_2 = (I, w, x, InDef) \wedge \\ TD\_tr_2 = non \end{array} \right) \vee \\
\left(\begin{array}{l} TD_2 = (S, InDef, InDef, AbortRep) \wedge \\ TD\_tr_2 = non \end{array} \right) \vee \\
\left(\begin{array}{l} TD_2 = (S, InDef, InDef, AbortReq) \wedge \\ TD\_tr_2 = non \end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\forall w \in \textsc{Ident}, \exists x \in \textsc{Chainage} \\
{}_5C_2^9 \quad \Box \left(\begin{array}{l}
TD_2 = (R, w, InDef, InDef) \wedge TD\_tr_2 = non \wedge \\
Dialogue_2 = EnCours \wedge TC_2 = (I, \neg w, \hat{Chaîné}, InDef) \\
\Rightarrow \Diamond \left(\begin{array}{l}
AckTC\_tr_2 = oui \vee Dialogue_2 = Interrompu \vee \\
\left(\begin{array}{l} TC_2 = (I, w, x, InDef) \wedge \\ TC\_tr_2 = non \end{array} \right) \vee \\
\left(\begin{array}{l} TC_2 = (S, InDef, InDef, AbortRep) \wedge \\ TC\_tr_2 = non \end{array} \right) \vee \\
\left(\begin{array}{l} TC_2 = (S, InDef, InDef, AbortReq) \wedge \\ TC\_tr_2 = non \end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array}
$$

– – **Règle 3: à une demande de temps supplémentaire (*WtxReq*) de la *carte*, le *lecteur* répond par une acceptation(*WtxRep*)**

$$ {}_3C_2^{10} \quad \Box \left(\begin{array}{l} TC_2 = (S, InDef, InDef, WtxReq) \wedge \\ TC\_tr_2 = non \wedge Dialogue_2 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\; TD_2 = (S, InDef, InDef, WtxReq) \wedge TD\_tr_2 = non \; \right) \vee \\ Dialogue_2 = Interrompu \end{array} \right) \end{array} \right) $$

– – **Règle 4: à une demande de modification de longueur de bloc échangé (*IfsReq*) envoyée par A, B répond par une acceptation (*IfsRep*)**

$$ {}_4C_2^{11} \quad \Box \left(\begin{array}{l} TC_2 = (S, InDef, InDef, IfsReq) \wedge \\ TC\_tr_2 = non \wedge Dialogue_2 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\; TD_2 = (S, InDef, InDef, IfsRep) \wedge TD\_tr_2 = non \; \right) \vee \\ Dialogue_2 = Interrompu \end{array} \right) \end{array} \right) $$

$$ {}_4C_2^{12} \quad \Box \left(\begin{array}{l} TD_2 = (S, InDef, InDef, IfsReq) \wedge \\ TD\_tr_2 = non \wedge Dialogue_2 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\; TC_2 = (S, InDef, InDef, IfsRep) \wedge TC\_tr_2 = non \; \right) \vee \\ Dialogue_2 = Interrompu \end{array} \right) \end{array} \right) $$

– – **Règle 6.2: Une demande de resynchronisation (*ResynchReq*) doit être acquittée par une acceptation de resynchronisation (*ResynchRep*)**

$$ {}_{6.2}C_2^{13} \quad \Box \left(\begin{array}{l} TD_2 = (S, InDef, InDef, ResynchReq) \wedge \\ TD\_tr_2 = non \wedge Dialogue_2 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\; TC_2 = (S, InDef, InDef, ResynchRep) \wedge TC\_tr_2 = non \; \right) \vee \\ Dialogue_2 = EnCours \end{array} \right) \end{array} \right) $$

– – **Règle 6.3: Après une acceptation à une demande de resynchronisation, le protocole est réinitialisé par le *lecteur* (cf. règle 1).**

$$ \begin{array}{l} \exists y \in \{ResynchReq, IfsReq\}, \exists x \in \textsc{Chainage} \\ {}_{6.3}C_2^{14} \quad \Box \left(\begin{array}{l} TC_2 = (S, InDef, InDef, ResynchRep) \wedge \\ TC\_tr_2 = non \wedge Dialogue_2 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} TD_2 = (I, 0, x, InDef) \wedge \\ AckTD_2 = TD_2 \wedge AckTD\_tr_2 = non \end{array} \right) \vee \\ \left(\begin{array}{l} TD_2 = (S, InDef, InDef, y) \wedge \\ AckTD_2 = (I, 1, \hat{NonChaîné}, InDef) \wedge \\ AckTD\_tr_2 = oui \end{array} \right) \end{array} \right) \wedge \\ AckTC\_tr_2 = oui \wedge TD\_tr_2 = non \wedge \\ AckTC_2 = (I, 1, \hat{NonChaîné}, InDef) \wedge \\ Dialogue_2 = EnCours \end{array} \right) \end{array} \right) \end{array} $$

– – **Règle 9: L'avortement d'un échange chaîné peut être à l'initiative de l'émetteur ou du récepteur. Une demande d'avortement d'échange chaîné (*AbortReq*) doit être acquitté par une acceptation (*AbortRep*). Après cet échange une R-trame peut être envoyée pour redonner le droit d'émettre.**

$$ {}_9C_2^{15} \quad \Box \left(\begin{array}{l} TC_2 = (S, InDef, InDef, AbortReq) \wedge \\ TC\_tr_2 = non \wedge Dialogue_2 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} (\ TD_2 = (S, InDef, InDef, AbortRep) \wedge TD\_tr_2 = non\ ) \vee \\ Dialogue_2 = Interrompu \end{array} \right) \end{array} \right) $$

$$ {}_9C_2^{16} \quad \Box \left(\begin{array}{l} TD_2 = (S, InDef, InDef, AbortReq) \wedge \\ TD\_tr_2 = non \wedge Dialogue_2 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} (\ TC_2 = (S, InDef, InDef, AbortRep) \wedge TC\_tr_2 = non\ ) \vee \\ Dialogue_2 = Interrompu \end{array} \right) \end{array} \right) $$

C.2.4 Raffinement n ° 3

But

Dans ce niveau de raffinement, les erreurs dues au support de communication et le délai d'attente d'une réponse du *lecteur* sont pris en compte.

Nouvelles règles concernées: 6.4, 6.5, 7.1 à 7.6, 7.4.1 à 7.4.3, 8.

Données

On introduit les ensembles :

$\textsc{Erreur} = \{oui, non\}$

$\textsc{Trame} = \textsc{TypeTrame} \times \textsc{Ident} \times \textsc{Chainage} \times \textsc{Sup} \times \textsc{Erreur}$

On conserve les variables du niveau précédent aux indices près. Les variables suivantes sont introduites :

- RTC_3: dernière trame reçue par la *carte*.
- RTD_3: idem pour le *lecteur*.
- RTC_tr_3: La dernière trame reçue par la *carte* a été traitée.
- RTD_tr_3: idem pour le *lecteur*.
- $\#EssaiResynch_3$: cette variable permet de compter le nombre de resynchronisations successives réalisées par le *lecteur*.
- $\#ErreurC_3$: compteur du nombre de réceptions successives de trames erronées chez la *carte*.
- $TimeOut_3$: signal indiquant que le délai d'attente du *lecteur* est dépassé.

Invariant de collage

– – **Les variables suivantes sont identiques à leurs abstractions**

$TC\_tr_3 = TC\_tr_2 \wedge$
$TD\_tr_3 = TD\_tr_2 \wedge$
$Dialogue_3 = Dialogue_2 \wedge$
$AckTC\_tr_3 = AckTC\_tr_2 \wedge$
$AckTD\_tr_3 = AckTD\_tr_2 \wedge$

– – **Typages des nouvelles variables**

$TC_3 \in \text{TRAME} \wedge$
$TD_3 \in \text{TRAME} \wedge$
$AckTC_3 \in \text{TRAME} \wedge$
$AckTD_3 \in \text{TRAME} \wedge$
$RTC_3 \in \text{TRAME} \wedge$
$RTD_3 \in \text{TRAME} \wedge$
$RTC\_tr_3 \in \{oui, non\} \wedge$
$RTD\_tr_3 \in \{oui, non\} \wedge$
$\#EssaiResynch_3 \in \mathbb{N} \wedge$
$\#ErreurC_3 \in \mathbb{N} \wedge$
$TimeOut_3 \in \{oui, non\} \wedge$

– – **Restrictions sur le typage**

$\forall w \in \text{IDENT}, \forall x \in \text{CHAINAGE}, \forall y \in \text{SUP} \wedge \forall z \in \text{ERREUR}$

$$\left(\begin{array}{l} \left(\begin{array}{l} TD_3 = (I,w,x,y,z) \vee TC_3 = (I,w,x,y,z) \vee \\ AckTC_3 = (I,w,x,y,z) \vee AckTD_3 = (I,w,x,y,z) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} w \neq InDef \wedge x \neq InDef \wedge \\ y = InDef \wedge z = non \end{array} \right) \end{array} \right) \wedge$$

$$\left(\begin{array}{l} \left(\begin{array}{l} TD_3 = (S,w,x,y,z) \vee TC_3 = (S,w,x,y,z) \vee \\ AckTC_3 = (S,w,x,y,z) \vee AckTD_3 = (S,w,x,y,z) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} w = InDef \wedge x = InDef \wedge \\ y \neq InDef \wedge z = non \end{array} \right) \end{array} \right) \wedge$$

$$\left(\begin{array}{l} \left(\begin{array}{l} TD_3 = (R,w,x,y,z) \vee TC_3 = (R,w,x,y,z) \vee \\ AckTC_3 = (R,w,x,y,z) \vee AckTD_3 = (R,w,x,y,z) \end{array} \right) \\ \Rightarrow \left(\begin{array}{l} w \neq InDef \wedge x = InDef \wedge \\ y = InDef \wedge z = non \end{array} \right) \end{array} \right) \wedge$$

– – **Relations des variables avec leurs abstractions**

$\forall k \in \text{TYPETRAME}, \forall w \in \text{IDENT},$
$\forall x \in \text{CHAINAGE}, \forall y \in \text{SUP}$
$\left(\; TC_3 = (k,w,x,y,non) \Rightarrow TC_2 = (k,w,x,y) \; \right) \wedge$
$\left(\; TD_3 = (k,w,x,y,non) \Rightarrow TD_2 = (k,w,x,y) \; \right) \wedge$
$\left(\; AckTC_3 = (I,w,x,InDef,non) \Rightarrow AckTC_2 = (I,w,x,InDef) \; \right) \wedge$
$\left(\; AckTD_3 = (I,w,x,InDef,non) \Rightarrow AckTD_2 = (I,w,x,InDef) \; \right) \wedge$

– – **Les erreurs sont dues à la transmission, donc seulement dans RTD_3 et RTC_3**

$\forall w \in \text{IDENT}, \forall x \in \text{CHAINAGE},$
$\forall y \in \text{SUP}, \forall z \in \text{ERREUR}$
$\left(\; \left(\; RTC_3 = (I,w,x,y,z) \vee RTD_3 = (I,w,x,y,z) \; \right) \Rightarrow \left(\; w \neq InDef \wedge x \neq InDef \wedge y = InDef \; \right) \right) \wedge$
$\left(\; \left(\; RTC_3 = (S,w,x,y,z) \vee RTD_3 = (S,w,x,y,z) \; \right) \Rightarrow \left(\; w = InDef \wedge x = InDef \wedge y \neq InDef \; \right) \right) \wedge$
$\left(\; \left(\; RTC_3 = (R,w,x,y,z) \vee RTD_3 = (R,w,x,y,z) \; \right) \Rightarrow \left(\; w \neq InDef \wedge x = InDef \wedge y = InDef \; \right) \right) \wedge$

– – **Règle 6: Une demande de resynchronisation (*Resynch*) ne peut pas être émise par la carte**

$\forall z \in \text{ERREUR}$

$$\left(\begin{array}{l} TC_3 \neq (S, InDef, InDef, ResynchReq, non) \wedge \\ TD_3 \neq (S, InDef, InDef, ResynchRep, non) \wedge \\ RTD_3 \neq (S, InDef, InDef, WtxReq, z) \wedge \\ RTC_3 \neq (S, InDef, InDef, ResynchRep, z) \end{array} \right) \wedge$$

– – **Règle 3: Une demande de temps supplémentaire ne peut pas être émise par le *lecteur*. On n'exprime que partiellement la règle 3.**

$$\forall z \in \text{Erreur} \quad \left(\begin{array}{l} TD_3 \neq (S, InDef, InDef, WtxReq, non) \wedge \\ TC_3 \neq (S, InDef, InDef, WtxRep, non) \wedge \\ RTC_3 \neq (S, InDef, InDef, WtxReq, z) \wedge \\ RTD_3 \neq (S, InDef, InDef, WtxRep, z) \end{array} \right)$$

Propriétés temporelles

Propriétés temporelles raffinées

– – **Règle 1: La première trame doit être une I-trame émise par le *lecteur***

$$\begin{array}{l} \exists y \in \{ResynchReq, IfsReq\}, \exists x \in \text{Chainage} \\ {}_1C_3^1 \quad \left(\begin{array}{l} \left(\begin{array}{l} TD_3 = (I, 0, x, InDef, non) \wedge \\ AckTD_3 = TD_3 \wedge AckTD\_tr_3 = non \end{array} \right) \vee \\ \left(\begin{array}{l} TD_3 = (S, InDef, InDef, y, non) \wedge \\ AckTD_3 = (I, 1, \hat{NonChaîné}, InDef) \wedge \\ AckTD\_tr_3 = oui \end{array} \right) \end{array} \right) \wedge \\ AckTC\_tr_3 = oui \wedge AckTC_3 = (I, 1, \hat{NonChaîné}, InDef) \wedge \\ TD\_tr_3 = non \wedge Dialogue_3 = EnCours \wedge \\ \#EssaiResynch_3 = 0 \wedge TimeOut_3 = non \wedge \#ErreurC_3 = 0 \end{array}$$

– – **Règle 2.1: à une I-trame envoyée par le dispositif A, le dispositif B répond par une I-trame pour transmettre des données et pour indiquer qu'il peut recevoir la trame suivante de A**

$$\begin{array}{l} \forall w, w' \in \text{Ident}, \forall x' \in \text{Chainage}, \exists x \in \text{Chainage} \\ {}_{2.1}C_3^2 \quad \Box \left(\begin{array}{l} AckTC_3 = (I, w, \hat{NonChaîné}, InDef, non) \wedge AckTC\_tr_3 = non \wedge \\ Dialogue_3 = EnCours \wedge AckTD_3 = (I, w', x', InDef) \\ \Rightarrow \Diamond \left(\begin{array}{l} RTC_3 = (I, \neg w', x, InDef, non) \wedge \\ RTC\_tr_3 = non \wedge AckTC\_tr_3 = oui \end{array} \right) \end{array} \right) \end{array}$$

$$\begin{array}{l} \forall w, w' \in \text{Ident}, \forall x' \in \text{Chainage}, \exists x \in \text{Chainage} \\ {}_{2.1}C_3^3 \quad \Box \left(\begin{array}{l} AckTD_3 = (I, w, \hat{NonChaîné}, InDef, non) \wedge AckTD\_tr_3 = non \wedge \\ Dialogue_3 = EnCours \wedge AckTC_3 = (I, w', x', InDef) \\ \Rightarrow \Diamond \left(\begin{array}{l} RTD_3 = (I, \neg w', x, InDef, non) \wedge \\ RTD\_tr_3 = non \wedge AckTD\_tr_3 = oui \end{array} \right) \end{array} \right) \end{array}$$

– – **Règle 2.2: à une I-trame envoyée par A, B répond par une R-trame pour indiquer que le bloc reçu est correct et qu'un autre bloc peut être envoyé**

$$
{}_{2.2}C_3^4 \quad \Box \left(\begin{array}{l} \forall w \in \textsc{Ident} \\ \left(\begin{array}{l} AckTC_3 = (I, w, Chaîné, InDef, non) \wedge AckTC\_tr_3 = non \\ \Rightarrow \Diamond \left(\left(\begin{array}{l} RTC_3 = (R, \neg w, InDef, InDef, non) \vee \\ RTC_3 = (S, InDef, InDef, AbortReq, non) \vee \\ RTC_3 = (S, InDef, InDef, AbortRep, non) \end{array} \right) \wedge \right) \\ RTC\_tr_3 = non \wedge AckTC\_tr_3 = oui \end{array} \right) \end{array} \right)
$$

$$
{}_{2.2}C_3^5 \quad \Box \left(\begin{array}{l} \forall w \in \textsc{Ident} \\ \left(\begin{array}{l} AckTD_3 = (I, w, Chaîné, InDef, non) \wedge AckTD\_tr_3 = non \\ \Rightarrow \Diamond \left(\left(\begin{array}{l} RTD_3 = (R, \neg w, InDef, InDef, non) \vee \\ RTD_3 = (S, InDef, InDef, AbortReq, non) \vee \\ RTD_3 = (S, InDef, InDef, AbortRep, non) \end{array} \right) \wedge \right) \\ RTD\_tr_3 = non \wedge AckTD\_tr_3 = oui \end{array} \right) \end{array} \right)
$$

– – **Règle 5: Une I-trame non chaînée est un message non chaîné ou le dernier bloc d'un message chaîné. Une I-trame chaînée doit être suivie d'un autre bloc. Une R-trame accuse réception du dernier bloc envoyé et demande la transmission du suivant**

$$
\begin{array}{l}
\forall w \in \textsc{Ident}, \exists w' \in \textsc{Ident} \\
{}_5C_3^6 \quad \Box \left(\begin{array}{l}
AckTD_3 = (I, w, \hat{Chaîné}, InDef) \wedge \\
AckTD\_tr_3 = non \wedge Dialogue_3 = EnCours \\
\Rightarrow \Diamond \left(\begin{array}{l}
\left(\begin{array}{l} AckTD_3 = (I, w', \hat{NonChaîné}, InDef) \wedge \\ AckTD\_tr_3 = non \end{array} \right) \vee \\
\left(\begin{array}{l} RTD_3 = (S, InDef, InDef, AbortRep) \wedge \\ RTD\_tr_3 = non \end{array} \right) \vee \\
\left(\begin{array}{l} RTD_3 = (S, InDef, InDef, AbortReq) \wedge \\ RTD\_tr_3 = non \end{array} \right) \vee \\
Dialogue_3 = Interrompu
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\forall w \in \textsc{Ident}, \exists w' \in \textsc{Ident} \\
{}_5C_3^7 \quad \Box \left(\begin{array}{l}
AckTC_3 = (I, w, \hat{Chaîné}, InDef) \wedge \\
AckTC\_tr_3 = non \wedge Dialogue_3 = EnCours \\
\Rightarrow \Diamond \left(\begin{array}{l}
(\ AckTC_3 = (I, w', \hat{NonChaîné}, InDef) \wedge AckTC\_tr_3 = non \ ) \vee \\
(\ RTC_3 = (S, InDef, InDef, AbortRep) \wedge RTD\_tr_3 = non \ ) \vee \\
(\ RTC_3 = (S, InDef, InDef, AbortReq) \wedge RTC\_tr_3 = non \ ) \vee \\
Dialogue_3 = Interrompu
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\forall w \in \textsc{Ident}, \exists x \in \textsc{Chainage} \\
{}_5C_3^8 \quad \Box \left(\begin{array}{l}
RTD_3 = (R, w, InDef, InDef) \wedge RTD\_tr_3 = non \wedge \\
Dialogue_3 = EnCours \\
\Rightarrow \Diamond \left(\begin{array}{l}
\left(\begin{array}{l} AckTD\_tr_3 = oui \wedge TD\_tr_3 = non \wedge \\ TD_3 = (I, w, x, InDef, non) \end{array} \right) \vee \\
\left(\begin{array}{l} TD_3 = (S, InDef, InDef, AbortRep, non) \wedge \\ TD\_tr_3 = non \end{array} \right) \vee \\
\left(\begin{array}{l} TD_3 = (S, InDef, InDef, AbortReq, non) \wedge \\ TD\_tr_3 = non \end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\forall w \in \textsc{Ident}, \exists x \in \textsc{Chainage} \\
{}_5C_3^9 \quad \Box \left(\begin{array}{l}
RTC_3 = (R, w, InDef, InDef) \wedge RTC\_tr_3 = non \wedge \\
Dialogue_3 = EnCours \\
\Rightarrow \Diamond \left(\begin{array}{l}
\left(\begin{array}{l} AckTC\_tr_3 = oui \wedge TC\_tr_3 = non \wedge \\ TC_3 = (I, w, x, InDef, non) \end{array} \right) \vee \\
\left(\begin{array}{l} TC_3 = (S, InDef, InDef, AbortRep, non) \wedge \\ TC\_tr_3 = non \end{array} \right) \vee \\
\left(\begin{array}{l} TC_3 = (S, InDef, InDef, AbortReq, non) \wedge \\ TC\_tr_3 = non \end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array}
$$

– – **Règle 3: à une demande de temps supplémentaire (*WtxReq*) de la *carte*, le *lecteur* répond par une acceptation(*WtxRep*)**

$$
{}_3C_3^{10} \quad \Box \left(\begin{array}{l}
TC_3 = (S, InDef, InDef, WtxReq, non) \wedge \\
TC\_tr_3 = non \wedge Dialogue_3 = EnCours \\
\Rightarrow \Diamond \left(\ TD_3 = (S, InDef, InDef, WtxReq, F) \wedge TD\_tr_3 = non \ \right)
\end{array} \right)
$$

– – **Règle 4: à une demande de modification de longueur de bloc échangé (*IfsReq*) envoyée par A, B répond par une acceptation (*IfsRep*)**

$$
{}_4C_3^{11} \quad \Box \left(\begin{array}{l} TC_3 = (S, InDef, InDef, IfsReq, F) \wedge \\ TC\_tr_3 = non \wedge Dialogue_3 = EnCours \\ \Rightarrow \Diamond \left(\; TC_3 = (S, InDef, InDef, IfsRep, F) \wedge TC\_tr_3 = non \; \right) \end{array} \right)
$$

$$
{}_4C_3^{12} \quad \Box \left(\begin{array}{l} TD_3 = (S, InDef, InDef, IfsReq, F) \wedge \\ TD\_tr_3 = non \wedge Dialogue_3 = EnCours \\ \Rightarrow \Diamond \left(\; TD_3 = (S, InDef, InDef, IfsRep, F) \wedge TD\_tr_3 = non \; \right) \end{array} \right)
$$

– – **Règle 6.2: Une demande de resynchronisation (*ResynchReq*) doit être acquittée par une acceptation de resynchronisation (*ResynchRep*)**

$$
{}_{6.2}C_3^{13} \quad \Box \left(\begin{array}{l} TD_3 = (S, InDef, InDef, ResynchReq, non) \wedge \\ TD\_tr_3 = non \wedge Dialogue_3 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\; RTD_3 = (S, InDef, InDef, ResynchRep, non) \wedge TD\_tr_3 = non \; \right) \vee \\ Dialogue_3 = EnCours \end{array} \right) \end{array} \right)
$$

– – **Règle 6.3: Après une acceptation à une demande de resynchronisation, le protocole est réinitialisé par le *lecteur* (cf. règle 1).**

$$
\begin{array}{l} \exists y \in \{ResynchReq, IfsReq\}, \exists x \in \textsc{Chainage} \\ {}_{6.3}C_3^{14} \quad \Box \left(\begin{array}{l} RTD_3 = (S, InDef, InDef, ResynchRep, non) \wedge \\ TD\_tr_3 = non \wedge Dialogue_3 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} TD_3 = (I, 0, x, InDef, non) \wedge \\ AckTD_3 = TD_3 \wedge AckTD\_tr_3 = non \end{array} \right) \vee \\ \left(\begin{array}{l} TD_3 = (S, InDef, InDef, y, non) \wedge \\ AckTD_3 = (I, 1, NonChaîné, InDef) \wedge \\ AckTD\_tr_3 = oui \end{array} \right) \end{array} \right) \wedge \\ AckTC\_tr_3 = oui \wedge TD\_tr_3 = non \wedge \\ AckTC_3 = (I, 1, NonChaîné, InDef) \wedge \\ Dialogue_3 = EnCours \wedge \#EssaiResynch_3 = 0 \wedge \\ \#ErreurC_3 = 0 \end{array} \right) \end{array} \right) \end{array}
$$

– – **Règle 9: L'avortement d'un échange chaîné peut être à l'initiative de l'émetteur ou du récepteur. Une demande d'avortement d'échange chaîné (*AbortReq*) doit être acquitté par une acceptation (*AbortRep*). Après cet échange une R-trame peut être emvoyée pour redonner le droit d'émettre.**

$$
{}_9C_3^{15} \quad \Box \left(\begin{array}{l} TC_3 = (S, InDef, InDef, AbortReq) \wedge \\ TC\_tr_3 = non \wedge Dialogue_3 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\; TD_3 = (S, InDef, InDef, AbortRep) \wedge TD\_tr_3 = non \; \right) \vee \\ Dialogue_3 = Interrompu \end{array} \right) \end{array} \right)
$$

$$
{}_9C_3^{16} \quad \Box \left(\begin{array}{l} TD_3 = (S, InDef, InDef, AbortReq) \wedge \\ TD\_tr_3 = non \wedge Dialogue_3 = EnCours \\ \Rightarrow \bigcirc \left(\begin{array}{l} \left(\; TC_3 = (S, InDef, InDef, AbortRep) \wedge TC\_tr_3 = non \; \right) \vee \\ Dialogue_3 = Interrompu \end{array} \right) \end{array} \right)
$$

– – **Règle** 6.4**: Après trois tentatives successives de resynchronisation par une S-trame $R(ResynchReq)$, le *lecteur* fait une remise à zéro de la *carte*.**

$$ {}_{6.4}C_3^{17} \quad \Box \left(\begin{array}{l} \#EssaiResynch_3 = 3 \wedge TimeOut_3 \wedge \\ TD_3 = (S, InDef, InDef, ResynchReq, non) \wedge \\ RTD\_tr_3 = oui \\ \Rightarrow \bigcirc \left(\; Dialogue_3 = Interrompu \; \right) \end{array} \right) $$

$$ {}_{6.4}C_3^{18} \quad \Box \left(\begin{array}{l} \forall n \in \mathbb{N} \\ TimeOut_3 \wedge RTD\_tr_3 = oui \wedge \\ TD_3 = (S, InDef, InDef, ResynchReq, non) \wedge \\ \#EssaiResynch_3 < 3 \wedge \#EssaiResynch_3 = n \\ \Rightarrow \bigcirc \left(\begin{array}{l} \#EssaiResynch_3 = n+1 \wedge TD\_tr_3 = non \wedge \\ TD_3 = (S, InDef, InDef, ResynchReq, non) \end{array} \right) \end{array} \right) $$

– – **Règle** 6.5**: Quand la carte reçoit une S-trame $S(ResynchReq)$, elle considère que le bloc précédement émis n'a pas été reçu**

$$ {}_{6.5}C_3^{19} \quad \Box \left(\begin{array}{l} RTC_3 = (S, InDef, InDef, ResynchReq, non) \wedge \\ RTC\_tr_3 = non \wedge Dialogue_3 = EnCours \\ \Rightarrow AckTD\_tr_3 = non \end{array} \right) $$

– – **Règle** 7.1**: Quand une I-trame I est envoyée et qu'une trame invalide est reçue ou qu'un "timeout" survient, une trame R de demande de réponse est envoyée; celle-ci précise l'identificateur $N(R)$ de la trame I d'information dont on attend la réponse avec $N(R) = N(S)$.**

$$ {}_{7.1}C_3^{20} \quad \Box \left(\begin{array}{l} \forall w, w' \in \textsc{Ident}, \forall x, x' \in \textsc{Chainage}, \forall k \in \textsc{TypeTrame}, \forall y \in \textsc{Sup} \\ AckTC_3 = (I, w, x, InDef, non) \wedge \\ AckTC\_tr_3 = non \wedge RTC\_tr_3 = non \wedge \\ RTC_3 = (k, w', x', y, oui) \\ \Rightarrow \bigcirc \left(\begin{array}{l} TC_3 = (R, w, InDef, InDef, non) \wedge \\ TC\_tr_3 = non \end{array} \right) \end{array} \right) $$

$$ {}_{7.1}C_3^{21} \quad \Box \left(\begin{array}{l} \forall w, w' \in \textsc{Ident}, \forall x, x' \in \textsc{Chainage}, \forall k \in \textsc{TypeTrame}, \forall y \in \textsc{Sup} \\ AckTD_3 = (I, w, x, InDef, non) \wedge \\ AckTD\_tr_3 = non \wedge RTD\_tr_3 = non \wedge \\ RTD_3 = (k, w', x', y, oui) \\ \Rightarrow \bigcirc \left(\begin{array}{l} TD_3 = (R, w, InDef, InDef, non) \wedge \\ TD\_tr_3 = non \end{array} \right) \end{array} \right) $$

– – **Règle** 7.2**: Quand une R-trame a été envoyée et qu'une trame invalide est reçue ou qu'un "$TimeOut_3$" survient la trame R est retransmise**

$$ {}_{7.2}C_3^{22} \quad \Box \left(\begin{array}{l} \forall k \in \textsc{TypeTrame}, \forall w, w' \in \textsc{Ident}, \forall x \in \textsc{Chainage}, \forall y \in \textsc{Sup} \\ TC_3 = (R, w, InDef, InDef, non) \wedge \\ RTC_3 = (k, w', x, y, oui) \wedge \\ TC\_tr_3 = non \\ \Rightarrow \bigcirc \left(\begin{array}{l} TC_3 = (R, w, InDef, InDef, non) \wedge \\ TC\_tr_3 = non \end{array} \right) \end{array} \right) $$

$$ {}_{7.2}C_3^{23} \quad \Box \left(\begin{array}{l} \forall k \in \textsc{TypeTrame}, \forall w, w' \in \textsc{Ident}, \forall x \in \textsc{Chainage}, \forall y \in \textsc{Sup} \\ TD_3 = (R, w, InDef, InDef, non) \wedge \\ RTD_3 = (k, w', x, y, oui) \wedge \\ TD\_tr_3 = non \\ \Rightarrow \bigcirc \left(\begin{array}{l} TD_3 = (R, w, InDef, InDef, non) \wedge \\ TD\_tr_3 = non \end{array} \right) \end{array} \right) $$

– – **Règle 7.3: Si on ne répond pas à une S-trame $S(XReq)$ par une S-trame $\boldsymbol{S(XRep)}$ ou qu'il survient un time out côté *lecteur*, il faut réémettre la S-trame $\boldsymbol{S(XReq)}$**

$$\forall X \in \{Resynch, Ifs, Abort\}$$
$${}_{7.3}C_3^{24} \quad \square \quad \left(\begin{array}{l} TD_3 = (S, InDef, InDef, XReq, non) \wedge \\ \left(\begin{array}{l} \left(\begin{array}{l} RTD_3 \neq (S, InDef, Indef, XRep, non) \wedge \\ RTD\_tr_3 = non \end{array} \right) \vee \\ TimeOut_3 \end{array} \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} TD_3 = (S, InDef, InDef, XReq, non) \wedge \\ TD\_tr_3 = non \wedge RTD\_tr_3 = oui \end{array} \right) \end{array} \right)$$

$$\forall X \in \{Resynch, Ifs, Abort\}$$
$${}_{7.3}C_3^{25} \quad \square \quad \left(\begin{array}{l} TD_3 = (S, InDef, InDef, XReq, non) \wedge \\ \left(\begin{array}{l} \left(\begin{array}{l} RTC_3 \neq (S, InDef, Indef, XRep, non) \wedge \\ RTC\_tr_3 = non \end{array} \right) \vee \\ TimeOut_3 \end{array} \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} TC_3 = (S, InDef, InDef, XReq, non) \wedge \\ TC\_tr_3 = non \wedge RTC\_tr_3 = oui \end{array} \right) \end{array} \right)$$

– – **Règle 7.3*bis*: Si on envoie une S-trame $\boldsymbol{S(XRep)}$ et au'une trane erronée est reçue ou qu'un time out survient, une R-trame est envoyée**

$$\forall k \in \text{TypeTrame}, \forall w \in \text{Ident}, \forall x \in \text{Chainage}, \forall y \in \text{Sup}, \forall X \in \{Wtx, Ifs, Abort\}$$
$${}_{7.3}C_3^{26} \quad \square \quad \left(\begin{array}{l} TD_3 = (S, InDef, InDef, XRep, non) \wedge \\ TD\_tr_3 = non \wedge \\ \left(\begin{array}{l} \left(\begin{array}{l} RTD_3 = (k, w, x, y, oui) \wedge \\ RTD\_tr_3 = non \end{array} \right) \vee TimeOut_3 \end{array} \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} TC_3 = (R, 0, InDef, InDef, non) \wedge TC\_tr_3 = non \end{array} \right) \end{array} \right)$$

$$\forall k \in \text{TypeTrame}, \forall w \in \text{Ident}, \forall x \in \text{Chainage}, \forall y \in \text{Sup}, \forall X \in \{Wtx, Ifs, Abort\}$$
$${}_{7.3}C_3^{27} \quad \square \quad \left(\begin{array}{l} TC_3 = (S, InDef, InDef, XRep, non) \wedge \\ TC\_tr_3 = non \wedge \\ \left(\begin{array}{l} \left(\begin{array}{l} RTC_3 = (k, w, x, y, oui) \wedge \\ RTC\_tr_3 = non \end{array} \right) \vee TimeOut_3 \end{array} \right) \\ \Rightarrow \bigcirc \left(\begin{array}{l} TD_3 = (R, 0, InDef, InDef, non) \wedge TD\_tr_3 = non \end{array} \right) \end{array} \right)$$

– – **Règle 7.4.3: Après deux essais infructeux de réception sans erreur par le carte, celle-ci reste en réception**

$$\exists k \in \text{TypeTrame}, \exists w \text{Ident}, \exists x \in \text{Chainage}, \exists y \in \text{Sup}$$
$${}_{7.4.3}C_3^{28} \quad \square \quad \left(\begin{array}{l} \#ErreurC_3 = 3 \wedge RTC\_tr_3 = oui \\ \Rightarrow \left(\begin{array}{l} TC\_tr_3 = non \end{array} \right) \mathcal{U} \left(\begin{array}{l} RTC\_tr_3 = non \wedge \\ RTC_3 \neq (k, w, x, y, oui) \end{array} \right) \end{array} \right)$$

$$\forall k \in \text{TypeTrame}, \forall w \in \text{Ident}, \forall x \in \text{Chainage}, \forall y \in \text{Sup}, \forall n \in \mathbb{N}$$
$${}_{7.4.3}C_3^{29} \quad \square \quad \left(\begin{array}{l} RTC_3 = (k, w, x, y, oui) \wedge RTC\_tr_3 = non \wedge \\ \#ErreurC_3 = n \wedge \#ErreurC_3 < 3 \\ \Rightarrow \bigcirc \left(\begin{array}{l} \#ErreurC_3 = n + 1 \wedge RTC\_tr_3 = oui \end{array} \right) \end{array} \right)$$

– – **Règle 8: Quand la carte envoie une S-trame et reçoit une trame erronée, elle retransmet au plus une fois sa requête. Après la deuxième tentative, elle reste en mode réception.**

$$ {}_8C_3^{30} \quad \Box \left(\begin{array}{l} \#ErreurC_3 = 1 \wedge TC_3 = (S, InDef, InDef, IfsReq, non) \wedge \\ TC\_tr_3 = oui \wedge RTC\_tr_3 = oui \\ \Rightarrow \left(\; TC\_tr_3 = non \; \right) \mathcal{U} \left(\begin{array}{l} RTC_3 = (S, InDef, InDef, IfsRep, non) \wedge \\ RTC\_tr_3 = non \end{array} \right) \end{array} \right) $$

$$ {}_8C_3^{31} \quad \Box \left(\begin{array}{l} RTC_3 = 0 \wedge TC\_tr_3 = oui \wedge \\ TC_3 = (S, InDef, InDef, IfsReq, non) \wedge \\ RTC_3 \neq (S, InDef, InDef, IfsRep, non) \wedge \\ RTC\_tr_3 = non \\ \Rightarrow \bigcirc \left(\begin{array}{l} \#ErreurC_3 = 1 \wedge TC\_tr_3 = non \wedge \\ TC_3 = (S, InDef, InDef, IfsReq, non) \end{array} \right) \end{array} \right) $$

– – **Règle 9: L'avortement d'un échange chaîné peut être à l'initiative de l'émetteur ou du récepteur. Une demande d'avortement d'échange chaîné doit être acquitté par une acceptation. après cet échange, une R-trame peut être envoyée pour redonner le droit d'émettre.**

$$ {}_9C_3^{32} \quad \Box \left(\begin{array}{l} TC_3 = (S, InDef, InDef, AbortReq) \wedge \\ TC\_tr_3 = non \wedge Dialogue_3 = EnCours \\ \Rightarrow \Diamond \left(\begin{array}{l} RTC_3 = (S, InDef, InDef, AbortRep) \wedge \\ RTC\_tr_3 = non \end{array} \right) \end{array} \right) $$

$$ {}_9C_3^{33} \quad \Box \left(\begin{array}{l} TD_3 = (S, InDef, InDef, AbortReq) \wedge \\ TD\_tr_3 = non \wedge Dialogue_3 = EnCours \\ \Rightarrow \Diamond \left(\begin{array}{l} RTD_3 = (S, InDef, InDef, AbortRep) \wedge \\ RTD\_tr_3 = non \end{array} \right) \end{array} \right) $$

Résumé

La puissance des outils informatiques permet aujourd'hui à des logiciels dont le fonctionnement est de plus en plus sensible, de gérer des systèmes de plus en plus complexes. La sûreté du déroulement de ces applications est un problème sérieux car le dysfonctionnement de tels programmes peut avoir des conséquences disproportionnées par rapport aux moyens mis en oeuvre. Pour toutes ces raisons, les environnements proposant de valider le fonctionnement d'un logiciel sont actuellement en plein essor.

Mon travail est situé dans le cadre du développement d'une extension de B par des propriétés dynamiques exprimées en Logique Temporelle Linéraire. Le but de cette étude est de dégager, à partir de plusieurs exemples, les formes de propriétés dynamiques ainsi que leurs possibilités d'évolution durant les raffinements composant les spécifications. Un autre aspect du travail réalisé est le développement d'un algorithme permettant de vérifier l'existence d'un raffinement entre deux graphes d'accessibilité.

Mots clés

Validation de logiciels
Systèmes réactifs
Logique Temporelle Linéaire
Spécification
Raffinements
Conception
Vérification
Graphes d'accessibilité