UNIVERSITÉ ABDEHAMID IBN BADIS MOSTAGANEM

N° attribué par la bibliothèque

THÈSE

pour obtenir le grade de

Docteur

Spécialité : **Apprentissage Automatique et Web Intelligent**

préparée au laboratoire **ACADIE**

dans le cadre de l'École Doctorale **Université Abdelhamid Ibn Badis Mostaganem**

présentée et soutenue publiquement
par

Sarah BENYAGOUB

le date

Titre:

Vers une évolution cohérente des chorégraphies par les méthodes formelles

Directeur de thèse: **Ahmed MEDAGHRI**
Co-directeur de thèse: **Yamine AIT AMEUR et Meriem OUEDERNI**

Jury

Amir Abdessamad. Professeur,Université de Mostaganem,
Sehaba Karim. Maitre de Conférences A; Université de Mostaganem,
Dominique MERY. Professeur, LORIA Université de Nancy France,
Ait Ameur Yamine. Professeur, ENSEEIHT Toulouse France,
Medeghri Ahmed. Professeur Université de Mostaganem Abdelhamid Ibn Badis,
Ouederni Meriem. Maitre de Conférences A, ENSEEIHT Toulouse France,
André-Luc BEYLOT. Professeur l'ENSEEIHT Toulouse France,
Ladjel Bellatreche. Professeur, Université LIAS/ISAE-ENSMA POITIERS France,
Henni Fouad. Maitre de Conférences B; Université de Mostaganem,

Résumé

Les systèmes contemporains complexes basés sur les interactions sont souvent construits en réutilisant des entités de communications distribués existants qui doivent se coordonner pour répondre aux exigences du client, du système et de l'environnement. Dans cette thèse, nous abordons la conception de systèmes distribués composés d'entités (systèmes de transitions d'états) communiquant via des échanges de messages. Nous considérons les chorégraphies comme le modèle formel permettant au développeur de décrire et de spécifier la coordination entre entités comme un ensemble de conversations, c'est-à-dire toutes les séquences de messages échangées entre les entités en communication. En procédant ainsi, ne nécessite pas de construction des entités individuels ni de leur composition car ils peuvent être obtenus par la projection de la chorégraphique. La rectitude de la conservation de tels messages échangés par chaque entité, obtenus par projection, est une problématique majeur, connu sous le nom de problème de réalisabilité. Il est impératif de vérifier la réalisabilité de la chorégraphie pour créer des applications tierces sans erreur de coordination, par exemple, l'absence d'interblocages, la perte et le désordre des messages, ... Dans nos travaux [18–20], nous avons proposé un ensemble d'opérateurs de composition permettant aux concepteurs de construire des chorégraphies **réalisables** *décrites sous forme de protocole de conversation. La réalisabilité est garantie par construction. Nous nous appuyons sur la méthode B-événementiel, correcte-par-construction, pour prouver que chaque protocole de conversation construit avec nos opérateurs est réalisable. Tel que, notre approche s'applique et évolue à un ensemble de cas d'utilisation empruntés à la littérature et utilisés par la communauté des chercheurs. Notre approche permet également de détecter les défaillances et que la récupération après défaillance n'est pas réalisable.*

Mots clés. *Réalisabilité de la chorégraphie, protocoles de conversation, méthodes correctes-par-construction, B-événementiel, méthodes de vérification et de raffinement, systèmes répartis.*

Abstract

Contemporary interaction-based complex systems are often built by reusing existing distributed peers which have to coordinate with each other to fulfill the client, system, and environment requirements. In this thesis, we address the design of distributed systems composed of peers (state-transitions systems) communicating through message exchanges. We consider choreographies as the formal model allowing a developer to describe and specify peers coordination as a set of conversations, i.e., all sequences of messages exchanged between the communicating peers. Proceeding this way neither require building the individual peers nor their composition as they may be obtained by the choreography projection. The correctness of the preservation of such messages exchanges by each peer obtained after projection is a key issue, known as the realizability problem. Checking choreography realizability is mandatory to build third-party applications with no coordination error, e.g., absence of deadlocks, missing messages, and erroneous messaging order. In our works [18–20], we have proposed a set of composition operators allowing designers to build **realizable** *choreographies that are represented by conversation protocols. Realizability is guaranteed by construction. We rely on the correct-by-construction Event-B method to prove that each CP constructed using our operators is realizable. Our approach applies and scales to a set of use cases borrowed from the literature and used by the research community. Our approach allows also to detect failures and failure recovery in case realizability does not hold.*

Keywords. *Choreography Realizability, Conversation Protocols, Correct-by-Construction, Event-B, Proof and Refinement-based Methods, Distributed Systems.*

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1 Contexte de l'étude

Les systèmes informatiques consistent généralement en un ensemble de composants individuels qui interagissent les uns avec les autres pour accomplir un but commun, calculant par exemple un résultat ou prenant une décision con-sensuelle. Les composants n'ont pas l'accès direct à une connaissance partagée et doivent échanger des informations l'un entre l'autre pour atteindre leur but. Ce qui est intéressant c'est que, les situations de la vie quotidienne nous rappellent souvent la difficulté de cette tâche. Les amis ont en effet toujours eu du mal à s'accorder sur une heure de rendez-vous pour une soirée au cinéma ; certains ne reçoivent jamais la dernière mise à jour à temps ; d'autres sont confrontés à des informations contradictoires ; et des amis trop polis ont parfois raté le début du spectacle en s'attendant les uns les autres pour franchir les portes du théâtre en premier. Tous ces problèmes trouvent des échos dans les systèmes distribués réels comme des incompatibilités difficiles et stimulantes. La garantie d'une collection de composants compatibles est en effet loin d'être trivial. Avec l'avènement de l'informatique en nuage (cloud computing), de l'internet des objets (Internet of Things) et l'interconnexion des objectifs informatiques allant des bases de données volumineux centralisées aux minuscules appareils de cuisine, les systèmes distribués acquièrent une influence et un effet encore plus fort sur nos vies. En conséquence, assurant leur fonctionnement approprié est crucial. Un simple défaut peut ainsi entraîner des complication physique, environnementale, économique, ou même mortelle. De nombreux systèmes critiques sont en effet intrinsèquement distribués : les exemples classiques incluent les capteurs et les unités de calcul dans les avions ou les voitures autonomes, les réseaux de télécommunication d'urgence ou le traitement parallèle des données médicales. Les méthodes

formelles proposent de répondre à ces exigences de sécurité et de performance en instrumentant des structures et des preuves mathématiques pour décrire, modéliser et raisonner la conception des systèmes de manière à minimiser les risques de bugs et la charge de dépannage. Ils proposent de certifier que les systèmes sont effectivement conformes à un ensemble de spécifications et de propriétés de comportement correct. À cet égard, Ce travail contribue à la formalisation d'un domaine d'interaction spécifique dans les systèmes distribués, appelé interactions asynchrones.

2 Position du Problème

Dans une conception descendante des systèmes distribués, les interactions entre entités communicantes est généralement définie à l'aide d'une spécification globale appelée chorégraphie ou protocoles de conversation (CP). Ce modèle spécifie les interactions comportementales entre les entités en décrivant l'ordre des messages échangés.

Étant donné un protocole de conversation, il ya quelques questions difficiles à aborder, à savoir, est-ce qu'il existe un ensemble d'entités distribués où leur comportement composé est le même spécifié par le protocole de conversation ? Ceci est appelé un problème de « Réalisabilité ».

Étant donné que les spécifications du système peuvent se développer avec le temps, comment pouvons-nous assurer la cohérence de l'évolution ? Comment cela peut être vérifié et prouvé formellement ? De plus, à partir d'un protocole de conversation réalisable qui a été mis à jour pour certaines raisons, comment la réalisabilité peut être vérifiée (partiellement), après la mise à jour du système ? Sinon, nous supposons qu'il existe un ensemble d'entités communicantes qui mettent en œuvre un protocole de conversation, et les comportements locaux des entités formant le *CP* ont été modifiés par leurs fournisseurs. Comment cette évolution peut être refléter dans la spécification globale, c.à.d le protocole de conversation ? de tel sorte que toutes les erreurs de coordination doivent être évitées.

Une problématique principale est déjà abordée par la communauté de la recherche, qui est : la vérification de la réalisabilité des *CPs*. Cela se réfère à la vérification d'existence d'un ensemble d'entités communicantes où leur composition génère les mêmes séquences de messages que celles spécifiées par le *CP*. Considérant un système asynchrone, le problème de réalisabilité est indécidable [26] avec un tel mode de communication, en raison du mécanisme de file d'attente illimité (non bornées). Le travail récent de [10] a proposé une condition nécessaire et suffisante pour vérifier la possibilité de composition de *CP* à partir d'un ensemble d'entités communicants de manière asynchrone en utilisant des files d'attentes de taille illimités. Ce travail résout le problème de la réalisabilité pour une sous-classe d'entités d'un système asynchrones, à savoir, les systèmes synchronisables, c'est-à-dire que le système composé de ces entités se comportent de la même façon en passant d'une communication synchrone à une autre asynchrone. Un *CP* est réalisable s'il existe un ensemble d'entités implémentant ce *CP*, autrement dit, les entités formant le *CP* envoient des messages les uns aux autres dans le même ordre que le *CP*, et leur composition est synchronisable. Dans [10], la vérification de la réalisabilité du *CP* suit les étapes suivantes : i) la projection des entités communicantes à partir du *CP*, ii) la vérification de la synchronisation ç.à.d l'équivalence entre les compositions

synchrone et asynchrone des entités projetées, iii) la vérification d'équivalence entre le *CP* et les compositions synchrone et asynchrone des entités projetées, iv) ainsi que l'ensemble des files d'attente utilisées par les entités communicantes doivent être vide à la fin de la recomposition du système. Ce travail propose une vérification des systèmes asynchrone de taille raisonnable de point de vu nombre : d'états, transitions et entités communicantes.

3 Contributions

Nous considérons un système distribué par son protocole de conversation indépendamment des langages de chorégraphies (CDL, BPMN, ...). Le *CP* spécifie le niveau conversationnel du système tel que, la Figure 1.1 présente un aperçu de nos contributions :

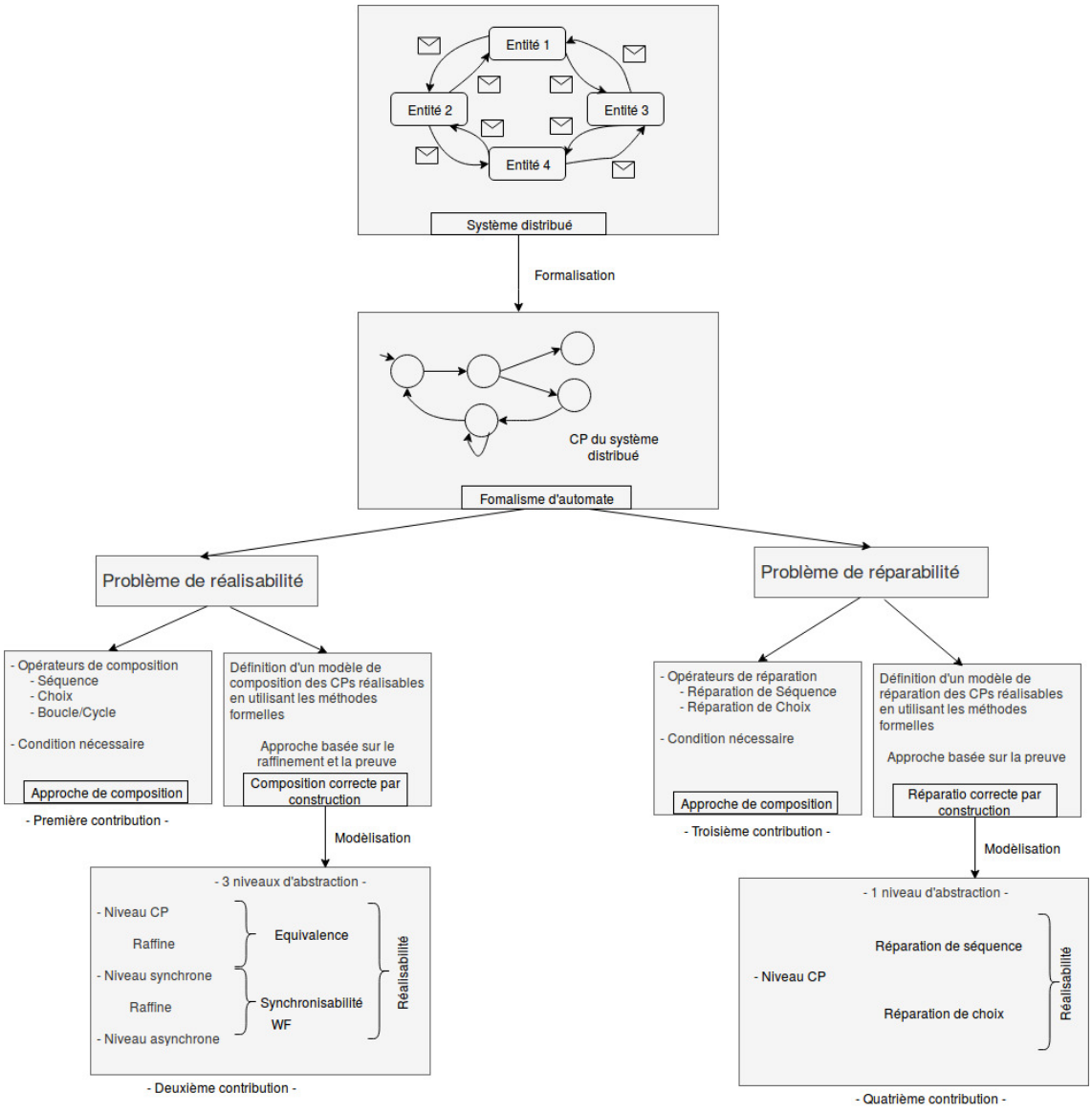


FIGURE *1.1 – Aperçu des contributions.*

Nous contributions sont présentées sur trois principaux volets :

— **Langage de composition correcte-par-construction**, où nous traitons la conception descendante des *CPs* réalisables suivant une méthode correcte par construction, ç.à.d, au lieu d'utiliser une approche a posteriori telle que réalisée par les travaux existant où la réalisabilité est vérifiée sur un *CP* entier donné (entités communicantes, traces de la composition synchrone et asynchrone du système), ce qui complique les étapes de vérification. Nous proposons de vérifier la réalisabilité au niveau *CP* indépendemment du comportement séparé des entités communicantes.
Notre objectif est de simplifier et, en conséquence, de réduire considérablement la complexité de la tâche de vérification. Ainsi, nous économisons des ressources (temps et mémoire) et les efforts nécessaires pour vérifier les systèmes complexes du monde réel. C'est pourquoi nous proposons une méthode a priori correcte-par-construction pour composer incrémentalement des *CPs* réalisables à partir d'un *CP* vide.
Pour se faire, nous définissons un nouveau langage de composition en utilisant une algèbre d'opérateurs, à savoir les opérateurs de séquence, de choix et de boucle (cycle). À partir d'un *CP* initial de base (*CP* avec une seul transition), nous composons incrémentalement un *CP* réalisable en ajoutant a un *CP* de départ des *CPs* basique où la composition préserve toujours la réalisation du *CP* résultat.
Nous avons identifié des conditions suffisantes relative a chacun des opérateurs, afin de garantir la réalisabilité du système composé.

— **Modélisation et preuve**, où nous présentons sous la forme d'une architecture de plusieurs niveaux de modélisations. Partant d'un modèle de conversation abstrait où nous identifions un modèle générique pour chaque opérateur de composition, tel que, si les conditions suffisantes sont satisfaites, cela implique une préservation total de réalisabilité du système.
Nous dégageons, par raffinement et preuve de théorème, un premier niveau concret, exprimant un modèle d'interaction synchrone. Ensuite, un deuxième niveau plus concret, obtenu aussi par raffinement et preuves, donnant une concrétisation asynchrone du protocole de la conversation.

— **Réparation**, où nous étudions la possibilité de réparation de chorégraphies non réalisables, l'objectif étant d'identifier un ensemble de modifications apportées à une chorégraphie non réalisable afin de restaurer la réalisablité, tout en se basant sur notre algèbre définit pour la composition correcte-par-construction des *CPs* réalisable. Nous présentons une technique de réparation automatique des chorégraphies non réalisables et fournissons des garanties formelles d'exactitude.

4 Organisation du manuscrit

Le contenu du manuscrit de thèse est organisé comme suit :
— La première partie est composée de trois chapitre.

Le premier chapitre résume les formalismes en relation avec les travaux réalisés comme suit :

— Nous commençons par le formalisme du protocole de conversation dans lequel nous définissons les entités communicantes, les *CPs*, la projection (le comportement de chacune des entités communicantes) ainsi que les deux mode de communication des systèmes synchrone et asynchrone.

— Nous passons ensuite a l'exposition des méthodes formelles tel que nous présentons la logique de B-événementiel, les obligations de preuve l'animation par ProB ainsi qu'un exemple du modèle en B-événementiel.

— Le dernier chapitre de la première partie concerne l'étude de l'état de l'art des travaux qui traitent des problématiques similaires, elle est présenté sur trois parties :

  — La première partie est dédié aux approches de vérification et validation de la réalisabilité.
  — La deuxième concerne les approches d'évolution des *CPs*.
  — La dernière est consacré aux approches correctes-par-construction.
  — Nous concluons le chapitre par la comparaison et la définition des limites de chaque catégorie des approches présentées.

— La deuxième partie comporte les détails de nos contributions, tel que :

— La première partie concerne notre approche correcte-par-construction où nous définissons l'ensemble des propriétés et opérateurs de composition des *CPs* réalisables. La validation des définitions et l'exactitude de l'approche seront prouvées par des théorèmes et des preuves mathématique.

— Nous détaillons ensuite notre seconde contribution qui est la validation de notre approche de composition correcte-par-construction. Tel que les démarches de modélisation adoptée ainsi que les types de raffinage suivi seront donné en détail. Nous expliquons ensuite toutes les déclarations formelles utilisés dans la construction du modèle sur les différents niveaux de raffinement. Nous validons aussi notre proposition avec un cas d'étude réel réalisable emprunté de la littérature.

— Nous présentons ensuite notre troisième contribution, qui est une stratégie de réparation incrémentale, où nous présentons le principe de réparation baser sur notre langage de composition ainsi que la preuve formelle de la proposition, effectuée en implémentant et prouvant la stratégie de réparation en B-événementiel. Nous validons aussi notre proposition avec un cas d'étude réel identifié comme non-réalisable, emprunté de la littérature.

— Des implémentations, des tests et des validations de nos deux contributions seront donné tel que, l'application des stratégies de composition

ainsi que la réparation sur plusieurs cas d'utilisation réels sont présentés en détail.

— La dernière partie est la conclusion générale qui présente une synthèse des travaux réalisés. Il dégage aussi les perspectives importantes qui peuvent être une suite à ce travail.

5 Diffusion scientifique

Les travaux présentés dans ce manuscrit ont fait l'objet de plusieurs publications listées ci-dessous.

Revue scientifique avec comité de lecture

— S. Benyagoub, Y. Aït-Ameur, M.Ouederni, A. Mashkoor and A.Medeghri. Formal design of scalable conversation protocols using Event-B : Validation, experiments and benchmarks. *Journal of Software : Evolution and Process, 2019. to appear.*
— COMLAN

Conférence internationale avec comité de lecture

— Sarah Benyagoub, Yamine Aït Ameur, Meriem Ouederni, Atif Mashkoor : Scalable Correct-by-Construction Conversation Protocols with Event-B : Validation, Experiments and Benchmarks. *ICECCS 2018 : 209-212*
— Sarah Benyagoub, Yamine Aït Ameur, Meriem Ouederni, Atif Mashkoor : Handling Reparation in Incremental Construction of Realizable Conversation Protocols. *MEDI Workshops 2018 : 159-166*
— Sarah Benyagoub, Meriem Ouederni, Yamine Aït Ameur, Atif Mashkoor : Incremental Construction of Realizable Choreographies. *NFM 2018 : 1-19*
— Sarah Benyagoub, Meriem Ouederni, Neeraj Kumar Singh, Yamine Aït Ameur : Correct-by-Construction Evolution of Realisable Conversation Protocols. *MEDI 2016 : 260-273*
— Sarah Benyagoub, Meriem Ouederni, Yamine Aït Ameur : Towards correct Evolution of Conversation Protocols. 2016 : 193-201

Première partie

Contexte

Chapitre 2

Formalisme d'automates communicants

Sommaire

La contribution principale de ce chapitre est la formalisation du modèle du protocole de communication ainsi que leurs comportement dans les deux modes de communications en ligne et hors ligne.

1 Modèle du protocole de communication (CP)

Nous formalisons ci-dessous le modèle pour l'entité communicante ($\mathcal{P}$) (Définition 1), l'élément de base de nos protocoles de conversation (CP) (Définition 2), ainsi que la formalisation des traces du ($\mathcal{P}$) et (CP) (Définition 3). Nous considérons dans nos travaux de thèse les protocoles de conversation déterministe définie dans (Définition 4). Nous définissons aussi le protocole de conversation basique formé de deux états et une seule transition Définition 5. Le comportement des entités communicantes est calculé a partir (CP), en appliquant la fonction de $Projection$ (Définition 6). Sachant que, à partir de l'ensemble des entités calculés, nous pouvons recomposer le système en mode de communication synchrone dans le cas d'envoi-réception ou asynchrone en utilisant des files d'attente de taille illimité dans le cas de plusieurs envois puis réception. La définition des deux modes de communications est donnée dans les Définitions 7, 8.

Nous utilisons des systèmes de transitions étiqueté (LTS) pour modéliser le protocole de conversation (CP) et les entités communicantes ($\mathcal{P}$) incluses dans cette spécification. Ce modèle de comportement définit l'ordre des messages envoyés dans le CP. Au niveau des entités communicantes, le LTS peut être calculé par projection à partir de CP et décrit l'ordre dans lequel ces entités déclenchent leurs actions d'envoi et de réception.

Définition 1 (entité communicante ($\mathcal{P}$)) *Une entité communicante est un quadruplet $\mathcal{P} = (S, s^0, \Sigma, T)$ tel que, S est l'ensemble fini des états, $s^0 \in S$ est l'état initiale du système, $\Sigma = \Sigma^! \cup \Sigma^? \cup \{\tau\}$ est l'ensemble fini d'alphabet divisé en un ensemble de messages envoyés $\{m!, m'!, \ldots\}$, l'ensemble des messages reçus $\{m?, m'?, \ldots\}$ et*

l'ensemble des actions internes τ (les transitions vides), et $T \subseteq S \times \Sigma \times S$ est la relation de transition.

Nous nous référons à $t = s \xrightarrow{l} s'$, où $l \in \Sigma$ *et* $t \in T$, en tant que transition d'envoi si $l = m!$ ($m! \in \Sigma^!$) et comme transition de réception si $l = m?$ ($m? \in \Sigma^?$). Notez que nous appelons un état s^f *dans* S^f final si l'entité correspondante peut arrêter de fonctionner à cet état.

Définition 2 (Protocole de conversation (CP)) *Un protocole de conversation CP pour un ensemble d'entités communicantes $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\}$ est un système de transition étiqueté $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ où S_{cp} est un ensemble fini d'états globaux et s^0_{CP} est l'état initial global, L_{CP} est l'ensemble des étiquettes où chaque étiquette est dénoté par $m^{\mathcal{P}_i,\mathcal{P}_j}$ tel que $\mathcal{P}_i$ et $\mathcal{P}_j$ sont, respectivement, les entités émettrices et réceptrices du message m avec $\mathcal{P}_i \neq \mathcal{P}_j$. Enfin $T_{CP} \subseteq S_{CP} \times L_{CP} \times S_{CP}$ est la relation de transition.*

Nous exigeons que chaque message ait un expéditeur et un destinataire uniques : $\forall\{(\mathcal{P}_i, m, \mathcal{P}_j), (\mathcal{P}'_i, m', \mathcal{P}'_j)\} \subseteq L_{CP} : m = m' \implies \mathcal{P}_i = \mathcal{P}'_i \wedge \mathcal{P}_j = \mathcal{P}'_j$.

Nous utilisons les notations suivantes dans la suite :

- $t = s \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s'$ *indique une transition $t \in T_{CP}$ (également appelée transition envoi-réception) où s et s' sont les états source et cible et $m^{\mathcal{P}_i \to \mathcal{P}_j}$ est l'étiquette de transition, et*
- $S^f_{CP} \neq \varnothing$ *et $s^f_{CP} \subseteq S_{CP}$ dénotes l'ensemble non-vide d'états finaux, ç.a.d., un état dans lequel le système peut cesser de fonctionner.*

Définition 3 (Traces) *Les traces sont définies aux deux niveaux CP et $\mathcal{P}$, respectivement, comme suit :*

Traces de CP.

- *Pour un $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ donné, une trace est une séquence finie de messages d'envoi-réception de la forme $m^{\mathcal{P}_i \to \mathcal{P}_j}$ où il existe $\{s_{CP} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s'_{CP}\} \subseteq T_{CP}$.*
- *Tr_{CP} désigne l'ensemble de ces traces où $\{s_0 \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_1, \ldots, s_n \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s_{n+1}\} \subseteq Tr_{CP}$. s_i est le i^{eme} état d'une trace de séquence dans Tr_{CP}. Lorsque s_{n+1} est un état final, la trace est dite complète.*

Traces d'entité communicante.

- *Pour une entité donné $\mathcal{P} = (S, s^0, \Sigma, T)$, une trace est une séquence finie d'envoi (de la forme m!) ou de la réception (de la forme m?) correspondant aux libellés des transitions dans $\mathcal{P}$ tel que $s_{\mathcal{P}} \xrightarrow{m!} s'_{\mathcal{P}} \subseteq T$ est valide ($s_{\mathcal{P}} \xrightarrow{m?} s'_{\mathcal{P}}$ détient, respectivement).*
- *Tr_k désigne l'ensemble de ces traces où $\{s^k_0 \xrightarrow{l} s^k_1, \ldots, s^k_n \xrightarrow{l'} s^k_{n+1}\} \subseteq Tr_k$. s^i_n est le i^{eme} état d'une trace de séquence dans Tr_k et $l \in \Sigma$, $l = m!$ ou $l = m?$. Lorsque s_{n+1} (resp. S^k_{n+1}) est un état final, la trace est dite complète.*

Définition 4 (Choix Déterministe (DC)) *Nous considérons les protocoles de conversation déterministe, ı e des CPs qui réagissent toujours de la même façon à un*

événement, c'est-à-dire que, quoi qu'il se soit passé auparavant, à partir du moment où le CP arrive dans un état donné, son évolution sera toujours identique. En automatique, les systèmes déterministes forment une classe de systèmes qui, à une suite d'événements entrants, produit une suite d'événements sortants, selon un ordre déterminé par l'ordre des événements entrants. Formellement, nous définissons un CP déterministe comme suit.

Soit DC l'ensemble des CPs déterministes. Pour tout protocole de co-nversation CP, on dit que $CP \in DC$ si et seulement si $\forall s_{CP} \in S_{CP}$, $\nexists\{s_{CP} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s'_{CP}, s_{CP} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s''_{CP}\} \subseteq T_{CP}$ tel que $s'_{CP} \neq s''_{CP}$.

Définition 5 (CP basique) *Un CP_b est un CP avec une seule transition définie comme $CP_b =< S_{CP_b}, s^0_{CP_b}, L_{CP_b}, T_{CP_b} >$ avec $T_{CP_b} = \{s^0_{CP_b} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s^f_{CP_b}\}$ tel que $s^0_{CP_b} \neq s^f_{CP_b}$.*

Supposons donc que nous disposons de n entités capables de communiquer. chacune de ces entités, que nous noterons $\mathcal{P}$, reçoit et envoie un ensemble de messages échangés entre les n composants (incluant elle-même) du protocole de conversation. Il existe plusieurs stratégies pour calculer le comportement des entités communicantes à partir du CP.

La stratégie *Découpage statique*, la stratégie *Maître-esclave*, la stratégie *Work stealing*, ... etc. Notre fonction de projection est présentée formellement comme suit.

Définition 6 (Projection) *La projection $\downarrow CP$ d'un protocole de conversation CP renvoie un ensemble d'entités $\Gamma = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ où chaque entité $\mathcal{P}_i, i \in [1..n], n \in \mathbb{N}$, est un LTS, $\mathcal{P}_i =< S_i, s^0_i, \Sigma_i, T_i >$ obtenu en remplaçant dans $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ chaque étiquette par :*

- *$m!$, $m! \in \Sigma_j$ si $j = i$*
- *$m?$, $m? \in \Sigma_k$ si $k = i$*
- *τ (action interne) sinon.*

Des algorithmes de détermination et de minimisation [60] peuvent être appliqués à chaque entité obtenue pour supprimer les actions internes.

2 Système de composition

Notre travail repose sur deux modèles de communication. Le modèle synchrone [96] de protocoles d'interaction des entités communicantes, et le modèle asynchrone [27]. La sémantique synchrone nous permet de simplifier la spécification de protocole en évitant la description des canaux de message (file d'attente). Dans une communication binaire, cette sémantique exige que les deux entités s'accordent sur l'ordre des messages envoyés et reçus pour passer d'un état à un autre. En faisant cela, cette exigence facilite le raisonnement sur les LTSs des entités communicantes en évitant les problèmes d'indécidabilité de la sémantique asynchrone (blocage, perte d'information, désordre des messages échangés, ..., etc). Bien que le modèle de communication asynchrone conduise à plusieurs limites de spécification et de vérification, cette sémantique peut être utile dans des cas réels. Il est donc important de

chercher des solutions à ces cas. Dans ce qui suit, nous définissons d'abord le modèle de communication synchrone pour nos LTSs, que nous appliquons dans nos techniques de vérification, puis nous discutons de la possibilité de résoudre la vérification des systèmes fonctionnant en sémantique asynchrone.

Définition 7 (CP synchrone) *Considérant la sémantique synchrone, les systèmes reposent sur des communications binaires de rendez-vous. Une entités communicante ne peut envoyer un message m à son partenaire que si ce dernier est dans un état lui permettant d'envoyer m, et si son partenaire est dans un état lui permettant de recevoir ce message. Par conséquence, aucun canal de message n'est requis.*

Pour un ensemble d'entités $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *avec* $\mathcal{P}_i = (S_i, s_i^0, \Sigma_i, T_i)$*, le modèle CP synchrone* $Sys_{sync}(\Gamma) = (S_{sync}, s_{sync}^0, \Sigma_{sync}, T_{sync})$ *tel que :*

- $S_{sync} \subseteq S_1 \times \ldots \times S_n$
- $s_{sync}^0 \in S$ *tel que* $s^0 = (s_1^0, \ldots, s_n^0)$
- $\Sigma_{sync} = \cup_i \Sigma_i$
- $T_{sync} \subseteq S \times \Sigma \times S$*, et pour* $s = (s_1, \ldots, s_n) \in S$ *et* $s' = (s'_1, \ldots, s'_n) \in S$ *(interact)* $s_{sync} \xrightarrow{m} s'_{sync} \in T_{sync}$ *if* $\exists i, j \in \{1, \ldots, n\}, i \neq j$*,* $m \in \Sigma$ *avec* $m! \in \Sigma_i^!$*,* $m? \in \Sigma_j^?$ *où* $\exists\ s_i \xrightarrow{m!} s'_i \in T_i$*, et* $s_j \xrightarrow{m?} s'_j \in T_j$ *tel que* $\forall k \in \{1, \ldots, n\}, k \neq i \wedge k \neq j \Rightarrow s'_k = s_k$

Définition 8 (CP asynchrone) *De nombreux systèmes interagissent de manière asynchrone en envoyant et en recevant des messages sur des canaux de communication (non)bornés. Sous la sémantique asynchrone, une entité communicante peut envoyer un message m lorsqu'elle se trouve dans un état permettant l'envoi de m, même si son partenaire n'est pas dans un état lui permettant de recevoir m. Dans ce modèle, chaque entité a un canal (ou plusieurs, par exemple un pour les messages d'entrée et un pour les messages de sortie) qui stocke les messages qui lui ont été envoyés. Un message peut être consommé depuis un canal chaque fois que le service de canal est dans un état dans lequel ce message peut être reçu. Si une communication asynchrone est utilisée, l'espace d'état du système global peut être infini même si les entités communicantes impliquées sont initialement décrites à l'aide d'un espace d'état fini. Par conséquent, dans ce contexte, la vérification des propriétés du système, par exemple, l'atteignabilité ou l'absence du blocage, est considérée comme étant indécidable [27].*

Les systèmes à communication asynchrone donnent lieu à plusieurs limites de vérification, notant par exemple les travaux récents proposés dans [11,47] qui montrent qu'une grande classe de services communiquant sous une sémantique asynchrone peuvent être analysés à l'aide de techniques et d'outils existants pour le modèle de communication synchrone.

Un ensemble d'entités communicantes est dit synchronisable si et seulement si l'ordre des échanges de messages reste le même lorsque la communication asynchrone est remplacée par la communication synchrone.

Par conséquent, si un système est synchronisable, la vérification peut être effectuée sur la version synchrone du système et les résultats sont conservés pour le cas asynchrone. La synchronisabilité peut être vérifiée à l'aide de techniques de vérification d'équivalence mises en œuvre dans la boîte à outils du CADP.

En conséquence, nous considérons deux entités communiquant de manière asynchrone comme compatible, s'ils sont compatibles sous la sémantique synchrone et

qu'ils sont synchronisables en utilisant les techniques proposées dans [11, 47]. Cette hypothèse nous permet de vérifier la compatibilité asynchrone d'une sous-classe d'entités respectant les exigences de synchronisabilité.

Pour un ensemble d'entités $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *avec* $\mathcal{P}_i = (S_i, s_i^0, \Sigma_i, T_i)$, *et* Q_i *étant sa file d'attente associé, la composition asynchrone* Sys_{async} *est* $(S_a, s_a^0, \Sigma_a, T_a)$ *où :*

1. $S_a \subseteq S_1 \times Q_1 \times \ldots \times S_n \times Q_n$ *où* $\forall i \in \{1, \ldots, n\}$, $Q_i \subseteq (\Sigma_i^?)*$
2. $s_a^0 \in S_a$ *tel que* $s_a^0 = (s_1^0, \epsilon, \ldots, s_n^0, \epsilon)$ *(où* ϵ *Désigne une file d'attente vide)*
3. $\Sigma_a = \cup_i \Sigma_i$
4. $T_a \subseteq S_a \times \Sigma_a \times S_a$, *et pour* $s = (s_1, Q_1, \ldots, s_n, Q_n) \in S_a$ *et* $s' = (s'_1, Q'_1, \ldots s'_n, Q'_n) \in S_a$

(Envoi) $s \xrightarrow{m!} s' \in T_a$ *if* $\exists i, j \in \{1, \ldots, n\}$ *où* $i \neq j : m \in \Sigma_i^! \cap \Sigma_j^?$, *(i)* $s_i \xrightarrow{m!} s'_i \in T_i$, *(ii)* $Q'_j = Q_j m$, *(iii)* $\forall k \in \{1, \ldots, n\} : k \neq j \Rightarrow Q'_k = Q_k$, *et (iv)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

(consommation) $s \xrightarrow{\tau} s' \in T_a$ *if* $\exists i \in \{1, \ldots, n\} : m \in \Sigma_i^?$, *(i)* $s_i \xrightarrow{m?} s'_i \in T_i$, *(ii)* $mQ'_i = Q_i$, *(iii)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow Q'_k = Q_k$, *et (iv)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

(interaction interne) $s \xrightarrow{\tau} s' \in T_a$ *if* $\exists i \in \{1, \ldots, n\}$, *(i)* $s_i \xrightarrow{\tau} s'_i \in T_i$, *(ii)* $\forall k \in \{1, \ldots, n\} : Q'_k = Q_k$, *et (iii)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

Chapitre 3

Modélisation de système avec B-événementiel

Sommaire

Cette thèse vise la modélisation et la vérification de systèmes composés d'entités communicantes pouvant échanger des messages, en mode hors ligne ou en ligne. Ces échanges peuvent être bien modélisés à l'aide de changements d'état du système. Nous avons donc décidé de nous baser sur des systèmes de transition d'états comme modèle de formalisation, sur la méthode B-événementiel et sur la plate-forme Rodin pour la modélisation du système et les preuves de satisfaction des exigences structurées à l'aide des raffinements. Nous présentons dans ce qui suit l'ensemble de ces techniques formelles.

1 Modèles de systèmes

Les systèmes de transition ont été identifiés comme un modèle générique approprié pour les systèmes. Ils prennent en charge la définition des systèmes et leur comportement et permettent aux développeurs de raisonner sur leurs traces d'exécution. Une des méthodologies de conception associées aux systèmes de transition consiste à décrire une séquence st_i de ces systèmes où st_i raffine st_{i-1}. Le raffinement introduit de plus en plus de détails passant d'un système abstrait à un système plus concret. De plus, nous ciblons la définition de systèmes correctes éventuellement paramétrés. Par conséquent, il est nécessaire de prouver l'exactitude des modèles conçus au-delà des tests (partiels) ou de la vérification des modèles bornés.

Plusieurs méthodes formelles ont été proposées dans la littérature pour définir et modéliser de tels systèmes. La première classe de méthodes formelles est basée sur la définition des algèbres de processus CCS [75] ou LOTOS [40, 72]. Ces techniques n'offrent pas d'opérations de raffinement. Nous ne les avons donc pas considérées dans notre travail.

La deuxième classe de méthodes formelles sont appelées méthodes formelles basées sur un état. Ces méthodes ont attiré l'attention de plusieurs chercheurs. Elles reposent sur la définition d'états de systèmes (via un ensemble de variables d'état) et de transitions (d'un état à l'autre) dotées en général de pré-conditions et post-conditions [57] permettant d'offrir des capacités de raisonnement. De plus, ces modèles formels peuvent être associés à un mécanisme de raffinement permettant de définir une séquence de modèles liés par cette relation. Parmi ces méthodes, on peut citer Z [91, 93], VDM [23], B [1], TLA$^+$ [65], B-événementiel [4] et Statecharts [53]. Dans les développements récents, ces méthodes ont été associées à plusieurs techniques et outils de vérification de modèles offrant des capacités de vérification et/ou d'animation de modèles. Des exemples de vérificateurs de ce type de modèles sont NuSMV [28], CADP [49], PROMELA/SPIN [58], ProB [69] et TINA [21].

Une troisième classe de méthodes formelles concerne les «méthodes formelles d'ordre supérieur». Grâce à leurs caractéristiques d'ordre supérieur, ces méthodes permettent de décrire les modèles de système et la procédure de vérification associée de manière uniforme. Les méthodes pourraient être utilisés à un niveau «méta» : ils auraient besoin d'un codage des notions d'état et de transition à l'aide de fonctions d'ordre supérieur. Ces méthodes sont Isabelle/HOL [76], PVS [77] ou Coq [9, 22].

Afin de bénéficier d'une méthodologie basée sur les notions d'état, de transition, de raffinement, de preuves et de la disponibilité d'un puissant outil de support (la plate-forme Rodin), nous avons choisi la méthode formelle B-événementiel pour exprimer nos modèles et prouver les propriétés associées.

La méthode B-événementiel [4] est une évolution récente de la méthode B [1]. Cette méthode est basée sur les notions de pré-conditions et post-conditions de Hoare [57], la pré-condition la plus faible de Dijkstra [39] et le calcul de substitution [1]. C'est une méthode formelle basée sur des fondements mathématiques : logique du premier ordre et théorie des ensembles.

2 Modèles B-événementiel

Une machine B-événementiel est composé de plusieurs composants de deux types : les machines et les contextes. Les machines contiennent les parties dynamiques (états et transitions) d'un modèle, tandis que les contextes contiennent les parties statiques (axiomatisation et théories) d'un modèle. Une machine peut être raffinée par une autre et un contexte peut être étendu par un autre contexte. De plus, une machine peut *voir* un ou plusieurs contextes.

Un modèle B-événementiel est caractérisé par un ensemble de variables, définies dans la clause $Variables$ et évoluant grâce aux événements définis dans la clause $Events$. Il code un système de transition d'état où les variables représentent l'état

et les événements représentent les transitions d'un état à un autre. Au cours de l'exécution, les événements sont entrelacés (c'est-à-dire qu'un seul événement est exécuté).

Un contexte est défini par un ensemble de clauses (Table 1.1) comme suit.

— *Context* représente le nom du composant qui devrait être unique dans un modèle.
— *Extends* déclare le(s) contexte(s) étendu(s) par le contexte décrit.
— *Sets* décrit un ensemble de types abstraits et énumérés.
— *Constants* représente les constantes utilisées par un modèle.
— *Axioms* décrit, dans des expressions logiques de premier ordre, les propriétés (définitions) des attributs déclarés dans les clauses *Constants* et *Sets*. Les types et les contraintes sont également décrits dans cette clause.
— *Theorems* sont des expressions logiques pouvant être déduites des axiomes.

CONTEXT	**MACHINE**
ctxt_id_2	*machine_id_2*
EXTENDS	**REFINES**
ctxt_id_1	*machine_id_1*
SETS	**SEES**
s	*ctxt_id_2*
CONSTANTS	**VARIABLES**
c	v
AXIOMS	**INVARIANTS**
$A(s, c)$	$I(s, c, v)$
THEOREMS	**THEOREMS**
$T_c(s, c)$	$T_m(s, c, v)$
END	**VARIANT**
	$V(s, c, v)$
	EVENTS
	Event *evt*
	any x
	where $G(s, c, v, x)$
	then
	$v :\mid BA(s, c, v, x, v')$
	end
	END

TABLE *3.1 – Structures de contextes et de machines B-événementiel*

De manière similaire aux contextes, les machines sont définies par un ensemble de clauses (Table 3.1).

— *Machine* représente le nom du composant qui devrait être unique dans un modèle.
— *Refines* déclare la machine raffinée par la machine décrite.
— *Sees* déclare la liste des contextes importés par la machine décrite.
— *Variables* représente les variables d'état du modèle de la spécification. Les raffinements peuvent introduire de nouvelles variables afin d'enrichir le système décrit.
— *Invariants* décrivent, à l'aide d'expressions logiques de premier ordre, les propriétés des variables déclarées dans la clause *Variables*. Les types d'informations, les propriétés fonctionnelles et de sécurité sont généralement données dans cette clause. Ces propriétés doivent rester vraies à tout moment. Cela signifie que les invariants doivent être valide à l'initialisation et que les événements (plus précisément leurs actions) doivent les préserver. Cela suffit pour garantir que les invariants sont maintenus par induction mathématique. Il exprime également l'invariant de collage requis par chaque raffinement.

— *Theorems* définissent un ensemble d'expressions logiques pouvant être déduites des invariants et/ou du (des) contexte(s). Ils n'ont pas besoin d'être prouvés pour chaque événement, contrairement aux invariants.
— *Variant* introduit un nombre naturel ou un ensemble fini qui sera utilisé pour garantir les propriétés de la terminaison.
— *Events* définit tous les événements (transitions) pouvant survenir dans un modèle donné. Chaque événement est caractérisé par sa garde et est décrit par un ensemble d'actions. Chaque machine doit contenir un événement d'initialisation. Les événements survenant dans un modèle B-événementiel affectent l'état décrit dans la clause *Variables*.
Un événement comprend les clauses suivantes (Table 1.1) :
— *Refines* déclare la liste des événements raffinés par l'événement décrit.
— *Any* liste les paramètres de l'événement.
— *Where* exprime la condition de l'événement. Un événement peut être déclenché lorsque sa ou ses conditions (guard) est évaluée à vrai. Si plusieurs conditions sont évalués comme étant vrais, uniquement un évènement peut être déclenché avec un choix non-déterministe.
un seul choix peut être déclenché avec un choix non déterministe.
— *Then* Contient les actions de l'événement utilisées pour modifier les variables et donc l'état du système.
Pour modéliser les propriétés de terminaison, les événements sont marqués comme suit :
— *ordinary* : il n'y a pas de restriction concernant la variante,
— *convergent* : la variante doit diminuer,
— *anticipated* : la variante ne doit pas augmenter. Ceci est destiné à être utilisé avec raffinement.
B-événementiel propose trois types d'actions (substitutions) :
— *affectation* (x := E) où la variable devient égale à la valeur d'une expression particulière. Cette action est déterministe.
Exemple : x := 4
— *choix* ($x :\in S$) où la variable prend une valeur d'un ensemble, de manière non déterministe.
Exemple : $x :\in N \setminus \{2\}$
où la variable x prend pour valeur tout nombre naturel autre que 2.
— Le prédicat *avant-après* (x :| BA(x, x')) est la forme d'action la plus générale. Les nouvelles valeurs des variables deviennent telles que le prédicat avant-après donné est valable. Les futures valeurs sont citées, les valeurs actuelles ne le sont pas. C'est la notation la plus puissante puisqu'elle peut exprimer toutes les autres. Il est obligatoire pour exprimer les relations entre les futures valeurs de plusieurs variables dans une action, sinon les actions sont indépendantes. Cependant, en ajoutant des paramètres avec des conditions, la première forme := est suffisante.
Exemple : x,y :| x'>x ∧, x'+y' = 5
Il affirme que x et y prennent des valeurs telles que x devient supérieur à sa valeur précédente et que la somme des nouvelles valeurs de x et y est égale à 5.

3 Règles d'obligation de preuve

Les obligations de preuve (PO) sont associées à tout modèle B-événementiel afin d'exprimer l'exactitude des développements et des raffinements. Ils doivent être prouvés pour assurer l'exactitude du modèle.

Les règles pour générer des obligations de preuve suivent le calcul des substitutions [1, 4], proche du calcul du pré-condition le plus faible de Dijkstra [39]. Afin de définir les règles d'obligation de preuve, nous utilisons les notations définies dans la Table 3.2, où s indique les ensembles vus, c les constantes vues et v les variables de la machine. Les axiomes vus sont désignés par $A(s, c)$ et les théorèmes par $T_c(s, c)$, tandis que les invariants sont désignés par les théorèmes $I(s, c, v)$ et local (spécifiques à l'événement) par $T_m(s, c, v)$. Pour un événement, la condition (guard) est dénotée par $G(s, c, v, x)$ et l'action par le prédicat avant-après $BA(s, c, v, x, v')$. La notation principale v' désigne la variable v après exécution de l'action.

Théorèmes	$A(s,c) \Rightarrow T_c(s,c)$	(a)
	$A(s,c) \wedge I(s,c,v) \Rightarrow T_m(s,c,v)$	(b)
Préservation d'invariant	$A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \wedge BA(s,c,v,x,v') \Rightarrow I(s,c,v')$	(c)
Faisabilité de l'événement	$A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \Rightarrow \exists v'.BA(s,c,v,x,v')$	(d)
Variant naturel	$A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \Rightarrow V(s,c,v') \in N$	(e)
Progression de variant	$A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \wedge BA(s,c,v,x,v') \Rightarrow V(s,c,v') < V(s,c,v)$	(f)

TABLE *3.2 – Exemples d'obligations de preuve pour un modèle B-événementiel*

La Table 3.2 présente les principales règles d'obligation de preuve associées à un modèle B-événementiel.

- Les *règles d'obligation de preuve de théorème* (a) et (b) garantissent qu'un théorème de contexte proposé (a) ou un théorème de machine (b) est bien correcte : il peut être déduit des axiomes et des invariants.
- La *règle d'obligation de preuve de conservation d'invariant* (c) garantit que chaque invariant d'une machine est préservé par chaque événement.
- La *règle d'obligation de preuve de faisabilité* (d) garantit qu'une action non déterministe est réalisable.
- La *règle d'obligation de preuve de variant naturel* (e) garantit que, sous les conditions de chaque événement convergent ou anticipé, un variant numérique proposé est en effet un nombre naturel
- La *règle d'obligation de preuve de variant* (f) indique que chaque événement convergent diminue le variant numérique proposé.

Il existe d'autres règles pour générer des obligations de preuve afin de prouver l'exactitude du raffinement. Les définitions complètes sont données dans [4].

4 Sémantique

Le nouvel aspect de la méthode B-événementiel [4], par rapport à la méthode B classique [1], est lié à la sémantique. En effet, les événements d'un modèle sont

des événements atomiques d'un système à transition d'état. La sémantique d'un modèle B-événementiel est une sémantique basée sur des traces avec des événements entrelacés. Un système est caractérisé par un ensemble de traces licites correspondant aux événements déclenchés du modèle et respectant les propriétés décrites. Les traces définissent une séquence d'états pouvant être observés par les propriétés. Toutes les propriétés seront exprimées sur ces traces.

5 Raffinement

L'opération de raffinement [3] proposée par B-événementiel permet le développement de modèle par étapes. Un système de transition d'état est transformé en un autre système comportant de plus en plus de décisions de conception tout en passant d'un niveau abstrait à un niveau moins abstrait. Une machine raffinée est définie en ajoutant de nouveaux événements, de nouvelles variables d'état et un invariant de collage. Chaque événement du modèle abstrait est raffiné dans le modèle concret en ajoutant de nouvelles informations indiquant comment le nouvel ensemble de variables et les nouveaux événements évoluent. Tous les nouveaux événements apparaissant dans le raffinement raffinent l'événement de saut (skip event) (c'est-à-dire l'événement qui ne fait rien et qui peut se produire à tout moment). Le raffinement préserve les propriétés prouvées et il n'est donc pas nécessaire de les prouver à nouveau dans le système de transition raffiné, généralement plus détaillé. Cela aide à garder la taille des preuves raisonnable en répartissant l'effort de preuve le long de l'arbre de raffinement.
Afin de prouver l'exactitude du développement, il est nécessaire de prouver l'exactitude des différents raffinements qu'il contient. Les obligations de preuve suivantes sont les deux obligations de preuve clés.

— Renforcement de la garde : un événement concret ne doit être activé que si l'événement abstrait est activé. Pour chaque condition abstraite G_i^A,
$$A \wedge I^A \wedge I^C \wedge G^C \wedge W \Rightarrow G_i^A$$
où, pour rappel, A désigne la conjonction des axiomes, I les invariants, G les conditions, W les témoins (prédicats liant des variables concrètes et abstraites) et BA les prédicats avant-après (actions) ; et $\cdot^A$ concerne la machine abstraite tandis que $\cdot^C$ concerne la machine concrète.
— Simulation d'action : si l'action d'un événement abstrait attribue une valeur à une variable également déclarée dans la machine concrète, il faut prouver que le comportement de l'événement abstrait correspond au comportement concret.
$$A \wedge I^A \wedge I^C \wedge G^C \wedge W \wedge BA^C \Rightarrow BA_i^A$$

Remarque Notez que de nombreux raffinements peuvent raffiner la même machine abstraite. Chaque machine de raffinement correspond à un comportement possible, à la mise en oeuvre ou à la concrétisation de la machine abstraite. Ainsi, plusieurs améliorations possibles sont proposées pour une machine abstraite donnée. Ceci sera utilisé dans les chapitres suivants pour caractériser l'ensemble des systèmes corrects qui se comportent comme décrit par une description abstraite du système. La méthode B-événementiel a prouvé sa capacité à représenter des systèmes événementiels tels que les systèmes ferroviaires, les systèmes intégrés ou les services Web. De plus, des systèmes complexes peuvent être construits progressivement de

manière incrémentale en préservant les propriétés initiales grâce à la préservation d'un invariant de collage.

6 Propriétés de vivacité

Les installations intégrées de B-événementiel sont principalement orientées vers la garantie des propriétés de sécurité (absence de mauvais états) grâce à la préservation des invariants. Cependant, il est également possible de vérifier certaines propriétés d'activité :

— dans B-événementiel où les formules LTL peuvent être directement codées [56] bien que cela ne soit pas vraiment pratique pour les formules volumineuses.
— en utilisant des outils externes tels que le modèle de vérification de modèle ProB, qui peuvent vérifier les formules LTL sur des modèles B-événementiel liés [79].

Il est important de noter que, contrairement aux propriétés de sécurité, les propriétés d'activité ne sont pas systématiquement préservées par raffinement.

7 Outils

Le principal outil disponible pour mener des développements basés sur B-événementiel est la plate-forme Rodin [1] [2]. Il s'agit d'un environnement de développement intégré doté d'éditeurs de contextes et de machines, d'un générateur d'obligation de preuve, de démonstrateurs automatiques et de capacités de vérification interactives. De plus, une large gamme de plug-ins sont disponibles, ce qui peut étendre les capacités de modélisation (avec des théories, par exemple) ou de vérification (comme le vérificateur de modèle ProB [70] ou l'utilisation de solveurs SMT).

7.1 Animation

Il est également possible d'instancier les modèles dans la plate-forme Rodin et de les animer. Ceci est très utile pour vérifier avec les ingénieurs du domaine si la spécification produit les comportements prévus et pour vérifier si les modèles, en plus de ne pas violer les invariants, peuvent réellement exister.

7.2 Prouveurs automatiques

Parmi les prouveurs automatiques, on peut différencier les solveurs SMT (Satisfiability Modulo Theories) comme $Alt-Ergo$ [41], $CVC3$ [29] et $Z3$ [36] qui utilisent des solveurs SAT (pour les problèmes de satisfaisabilité booléenne), des prouveurs du premier ordre comme Zenon [81] et Vampire [82], dans la mesure où les premiers vérifient la satisfaisabilité d'une formule par rapport à une théorie donnée, les seconds recherchent une preuve directement de la formule.

1. http ://www.event-b.org/

Pour chaque famille de prouveurs, il existe un format d'entrée commun : $TPTP$ et $SMTLIB$. Concernant la trace, il n'y a actuellement pas de format de sortie standard, elle peut être :

- une information sur le fait que le prouveur a trouvé ou non une preuve (OK/KO) ;
- un contre-exemple ;
- des éléments de preuve ;
- des preuves dans un format propre à l'outil ;
- des preuves lisibles par un humain ;
- des preuves vérifiables automatiquement par un assistant à la preuve.

Chapitre 4

Réalisabilité de systèmes de communication

Sommaire

Un protocole de conversation est un ensemble de règles qui définissent comment se produit une communication dans un système distribué, la question qui se pose est : existe t-il un ensemble d'entités dont le comportement composé est le même que les spécification du protocole de conversation ? Ce qui est connu sous le nom du problème de réalisabilité des protocoles. Autrement dit, un système est dit réalisable si est seulement si leur comportement reste le même quand la communication synchrone est remplacé par la communication asynchrone. Notre étude de l'état de l'art comporte trois branches : Une dédié au approches de vérification et validation de réalisabilité des systèmes distribué. une autre pour tout ce qui concerne l'assurance de la réalisabilité lors de l'évolution des systèmes suite a un changement de comportement interne ou externe. une branche pour les approches qui proposent des modèles des systèmes correctes-par-construction et une dernière pour les approches qui proposent des réparations des systèmes identifier comme non-réalisables.

1 Approches de vérification et validation de la réalisabilité

Dans cette section, nous nous concentrons sur les approches de vérification et validation des propriétés du système. La vérification et la validation des systèmes répartis [10, 46, 71] a toujours été un sujet de recherche très actif. Le principe fondamental de ces travaux est de modéliser le protocole d'échange d'information entre entités communicante à partir du système global. La représentation d'un tel modèle peut être sous la forme de : algèbre de processus, transition d'états, réseau de pétri

ou autre. Ils appliquent en suite une technique de vérification sur le modèle obtenu tel que, chaque approche propose un contexte différents de vérification.

Divers travaux modélisent et vérifient des protocoles multi-partites, notamment des chorégraphies, à l'aide de méthodes basées sur la théorie des automates, en représentant chaque partie sous la forme de machine à état finis CFSM [27].

La sécurité de leurs interactions (généralement indécidables) est vérifiée selon deux approches principales : (a) supposer la décidabilité d'une propriété de synchronisabilité [14, 15], puis vérifient les propriétés temporelles des modules CFSM via la vérification du modèle ; (b) vérifier les conditions d'exécution synchrones décidables sur les modules CFSM et prouver qu'elles garantissent des exécutions asynchrones sécurisées [24, 38, 59].

Le principal but du travail présenté dans [10] est de répondre à la question suivante : *Étant donné une chorégraphie, est-il possible de composer un système distribué qui répond exactement au spécifications d'une telle chorégraphie ?*

Ce qui est connu sous le nom de *problème de réalisabilité*. Pour cela, [10] fourni des conditions nécessaires et suffisantes pour la réalisation des chorégraphies qui sont : la condition d'équivalence (Définition 9), la condition de synchronisabilité (Définition 10) et la condition du système bien formé (Well-Formedness WF) (Définition 11). L'efficacité de l'approche proposée a était démontrée sur (1) des chorégraphies de services Web, (2) des diagrammes de collaboration UML et autres.

Les auteurs de [46] s'intéressent par le problème de la réalisabilité des systèmes asynchrone avec files d'attentes de capacité infini.

La proposition consiste en : au départ, le système à vérifier doit être modélisé sous forme d'un automate de büchi, le protocole est dit réalisable si et seulement si les trois conditions de réalisabilité suivantes sont satisfaites par l'automate. (1) Le protocole de communication est spécifié par un automate de büchi réalisable, (2) Les propriétés du *CP* sont vérifiées par l'automate de büchi, (3) le *CP* est formé à partir des entités communicantes via la projection

Dans [71], le problème de la réalisabilité a été relier au problème de contrôlabilité. L'approche proposée est basée sur des outils et algorithmes capable de contrôler la faisabilité des chorégraphies. Le résultat de l'approche permet la spécification et la synthèse des implémentations asynchrones des entités communicantes.
Une telle proposition évite l'analyse ultérieure et la correction des systèmes asynchrones composés à partir des entités communicantes.

Les approches proposées dans [62] et [78] sont basées sur la modélisation par transitions d'états. Les auteurs considèrent deux définitions de réalisabilité des systèmes qui sont : la réalisabilté forte et la réalisabilité faible. Cette classification de la réalisabilité permet de traiter efficacement ce problème vu que nous pouvons obtenir des degrés de réalisabilité du protocole de conversation, ce qui permet de traiter d'autres propriétés intéressantes des compositions des systèmes.
La réalisabilté ici est étudié au niveau global (les interactions du système) et aussi au niveau local (comportement de chaque entité qui forme le système).

Les approches proposées permettent de modéliser et d'analyser les échanges au coeur du système en mode de communication asynchrone avec des files d'attentes de capacité infinie.

Heussner et al, ont développé l'approche CEGAR basée sur la vérification régulière des modèles [54].

Les classifications des topologies de communication en fonction de la décidabilité des problèmes d'accessibilité sont connues pour les communications FIFO, FIFO+lossy et FIFO+bag [31, 34].

Dans [55, 64], le problème d'accessibilité lié au commutateur de contexte pour les machines en communication étendues avec des piles locales modélisant des appels de fonction récursifs est présenté comme décidable sous diverses hypothèses.

Des dialectes de type session ont été introduits pour les systèmes de machines à états finis en communication [37] et ont montré qu'ils appliquaient diverses propriétés souhaitables. Plusieurs notions similaires à celle de synchronisabilité ont également été étudiées dans différents contextes. L'élasticité relâchée (Slack elasticity) est le nom donné au système distribué asynchrones qui se comporte de la même manière quel que soit le jeu des communications.

Cette propriété a été étudiée dans la conception matérielle [73], dans le but de garantir que certaines transformations de code conservent la sémantique, dans le calcul haute performance, afin de garantir l'absence de blocage et autres bugs dans les programmes d'interface de passage de message MPI [89, 92], mais également pour communiquer avec des machines à états finis, comme dans ce travail, avec une manière légèrement différente de comparaison des comportements du système avec différentes files d'attente de taille limité.

Genest et ses collaborateurs ont introduit la notion des algorithmes de vérification de modèles pour les automates communicants liés de manière existentielle, ou ils définissent des traces de Mazurkiewicz, également appelées graphes de séquence de messages dans le contexte des machines à états finis en communication [50].

Enfin, une notion similaire à celle de systèmes liés de manière existentielle appelée «systèmes k-synchrones» a été introduite dans le travail de [25].

Malheureusement, Finkel et Lozes [45] ont récemment prouvé que la synchronisabilité était impossible : *ç.a.d.* les méthodes proposés au références ci-dessus pourrait être inutilisable.

Finkel et al, dans leur travail [45] se concentrent sur le mode de communication peer-to-peer, tel que. Ils fournissent des contre-exemples liées aux propriétés des modèles de communication, à la fois pour la sémantique peer-to-peer et pour la sémantique de boîte aux lettres (Mailbox), qui illustrent que les revendications dans [14, 15] ne sont pas prouvées correctement. Ils montrent aussi que l'affirmation sur la décidabilité de la synchronisabilité pour la sémantique peer-to-peer est en réa-

lité fausse, et établissons l'indécidabilité de la synchronisabilité. Ils montrent que la synchronisabilité du modèle est valable pour les systèmes formés de machines communicantes sur un anneau orienté (oriented-ring), à la fois sous la boîte aux lettres (Mailbox) et la sémantique peer-to-peer (les deux sont en réalité les mêmes dans ce cas), et donc que la synchronisabilité est décidable pour les anneaux orientés. Ils montrent également que l'ensemble d'atteignabilité de tels systèmes est reconnaissable par canal (channel-recognizable), c'est-à-dire que l'ensemble des configurations d'accès est régulier. Enfin, ils montrent que les contre-exemples qu'ils ont donnés invalident d'autres revendications, en particulier le résultat utilisé pour vérifier la stabilité [5, 6].

Dans [88], Scalas et al proposent une nouvelle théorie généralisée des types de sessions multi-parties (MPST).

Ils démontrent qu'une révision des fondements théoriques de la MPST est nécessaire : les MPSTs classiques ont une propriété de cohérence limitée, avec des restrictions inhérentes qui peuvent être facilement négligés lors de la révision, ce qui a conduit dans les travaux antérieurs à des preuves défective de sécurité. La théorie proposée supprime de telles restrictions et corrige de tels défauts et preuves. Ils apportent une nouvelle théorie MPST moins compliquée, mais plus générale, que la théorie classique : elle ne nécessite pas de types de session multi-parties globaux ni de types de sessions binaires, mais repose sur des propriétés comportemental générales , et prouve la sécurité des protocoles et processus. Ils produisent une analyse détaillée des propriétés, montrant comment, dans la nouvelle théorie, ils permettent d'assurer la décidabilité du contrôle des propriétés et garantissent de manière statique que les processus bénéficient, par exemple, d'absence de blocage et de la vivacité au moment de l'exécution.

Dans [67] les auteurs proposent une procédure de vérification des propriétés des automates de session communicants (csa), c'est-à-dire des automates en communication correspondant à des types de session à multi-parties. Ils introduisent une nouvelle propriété de compatibilité pour csa asynchrone, appelée compatibilité k-multipartie (k-mc), qui est un sur-ensemble strict de la compatibilité multipartie synchrone proposée dans la littérature. La propriété est décomposé en deux propriétés liées : (i) une condition appelée k-sécurité qui garantit que, dans la limite, tous les messages envoyés peuvent être reçus et que chaque automate peut effectuer un changement d'état ; et (ii) une condition appelée k-exhaustivité qui garantit que toutes les actions k-accessible d'envoi peuvent être déclenchées. Ils montrent aussi que les systèmes k-exhaustifs caractérisent correctement et complètement les systèmes, tel que, chaque automate se comporte de manière uniforme pour toute borne supérieure ou égale à "k". Leur proposition peut être effectuée efficacement sur de grands systèmes en utilisant des techniques de réduction d'ordre partiel. Ils démontrent aussi que plusieurs exemples de la littérature sont k-mc.

Le but de l'étude des approches présentées précédemment est de comprendre la notion de la réalisabilité des systèmes et de pouvoir distinguer par la suite les différentes conditions de validation de systèmes, afin de pouvoir classer les conditions les plus pertinentes par rapport à nos contributions.

L'objectif est de vérifier la compatibilité de la composition d'un ensemble d'entités communicantes, à partir d'une description comportementale de ces derniers et d'un modèle de communication (CP). Pour avoir une intuition, considérons la compatibilité en termes de terminaison de la composition de deux entités spécifiée comme suit :

Utilisateur

1. Envoyer le nom d'utilisateur
2. Envoyer le code PIN
3. Recevoir un rapport
4. Terminer

Service de connexion

1. Recevoir un identifiant
2. Recevoir un code PIN
3. Vérifier la validité
4. Envoyer un rapport
5. Terminer

Dans le monde synchrone, la compatibilité de ces deux entités est bien définie comme suit. Les deux correspondent pour un premier rendez-vous sur le nom d'utilisateur, puis procèdent à un deuxième rendez-vous sur le code PIN, puis sur le rapport et se terminent éventuellement. Cependant, cela est moins clair dans le monde asynchrone. Traditionnellement, du point de vue des systèmes distribués, on considère que le moyen de communication contrôle les envois de messages, cela signifie que les messages sont transmis aux entités communicantes. Les entités communicantes ne peuvent pas imposer un ordre de réception.

Dans notre exemple, si le moyen de communication assure l'ordre FIFO, le nom d'utilisateur est fourni avant le code PIN (car ils sont envoyés dans cet ordre) et les deux entités se terminent : nous disons qu'ils sont compatibles.

Cependant, si le moyen de communication est totalement asynchrone, le code PIN peut être reçu en premier. Le service de connexion pourrait ne pas être en mesure de faire face à une telle situation. À partir de là, la fin du service de connexion est incertaine. De plus, rien ne garantit qu'un rapport sera envoyé : l'utilisateur peut l'attendre et ne jamais se terminer. Qu'une sortie positive du système existe ou non, la composition est jugée incompatible en communication totalement asynchrone.

Le contrôle de la compatibilité peut consister à construire de manière exhaustive l'ensemble de toutes les exécutions de la composition des entités communicantes, en le réduisant, en filtrant les exécutions non pertinentes dans le modèle de communication choisis, et en évaluant les propriétés sur cet ensemble (blocage, perte ou désordre des messages, ..., etc).

La spécification des entités communicantes d'une composition, des modèles de communication, du système global dans lequel ils interagissent et des propriétés de compatibilité constitue un cadre pour la vérification de la compatibilité des entités communicantes, ou ce que nous appelons réalisabilité des protocoles de conversation. La suite de la section décrit et formalise ce cadre et démontre sa validité en termes d'exactitude et de complétude.

Nous considérons la condition nécessaire et suffisante de la réalisabilité des CPs, le résultat fondamental du travail présenté dans [10]. La propriété de réalisabilité

comporte principalement. (1) la condition d'équivalence qui est vérifié entre le modèle du *CP* et la recomposition synchrone du système, (2) la condition de synchronisabilité qui doit être vérifié entre la recomposition du *CP* en mode de communication synchrone et la recomposition asynchrone (3) le système à vérifier qui doit être bien formé, c'est ce qu'on appelle la condition WF (well-formedness) qui concerne la vérification des files d'attentes à la fin de la recomposition asynchrone [10].

Le travail présenté dans [90] est axé sur deux notions, à savoir la simulation et la bisimulation, afin de mesurer l'équivalence des systèmes décrits à l'aide de systèmes de transition étiquetés (LTS).

La première notion suppose qu'un état s_1 d'un système de transition étiqueté LTS_1, simule un autre état s_2 dans un LTS_2 si chaque transition sortante de s_1 est étiquetée avec un message qui peut être mis en correspondance avec une étiquette sortante de s_2, de telle sorte que la synchronisation sur les deux transitions conduise à des états liés entre eux par la même relation de simulation.

La bisimulation est simplement comprise comme la version symétrique de la relation de simulation.

Dans [95], les auteurs proposent une notion de compatibilité pour vérifier deux services décrits en utilisant le π-calculus. Selon leur proposition, deux services sont compatibles s'il existe toujours au moins une séquence de transition entre eux, jusqu'à atteindre les états finaux. Leur travail ne calcule pas la compatibilité détaillée des différents états de protocole et il n'y a pas de détection d'incohérence.

Nous définissons formellement la relation d'équivalence entre le protocole de conversation et la recomposition synchrone du *CP* a partir des entités projetées comme suit :

Définition 9 (Equivalence) *CP est equivalent à Sys_{sync} , noté formellement, $CP \equiv Sys_{sync}$, s'ils ont des séquences de message égaux, ç.a.d., des traces équivalentes.*

Afin de vérifier la compatibilité du système, nous devons vérifier que chaque état global formant le système est atteignable lors de l'exécution du système. Par conséquence, nous utilisons une fonction accessibilité ${}_{PS}((s_1, \ldots, s_n))$ qui fournit l'ensemble des états globaux que n inter-opérant (entités) peuvent atteindre, en une ou plusieurs étapes, à partir d'un état global actuel $(s_1, ..., s_n)$ à travers des synchronisations ou des évolutions indépendantes.

Dans notre définition de l'accessibilité, les séquences de τ-actions sont ignorées. Les états atteints avec les actions τ sont dans l'ensemble des états accessibles uniquement s'ils sont finaux ou s'il existe des actions observables pouvant être exécutées à partir d'eux.

Nous expliquons ici le concept de compatibilité d'état sur lequel reposent certaines des notions de compatibilité que nous formalisons dans ce travail. L'interaction du protocoles de conversation dépend essentiellement des synchronisations sur les actions observables et peut ensuite être définie à l'aide d'un critère défini. Le critère est utilisé pour vérifier la compatibilité des états comme suit.

Pour un état global donné $(s_1, \ldots, s_n)$, cet état est considéré comme compatible si chaque message envoyé (respectivement reçu) par une entité communicante $\mathcal{P}_i$ dans l'état s_i sera finalement reçu (respectivement envoyé) par une autre entité communicante $\mathcal{P}_j$ à l'état sj où $i, j \in 1, \ldots, n$ et $i \neq j$, de tel sorte que toutes les entités communicantes évoluent vers un état global compatible.

S'il n'existe pas d'entité communicante de ce type capable d'interagir avec l'action de l'entité communicante $\mathcal{P}_i$, les entités communicantes $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ devraient pouvoir atteindre un état global dans lequel cette action sera activée. Comme les entités communicantes peuvent évoluer indépendamment à travers de certaines transitions τ non observables, la compatibilité comportementale exige que chaque évolution interne conduise toutes les entités dans des états compatibles [30, 35]. Nous définissons formellement la propriété de synchronisabilité comme suit.

Définition 10 (Synchronisabilité) *Soit un ensemble d'entités $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, le CP synchrone $Sys_{sync} = (S_s, s_s^0, L_s, T_s)$, et le CP asynchrone $Sys_{async} = (S_a, s_a^0, L_a, T_a)$, et deux états $r \in S_s$ et $s \in S_a$ sont synchronisable s'il existe une relation $Sync\_st$ entre les états tel que $Sync\_st(r, s)$ et :*

- *pour chaque $r \xrightarrow{m} r' \in T_s$, il existe $s \xrightarrow{m!} s' \in T_a$, tel que $Sync\_st(r', s')$;*
- *pour chaque $s \xrightarrow{m!} s' \in T_a$, il existe $r \xrightarrow{m} r' \in T_s$, tel que $Sync\_st(r', s')$;*
- *pour chaque $s \xrightarrow{m?} s' \in T_a$, $Sync\_st(r, s')$.*

Sync est l'ensemble synchronisable Sys_{async} où $Sys_{async} \in Sync \Leftrightarrow Sync\_st(s_s^0, s_a^0)$.

WF indique qu'à chaque fois que la taille d'une file d'attente de réception, Q_i, de la i^{eme} entité est supérieur à 0 (*ç.a.d.*, Q_i est non vide), le *CP* asynchrone peut éventuellement passer à un état où Q_i devient vide.

Définition 11 (WF) *Soit WF l'ensemble des systèmes bien formés. Un CP asynchrone $Sys_{async} = (S_a, s_a^0, \Sigma_a, T_a)$ défini sur un ensemble d'entités $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ est bien formé, c.à.d, $Sys_{async} \in WF$, si et seulement si $\forall s_a = (s_1, Q_1, \ldots, s_n, Q_n) \in S_a$, où s_a est accessible depuis $s_a^0 = (s_1^0, \epsilon, \ldots, s_n^0, \epsilon)$, ce qui suit s'applique : s'il existe Q_i tel que $\mid Q_i \mid > 0$, alors il existe $s_a \Rightarrow^* s'_a \subseteq T_a$ où $s'_a = (s'_1, Q'_1, \ldots, s'_n, Q'_n) \in S_a$ et $\forall Q'_i, \mid Q'_i \mid = 0$.*

Dans un système distribué, il est difficile d'assurer l'atomicité des transactions car plusieurs processus peuvent participer à la réalisation d'une transaction unique. La défaillance d'un des participants à la transaction ou d'une des liaisons de communication peut aboutir à des résultats erronés.

Dans les implémentations distribuées des modèles de communication, la transmission de messages est le seul moyen d'échanger des informations.

Un système est réalisable avec une communication synchrone si chaque événement d'envoi est immédiatement suivi de l'événement de réception correspondant [32, 63]. Si le couple d'événements d'envoi et de réception est visualisé de manière atomique, cela correspond à une exécution de communication synchrone. En communication multi-diffusion, cela signifie qu'un événement d'envoi est immédiatement suivi des événements de réception ou des événements internes correspondants. Après l'envoi d'un autre message, aucune réception du message précédent ne peut se produire lors de l'exécution.

La réalisabilité d'un système avec une communication asynchrone est définit formellement comme suit.

Définition 12 (Réalisabilité) *Un protocole de conversation CP est réalisable si et seulement si (i) les entités calculées par projection à partir de ce protocole sont synchronisables, (ii) le CP asynchrone résultant de la composition des entité est*

bien formé (WF est vérifié), et (iii) la version synchrone du système distribué (CP synchrone) $\{P1, \ldots, Pn\}$ *est equivalent au CP du départ.*

L'exactitude de cette approche est donnée dans [42].

Soit R l'ensemble des CPs réalisables. R est défini comme un sous-ensemble de CPs déterministes tels que :

$$\forall CP \in DC, \qquad CP \in R \Longleftrightarrow \begin{cases} (1) & CP \equiv Sys_{sync}(\downarrow CP) \\ & \wedge \\ (2) & Sys_{async}(\downarrow CP) \in Synchronizable \\ & \wedge \\ (3) & Sys_{async}(\downarrow CP) \in WF \end{cases}$$

où $\downarrow CP = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *est l'ensemble des entités obtenues par projection de CP conformément à la définition 6.*

La définition ci-dessus est prise de [10] tel que la vérification de la réalisabilité de CP est faite tout au long des étapes suivantes :

- *Calcul de l'ensemble des entités impliquées* $\downarrow CP = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$ *par projection à partir de CP*
- *Calcul des compositions synchrones et asynchrones*
- *Vérification de la synchronisation*
- *Vérification de l'équivalence entre CP et la composition des entités communicantes.*

Dans ce qui suit, nous présentons les approches dédiées à l'évolution des systèmes distribués.

2 Approches d'évolution des systèmes

Cette section est consacrée aux travaux qui s'intéressent par l'évolution ou reconfiguration des systèmes distribués par ajout, élimination ou remplacement de certains interactions des entités formant le système.

Nous nous concentrons sur les solutions proposées afin d'assurer la réalisabilité après chaque mise à jour du système. Tel que, considérant par exemple le succès économique d'une entreprise, qui dépend de plus en plus de sa capacité à réagir avec souplesse et rapidité aux changements intervenant sur le marché, le développement ou la fabrication. C'est pourquoi les entreprises s'intéressent de plus en plus à l'amélioration de l'efficacité et de la qualité de leurs processus internes et à l'optimisation de leurs interactions avec leurs partenaires et leurs clients. Les auteurs de [83] ont constaté une adoption croissante des technologies d'automatisation des processus d'affaires (BP) par les entreprises, ainsi que des normes émergentes pour l'orchestration et la chorégraphie de BP afin de répondre à ces objectifs. Ces derniers permettent la définition, l'exécution et la surveillance des processus opérationnels

d'une entreprise. En outre, en liaison avec la technologie de service Web, les avantages de l'automatisation de l'entreprise au sein d'une même entreprise peuvent également être transférés à d'autre entreprises inter-organisationnelles (chorégraphies de processus). La prochaine étape de cette évolution sera l'émergence d'une entreprise agile capable de mettre en place rapidement de nouveaux processus et d'adapter rapidement ceux qui existent à l'évolution de son environnement.

Un défi important qui n'a pas encore été traité de manière adéquate concerne l'évolution des chorégraphies de processus, c'est-à-dire le changement contrôlé des interactions entre les processus des partenaires dans un contexte inter-organisationnel. Si une partie modifie son processus de manière incontrôlée, des incohérences ou des erreurs concernant ces interactions peuvent survenir dans la suite. Généralement, les partenaires impliqués dans une chorégraphie de processus échangent des messages via leurs processus publics, qui peuvent être considérés comme des vues spéciales sur leurs processus privés (c'est-à-dire les orchestrations de processus). Si l'un de ces partenaires doit modifier la mise en œuvre de son processus privé (par exemple, l'adapter à de nouvelles lois ou à des processus optimisés), la difficulté majeur dans ce cas là est de savoir si ce changement affecte également les interactions avec les processus partenaires et leur mise en œuvre. De toute évidence, tant qu'un processus métier modifié ne fait pas partie d'une chorégraphie de processus, les effets de changement peuvent être conservés localement. Le même raisonnement s'applique si les modifications d'un processus privé n'ont aucun impact sur les vues publiques associées.

En général, nous ne pouvons pas supposer cela. La modification d'un processus privé peut non seulement influer sur le processus public correspondant, mais également sur les processus publics et privés de ses partenaires. Nous avons donc besoin de méthodes pour propager (automatiquement) les modifications d'un processus privé vers les processus partenaires (si nécessaire). Les adaptations de processus chorégraphiques se sont à la fois coûteuses et sujettes aux erreurs.

[68] examine les approches traitant l'évolution des services Web jusqu'à l'année 2013. L'étude de ce travail nous a permis de tirer les propositions les plus liées à nos contributions qui sont : [61, 83, 85, 86, 94]. L'étude de l'évolution des systèmes a été abordée dans [74]. Les solutions proposées dans ces travaux concernent la reconfiguration dynamique des systèmes. La problèmatique a été abordée dans différents contextes comme, les systèmes distribués, les architectures logiciels, les transformations graphiques et autres, mais, aucun travail n'étudie les systèmes dont les interactions sont décrites sous forme des *CPs*.

Nous limitons notre étude aux approches axées sur l'évolution des protocoles de conversation. Les auteurs de [86] proposent une méthode de vérification de la reconfigurabilité des *CPs*. L'approche considère deux *CPs* réalisables, à savoir, un CP et un CP', et deux ensembles d'entités communicantes PS et PS' obtenus par projection des deux *CPs*, respectivement.

Étant donné une trace t qui correspond au CP (c.à.d, la séquence d'interactions

entre les entités), si t peut également être exécuté à partir des entités reconfigurées et générées à partir de CP', alors la reconfiguration peut avoir lieu.

[94] utilise un modèle formel basé sur les automates d'état fini (aFSA) pour décrire les interactions des services Web. L'approche proposée est implémentée en DYCHOR, elle harmonise les changements entre la chorégraphie mise à jour et l'orchestration correspondante. Les changements proposés sont soit des ajouts ou des éliminations des séquences de messages. Ces changements sont propagé aux entités distribuées tout en gardant la cohérence de la chorégraphie.

Dans [83, 85], les auteurs proposent également une méthode d'évolution des systèmes contrôlée où la propagation des changements apportés à une entité nécessite de vérifier son effet sur les autres entités formant la chorégraphie.

[44] se concentre sur le domaine de gestion des processus métier (BPML) et l'architecture orientée services (SOA). Il décrit les chorégraphies des services en utilisant un modèle arborescent. Les auteurs envisagent certains changements tels que : la mise à jour, l'insertion ou la suppression des fragments d'interaction au cœur des systèmes.

Les auteurs de [80] définissent un nouveau langage nommé DIOC destiné à la programmation cohérente des applications distribuées. La sémantique du langage DIOC repose sur les systèmes de transition étiquetés. L'approche donnée dans [80] permet de mettre à jour des fragments de code des entités distribuées. Cela peut être spécifié au niveau de la chorégraphie où des blocs de code qui peuvent être mis à jour dynamiquement à l'aide des "scopes". Ces blocs scope doivent être connus a priori lors de la description de la chorégraphie.

Les auteurs de [33] se basent sur l'analyse des systèmes abstrait afin de vérifier cetraines propriétés de la logique temporelle, si ces propriétés sont satisfaites par le système abstrait cela signifie qu'elles le seront aussi dans leur système concret. Dans le cas contraire un contre exemple est généré par le modèle checker afin de pouvoir rafiner le système a une abstraction différente, le processus est itératif jusqu'à arriver à un système cohérent.

3 Approches correctes-par-construction

Cette section est dédié au processus formel de vérification des propriétés des systèmes. Ces approches permet la spécification de systèmes selon une méthodologie correcte-par-construction basée sur le raffinement, commençant à partir d'un niveau de spécification abstrait élevé arrivant au niveau le plus concret. La cohérence de la spécification est préservée à tous ses niveaux car les propriétés du modèle sont liées au comportement. En effet, définir les propriétés du modèle en fonction de son comportement permet d'obtenir une vérification a priori de l'exactitude de la spécification.

Les auteurs de [43] utilisent les systèmes d'états étiquetés comme modèle de

base pour la description des chorégraphies des systèmes distribués. En se basant sur la méthode B-événementiel, ils proposent un modèle correcte-par-construction qui permet de composer des systèmes distribués asynchrones qui a priori réalisent leurs chorégraphies. Le travail établi par les auteurs de [43] se base principalement sur l'approche proposée dans [10]. Le principal but de l'approche est la modélisation les conditions nécessaires et suffisantes de réalisabilité proposées dans [10] en B-événementiel. Ils appliquent plusieurs raffinement afin de générer les entités distribuées de communication. Le premier raffinement retourne les comportements des entités obtenues par projection synchrone. Le système calculé précédemment est ensuite raffiné dans sa version asynchrone avec des files d'attentes de capacité illimitée. Grâce au invariants qu'une séquence de messages échangés est préservée à chaque étape de raffinement. L'approche fournie une preuve formelle de l'algorithme de la réalisabilité des chorégraphies déterministes, mais avec une seule file d'attente globale indexé afin d'assuré le principe d'annaux de communication proposé par Finkel et al [45].

Dans le travail de [48], les auteurs proposent une approche de spécification formelle du contrôle d'accès basé sur les attributs (ABAC) en utilisant la méthode B-événementiel. Ils appliquent une vérification formelle a priori pour construire incrémentalement un modèle correcte-par-construction, *ç.a.d.* L'exactitude du modèle de spécification est assurée pendant les étapes de construction. Le modèle est composé de niveaux d'abstraction générés par des opérations de raffinement. Un ensemble de propriétés ABAC est défini dans chaque niveau de raffinement. Ces propriétés sont préservées par les preuves avec la spécification de comportement.

4 Approches de réparation

Lorsqu'une chorégraphie est considérée comme non réalisable, est-il possible de réparer automatiquement la chorégraphie de tel sorte que la version réparée soit réalisable ? Nous nous référerons à ce problème en tant que problème de réparabilité de chorégraphie. Son importance provient du fait que l'automatisation de la réparation de la chorégraphie permettra un développement plus rapide de systèmes distribués avec des garanties formelles de correction. Dans ce qui suit, nous résumons les approches connexes qui abordent le même problème.

La réalisabilté dans [51] est renforcé par l'introduction des modules appelés moniteurs de synchronisation. Un moniteur est créé pour chaque entité du système. Son rôle est de synchronisé les comportements des entités communicantes afin de satisfaire le protocole de conversation globale souhaité.
Des testes de réalisabilité sont effectués afin de détecter les comportements incohérents que nous souhaitons éviter. suivi par l'adaptation des comportements des entités via leurs moniteurs. L'approche proposée est basée sur le teste par contre exemple.

Dans [12], Basu et al étudient la possibilité de réparation des chorégraphies identifiées comme non réalisables, l'objectif étant d'identifier un ensemble de modifications d'une chorégraphie non réalisable donnée qui le rendra réalisable. Ils

proposent une technique permettant de réparer automatiquement les chorégraphies non réalisables et fournissent des garanties formelles d'exactitude et de terminaison. Ils appliquent la technique avec succès à plusieurs chorégraphies identifiées comme non réalisables au domaine du contrat Singulary OS et des services Web.

Dans [8], Autili et al formalisent une méthode d'enforcement de la chorégraphie basée sur la coordination distribuée des participants de l'extérieur. La méthode fait référence à un registre de services pour découvrir les services appropriés qui joue le rôle de la chorégraphie. Le registre contient des services publiés par les fournisseurs qui ont identifié des opportunités commerciales dans le domaine d'intérêt. Le registre contient également l'enregistrement des utilisateurs intéressés à exploiter la chorégraphie via des applications client. Les délégués de coordination (CDs) effectuent une coordination pure au niveau d'entreprise en analysant l'interaction de service et en la médian en échangeant les informations de coordination contenues dans leurs propres modèle de coordination (CM), de manière totalement distribuée. De cette manière, les CD empêchent les interactions indésirables. Ces dernières sont les interactions qui n'appartiennent pas à l'ensemble des interactions autorisées par la chorégraphie et peuvent se produire lorsque les services collaborent de manière incontrôlée. De plus, Ils décrivent un algorithme de coordination distribué implémenté par les CD pour réaliser la logique de coordination.

Dans leur travail récent [7] Autili et al proposent une approche formelle d'enforcement de la réalisabilité de la chorégraphie à travers la synthèse automatique des délégués de coordination (CD) distribués. Les CD sont des entités logicielles supplémentaires vis-à-vis des participants à la chorégraphie. Ils sont synthétisés afin de représenter et de contrôler l'interaction des services aux entités communicantes. Lors d'une intervention les entités synthétisées appliquent la collaboration prescrite par la spécification de chorégraphie. Ils affirment aussi l'exactitude de leur méthode de synthèse et illustrent sa formalisation sur un exemple explicatif.

Dans [52], les auteurs présentent une approche de vérification de chorégraphies à l'aide de techniques de contrôle des modèles et de l'équivalence. Leur proposition permet la vérification de certaines tâches d'analyse, c'est-à-dire la réparabilité, la réalisabilité, la conformité, la synchronisalité et le contrôle de renforcement de la réalisabilité. Afin de vérifier en séquence la synchronisabilité et la réalisabilité du système à l'aide du contrôle d'équivalence, les contrôleurs distribués sont générés via un processus itératif pris principalement du [51]. Les contre-exemples sont exploités pour raffiner le comportement des contrôleurs en ajoutant de nouveaux messages de synchronisation jusqu'à ce que la synchronisabilité et la réalisabilité puissent être appliquées.

5 Discussion

Les approches de modélisation et de la vérification des systèmes distribués étudiés dans la section précédente répondent à trois besoins principaux :

— *Vérification de la validité des systèmes distribués existants et des algorithmes.* Cela signifie prouver que les systèmes existants répondent à leurs spécifica-

tions. La vérification est souvent statique et repose sur des assistants de preuve dotés d'un degré variable d'automatisation ad-hoc.

— *Conception de systèmes distribués correctes-par-construction.* En d'autres termes, dériver des systèmes de leurs spécifications. Cela repose généralement sur une approche descendante avec un raffinement mécanique incrémentale, éventuellement complétée par une traduction certifiée conforme au code source.

— *Détecter et éviter les erreurs d'exécution des implémentations de systèmes distribués.* En pratique, il est possible que les implémentations de système distribué n'aient pas été construites ou prouver que se soient sans erreurs ou que des erreurs puissent survenir après de longues exécutions.
La vérification de l'exécution vise à détecter ou à éviter certains erreurs d'exécution.

Le formalisme et le contexte d'analyse et de vérification des protocoles de conversation des travaux connexes sont définis de différentes façons. La table 4.1 résume les quatre classes de notre état de l'art tel que. Indépendamment du formalisme adopté, les approches étudiées peuvent être classées comme suit :

Approches de vérification et validation de la réalisabilité	[10], [46], [71], [27], [14], [15], [24], [38], [59].
Approches d'évolution des systèmes	[83], [68], [86], [94], [61], [85], [83], [74], [44], [80], [33].
Approches correctes-par-construction	[43], [48].
Approches de réparation	[51], [12], [8], [7], [52].

TABLE *4.1 – Les trois classes d'approches existantes*

La première famille d'approches englobe les processus d'analyse basés sur les outils de vérification de modèles (CADP, . . ., etc), tel que, les auteurs définissent un ensemble de contraintes à vérifier par les protocoles de conversation modélisant des systèmes répartis afin de valider le bon fonctionnement du système.

La deuxième classe se base sur les simulations et test l'analyse d'évolution des systèmes distribués.

La troisième et dernière classe concerne les approches de vérification formelle correcte-par-construction des propriétés des modèles des systèmes basés sur la méthode B-événementiel.

La dernière classe est relative à la réparation des systèmes générant des erreurs de type, blocage, perte d'information, ou autre.

Les inconvénients majeurs des approches connexes sont :

— *la vérification non formelle ainsi qu'elle est effectuée sur l'ensemble du CP, ce qui incrémente la complexité des processus de vérification,*
— *le problème de passage à l'échelle en raison de l'explosion combinatoire d'espace d'états lors des validations des modèles complexes*, et
— *la difficulté des obligations de preuve et la complexité des étapes suivi afin de modéliser, vérifier et valider la réalisabilité des modèles de système.*

L'approche proposée dans [43] a pu minimiser le problème de l'explosion combinatoire qui présente une limitation majeure des processus de validation des modèles de composition par vérification des modèles, par proposition de la technique RLRA (Reverse Leaping Reachability Analysis). Cette technique, permet la détection des interblocages dans les spécifications des protocoles. L'approche de validation est basée sur le principe du retour arrière, ils commencent par identifier les états d'interblocage et de valider ensuite parmi les états suspects, ceux qui présentent réellement une erreur, à travers la recherche des chemins de retours vers la racine du graphe d'exécution.

Afin de réduire le problème de l'explosion combinatoire, ils ont introduit la notion de graphes de sauts (leap graphe) qu'ils appliquent dans la construction des chemins de retour arrière, ce processus permet d'avoir une réduction importante de la taille du graphe à parcourir.

La résolution du troisième inconvénient majeur qui est l'aspect global des approche de vérification de réalisabilité est l'objectif fondamental de cette thèse.

Deuxième partie

Contributions

Chapitre 5

Construction incrémentale : algèbre de composition

Sommaire

Ce chapitre est consacré à la présentation de nos contributions de composition des protocoles de communications réalisables. D'abord nous commençons par l'exposition de notre solution théorique qui se résume sous forme d'un ensemble d'opérateurs et de propriétés de composition des *CP*s avec des théorèmes et preuves mathématique montrant la correction de notre proposition.

Nous définissons ensuite les propriétés suffisantes garantissant la réalisabilité des *CP*s construits en utilisant les opérateurs de composition.

1 Opérateurs de composition

Notre proposition concernant la construction des *CP*s réalisables. Nous identifions un ensemble d'opérateurs de composition permettant la construction incrémentale correcte-par-construction des *CP*s réalisables. L'ensemble des opérateurs que nous avons identifiés pour construire un *CP* par la composition incrémentale de plusieurs CP_b désigne une composition séquentielle dénotée par $\otimes_{(\gg, s^f_{CP})}$, une composition en branche dénotée par $\otimes_{(+, s^f_{CP})}$ et une composition en boucle dénotée par $\otimes_{(\circlearrowleft, s^f_{CP})}$. Chacun des opérateurs est caractérisé par deux paramètres, le premier est un *CP réalisation* avec un état final $s^f_{CP} \in S^f_{CP}$ et le second est un (un en-

semble de) protocole de conversation basique CP_b. Chaque expression de la forme $\otimes_{(op,s^f_{CP})}(CP, CP_b)$ signifie que l'état initial de CP_b est fusionné avec l'état s^f_{CP}. En d'autres termes, CP_b est ajouté (collé) à CP à l'état s^f_{CP}.

Nous indiquons également que la composition du CP vide, référé comme $\varnothing$, avec tout CP_b (resp. $\{CP_{bi}\}$) produit un CP_b (resp. $\{CP_{bi}\}$) *ç.a.d.* $\otimes_{(op,s^f_{CP})}(\varnothing, CP_b) = CP_b$ (resp. $\otimes_{(op,s^f_{CP})}(\varnothing, \{CP_{bi}\}) = \{CP_{bi}\}$ (élément neutre).

Dans ce qui suit, nous définissons formelles chaque opérateur d'un point de vue structurel. Les conditions dans lesquelles la réalisabilité est préservée sont données dans la section 2.

Définition 13 *Composition séquentielle* $\otimes_{(\gg,s^f_{CP})}$.

Soit

- $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *est un protocole de conversation avec un état final* $s^f_{CP} \in S_{CP}$ *et*
- CP_b *est un protocole de conversation basique avec* $T_{CP_b} = \{s_{CP_b} \xrightarrow{l_{CP_b}} s'_{CP_b}\}$ *tel que* $\{s_{CP_b}, s'_{CP_b}\} \not\subset S_{CP}$.

La composition séquentielle de CP avec CP_b *à l'état* s^f_{CP} *dénoté par* $CP_{\gg} = \otimes_{(\gg,s^f_{CP})}(CP, CP_b)$ *est défini comme suit :*

- $S_{CP_{\gg}} = S_{CP} \cup \{s'_{CP_b} |\ s_{CP_b} \xrightarrow{l_{CP_b}} s'_{CP_b} \in T_{CP_b}\}$
- $L_{CP_{\gg}} = L_{CP} \cup \{l_{CP_b}\}$
- $T_{CP_{\gg}} = T_{CP} \cup \{s^f_{CP} \xrightarrow{l_{CP_b}} s'_{CP_b}\}$
- $S^f_{CP_{\gg}} = (S^f_{CP} \setminus \{s^f_{CP}\}) \cup \{s'_{CP_b}\}$

Tel que, la composition séquentielle $CP_{\gg} = \otimes_{(\gg,s^f_{CP})}(CP, CP_b)$ ajoute un CP_b à la fin d'un CP, et mis à jour l'ensemble des états $S_{CP_{\gg}}$, l'ensemble des messages échangés $L_{CP_{\gg}}$, l'ensemble de transitions $T_{CP_{\gg}}$ et l'ensemble des états finaux $S^f_{CP_{\gg}}$.

Définition 14 *Composition en branche* $\otimes_{(+,s^f_{CP})}$.

Soit

- $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *est un protocole de conversation avec un état final* $s^f_{CP} \in S_{CP}$ *et*
- $\{CP_{bi} \mid 1 \leq i \leq n, n \geq 2\}$ *est un ensemble fini de protocoles de conversation basique avec* $T_{CP_{bi}} = \{s_{CP_{bi}} \xrightarrow{l_{CP_{bi}}} s'_{CP_{bi}}\}$, $\{s_{CP_{bi}}, s'_{CP_{bi}}\} \not\subset S_{CP}$, *et les états finaux de* CP_{bi}, $1 \leq i \leq n, n \geq 2$.

La composition en branche de CP avec $\{CP_{bi} \mid 1 \leq i \leq n, n \geq 2\}$ *à l'état* s^f_{CP} *dénoté par* $CP_+ = \otimes_{(+,s^f_{CP})}(CP, \{CP_{bi}\})$ *est définit comme suit :*

- $S_{CP_+} = S_{CP} \cup \{s'_{CP_{b1}}, \ldots, s'_{CP_{bn}} |\ s_{CP_{bi}} \xrightarrow{l_{CP_{bi}}} s'_{CP_{bi}} \in T_{CP_{bi}}\}$
- $L_{CP_+} = L_{CP} \cup \{l_{CP_{b1}}, \ldots, l_{CP_{bn}}\}$
- $T_{CP_+} = T_{CP} \cup \{s^f_{CP} \xrightarrow{l_{CP_{b1}}} s'_{CP_{b1}}, \ldots, s^f_{CP} \xrightarrow{l_{CP_{bn}}} s'_{CP_{bn}}\}$
- $S^f_{CP_+} = (S^f_{CP} \setminus \{s^f_{CP}\}) \cup \{s'_{CP_{b1}}, \ldots, s'_{CP_{bn}}\}$

La composition en choix $CP_+ = \otimes_{(+,s^f_{CP})}(CP, \{CP_{bi}\})$ ajoute un ensemble de protocoles de conversation basique à l'état final s^f_{CP} ce qui forme un branchement (un choix) et mis à jour l'ensemble des états S_{CP_+}, l'ensemble des messages échangés L_{CP_+}, l'ensemble de transitions T_{CP_+} et l'ensemble des états finaux $S^f_{CP_+}$.

Définition 15 *Composition en cycle* $\otimes_{(\circlearrowleft, s^f_{CP})}$.

Soit

— $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *est un protocole de conversation avec un état final* $s^f_{CP} \in S_{CP}$ *et*

— CP_b *est un protocole de conversation basique avec* $T_{CP_b} = \{s_{CP_b} \xrightarrow{l_{CP_b}} s'_{CP_b}\}$ *tel que* $s'_{CP_b} \in S_{CP}$.

La composition de CP avec CP_b *à l'état* s^f_{CP} *dénoté par* $CP_{\circlearrowleft} = \otimes_{(\circlearrowleft, s_{CP})}(CP, CP_b)$ *est défini comme suit :*

— $S_{CP_{\circlearrowleft}} = S_{CP}$

— $L_{CP_{\circlearrowleft}} = L_{CP} \cup \{l_{CP_b}\}$

— $T_{CP_{\circlearrowleft}} = T_{CP} \cup \{s_{CP} \xrightarrow{l_{CP_b}} s'_{CP_b}\}$

— $S^f_{CP_{\circlearrowleft}} = S^f_{CP}$

La condition $s'_{CP_b} \in S_{CP}$ *signifie que la composition en cycle* $CP_{\circlearrowleft} = \otimes_{(\circlearrowleft, s_{CP})}(CP, CP_b)$ *ajoute une transition avec l'état final* s^f_{CP} *en tant qu'état source (*$S^f_{CP_{\circlearrowleft}}$ *n'est pas mis à jour). Ainsi, deux cas peuvent être distingués :*

— CP_b *introduit un cycle, ç.a.d., itération dans le CP*

— CP_b *introduit un graphe dirigé, ç.a.d., la transition ajoutée ne définit pas de cycle dans* CP_b.

2 Réalisabilité-par-construction : définition des conditions suffisantes

Construire un *CP* en utilisant les opérateurs définis dans la section 1 ne garantit pas sa réalisabilité, comme indiqué dans la définition 12. Les définitions de ces opérateurs sont syntaxiques et structurelles. Ils ne donnent aucune information sémantique pour garantir la réalisabilité des *CPs* obtenus.

Dans notre précédent travail [16, 17], nous avons donné une définition préliminaire des conditions requises pour chaque opérateur de composition afin de garantir la réalisabilité de *CP* généré. Ces conditions ont été décrites de manière informelle et aucune preuve mathématique n'a donc été associée à leurs définitions.

Dans cette section, nous formalisons d'abord nos conditions suffisantes pour la réalisabilité. Ensuite, chaque condition est illustrée par un exemple ainsi qu'un contre exemple pour mieux la comprendre. Notant que ces conditions sont définies, sur les protocoles de conversation, au niveau structurel (syntaxique). Par conséquent, leur vérification est effectuée à un niveau syntaxique et ne nécessite pas d'opérations complexes du système, c'est-à-dire projection et composition du système en mode de communication synchrone et asynchrone.

Condition 1 (Choix déterministe DC) *Tout protocole de conversation CP passé en tant que paramètre de l'opérateur de composition en séquence, choix et cycle doit appartenir à l'ensemble DC (voir Définition 4).*

Example 1 *La figure 5.1 montre un exemple de CP non déterministe (Figure 5.1 (a)) et les entités obtenues par projection à partir de ce CP (Figure 5.1 (b)). Notant que, lors d'une communication asynchrone, les entités peuvent déclencher leurs*

transitions en pointillés. Considérant un tel scénario de communication, l'entité P1 *subira un blocage directement après l'envoi du message* a ! *étant donné que, la réception du message de* b ? *ne peut jamais être exécuté, ç.a.d., l'entité communicante* P2 *commence par recevoir* a ? *puis envoyer* c ! *à la file d'attente de* P1. *Un tel exemple illustre un problème de communication dû au non déterminisme. Dans notre algèbre, nous considérons que CP est déterministe pour éviter la violation de la réalisabilité due à un non-déterminisme.*

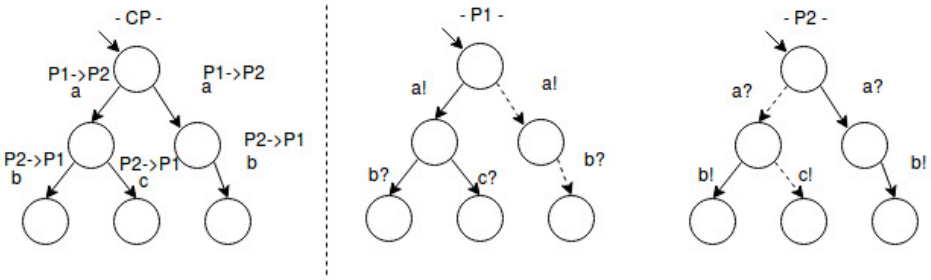


FIGURE *5.1 – Choix non déterministe CP*

Condition 2 (Absence de séquences indépendantes (ISeqF)) *La condition liée à deux échanges de messages consécutifs au niveau du CP est décomposé en deux conditions correspondantes à deux ordres d'échange de messages. La première condition préserve l'ordre d'envoi et de réception des messages en définissant une structure en anneau conformément à [45]. La deuxième condition conserve l'ordre d'envoi des messages et aucune contrainte n'est imposée à l'ordre de réception des messages.*

Préservation de l'ordre d'envoi-réception des messages. *Soit ISeqF est l'ensemble CPs sans séquences indépendantes. Donc, CP ∈ ISeqF si est seulement si*

$$\forall s_{CP} \in S_{CP}, \nexists\{s_{CP} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s'_{CP},\ s'_{CP} \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s''_{CP}\} \subseteq T_{CP} \text{ tel que } \mathcal{P}_j \neq \mathcal{P}_k.$$

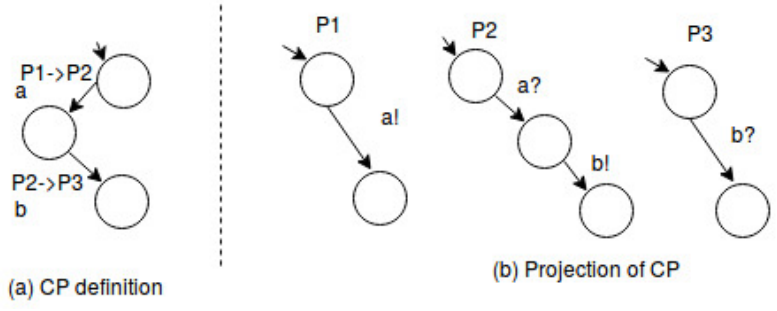


FIGURE *5.2 – Préservation de l'ordre des messages d'envoi-réception*

Cette condition conserve l'ordre d'envoi et de réception des messages. Elle impose une topologie en anneau d'entités communicantes.

Conservation de l'ordre d'envoi de messages. *Soit ISeqF est l'ensemble de CPs sans séquences indépendantes. Donc, CP ∈ ISeqF si est seulement si*

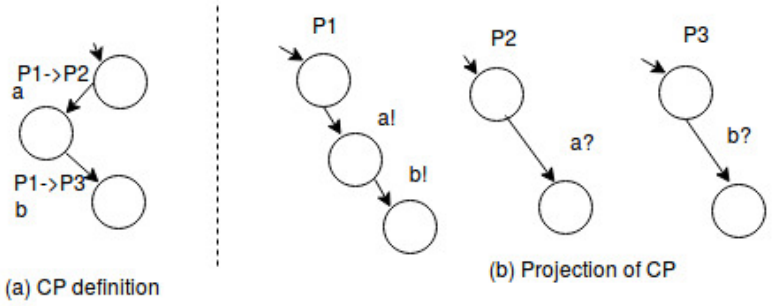


FIGURE *5.3 – Conservation de l'ordre d'envoi de messages*

$$\forall s_{CP} \in S_{CP}, \nexists\{s_{CP} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s'_{CP},\ s'_{CP} \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s''_{CP}\} \subseteq T_{CP} \text{ tel que } \mathcal{P}_j \neq \mathcal{P}_k \text{ et } \mathcal{P}_i \neq \mathcal{P}_k.$$

Cette condition conserve uniquement l'ordre d'envoi des messages.

La propriété ISeqF définit la séquence d'échanges de messages. En effet, supposons que *ISeqF* est valide. Donc,

— Quand $\mathcal{P}_k = \mathcal{P}_i$ à ce moment-là le message m' est envoyé par $\mathcal{P}_k = \mathcal{P}_i$ avant que m" soit envoyé par la même entité,

— quand $\mathcal{P}_k = \mathcal{P}_j$ alors le message m' est reçu par $\mathcal{P}_k = \mathcal{P}_j$ avant que m" soit envoyé par la même entité.

L'ordre d'envoi-réception de messages dans un protocole de conversation imposé par la condition ISeqF définit une séquence d'échanges de messages orientée dans ce protocole de conversation.

Example 2 *La figure 5.4 montre la satisfaction de la condition ISeqF par le CP réalisable. La communication des entités projetées indiquées du côté droit suit la même séquence que celle requise par le CP du côté gauche. En d'autres termes, procédant à la vérification du CP en suivant les étapes présentées précédemment dans la section 2 confirme que la réalisabilité est valable pour cet exemple. Ainsi, ici, la réalisabilité implique la condition 2.*

Example 3 (Contre-Exemple) *La figure 5.4 montre que le CP non réalisable ne satisfait pas la condition ISeqF. L'ensemble des entités (montrés à la figure 3(a)) obtenus par projection du CP côté gauche ne réalise pas leur CP (Figure 5.4 (a)) lors d'une communication asynchrone. La composition asynchrone permet de déclencher l'envoi du message* b ! *à l'état initial de* P3*, mais cela n'est pas spécifié à l'état initial du CP. Notant que le CP viole ISeqF et cela conduit à un échec de vérification de la réalisabilité.*

Condition 3 (Absence du choix parallèle (PCF)) *Soit PCF l'ensemble de CPs sans choix parallèle. Donc, CP ∈ PCF si est seulement si*

$$\forall s_{CP} \in S_{CP}, \nexists\{s_{CP} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s'_{CP} s_{CP} \xrightarrow{m'\ \mathcal{P}_k \to \mathcal{P}_q} s''_{CP}\} \subseteq T_{CP}, \text{ tel que } \mathcal{P}_i \neq \mathcal{P}_k \text{ et } s'_{CP} \neq s''_{CP}.$$

Le PCF renforce la propriété DC des protocoles de conversation. En faisant cela, un *CP* peut être composer d'un ensemble basique du CP_b en utilisant l'opérateur de choix, *ç.a.d.*, $\otimes_{(+,s^f_{CP})}(CP, \{CP_{bi}\})$, si et seulement si toutes les entités d'envois sont identiques sur les transitions sortantes de l'état de branchement s^f_{CP}.

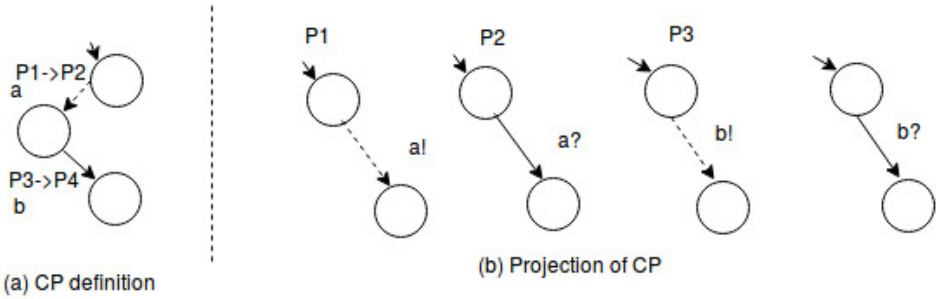


FIGURE 5.4 – *CP non réalisable violation de la condition ISeqF*

Example 4 *La figure 5.5 illustre un CP réalisable où PCF est respecté. Rappelant que l'on peut appliquer les étapes de vérification de la section 2 et en concluant que ce CP est réalisable. D'autre part, le même CP peut être construit en utilisant notre opérateur de choix tel que PCF est maintenue.*

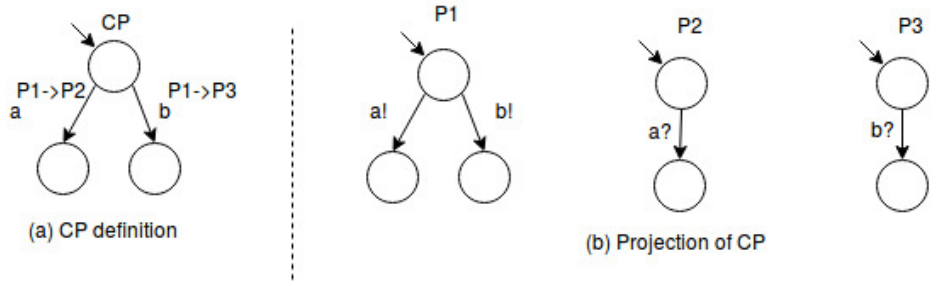


FIGURE 5.5 – *CP réalisable satisfaisant PCF*

Example 5 (Conter-exemple) *Nous désignons maintenant à la figure 5.6 un contre exemple où les transitions de branchement ne respectent pas PCF dans l'état initial, ç.a.d. Que deux entités différentes peuvent envoyer des messages à un même état du CP. Les entités montrées à droite de la figure 5.6 sont calculées par projection à partir du CP à gauche de la figure 5.6. Cependant, ces entités ne réalisent pas leurs CP car ils peuvent toutes les deux entrelacer les messages* a ! *et* b ! *lors d'une composition asynchrone, il est donc possible de déclencher soit* a ! *en premier et ensuite* b !*, ou* b ! *en premier suivi par* a !*. CP n'autorise pas ces comportements entrelacés, de tel sorte que la réalisabilité ne tient pas. Nous remarquons ici que la violation de PCF implique la non réalisabilité.*

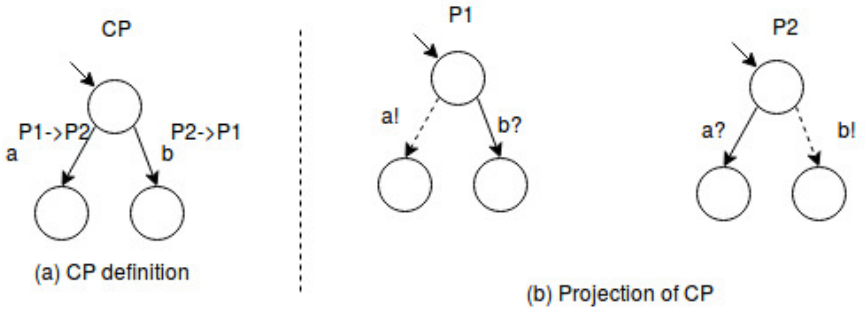


FIGURE 5.6 – *CP non-réalisable PCF violé*

Exemples 1, 3 et 5 donnés ci-dessus illustrent les problèmes d'interaction à éviter pour préserver la réalisation du système, et ceci peut être atteint par l'application des conditions 1, 2 et 3.

Remarque. L'intérêt de la propriété PCF est de faire en sorte que les entités d'envoi à l'état de branchement soient identiques. Lors de l'application de la propriété ISeqF à la présence d'un état de branchement, la propriété PCF garantit le raisonnement sur une seule entité lors de la vérification de la propriété ISeqF.

En conséquence, dans un état de branchement, l'ordre défini des messages d'envoi-réception appliqués par la séquence est toujours valable. La Figure 5.7 illustre cette situation.

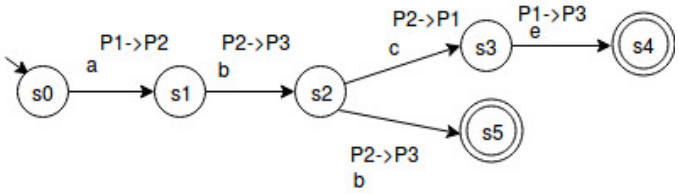


FIGURE *5.7 – Exemple illustratif d'un protocole de conversation avec séquence et choix satisfaisant ISeqF et PCF.*

La table 5.1 présente les théorèmes garantissant la réalisabilité par la construction incrémentale de CP, et cela utilise nos opérateurs de composition. Un théorème est associé à chacun de nos opérateurs. Ces théorèmes se basent sur les conditions définies auparavant. La preuve de chaque théorème est également donnée dans la section suivante.

Théorème 1	$CP_b \in R$
Théorème 2	$CP \in R \wedge CP_b \in R \wedge CP_{\gg} = \otimes_{(\gg, s^f_{CP})}(CP, CP_b) \in \text{ISeqF}$ $\Rightarrow CP_{\gg} \in R$
Théorème 3	$CP \in R \wedge \{CP_{bi}\} \subseteq R \wedge CP_+ = \otimes_{(+, s^f_{CP})}(CP, \{CP_{bi}\}) \wedge$ $CP_+ \in \text{DC} \wedge\ CP_+ \in \text{ISeqF} \wedge\ CP_+ \in \text{PCF}\ \Rightarrow\ CP_+ \in R$
Théorème 4	$CP \in R \wedge CP_b \in R \wedge CP_{\circlearrowleft} = \otimes_{(\circlearrowleft, s^f_{CP})}(CP, CP_b) \in \text{ISeqF}$ $\Rightarrow\ CP_{\circlearrowleft} \in R$

TABLE *5.1 – Théorèmes liés au CP réalisables-par-construction*

3 Réalisabilité-par-construction : théorèmes et preuves

Dans cette section, nous prouvons mathématiquement l'exactitude de notre proposition relative a la composition incrémentale des *CPs* réalisables-par-construction.

Dans la section 3.1, nous donnons le schéma de preuve ainsi que le modèle générique de preuve dans la section 3.2, qui sera instancié dans la section 3.3 pour chaque théorème de réalisation du CP_b, de la composition séquentielle, en branche et en cycle. La section 3.4 donne une description de formalisation de l'approche en B-événementiel qui sera détaillé dans la chapitre suivant.

3.1 Schéma de preuve.

Afin de démontrer l'exactitude des théorèmes donnés dans 5.1 , notre stratégie s'appuie sur ce qui suit.

— *Décomposition de la définition de réalisabilité.* La preuve repose sur une règle de preuve générique basée sur la décomposition de la condition de réalisabilité (Définition 12). Selon la condition de réalisation, il est nécessaire de prouver l'équivalence (Définition 9), la synchronisabilité (Définition 10) et la WF (Définition 11). La définition de l'équivalence, la synchronisabilité et la WF, nécessite le calcul de la projection $\downarrow CP_{op}$ (Définition 6). Sachant que, cette décomposition implique une preuve basée sur le raffinement. Ce processus est le même suivi pour formaliser les démonstrations des différents théorèmes de la table 5.1 en B-événement.

— *Établissement de la WF au niveau du système asynchrone.* Pour prouver la propriété WF, il est nécessaire de montrer que les files d'attente des messages du système asynchrone sont vides lorsque les états finaux de chaque entité communicante sont atteints et qu'un tel état est toujours accessible. Afin de prouver que les files d'attente sont vides, nous utilisons une variable de prophétie [65] pour prédire l'ensemble arbitraire de messages échangés. Cette variable définit une variante (cardinal de cet ensemble) qui se décrément à chaque fois qu'un message envoyé est reçu. Cet ensemble devient vide (le cardinal de l'ensemble devient égal à 0) lorsque tous les messages sont reçus.

— *Induction structurelle.* Les opérateurs de composition sont définis par induction en utilisant un cas de base sur un CP_b basique et un cas inductif sur un CP. La preuve est une induction structurelle sur les opérateurs définis. Soit $CP_b \in R$, $CP \in R$ et s_{CP}, respectivement, un CP basique réalisable, un CP réalisable et un état de collage. Donc, nous devons prouver que :

$$CP_{op} \in R \qquad \text{écriture équivalente à} \qquad \otimes_{(op,s_{CP})}(CP, CP_b) \in R$$

lorsque la condition suffisante définie Op_{cond} correspondant aux conditions 4,

3 et 2 de la section 2 définies auparavant et associées à chacun des opérateurs Op est satisfaite.

— Dans nos travaux de thèse, nous ne considérons pas les échanges de messages infinis. Toutes les traces considérées sont finies. Par conséquent, selon Lamport [65] qui a défini la notion de prophétie d'échange de messages, nous supposons qu'il existe un nombre naturel définissant le nombre d'échanges de messages. Pour chaque protocole de conversation considéré, nous considérons des traces *finies et complètes* (conduisant à des états finaux).

3.2 Modèle de preuve.

À partir du schéma de preuve présenté précédemment, nous formalisons le modèle de preuve dans l'équation 1.

$$CP \in R \wedge CP_b \in R \wedge \mathbf{Op_{cond}} \Longrightarrow \begin{cases} CP_{Op} \equiv Sys_{sync}(\downarrow CP_{\mathbf{Op}}) & (1.a) \\ \wedge & \\ Sys_{async}(\downarrow CP_{Op}) \in Synchronizable & (1.b)\ (1) \\ \wedge & \\ Sys_{async}(\downarrow CP_{Op}) \in WF & (1.c) \end{cases}$$

ou
— $CP_{Op} = \otimes_{(Op, s_{CP})}(CP, CP_b) \in R$
— Op_{cond} est la condition suffisante associée à l'opérateur de composition considéré Op.

3.3 Théorèmes de réalisabilité pour la séquence, le choix et l'itération.

Dans cette section, nous sommes en mesure de définir et de valider tous les théorèmes correspondant à la preuve par induction de la réalisabilité de tout CP construit à l'aide des opérateurs de composition définis. Quatre théorèmes (pour CP_b basique, séquence, choix et itération) sont donnés et prouvés.

Theorem 1 (Basic *CPs*).
Tout CP_b est réalisable.

Preuve 1. CP_b est constitué d'une seule transition de la forme $s \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s'$. Par conséquent, la projection produira deux entités communicantes $\mathcal{P}_i$ et $\mathcal{P}_j$ avec une seule transition où $\mathcal{P}_i$ envoie le message m a l'entité réceptrice $\mathcal{P}_j$. Cette projection est réalisable pour tous les cas possibles.

Theorem 2 (Séquence).
Soit $CP = <S_{CP}, s^0_{CP}, L_{CP}, T_{CP}>$, le protocole de conversation CP_b et $s^f_{CP} \in S^f_{CP}$ qui est un état final dans CP, alors les formules suivantes sont satisfaites.

Si $CP \in R$ *et* $CP_{\gg} = \otimes_{(\gg, s_{CP})}(CP, CP_b) \in$ *ISeqF alors*

$CP_{\gg} = \otimes_{(\gg, s^f_{CP})}(CP, CP_b) \in R$.

Preuve 2.

La preuve suit le modèle de preuve de l'équation 1 introduit dans la section 3.2 et schématisé dans la section 3.1.

Soit $CP_{\gg} = \otimes_{(\gg, s^f_{CP})}(CP, CP_b)$.

Ce modèle est réécrit (instancié) pour l'opérateur de séquence (équation 2) comme suit.

$$CP \in R \ \wedge \ CP_b \in R \ \wedge \ CP_{\gg} \in \text{ISeqF} \Longrightarrow \begin{cases} CP_{\gg} \equiv Sys_{sync}(\downarrow CP_{\gg}) & (2.a) \\ \wedge & \\ Sys_{async}(\downarrow CP_{\gg}) \in Synchronizable & (2.b)\ (2) \\ \wedge & \\ Sys_{async}(\downarrow CP_{\gg}) \in WF & (2.c) \end{cases}$$

Selon Théorème 1, $CP_b \in R$

La preuve est une induction.

Cas basique. Soit $CP = \varnothing$ et un CP_b tel que, $CP_{\gg} = \otimes_{(\gg, s^0_{CP})}(\varnothing, CP_b) = CP_b$. Alors, $CP_{\gg} \in R$ est valide selon Théorème 1.

Cas inductif. Soit CP $= <\mathrm{S}_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *et un* CP_b tel que, $CP \in R$ et $CP_b \in R$. Soit $s^f_{CP} \in S^f_{CP}$ est l'état de collage, (*ç.a.d.*, à la fois l'état final de CP et l'état initial de CP_b).

Selon le schéma de preuve de l'équation 2, nous devons prouver les propriétés 2.a, 2.b et 2.c

2.a Condition d'équivalence. Par hypothèse de récurrence nous écrivons $CP \equiv Sys_{sync}(\downarrow CP)$ et $CP_b \equiv Sys_{sync}(\downarrow CP_b)$. Supposons que la condition suffisante pour la séquence ISeqF est vérifiée, *ç.a.d.*, $CP_{\gg} \in$ ISeqF. Nous devons prouver que $CP_{\gg} \equiv Sys_{sync}(\downarrow CP_{\gg})$ (l'équation (2.a)).

Considérons les hypothèses suivantes :

— toute trace complète $\{s_0 \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_1, \ldots, s_n \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s_{n+1}\} \in Tr_{CP}$ du protocol de conversation réalisable CP,

— et la trace $Tr_{CP_b} = \{s_{b0} \xrightarrow{m''^{\mathcal{P}_t \to \mathcal{P}_z}} s_{b1}\}$ du CP_b réalisable.

Selon la définition de l'opérateur de séquence, les états finaux s_{n+1} et s_{b0} sont fusionnés et $s_{b1} = s_{n+2}$ est le nouvel état final du $CP_{\gg}$. La condition ISeqF étant satisfaite, deux cas sont distingués.

1. **Soit** $\mathcal{P}_k = \mathcal{P}_t$, alors les suffixes suivants des traces se produisent pour les entités $\mathcal{P}_k = \mathcal{P}_t$, $\mathcal{P}_q$ et $\mathcal{P}_z$
   — $\{\ldots, s^k_n \xrightarrow{m'!} s^k_{n+1}, s^k_{n+1} \xrightarrow{m''!} s^k_{n+2}\} \subseteq Tr_k$ ou s^k_{n+1} est l'état de collage et s^k_{n+2} est un nouvel état final du $\mathcal{P}_k$
   — $\{\ldots, s^q_n \xrightarrow{m'?} s^q_{n+1}\} \subseteq Tr_q$

— $\{\ldots, s^z_{n+1} \xrightarrow{m''?} s^z_{n+2}\} \subseteq Tr_z$ ou s^z_{n+1} est l'état de collage, s^z_{n+2} est un nouvel état final du $\mathcal{P}_z$,

2. **ou** $\mathcal{P}_q = \mathcal{P}_t$, donc les traces suivantes se produisent pour les entités $\mathcal{P}_q = \mathcal{P}_t$, $\mathcal{P}_k$ et $\mathcal{P}_z$
   — $\{\ldots, s^k_n \xrightarrow{m'!} s^k_{n+1}\} \subseteq Tr_k$
   — $\{\ldots, s^q_n \xrightarrow{m'?} s^q_{n+1}, s^q_{n+1} \xrightarrow{m''!} s^q_{n+2}\} \subseteq Tr_q$ ou s^q_{n+1} est l'état de collage et s^q_{n+2} est le nouvel état final $\mathcal{P}_q$
   — $\{\ldots, s^z_{n+1} \xrightarrow{m''?} s^z_{n+2}\} \subseteq Tr_z$ ou s^z_{n+1} est l'état de collage s^z_{n+2} est un nouvel état final du $\mathcal{P}_z$

Grâce à la propriété ISeqF, nous observons que la transition envoi/réception (qui est une transition synchrone) de CP_b nécessite que soit l'entité émettrice, soit l'entité réceptrice du CP_b devront être utilisés par la transition finale du CP réalisable. Cette condition définit un chemin orienté (séquence finie) dans la topologie de graphe formé des entités communicantes.
De plus, cette transition est toujours effectuée après que les transitions d'envoi/réception des traces de $Sys_{sync}(\downarrow CP_{\gg})$ du CP réalisable sont complétés. La transition envoi/réception de CP_b devient la dernière transition de la trace de $Sys_{sync}(\downarrow CP_{\gg})$ ($s_{b1} = s_{n+2}$ est le nouvel état final). Donc, $CP_{\gg} \equiv Sys_{sync}(\downarrow CP_{\gg})$.

2.b Condition de synchronisation.
Considérant les hypothèses suivantes :
— toute trace complète $\{s_0 \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_1, \ldots, s_n \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s_{n+1}\} \in Tr_{CP}$ dans le CP réalisable
— et la trace $Tr_{CP_b} = \{s_{b0} \xrightarrow{m''^{\mathcal{P}_t \to \mathcal{P}_z}} s_{b1}\}$ dans le CP_b réalisable
Par les hypothèses de récurrence, nous écrivons $Sys_{async}(\downarrow CP) \in$ *Synchronizable*, $Sys_{async}(\downarrow CP_b) \in$ *Synchronizable*.
Nous devons prouver que $Sys_{async}(\downarrow CP_{\gg}) \in$ *Synchronizable* (équation 2.b). L'ensemble *Synchronizable* est déduit de l'équivalence et de la condition *ISeqF*. En effet, la dernière transition des traces de $\downarrow CP_{\gg}$ correspond à $Sys_{sync}(\downarrow CP_b) = \{s_{b0} \xrightarrow{m''} s_{b1}\}$ dans le système synchrone et à $Sys_{async}(\downarrow CP_b) = \{s_{b0} \xrightarrow{m"!} s_b, s_b \xrightarrow{m''?} s_{b1}\}$ dans le système asynchrone. Ici s_b est un état intermédiaire dans le $Sys_{async}(\downarrow CP_{\gg})$. Dans cet état intermédiaire, la file d'attente de $\mathcal{P}_z$ contient le message m'' et devient vide dans l'état s_{b1}, le nouvel état final de $\mathcal{P}_z$. *ISeqF* garantit que l'ordre d'envoi-réception défini par CP_b est préservé dans la projection. Donc, $Sys_{async}(\downarrow CP_{\gg}) \in$ *Synchronizable*.

2.c Condition WF. Encore une fois, par hypothèses de récurrence, nous écrivons $Sys_{async}(\downarrow CP) \in WF$, $Sys_{async}(\downarrow CP_b) \in WF$. Nous devons prouver que $Sys_{async}(\downarrow CP_{\gg}) \in WF$ (équation 2.c). Cela signifie que, par hypothèse, les files d'attente sont vides dans l'état final de $Sys_{async}(\downarrow CP)$ puisqu'il est réalisable donc bien formé (WF). Nous devons montrer que la file d'attente est toujours vide après l'échange du massage du CP_b.
Considérant les hypothèses suivantes :
— toute trace $T_{CP} = \{s_0 \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_1, \ldots, s_n \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s_{n+1}\}$ dans le CP realizable
— et la trace $T_{CP_b} = \{s_{b0} \xrightarrow{m''^{\mathcal{P}_t \to \mathcal{P}_z}} s_{b1}\}$ dans le CP_b réalisable

Puisque la condition ISeqF est vérifiée *ç.a.d.* $\otimes_{(\gg, s^f_{CP})}(CP, CP_b) \in$ ISeqF, deux cas sont distingués.

1. **Si** $\mathcal{P}_k = \mathcal{P}_t$, alors les suffixes suivants des traces se produisent pour les entités $\mathcal{P}_k = \mathcal{P}_t$, $\mathcal{P}_q$ et $\mathcal{P}_z$
   - $\{\ldots, s^k_n \xrightarrow{m'!} s^k_{n+1}, s^k_{n+1} \xrightarrow{m''!} s^k_{n+2}\} \subseteq Tr_k$ ou s^k_{n+1} est l'état de collage et s^k_{n+2} est le nouvel état final dans $\mathcal{P}_k$
   - $\{\ldots, s^q_n \xrightarrow{m'?} s^q_{n+1}\} \subseteq Tr_q$
   - $\{\ldots, s^z_{n+1} \xrightarrow{m''?} s^z_{n+2}\} \subseteq Tr_z$ ou s^z_{n+1} est l'état de collage s^z_{n+2} est le nouvel état final $\mathcal{P}_z$.

   Grâce à l'hypothèse de récurrence et à la synchronisabilité, pour toutes les traces, les files d'attente sont vides dans l'état *n+1* et toutes les traces des entités $\mathcal{P}_i$ ou $i \neq k$ et $i \neq z$ sont complétées. A l'état s^k_{n+1}, l'entité $\mathcal{P}_k$ envoie le message m" et ses traces sont complétées. La file d'attente de l'entité $\mathcal{P}_z$ contient le message m" (La transition de $\mathcal{P}_k$ à l'état s^k_{n+2}). À ce niveau, l'entité $\mathcal{P}_z$ consome le message m" par la derniére transition $s^z_{n+1} \xrightarrow{m''?} s^z_{n+2}$ et vide sa file d'attente. Donc, toutes les queues sont vides à l'état $n+2$ et toutes les traces sont complétées.
2. **ou** $\mathcal{P}_q = \mathcal{P}_t$, puis les traces suivantes se produisent pour les entités $\mathcal{P}_q = \mathcal{P}_t$, $\mathcal{P}_k$ et $\mathcal{P}_z$
   - $\{\ldots, s^k_n \xrightarrow{m'!} s^k_{n+1}\} \subseteq Tr_k$
   - $\{\ldots, s^q_n \xrightarrow{m'?} s^q_{n+1}, s^q_{n+1} \xrightarrow{m''!} s^q_{n+2}\} \subseteq Tr_q$ ou s^q_{n+1} est l'état de collage et s^q_{n+2} est un nouvel état final dans $\mathcal{P}_q$
   - $\{\ldots, s^z_{n+1} \xrightarrow{m''?} s^z_{n+2}\} \subseteq Tr_z$ ou s^z_{n+1} est l'état de collage s^z_{n+2} est un nouvel état final dans $\mathcal{P}_z$

   Le même raisonnement est valable pour ce cas. Grâce à l'hypothèse de récurrence et à la synchronisabilité, pour toutes les traces, les files d'attente sont vides dans l'état $n+1$ et toutes les traces des entités $\mathcal{P}_i$ ou $i \neq q$ et $i \neq z$ sont complétées. A l'état s^q_{n+1}, l'entité $\mathcal{P}_q$ complète ses traces par l'envoi du message m" et sa trace devient complète.

   La queue de l'entité $\mathcal{P}_z$ contient le message m" (la transition de $\mathcal{P}_q$ à l'état s^q_{n+2}). A ce niveau, l'entité $\mathcal{P}_z$ consomme le message m" par la dernière transition $s^z_{n+1} \xrightarrow{m''?} s^z_{n+2}$ et vide sa queue. Donc, toutes les queues deviennent vide à l'état $n+2$ et toutes les traces seront complétées.

Nous concluons que l'opérateur de composition de séquence défini préserve la réalisabilité. □

Theorem 3 (Branchement).
Soit $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$, $\{CP_{bi} \mid 2 \leq i \leq n\}$ *est l'ensemble de* n $(n \geq 2)$ *protocoles de conversation basique et* $s^f_{CP} \in S^f_{CP}$ *un état final dans* CP, *donc les formules suivantes sont valides.*

Si $CP \in R$, $CP_+ = \otimes_{(+, s^f_{CP})}(CP, \{CP_{b1}, \ldots, CP_{bn}\}) \in$ *ISeqF et*
$CP_+ = \otimes_{(+, s^f_{CP})}(CP, \{CP_{b1}, \ldots, CP_{bn}\}) \in$ *PCF donc*
$CP_+ = \otimes_{(+, s^f_{CP})}(CP, \{CP_{b1}, \ldots, CP_{bn}\}) \in R$.

Preuve 3.

La preuve suit le modèle de preuve de l'équation 1 présenté à la Section 3.2 et schématisé dans la section 3.1.

Soit $CP_+ = \otimes_{(+,s_{CP})}(CP, \{CP_{b1}, \ldots, CP_{bn}\})$. Ce modèle est réécrit (instancié) pour l'opérateur de choix dans l'équation 3 comme suit.

$$\begin{cases} CP \in R \wedge \ \forall\ CP_{bi},\ CP_{bi}\ \in R \wedge \\ CP_+ \in \text{ ISeqF } \wedge\ CP_+ \in \text{ PCF} \end{cases} \Rightarrow \begin{cases} CP_+ \equiv Sys_{sync}(\downarrow CP_+) & (3.a) \\ \wedge & \\ Sys_{async}(\downarrow CP_+) \in Synchronizable & (3.b)\ (3) \\ \wedge & \\ Sys_{async}(\downarrow CP_+) \in WF & (3.c) \end{cases}$$

Selon théorème 1, pour tout $CP_b \in \{CP_{b1}, \ldots, CP_{bn}\}$, nous avons $CP_b \in R$.

La preuve est inductive.

Cas de base. Soit $CP = \varnothing$ et l'ensemble de protocole de conversation basique $\{CP_{bi}\},\ i \geq 2, i \in \mathbb{N}$ tel que,

$$CP_+ = \otimes_{(+,s^0_{CP})}(\varnothing, \{CP_{b1}, \ldots, CP_{bn}\}) \in R.$$

Chaque protocole de conversation de base dans $\{CP_{bi}\}, i \in \mathbb{N},\ i \geq 2$ contient une seule transition de la forme $T_{CP_{bi}} = \{s_{b0} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_{bi}\}$ ou s_{b0} est l'état de collage avec le CP vide ($CP = \varnothing$). Ces transitions définissent un CP déterministe.

En plus de la propriété DC, par la condition PCF, pour toute paire d'entités $CP_{bj} \in \{CP_{bi}\}$ et $CP_{bk} \in \{CP_{bi}\},\ i \geq 2, i \in \mathbb{N}, j \neq k$ avec
$T_{CP_{bj}} = \{s_{b0} \xrightarrow{m^{\mathcal{P}_m \to \mathcal{P}_n}} s_{bj}\}$ et $T_{CP_{bk}} = \{s_{b0} \xrightarrow{m'^{\mathcal{P}_s \to \mathcal{P}_t}} s_{bk}\}$ nous avons $\mathcal{P}_m = \mathcal{P}_s$.

Les protocoles de conversation obtenus, composés de protocoles de conversation avec un seul état initial, sont donc réalisables de manière triviale car ils sont déterministes et il n'existe qu'une seul entité émettrice suivant la condition PCF. En effet, si cette entité $\mathcal{P}_i$ est unique. Considérons la trace $\{s_{b0} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_{bi}\} \in Tr_{CP_+}$ de CP_+.

Les traces des entités correspondantes sont de la forme
— $\{s^i_{b0} \xrightarrow{m!} s^i_{b1}\} \subseteq T_i$ de l'entité $\mathcal{P}_i$
— $\{s^j_{b0} \xrightarrow{m?} s^j_{b1}\} \subseteq T_j$ de l'entité $\mathcal{P}_j$
A partir de ces traces, l'équivalence, la synchronisabilité et la WF sont garanties.

Cas inductif. Soit $CP \in R$ est un protocole de conversation,

$\{CP_{bi} \in R,\ i \geq 2, i \in \mathbb{N}\}$ est un ensemble de CP basique et $s_{CP} \in S^f_{CP}$ l'état de collage, (*ç.a.d.*, à la fois l'état final de CP et l'état initiale de chaque $CP_b \in \{CP_{bi},\ i \geq 2, i \in \mathbb{N}\}$

Selon le schéma de preuve de l'équation 3, nous devons prouver les propriétés 3.a, 3.b et 3.c lorsque CP n'est pas vide.

3.a Condition d'équivalence. Par hypothèse de récurrence nous écrivons $CP \equiv Sys_{sync}(\downarrow CP)$ et $\{CP_{bi}\} \equiv Sys_{sync}(\downarrow \{CP_{bi}\})$. Supposant que les conditions suffisantes $CP_+ \in$ ISeqF et $CP_+ \in$ PCF sont satisfaites. Nous devons prouver que $CP_+ \equiv Sys_{sync}(\downarrow CP_+)$ (l'équation 3.a).
Considérant les hypothèses suivantes :

— toute trace $T_{CP} = \{s_0 \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_1, \ldots, s_n \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s_{n+1}\}$ dans le CP réalisable

— et la trace $T_{CP_b} = \{s_{n+1} \xrightarrow{m''^{\mathcal{P}_t \to \mathcal{P}_z}} s_{n+2}\}$ dans $\{CP_{bi}\}$.

Selon la condition PCF et puisque le protocole de conversation est déterministe, $\mathcal{P}_t$ est unique (une seule entité émettrice pour tous les $\{CP_{bi} \in R,\ i \geq 2, i \in \mathbb{N}\}$). De plus, en raison de la condition ISeqF, deux cas sont distingués.

1. **Si** $\mathcal{P}_k = \mathcal{P}_t$, alors les suffixes suivants des traces se produisent pour les entités $\mathcal{P}_k = \mathcal{P}_t$, $\mathcal{P}_q$ et $\mathcal{P}_z$
   — $\{\ldots, s_n^k \xrightarrow{m'!} s_{n+1}^k, s_{n+1}^k \xrightarrow{m''!} s_{n+2}^k\} \subseteq T_k$
   — $\{\ldots, s_n^q \xrightarrow{m'?} s_{n+1}^q\} \subseteq T_q$
   — $\{\ldots, s_n^z \xrightarrow{m''?} s_{n+1}^z\} \subseteq T_z$.
2. **ou** $\mathcal{P}_q = \mathcal{P}_t$, puis les traces suivantes se produisent pour les entités $\mathcal{P}_q = \mathcal{P}_t$, $\mathcal{P}_k$ et $\mathcal{P}_z$
   — $\{\ldots, s_n^k \xrightarrow{m'!} s_{n+1}^k\} \subseteq T_k$
   — $\{\ldots, s_n^q \xrightarrow{m'?} s_{n+1}^q, s_{n+1}^q \xrightarrow{m''!} s_{n+2}^q\} \subseteq T_q$
   — $\{\ldots, s_n^z \xrightarrow{m''?} s_{n+1}^z\} \subseteq T_z$

Ici, le même raisonnement que pour la séquence s'applique. Grâce à l'hypothèse ISeqF, nous observons que la transition envoi/réception (transition synchrone) de CP_{bi} nécessite que soit l'entité émettrice ou l'entité réceptrice du CP_{bi} soit utilisé par la transition finale du CP réalisable. De plus, il est toujours effectué une fois que les transitions envoi-réception du $Sys_{sync}(\downarrow CP_+)$ du CP réalisable sont terminées. La transition envoi/réception de CP_{bi} devient la dernière transition du $Sys_{sync}(\downarrow CP_+)$. Donc, $CP_+ \equiv Sys_{sync}(\downarrow CP_+)$.

3.b Condition de synchronisabilité.
Considérant les hypothèses suivantes :

— toute trace complète $\{s_0 \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_1, \ldots, s_n \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s_{n+1}\} \in Tr_{CP}$ dans le CP réalisable

— et la trace $Tr_{CP_b} = \{s_{b0} \xrightarrow{m''^{\mathcal{P}_t \to \mathcal{P}_z}} s_{b1}\}$ dans le CP_b réalisable correspondant à la branche choisie de la $\{CP_{bi}\}$

Par les hypothèses de récurrence, nous écrivons $Sys_{async}(\downarrow CP) \in Synchronizable$, $Sys_{async}(\downarrow \{CP_{bi}\}) \in Synchronizable$. Nous devons prouver que $Sys_{async}(\downarrow CP_+) \in Synchronizable$ (équation (3.b)).

La synchronisabilité est déduite de l'équivalence et des deux conditions requises *ISeqF* et *PCF*. En effet, pour chaque trace de la composition synchrone $Sys_{sync}(\downarrow CP_+)$ et de la composition asynchrone $Sys_{async}(\downarrow CP_+)$

— selon la condition *PCF*, pour chaque trace, l'entité émettrice de la dernière

transition du $\downarrow CP_+$ sont identiques. Ensuite, la dernière entité émettrice est $\mathcal{P}_z$

— et par la *ISeqF* les dernières transitions de chaque trace complétée satisfaits la propriété de synchronisabilité de 2.c.

Ainsi, la dernière transition des traces de $\downarrow CP_+$ est l'une des branches correspondant à une transition de l'un des protocoles de base introduits par l'opérateur de choix, à l'état $s_{b0} = s^f_{CP}$. Cette dernière transition des traces correspond à $Sys_{sync}(\downarrow CP_b) = \{s_{b0} \xrightarrow{m''} s_{b1} = s_{n+2}\}$ dans le système synchrone et $Sys_{async}(\downarrow CP_b) = \{s_{b0} \xrightarrow{m"!} s_b, s_b \xrightarrow{m''?} s_{b1} = s_{n+2}\}$ dans le système asynchrone. Ici, s_b est un état intermédiaire dans le $Sys_{async}(\downarrow CP_+)$. Dans cet état intermédiaire, la file d'attente $\mathcal{P}_z$ contient le message m'' et devient vide dans l'état s_{b1}, le nouvel état final de $\mathcal{P}_z$. *ISeqF* garantit que l'ordre d'envoi-réception défini par CP_b est conservé dans la projection. Donc, $Sys_{async}(\downarrow CP_+) \in$ *Synchronizable*.

3.c Condition WF. Encore une fois, par hypothèses de récurrence, nous écrivons $Sys_{async}(\downarrow CP) \in WF$, $Sys_{async}(\downarrow \{CP_{bi}\}) \in WF$. Nous devons prouver $Sys_{async}(\downarrow CP_+) \in WF$ (équation (3.c)). Ainsi, par hypothèses, les files d'attente sont vides dans l'état final de $Sys_{async}(\downarrow CP)$ puisqu'il est réalisable (donc WF). Nous devons montrer que la file d'attente est toujours vide après l'exécution des échanges de messages de CP_{bi}.

Considérons les traces suivantes :

— toute trace $T_{CP} = \{s_0 \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_1, \ldots, s_n \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s_{n+1}\}$ dans le CP réalisable

— et la trace $T_{CP_b} = \{s_{n+1} \xrightarrow{m''^{\mathcal{P}_t \to \mathcal{P}_z}} s_{n+2}\}$ de CP_{bj} dans l'ensemble de $\{CP_{bi}\}$ correspond a l'opérateur de choix.

Par hypothèse, les conditions *ISeqF* et *PCF* sont satisfaite
ç.a.d. $\otimes_{(+,s^f_{CP})}(CP, \{CP_{bi}\}) \in$ *ISeqF* et $\otimes_{(+,s^f_{CP})}(CP, \{CP_{bi}\}) \in$ *PCF*.

Selon les propriétés du déterminisme (DC) et *PCF*, tous les CP_{bj} de l'ensemble $\{CP_{bi}\}$ ont la même entité émettrice $\mathcal{P}_t$, c'est-à-dire qu'il n'y a qu'une seule entité émettrice pour tous les $\{CP_{bi}\}$.

Selon la propriété *ISeqF*, comme dans 2.c, la preuve considère deux cas :

soit $\mathcal{P}_k = \mathcal{P}_t$ ou $\mathcal{P}_q = \mathcal{P}_t$ pour la trace ci-dessus. Ici, le même raisonnement que pour la séquence tient pour établir la WF de l'opérateur de choix.

À ce niveau, nous pouvons conclure que l'opérateur de composition à choix défini préserve la réalisabilité. □

Theorem 4 *(***Itération***)*

Soit $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *,* CP_b *un protocole de conversation basique et* $s^f_{CP} \in S^f_{CP}$ *un état final dans* CP*, tel que.*

Si $CP \in R$ *et* $CP_{\circlearrowleft} = \otimes_{(\circlearrowleft, s^f_{CP})}(CP, CP_b) \in$ *ISeqF alors* $CP_{\circlearrowleft} \in R$.

Preuve 4.

Lors de l'application de l'opérateur de cycle (boucle), il faut distinguer deux cas. Soit l'opérateur introduit une itération ou non.

Lorsque l'opérateur de cycle n'introduit pas d'itération, le *CP* obtenu définit un graphe cyclique dirigé sans boucle. Dans ce cas, l'opérateur correspond à l'ajout d'un choix. Ce cas est couvert par les opérateurs précédemment adressés.

Dans la suite, nous considérons le cas où l'opérateur de cycle introduit une boucle.

La preuve est inductive et suit le modèle preuve de l'équation 1. Ce modèle est réécrit (instancié) pour l'opérateur de cycle dans l'équation 5 comme suit.

$$CP \in R, CP_b \in R \wedge\ CP_{\circlearrowleft} \in \text{ISeqF} \Longrightarrow \begin{cases} CP_{\circlearrowleft} \equiv Sys_{sync}(\downarrow CP_{\circlearrowleft}) & (4.a) \\ \wedge & \\ Sys_{async}(\downarrow CP_{\circlearrowleft}) \in Synchronizable & (4.b)\ (4) \\ \wedge & \\ Sys_{async}(\downarrow CP_{\circlearrowleft}) \in WF & (4.c) \end{cases}$$

Cas basique. Soit $CP = \varnothing$ et CP_b est un *CP* basique. $CP_{\circlearrowleft} = \otimes_{(\circlearrowleft, s^0_{CP})}(\varnothing, CP_b)$ est réalisable. La preuve est triviale puisque $CP_{\circlearrowleft}$ contient un seul état avec une transition en cycle. La projection produit deux entités, l'une avec une boucle d'envoi du message $!m$ et la seconde avec une boucle de réception du message $?m$.

Cas inductif.

Pour prouver la réalisabilité dans le cas d'une boucle, nous utilisons le résultat de [45] établissant la réalisabilité pour les protocoles de conversation définissant des anneaux orientés.

Étant donné que le nombre d'échanges de messages est fini, on peut supposer qu'il existe un nombre fini d'échanges de messages et donc un ensemble de traces complètes.

Soit $CP \in R$ et $CP_b \in R$.

tel que, $T_{CP_b} = \{s_{b0} \xrightarrow{m''\mathcal{P}_t \to \mathcal{P}_z} s_{b1}\}$ où $\mathcal{P}_t$ et $\mathcal{P}_z$ sont les entités d'envoi et de réception.

Soit $s^f_{CP} \in S^f_{CP}$ est l'état de collage.

Par la définition de l'opérateur de cycle, nous obtenons

— s^f_{CP} est à la fois l'état final de *CP* et l'état initiale de CP_b, alors $s^f_{CP} = s_{b0}$, et

— une boucle (un cycle dans la topologie graphique des entités) donc $s_{b1} \in S_{CP}$

Puis, considérant que :

— la propriété *ISeqF* est valable pour $CP_{\circlearrowleft}$, c'est-à-dire que l'ordre des messages d'envoi-réception est garanti,

— il y a un cycle dans la topologie de graphe formé par les entités communicantes

nous concluons que la topologie de graphe formée par les entités définit un anneau orienté. Ensuite, selon [45], le protocole de conversation défini est réalisable.

À ce niveau, nous pouvons conclure que l'opérateur de composition de boucle défini préserve la réalisabilité. □

3.4 Formalisation avec la méthode B-événementiel

Cette section présente une description des modèles formels des opérateurs définis dans la section 1. Un événement B-événementiel est associé à chaque opérateur, ce qui permet de créer des *CP*s réalisables-par-construction. Chaque événement (*initialisation*, *Sequence*, *Choix* et *Cycle*) est préservé par les conditions suffisantes identifiées précédemment.

L'opérateur de raffinement offert par la méthode B-événementiel s'est avéré efficace pour gérer les preuves complexes associées à chaque opérateur. Cet opérateur nous a permis de gérer la propriété de réalisabilité de manière incrémentale en introduisant en premier l'équivalence, puis la synchronisabilité et enfin la WF dans des raffinements spécifiques. Par conséquent, la stratégie de raffinement suivante a été définie :

Modèle racine. Le modèle racine définit les protocoles de conversation. Il introduit le *CP* de base. Chaque opérateur de composition est défini comme un événement qui construit progressivement le *CP* final obtenu en introduisant un état final (des états finaux) à chaque étape de composition. Tous les *CPs* construits satisfont l'invariant du DC (Condition 1). Ce modèle déclare également la variable de prophétie comme variable d'état. Cette variable définit un nombre arbitraire de messages échangés et est utilisée pour définir une variante afin de prouver la WF de la composition.

Premier raffinement : le modèle synchrone. Le deuxième modèle est obtenu en raffinant chaque événement (opérateur de composition) pour définir la projection synchrone. Une invariante de collage liant le *CP* à la projection synchrone est introduit. La propriété d'équivalence est prouvée à ce niveau. Il est défini comme un invariant préservé par tous les événements encodant les opérateurs de composition. Cette projection représente le système synchrone, elle conserve l'ordre des échanges de messages entre entités et masque les échanges asynchrones.

Deuxième raffinement : le modèle asynchrone. Le dernier modèle introduit la projection asynchrone. Chaque événement (opérateur de composition) est raffiné pour gérer la communication asynchrone. Les projections synchrones et asynchrones sont liées par une autre invariante de collage. Des actions d'envoi et de réception, ainsi que des actions de traitement de la file d'attente et une variante décroissante de la variable de prophétie sont introduites. Ils sont nécessaires pour prouver la synchronisabilité et la WF exprimées en invariantes. Le raffinement des modèles synchrones dans un modèle asynchrone facilite le processus de preuve.

Lors du dernier niveau de raffinement, la réalisabilité est prouvée grâce à la préservation des invariantes et au processus de preuve inductive géré par B-événementiel sur la plate-forme Rodin.

Le chapitre suivant décrit ce développement. Pour chaque étape de raffinement, nous introduisons les définitions, axiomes et théorèmes nécessaires à la construction du modèle.

4 Cas d'étude

Afin d'illustrer notre approche, nous considérons une étude de cas complète issue de la littérature traitant le problème de la réalisabilité. Nous montrons dans cette section, comment notre approche s'applique à une étude de cas impliquant tous les opérateurs de composition. Cette étude de cas est prise de [83] qui ont étudié le problème de réalisabilité des systèmes.

Exigences

L'étude de cas concerne un processus d'achat en ligne. Dans [83], les exigences suivantes sont spécifiées.

Un simple processus d'achat dans un environnement virtuel est considérée. Ça comprend un client (B), un service de comptabilité (A) et un département logistique (L). Le service de comptabilité (A) approuve une commande (message Order) envoyé par le client (B) et le transmet au service logistique (L) (message Deliver) livre les marchandises demandées. Le service logistique (L) confirme la réception du message (en renvoyant le message Deliver_Conf) au service comptabilité (A), qui lui-même transfère ce message (avec la date de livraison prévue et le numéro de suivi du colis) au client (B) (message Delivery). Le client (B) peut terminer la commande en prévenant le service de la comptabilité et le service logistique (message Terminate) ou effectuer le suivi du colis (messages Get_Status et Status) des marchandises expédiées. Les messages correspondants sont transférés par le service comptabilité (A) au service logistique (L).

Un protocole de conversation possible

Pour commencer, nous avons conçu un protocole de conversation complet correspondant aux exigences décrites ci-dessus. Les concepts suivants ont été identifiés.

- Trois entités sont distingués : l'entité client (B), l'entité département de logistique (L) et l'entité département comptabilité (A).
- L'ensemble suivant $\{Order, Deliver, Deliver\_Conf, Delivery, Get\_Status, Get\_StatusL, StatusL, Status, Terminate, TerminateL\}$ définit l'ensemble des messages échangés par les différentes entités communicantes.
- Enfin, le protocole de conversation proposé est illustré à la figure 8.1.

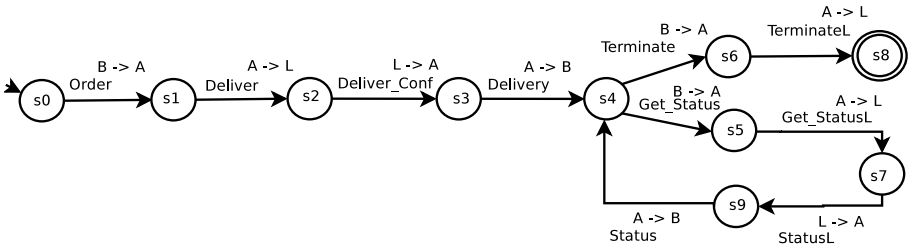


FIGURE *5.8 – Protocole de conversation d'un processus d'approvisionnement simple au sein d'une entreprise virtuelle*

Selon [83], le comportement des entités est conçu comme suit. *Initialement (état s_0), l'entité service de comptabilité (A) reçoit un message de commande envoyé par l'entité client (B) (état s_1). Ensuite, ce message est transmis à l'entité du service*

logistique (L) via le message Deliver (état s_2). Où, l'entité service logistique répond par un message Deliver_ Conf (à l'état s_3). Lorsque l'entité service comptabilité (A) reçoit ce message, il le transmet au client via le message Delivery (état s_4). A ce niveau (état s_4), deux alternatives sont valable. En effet, le service de comptabilité (A) peut recevoir un message Get_ status envoyé par le client (B) (état s_5), suivie d'une invocation du service logistique via le message Get_ StatusL (état s_7). Ensuite, deux messages Status et StatusL sont envoyés en tant que réponses par l'entité département logistique (L) (état s_9) et l'entité service de comptabilité (A) (état s_4).

A ce niveau (état s_4), ce processus peut être itéré. Alternativement, il est possible terminer l'entité service de comptabilité ainsi que l'entité département de logistique. Un message de terminaison est initié par l'entité service de compatibilité (B) (état s_6) envoyé au service de comptabilité (A), qui le transmet au service logistique (L) (état s_8). À cet état (s_8), les entité se terminent. Le protocole de conversation obtenu CP est formalisé avec 10 états, 7 messages échangés entre trois entités et 10 transitions. En utilisant les définitions introduites dans la section **??**. *Le protocole de conversation est formalisé comme suit.*

- $CP = <$
  - $S_{CP} = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$,
  - $s^0_{CP} = \{s_0\}$,
  - $L_{CP} = \{Order, Deliver, Deliver\_Conf, Delivery, Get\_Status, Get\_StatusL, Status, StatusL, Terminate, TerminateL\}$,
  - $T_{CP} = \{s_0 \xrightarrow{Order^{B\to A}} s_1, s_1 \xrightarrow{Deliver^{A\to L}} s_2, s_2 \xrightarrow{Deliver\_Conf^{L\to A}} s_3, s_3 \xrightarrow{Delivery^{A\to B}} s_4, s_4 \xrightarrow{Get\_Status^{B\to A}} s_5, s_4 \xrightarrow{Terminate^{B\to A}} s_6, s_5 \xrightarrow{Get\_StatusL^{A\to L}} s_7, s_6 \xrightarrow{TerminateL^{A\to L}} s_8, s_7 \xrightarrow{StatusL^{L\to A}} s_9, s_9 \xrightarrow{Status^{A\to B}} s_4\}$
  - $>$

Entités obtenues après projection

Selon la Définition 6, les entités communicantes sont obtenus à partir de *CP* par projection. La figure 5.9 illustre le service de la comptabilité (A), le client (B) et le service de logistique (L).

Les entités obtenues définissent les actions d'envoi (!) et de réception (?) des messages de l'ensemble L_{CP}. Ces entités réalisent les communications du protocole de conversation de la figure 8.1. En d'autres termes, le CP de la figure 8.1 est réalisable.

Dans ce qui suit, nous montrons, en utilisant l'approche que nous avons proposée, que le protocole de conversation de la figure 8.1 est réalisable.

Conception incrémentale du *CP* réalisables : un modèle formel

L'approche consiste à appliquer les différents opérateurs que nous avons définis sur un ensemble de protocoles de conversation basique en vérifiant que les conditions suffisantes associées à chaque opérateur de composition sont vérifiées à chaque fois qu'un opérateur est appliqué. Une séquence d'étapes est configurées pour construire le protocole de conversation de la figure 8.1 comme suit.

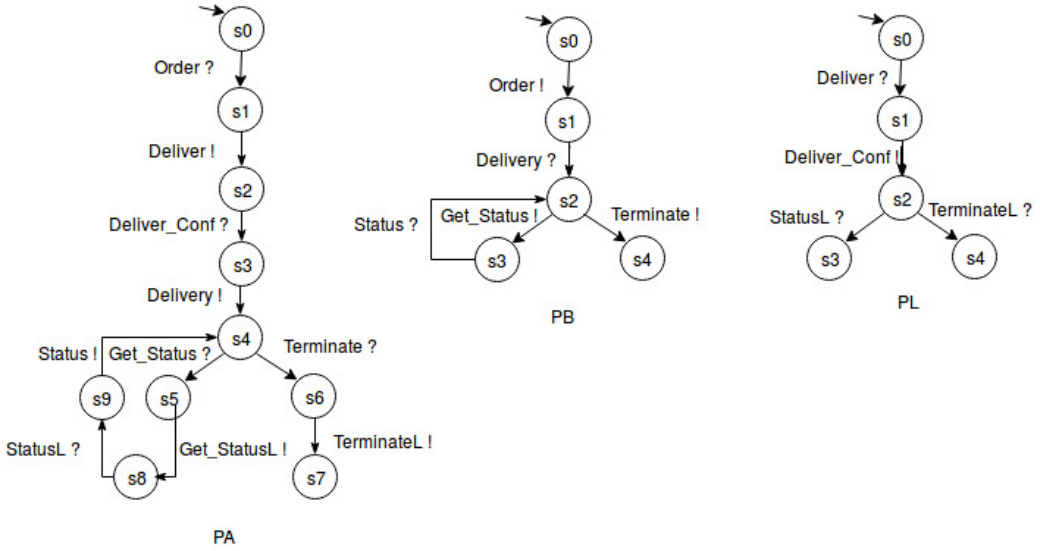


FIGURE *5.9 – Entités projetées d'un processus simple d'achat en ligne au sein d'une entreprise virtuelle*

Étape 1. Identification des protocoles de conversation basique CP_{bi}.

Étape 2. Initialisation du CP comme un CP vide.

Étape 3. Application des opérateurs de composition.

Étape 1. Identification des CP_b

À ce niveau, nous identifions l'ensemble des bases CP_{bi} comme suit. Seules les transitions de chaque CP_{bi} sont données.

- $CP_{b0} = s_0 \xrightarrow{Order^{B \to A}} s_1$
- $CP_{b1} = s_1 \xrightarrow{Deliver^{A \to L}} s_2$
- $CP_{b2} = s_2 \xrightarrow{Deliver\_Conf^{L \to A}} s_3$
- $CP_{b3} = s_3 \xrightarrow{Delivery^{A \to B}} s_4$
- $CP_{b4} = s_4 \xrightarrow{Get\_Status^{B \to A}} s_5$
- $CP_{b5} = s_4 \xrightarrow{Terminate^{B \to A}} s_6$
- $CP_{b6} = s_5 \xrightarrow{Get\_StatusL^{A \to L}} s_7$
- $CP_{b7} = s_6 \xrightarrow{TerminateL^{A \to L}} s_8$
- $CP_{b8} = s_7 \xrightarrow{StatusL^{L \to A}} s_9$
- $CP_{b9} = s_9 \xrightarrow{Status^{A \to B}} s_4$

Selon le théorème 1, tous les protocoles de conversation de base précédents sont réalisables.

Étape 2. Initialisation du CP

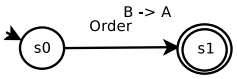
Soit CP le protocole de conversation correcte-par-construction en cours de construction. À l'initialisation, $CP_0 = \varnothing$. C'est un protocole de conversation réalisable.

Étape 3. Application des opérateurs de composition

A ce niveau, les opérateurs de composition sont appliqués. Chaque fois qu'un opérateur est appliqué, les conditions suffisantes sont vérifiées. Un nouveau CP est obtenu à chaque application d'opérateur. La séquence d'applications d'opérateurs commence à partir du CP vide définie à l'étape 2.

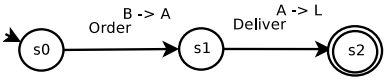
1. $CP_1 = \otimes_{(\gg, s^0_{CP})}(CP_0, CP_{b0})$. Le CP_0 vide et le protocole de conversation de base CP_{b0} sont composés à l'état initial s_0. Le CP_1 obtenu est réalisable par le théorème 1 avec le nouvel état final $s^f_{CP} = \{s_1\}$. Le protocole de conversation réalisable obtenue CP_1 est définie comme suit.
   — $CP_1 = <$
   — $S_{CP_1} = \{s_0, s_1\}$,
   — $s^0_{CP_1} = \{s_0\}$,
   — $L_{CP_1} = \{Order\}$,
   — $T_{CP_1} = \{s_0 \xrightarrow{Order^{B\to A}} s_1\}$
   $>$

   Le protocole de conversation réalisable CP_1 construit est décrit ci-dessous.



2. $CP_2 = \otimes_{(\gg, s^1_{CP_1})}(CP_1, CP_{b1})$. CP_1 est composé avec CP_{b1} à l'état final s_1 de CP_1 en utilisant l'opérateur de séquence $\otimes_{(\gg, s^f_{CP})}$. La condition suffisante ISeqF est satisfaite dans la composition, *ç.a.d.*, l'entité receptrice (A) du derniere transition CP_1 est la même entité émettrice que CP_{b1} avec le nouveau état final $s^f_{CP} = \{s_2\}$. Selon le théorème 2, le protocole de conversation réalisable obtenu CP_2 est définit comme suit.
   — $CP_2 = <$
   — $S_{CP_2} = \{s_0, s_1, s_2\}$,
   — $s^0_{CP_2} = \{s_0\}$,
   — $L_{CP_2} = \{Order, Deliver\}$,
   — $T_{CP_2} = \{s_0 \xrightarrow{Order^{B\to A}} s_1, s_1 \xrightarrow{Deliver^{A\to L}} s_2\}$
   $>$

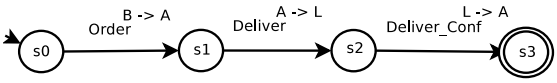
   Le protocole de conversation réalisable construit CP_2 est décrit ci-dessous.



3. $CP_3 = \otimes_{(\gg, s^2_{CP_2})}(CP_2, CP_{b2})$. Comme la composition précédente, CP_2 est composé avec le CP_{b2} basique en utilisant l'opérateur de séquence $\otimes_{(\gg, s^f_{CP})}$ à l'état final s_2 de CP_2. La condition suffisante ISeqF est valable pour cette composition, c'est-à-dire que l'entité récéptrice (L) de la dernière transition du CP_2 est la même entité émettrice de CP_{b2} avec le nouvel état final $s^f_{CP} = \{s_3\}$.

   Selon le théorème 2, le réalisable CP_3 obtenu est défini comme suit.
   — $CP_3 = <$
   — $S_{CP_3} = \{s_0, s_1, s_2, s_3\}$,
   — $s^0_{CP_3} = \{s_0\}$,
   — $L_{CP_3} = \{Order, Deliver, Deliver\_Conf\}$,
   — $T_{CP_3} = \{s_0 \xrightarrow{Order^{B\to A}} s_1, s_1 \xrightarrow{Deliver^{A\to L}} s_2, s_2 \xrightarrow{Deliver\_Conf^{L\to A}} s_3\}$
   $>$

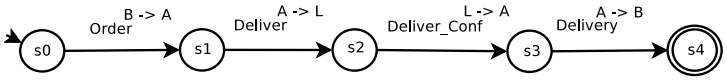
   Le protocole de conversation réalisable actuel construit par CP_3 est décrit ci-dessous.



4. $CP_4 = \otimes_{(\gg, s^3_{CP_3})}(CP_3, CP_{b3})$. Encore une fois, CP_3 est composé avec le protocole de conversation basique CP_{b3} en utilisant l'opérateur de séquence $\otimes_{(\gg, s^f_{CP})}$ à l'état final s_3 de CP_3. La condition suffisante ISeqF est valide pour cette composition, *ç.a.d.*, l'entité réceptrice (A) de la dernière transition du CP_3 est la même entité émettrice du CP_{b3} avec $s^f_{CP} = \{s_4\}$.

   Selon le théorème 2, le protocole de conversation CP_4 obtenu est définit comme suit.
   — $CP_4 = <$
      — $S_{CP_4} = \{s_0, s_1, s_2, s_3, s_4\}$,
      — $s^0_{CP_4} = \{s_0\}$,
      — $L_{CP_4} = \{Order, Deliver, Deliver\_Conf, Delivery\}$,
      — $T_{CP_4} = \{s_0 \xrightarrow{Order^{B\to A}} s_1, s_1 \xrightarrow{Deliver^{A\to L}} s_2, s_2 \xrightarrow{Deliver\_Conf^{L\to A}} s_3,$
      $s_3 \xrightarrow{Delivery^{A\to B}} s_4\}$
      $>$

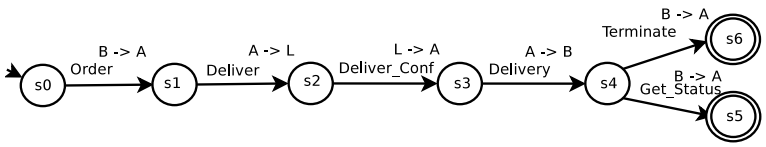
   Le protocole de conversation réalisable construit CP_4 est décrit ci-dessous.



5. $CP_5 = \otimes_{(+, s^4_{CP_4})}(CP_4, \{CP_{b4}, CP_{b5}\})$. A ce niveau, nous introduisons l'alternative entre vérifier le status ou terminer. CP_4 est composé à l'aide de l'opérateur de branchement $\otimes_{(+, s^f_{CP})}$ avec deux protocoles de conversation de base CP_{b4} et CP_{b5} à l'état final s_4 du CP_4. Selon le théorème 3, deux propriétés définissent la condition suffisante pour l'opérateur de branchement. Tout d'abord, la condition suffisante ISeqF est valable entre la dernière transition du CP_4 et chaque transition en branche CP_{b4} et CP_{b5}, c'est-à-dire l'entité réceptrice (B) de la dernière transition CP_4 est la même entité émettrice de CP_{b4} et de CP_{b5}. Deuxièmement, la condition PCF est valable pour chaque transition en branche, c'est-à-dire que la même entité émettrice (B) est présente sur chacune des transitions en branche de CP_{b4} et CP_{b5} avec les nouveaux états finaux $s^f_{CP} = \{s_5, s_6\}$. Ici, deux états finaux sont obtenus.

   Selon le théorème 3, le CP_5 réalisable obtenu est définie comme suit.
   — $CP_5 = <$
      — $S_{CP_5} = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$,
      — $s^0_{CP_5} = \{s_0\}$,
      — $L_{CP_5} = \{Order, Deliver, Deliver\_Conf, Delivery, Get\_Status,$
      $Terminate\}$,
      — $T_{CP_5} = \{s_0 \xrightarrow{Order^{B\to A}} s_1, s_1 \xrightarrow{Deliver^{A\to L}} s_2, s_2 \xrightarrow{Deliver\_Conf^{L\to A}} s_3,$
      $s_3 \xrightarrow{Delivery^{A\to B}} s_4, s_4 \xrightarrow{Get\_Status^{B\to A}} s_5, s_4 \xrightarrow{Terminate^{B\to A}} s_6\}$
      $>$

   Le protocole de conversation réalisable construit CP_5 est décrit ci-dessous.

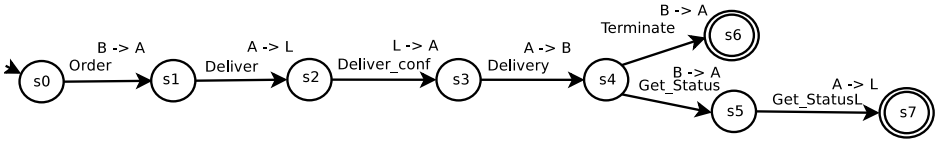


6. $CP_6 = \otimes_{(\gg, s^5_{CP_5})}(CP_5, CP_{b6})$. Nous suivons l'une des branches du CP obtenu CP_5. Le protocole de conversation CP_5 est composé avec le CP_{b6} basique en utilisant l'opérateur de séquence $\otimes_{(\gg, s^f_{CP})}$ à l'état final s_5 de CP_5. La condition suffisante ISeqF est valable pour cette composition, c'est-à-dire que l'entité réceptrice (A) de la dernière transition CP_5 est la même entité émettrice du CP_{b6} avec les nouveaux états finaux $s^f_{CP} = \{s_6, s_7\}$.

   Selon le théorème 2, le protocole de conversation réalisable obtenu CP_6 est défini comme suit.

   — $CP_6 = <$
   — $S_{CP_6} = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$,
   — $s^0_{CP_6} = \{s_0\}$,
   — $L_{CP_6} = \{Order, Deliver, Deliver\_Conf, Delivery, Get\_Status, Terminate, Get\_StatusL\}$,
   — $T_{CP_6} = \{s_0 \xrightarrow{Order^{B\to A}} s_1, s_1 \xrightarrow{Deliver^{A\to L}} s_2, s_2 \xrightarrow{Deliver\_Conf^{L\to A}} s_3, s_3 \xrightarrow{Delivery^{A\to B}} s_4, s_4 \xrightarrow{Get\_Status^{B\to A}} s_5, s_4 \xrightarrow{Terminate^{B\to A}} s_6, s_5 \xrightarrow{Get\_StatusL^{A\to L}} s_7\}$
   $>$

   Le protocole de conversation réalisable construit CP_6 est décrit ci-dessous.



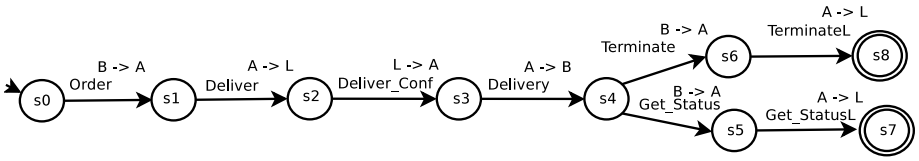
7. $CP_7 = \otimes_{(\gg, s^6_{CP_6})}(CP_6, CP_{b7})$. Ici, nous suivons et complétons l'autre branche. Le protocole de conversation CP_6 est composé avec le protocole de conversation basique CP_{b7} à l'aide de l'opérateur de séquence $\otimes_{(\gg, s^f_{CP})}$ à l'état final s_6 de CP_6. La condition suffisante ISeqF est valable pour cette composition, c'est-à-dire que l'entité réceptrice (A) de la dernière transition CP_6 est la même entité émettrice de CP_{b7} avec les nouveaux états finaux $s^f_{CP} = \{s_7, s_8\}$.

   Selon le théorème 2, le CP_7 réalisable obtenu est défini comme suit.

   — $CP_7 = <$
   — $S_{CP_7} = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$,
   — $s^0_{CP_7} = \{s_0\}$,
   — $L_{CP_7} = \{Order, Deliver, Deliver\_Conf, Delivery, Get\_Status, Get\_StatusL, Status, Terminate, TerminateL\}$,
   — $T_{CP_7} = \{s_0 \xrightarrow{Order^{B\to A}} s_1, s_1 \xrightarrow{Deliver^{A\to L}} s_2, s_2 \xrightarrow{Deliver\_Conf^{L\to A}} s_3, s_3 \xrightarrow{Delivery^{A\to B}} s_4, s_4 \xrightarrow{Get\_Status^{B\to A}} s_5, s_4 \xrightarrow{Terminate^{B\to A}} s_6, s_5 \xrightarrow{Get\_StatusL^{A\to L}} s_7, s_6 \xrightarrow{TerminateL^{A\to L}} s_8\}$

$>$

Le protocole de conversation réalisable construit CP_7 est décrit ci-dessous.

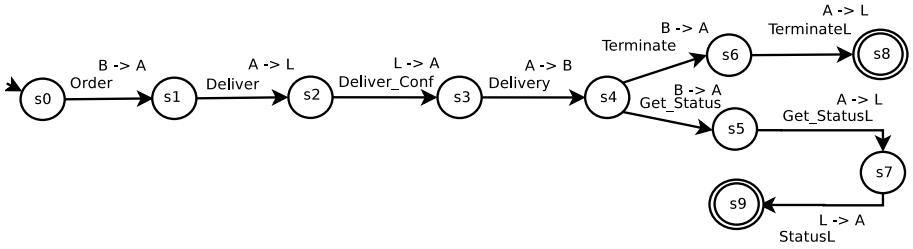


8. $CP_8 = \otimes_{(\gg, s^7_{CP_7})}(CP_7, CP_{b8})$. De retour à l'autre branche, CP_7 est composé avec CP_{b8} à l'aide de l'opérateur de séquence $\otimes_{(\gg, s^f_{CP})}$ à l'état final. s_7 de CP_6. La condition suffisante ISeqF est valable pour cette composition, c'est-à-dire que l'entité réceptrice (A) de la dernière transition du CP_7 est l'entité émettrice de CP_{b8}, avec les nouveaux états finaux $s^f_{CP} = \{s_8, s_9\}$.

   Selon le théorème 2, le CP réalisable obtenu est défini comme suit.

   — $CP_8 = <$
   — $S_{CP_8} = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$,
   — $s^0_{CP_8} = \{s_0\}$,
   — $L_{CP_8} = \{Order, Deliver, Deliver\_Conf, Delivery, Get\_Status, Get\_StatusL, Terminate, TerminateL\}$,
   — $T_{CP_8} = \{s_0 \xrightarrow{Order^{B\to A}} s_1, s_1 \xrightarrow{Deliver^{A\to L}} s_2, s_2 \xrightarrow{Deliver\_Conf^{L\to A}} s_3, s_3 \xrightarrow{Delivery^{A\to B}} s_4, s_4 \xrightarrow{Get\_Status^{B\to A}} s_5, s_4 \xrightarrow{Terminate^{B\to A}} s_6, s_5 \xrightarrow{Get\_Status^{A\to L}} s_7, s_6 \xrightarrow{TerminateL^{A\to L}} s_8, s_7 \xrightarrow{StatusL^{L\to A}} s_9\}$
   $>$

   Le protocole de conversation réalisable construit CP_8 est décrit ci-dessous.



9. $CP_9 = \otimes_{(\circlearrowleft, s^9_{CP_8})}(CP_8, CP_{b9})$. Enfin, nous introduisons la boucle ou l'itération en ajoutant un cycle dans CP_8 obtenu précédemment. CP_8 est composé à l'aide de l'opérateur d'itération $\otimes_{(\circlearrowleft, s^f_{CP})}$ avec CP_{b9} à l'état final s_9 de CP_8. La condition suffisante ISeqF est valable pour cette composition, c'est-à-dire que l'entité réceptrice (A) de la dernière transition CP_8 est la même entité émettrice de CP_{b9} avec le nouvel état final $s^f_{CP} = \{s_8\}$.

   Selon le théorème 4, le CP réalisable obtenu est défini comme suit.

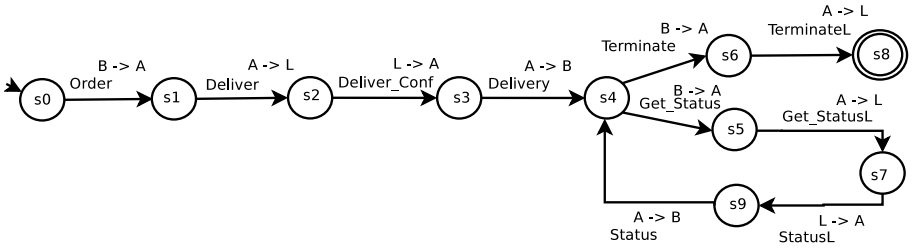
   — $CP_9 = <$
   — $S_{CP_9} = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$,
   — $s^0_{CP_9} = \{s_0\}$,

— $L_{CP_9} = \{Order, Deliver, Deliver\_Conf, Delivery, Get\_Status, Get\_StatusL, Status, Terminate, TerminateL\}$,

— $T_{CP_9} = \{s_0 \xrightarrow{Order^{B\to A}} s_1, s_1 \xrightarrow{Deliver^{A\to L}} s_2, s_2 \xrightarrow{Deliver\_Conf^{L\to A}} s_3, s_3 \xrightarrow{Delivery^{A\to B}} s_4, s_4 \xrightarrow{Get\_Status^{B\to A}} s_5, s_4 \xrightarrow{Terminate^{B\to A}} s_6, s_5 \xrightarrow{Get\_Status^{A\to L}} s_7, s_6 \xrightarrow{TerminateL^{A\to L}} s_8, s_7 \xrightarrow{StatusL^{L\to A}} s_9, s_9 \xrightarrow{Status^{A\to B}} s_4\}$

$>$

Le protocole de conversation **réalisable** et **final** CP_9 obtenu est décrit ci-dessous. Il correspond au protocole de conversation de la figure 8.1.



Chapitre 6

Formalisation : preuve formel avec B-événementiel

Sommaire

Dans ce chapitre, nous proposons une généralisation de notre cadre de composition incrémentale des *CPs* réalisables présenté dans le chapitre 5. Afin de le démontrer, nous l'instancierons sur le cas d'étude concret déjà présenté au chapitre 5 et obtiendrons un modèle final similaire.

Ce chapitre propose un modèle formel de composition incrémentale des systèmes réalisables développés prouvés et validés à l'aide de la stratégie de raffinement. B-événementiel prend en charge l'ensemble du développement formel des opérateurs de composition de *CP* réalisable. Le modèle générique développé peut être instancié sur un nombre quelconque de systèmes et l'approche proposée est générique : elle

ne dépend ni des composants internes des systèmes *ç.a.d.* entités communicantes, ni du type de composition.

1 Cadre mathématique pour la composition

Le paramètre mathématique formel permettant de gérer la composition de CP est présenté ci-dessous, fournissant les définitions mathématiques de base permettant de caractériser les systèmes réalisables par construction. Tous les éléments décrivant les systèmes et leur comportement sont introduits : variables, états, variantes, invariantes et événements.

1.1 Variables et états des systèmes de composition

Nos trois variables $Built\_CP$, $Built\_Synchrone$ et $Built\_Asynchrone$ représentent les états de composition des trois protocoles de conversation en cours de construction : CP, Sys_{sync} et Sys_{async}, respectivement. À l'initialisation les trois protocoles de conversation sont vide comme suit :

— $Built\_CP := \varnothing$
— $Built\_Synchrone := \varnothing$
— $Built\_Asynchrone := \varnothing$

Leurs valeurs sont prises dans les ensembles des, CP_bs basiques formant le CP réalisable par construction incrémentale, dénoté par CPs_B. L'ensemble des CP_bs basiques synchrones formant le Sys_{sync} dénoté par CPs_SYNC_B. Et l'ensemble des CP_bs basiques asynchrones formant le Sys_{async}, dénoté par CPs_ASYNC_B.

Les ensembles des transitions basiques sont définies comme suit :

— $CPs\_B \subseteq CP\_STATES \times PEERS \times MESSAGES \times PEERS \times CP\_STATES \times \mathbb{N}$
— $CPs\_SYNC\_B \subseteq CP\_STATES \times ACTIONS \times MESSAGES \times PEERS \times PEERS \times ACTIONS \times MESSAGES \times CP\_STATES \times \mathbb{N}$
— $CPs\_ASYNC\_B \in (A\_STATES \times ETIQ \times \mathbb{N}) \nrightarrow A\_STATES$

Tel que, CP_STATES est l'ensemble des états formant le CP. $PEERS \times MESSAGES \times PEERS$ est la définition de l'étiquette formant les transitions du CP afin de transmette un message dénoté par $MESSAGES$ entre deux entités émettrice et réceptrice, dénoté par $PEERS$ et d'un indexe dans l'ensemble des entiers naturels $\mathbb{N}$.

Les transitions basiques formant le CP synchrone Sys_{async} se diffère de celle du CP, où l'étiquette d'échange du message est découpé en deux parties, une première partie décrivant le message envoyé ou l'ensemble $ACTIONS$ est instancié par l'action d'envoi ! et une deuxième partie lié au message reçu instancié par l'action de réception ?. L'ensemble $ACTIONS$ définit aussi les transitions internes des entités communicantes, interprétée par une transition vide au niveau CP dénoté par τ.

L'ensemble CPs_ASYNC_B comporte les transitions basiques formant les entités communicantes. La transition est formée de deux états de l'ensemble A_STATES, d'une étiquette de l'ensemble $ETIQ$ et d'un indexe naturel.

1.2 Systèmes

Un système réalisable est un tuple défini comme une structure impliquant toutes les caractéristiques composant un système. Donc, pour tout système, nous définissons

$$Système = \langle variables, variantes, invariantes, initialisation, add\_operator \rangle$$

où :

— *variables* est un ensemble de variables représentant le CP à construire $Built\_CP$, son état initial $CP\_Initial\_state$ et l'ensemble des états finaux CP_Final_states. $Built\_Synchrone$ définit le système synchrone à construire Sys_{sync}. Le système asynchrone à construire dénoté par $Built\_Asynchrone$ utilise des files d'attente FIFO dénoté par *queue*. A_Traces représente les traces asynchrone de Sys_{async} comme suit :

$$\begin{aligned} Variables = \{&Built\_CP, CP\_Initial\_state, CP\_Final\_states, \\ &Built\_Synchrone, \\ &Built\_Asynchrone, A\_Traces, queue\} \end{aligned}$$

— *invariantes* est un prédicat défini sur les valeurs des variables :

$$\begin{aligned} Invariantes \in \{&Built\_CP \subseteq DC \wedge \\ &Built\_CP \twoheadrightarrow Built\_Synchrone \in Equivalence \wedge \\ &Built\_Synchrone \twoheadrightarrow A\_Trace \in Synchronisability \wedge \\ &A\_Traces \rightarrow Queue \in WF\} \end{aligned}$$

Afin de prouver la préservation de la propriéte de réalisabilité, les systèmes construits doivent préserver la propriété du choix déterministe dénoté par DC, la propriété d'équivalence entre CP et Sys_{sync} dénoté par $Equivalence$, la synchronisabilité entre Sys_{sync} et Sys_{async} dénoté par $Synchronisability$ et enfin la propriété du système bien formé WF.

— *variantes* est une opération produisant la valeur naturelle de la variante à partir d'une valorisation des variables :

$$Variantes = \{Prophecy\_of\_Sent\_Messages - Number\_of\_send\}$$

Tel que, $Prophecy\_of\_Sent\_Messages$ est une variable initialisé au nombre prévu d'échange de messages entre les entités communicantes, et $Number\_of\_send$ est le nombre des messages envoyés, initialisé à zéro et incrémenté de un, après chaque étapes de composition.

— *initialisation et add_operator* sont deux prédicats avant-après génériques qui changent d'état d'enregistrement.

1.3 Initialisation et composition

L'initialisation du système global initialise le $Built\_CP$, le $Built\_Synchrone$ et le $Buit\_Asynchrone$ à vide. Les événements de composition construit les CP, Sys_{sync} et Sys_{async} incrémentalement, tel que les trois systèmes satisfaits les invariants.

1.4 Conditions suffisantes

L'ensemble de nos conditions suffisantes préservant la réalisabilité des systèmes sont formalisées comme suit :

— $DC = CPs\_B \setminus Non\_Determinist\_CP$
— $ISeqF \subseteq CPs\_B$
— $PCF \in CP\_STATES \rightarrow \mathbb{P}(CPs\_B)$

Où, DC définit l'ensemble des systèmes déterministes. L'ensemble ISeqF caractérise la propriété qui préserve la dépendance des transitions en séquence. L'absence du choix parallèle est définit par la fonction totale PCF.

1.5 Propriété de réalisabilité

La propriété de réalisabilité à vérifier lors de la composition des systèmes est formalisée comme suit :

$$Réalisabilité = Equivalence \wedge Synchronisability \wedge WF$$

Tel que :

— $Equivalence \in CPs\_B \twoheadrightarrow CPs\_SYNC\_B$
— $Synchronisability \in CPs\_SYNC\_B \twoheadrightarrow R\_TRACE\_B$
— $WF \in A\_Traces \rightarrow queue$

La fonction de bijection *Equivalence* (one-to-one and onto relations) caractérise la relation d'équivalence entre *CP* identifié dans l'ensemble CPs_B et *syss* définit par CPs_SYNC_B.

Le même raisonnement s'applique pour la caractérisation de l'équivalence entre Sys_{sync} définit par l'ensemble CPs_SYNC_B et Sys_{async} définit par l'ensemble CPs_ASYNC_B.

La propriété WF est définit par une fonction totale vérifiant l'état de la file d'attente *queue* après l'exécution des traces asynchrones A_Traces.

2 Cadre générale de formalisation et preuve

Notre proposition théorique doit être prouvé par des théorèmes et preuves mathématique pour chaque type de composition. Ces preuves sont basées sur le principe de correction-par-construction.

La Figure 6.1 illustre les démarches générales que nous avons suivre afin d'implémenter notre modèle de composition des *CPs* réalisables en B-événementiel. Le modèle se compose de trois niveaux essentiels, le niveau *CP* (niveau abstrait), le niveau synchrone (le premier niveau de raffinement) et le niveau asynchrone (le deuxième niveau de raffinement).

La partie statique du modèle est décrite au niveau contexte. Afin de réduire la complexité des preuves, nous avons choisi de décomposer notre partie statique en trois sous parties, une dédié à la définition des éléments relatifs au niveau *CP*, une deuxième consacré à la définition des éléments du niveau synchrone et une dernière dans laquelle nous présentons les éléments nécessaire à la définition du niveau asynchrone, les trois contextes sont lié par la relation "REFINES" de raffinement. Nous

exprimons les interactions du système (partie comportementale) au niveau des machines B-événementiel.

Les machines et leurs relations constituent la partie dynamique du modèle. Dont le but est de faciliter le développement ainsi que la preuve du modèle, basé essentiellement sur les relations REFINES de raffinement entre les machines du modèle. La partie dynamique du modèle est lié à la partie statique par les relations SEES.

Au niveau *CP*, nous décrivons via une projection les entités communicantes composants le *CP*, chaque entité est muni d'un espace d'états formant les entités communicantes, un espace des messages (vocabulaire) et l'ensemble des transitions qui exprime les échanges des messages effectués entre entités. Au niveau synchrone nous identifions le comportement synchrone du système. Dans le dernier niveau nous exposons tous ce qui concerne le comportement asynchrone du système avec la définition de toutes les propriétés nécessaire pour la preuve de réalisabilité des *CPs* composés.

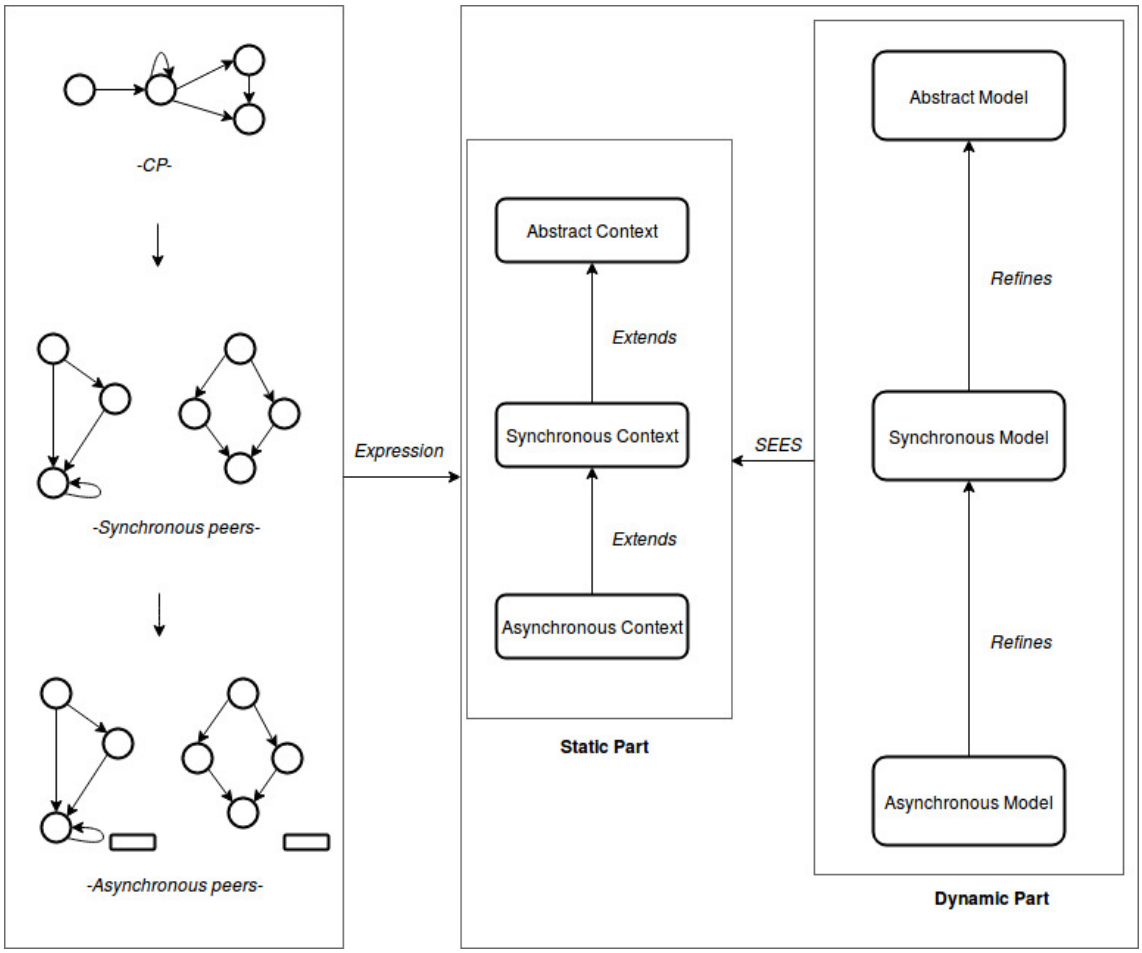


FIGURE *6.1 – Démarche générale*

Notre proposition est basée principalement sur la stratégie de raffinement hybride tel que nous combinons entre le raffinement vertical et horizontal.

La liaison entre les raffinements horizontales est effectué via les variables de collage (whithness). Chaque niveau de raffinement comporte de nouvelles variable et constantes, ce qui permet d'introduire progressivement les propriétés et conditions relatives au bon fonctionnement de chacun des niveaux du modèle. Les liens de collage servent aussi à préserver les propriétés du système à travers toute l'hiérarchie du modèle.

Le raffinement vertical est utilisé afin d'introduire de nouveaux comportements au niveau des modèles de raffinement ce qui permet de bien exprimer le modèle de raffinement.

L'intérêt de la stratégie de raffinement est la décomposition du modèle en plusieurs niveaux est de permettre la bonne maitrise du développement ainsi que de preuves. Les propriétés de chaque niveau sont préservé dans l'hiérarchie de raffinement.

Notre modèle B-événementiel se compose de trois contextes "$Lts\_Context$", "$Lts\_Sync\_Context$" et "$Lts\_Async\_Context$" aperçu (SEE) respectivement par trois machines "Lts_Model", "$Lts\_Synchronous\_Model$" et "$Lts\_Asynchronous\_Model$".

La première machine "Lts_Model" modélise le niveau abstrait du système, la machine est raffiné ensuite par la deuxième machine "$Lts\_Synchronous\_Model$" qui modélise le système en mode de communication synchrone. Ensuite la machine "$Lts\_Synchronous\_Model$" est raffiné par une troisième machine "$Lts\_Asynchronous\_Model$" qui modélise le mode de communication asynchrone du système.

Le modèle est illustré sur la Figure 6.2. Nous détaillons chacun des composants de notre modèles dans les section suivantes.

3 Un modèle B-événementiel pour la composition du système

Le paramètre mathématique décrit en Section 1 a été complètement formalisé en B-événementiel. Le développement complet du modèle B est disponible dans l'annexe A. Ce développement exprime d'abord la stratégie de composition de système à un niveau supérieur, puis réutilise ce développement pour chaque mode de communication en ligne et hors ligne adopté par le système spécifique.

Le système spécifique est obtenu par instanciation du modèle générique. L'instanciation est définie par une utilisation particulière du raffinement. Cette formalisation a conduit à la définition de trois contextes $Lts\_Context$, son raffinement $Lts\_Sync\_Context$ et le seconde raffinement $Lts\_Async\_Context$. Et de trois machines Lts_Model, son raffinement $Lts\_Synchronous\_Model$ et le seconde raffinement $Lts\_Asynchronous\ \_Model$.

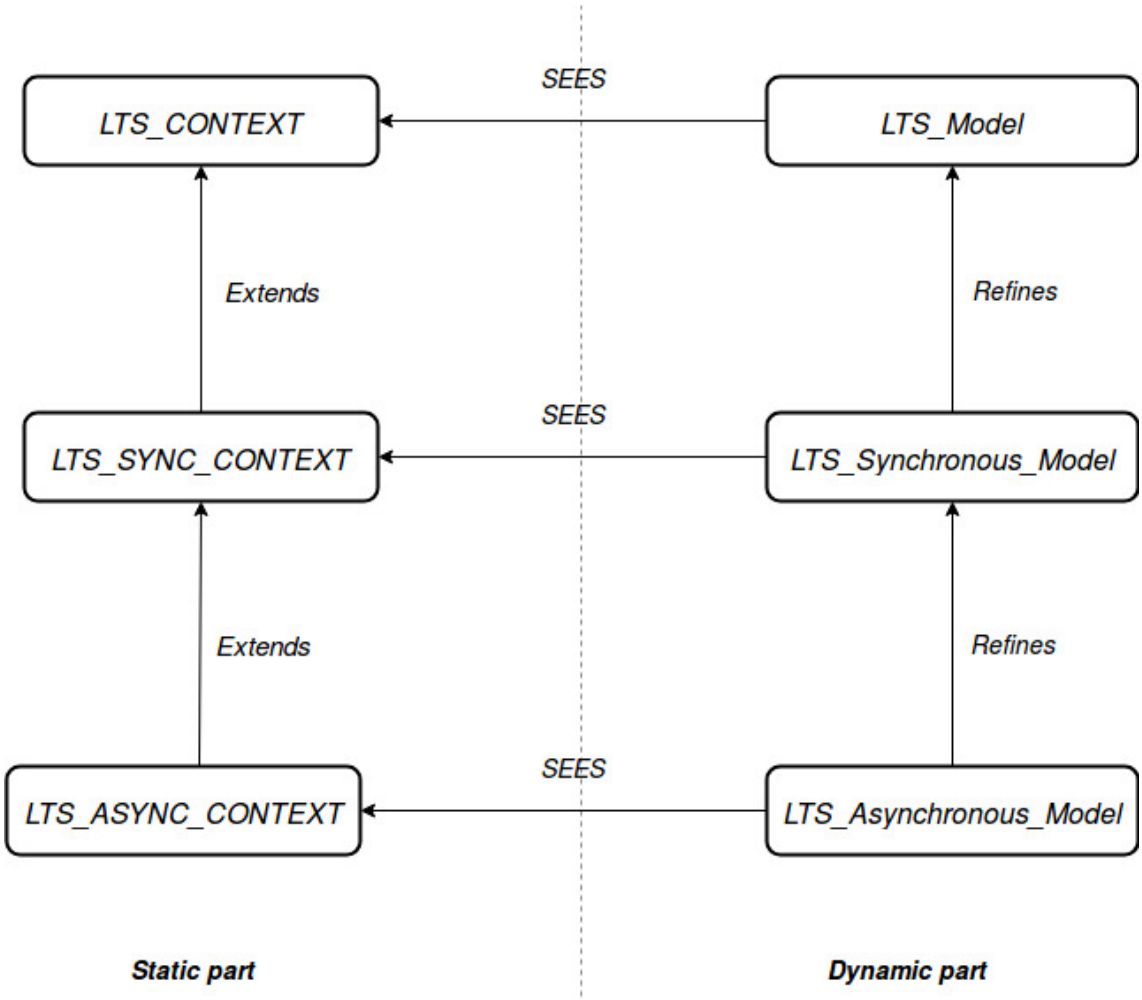


FIGURE *6.2 – Modèle en B-événementiel*

3.1 Partie statique : définitions requises

Le modèle racine. Il décrit la notion de CP et introduit la définition de chaque opérateur au niveau du CP. Chaque événement B-événementiel introduit correspond à la formalisation d'un opérateur défini dans la section 1.

Context *Lts_context*
Sets PEERS, MESSAGES , CP_STATES
Constants CPs_B, PEERs_B, DC, ISeqF, NDC, ...
Axioms

– – **Basic conversation protocols definition**
$axm1\_CP : CPs\_B \subseteq CP\_STATES \times PEERS \times MESSAGES \times PEERS \times CP\_STATES \times \mathbb{N}$

– – **Determinstic CP definition DC**
$axm3\_Cond1 : NDC \subseteq CPs\_B$
$axm4\_Cond1 : \forall Trans2, Trans1 \cdot ($
$Trans1 \in CPs\_B \wedge Trans2 \in CPs\_B \wedge SRC\_STATE(Trans1) = SRC\_STATE(Trans2) \wedge$
$LABEL(Trans1) = LABEL(Trans2) \wedge DST\_STATE(Trans1) \neq DST\_STATE(Trans2))$
$\Rightarrow \{Trans1, Trans2\} \subseteq NDC$
$axm5\_Cond1 : DC = CPs\_B \setminus NDC$

– – **Independent sequence freeness definition ISEQF**
$axm6\_Cond2 : ISeqF \subseteq CPs\_B$
$axm7\_Cond2 : \forall cp\_b \cdot (cp\_b \in CPs\_B \wedge (P\_SRC(cp\_b) = LAST\_SDR\_Ps(SRC\_STATE(cp\_b)) \vee$
$P\_SRC(cp\_b) = LAST\_RCV\_Ps(SRC\_STATE(cp\_b)))) \Rightarrow \{cp\_b\} \subseteq ISeqF$

– – **Parallel Choice freeness PCF**
$axm8\_Cond3 : PCF \in CP\_STATES \rightarrow \mathbb{P}(CPs\_B)$
$axm9\_Cond3 : \forall Trans1, Trans2.(Trans1 \in CPs\_B \wedge Trans2 \in CPs\_B \wedge$
$SRC\_STATE(Trans1) = SRC\_STATE(Trans2) \wedge$
$P\_SRC(Trans1) = P\_SRC(Trans2) \wedge$
$DST\_STATE(Trans1) \neq DST\_STATE(Trans2)) \wedge$
$\Rightarrow PCF(SRC\_STATE(Trans1)) = \{Trans1, Trans2\}$
...
End

Listing 6.1 – Un extrait du LTS_CONTEXT

Propriétés requises pour les CPs (voir Listing 6.1). Le listing 6.1 présente une partie du contexte B-événementiel utilisé au niveau abstrait. Nous introduisons, à l'aide d'ensembles et de constantes, l'ensemble des définitions de base des messages, des états de CP, des CPs basiques, etc.

Un ensemble d'axiomes est utilisé pour définir les propriétés pertinentes de ces définitions. Par exemple, dans l'axiome $axm1$, un CP est défini comme un ensemble de transitions avec un état source et un état cible, un message et des entités communicantes source et cible. Le choix déterministe (DC) est défini à partir des transitions choix non déterministes (NDC) dans les axiomes $axm3\_Cond1$ et $axm4\_Cond1$. Ensuite, les transitions déterministes DC sont obtenues par soustraction dans l'axiome $axm5\_Cond1$. La définition de ISeqF est donnée par les axiomes $axm6\_Cond2$ et $axm7\_Cond2$. Il compare l'entité source $P\_SRC(cp\_b)$ a l'entité émettrice $LAST\_SDR\_Ps$ ou à l'entité réceptrice $LAST\_RCV\_Ps$ de la dernière transition du CP. De même, afin de définir la propriété PCF, dans les axiomes $axm8\_Cond3$ et $axm9\_Cond3$, l'entité expéditrice $P\_SRC(Trans)$ des transitions impliquées dans une branche.

Le premier raffinement : modèle synchrone. L'objectif du premier raffinement est de construire la projection synchrone correspondant à la définition 7. Là encore, avant de construire cette projection, certaines définitions de propriétés sont nécessaires, notamment pour l'équivalence ($\equiv$), notée $EQUIV$ dans les modèles B-événementiel.

Context *Lts_Sync_Context* **Extends** *Lts_Context*
Sets ACTIONS
Constants CPs_B, PEERs_B, DC, ISeqF, NDC, ...
Axioms

$axm1 : CPs\_SYNC\_B \subseteq CP\_STATES \times ACTIONS \times MESSAGES \times PEERS \times$
$PEERS \times ACTIONS \times MESSAGES \times CP\_STATES \times \mathbb{N}$

– – **Equivalence of CP and Synchronous projection**
$axm\_1.a : EQUIV \in CPs\_B \rightarrowtail CPs\_SYNC\_B$
$axm\_1.a1 : EQUIV = \{Trans \mapsto S\_Trans | Trans \in CPs\_B \wedge S\_Trans \in CPs\_SYNC\_B \wedge$
$SRC\_STATE(Trans) = S\_SRC\_STATE(S\_Trans) \wedge$
$DST\_STATE(Trans) = S\_DST\_STATE(S\_Trans) \wedge$
$P\_SRC(Trans) = S\_P\_SRC(S\_Trans) \wedge$
$P\_DST(Trans) = S\_P\_DST(S\_Trans) \wedge$
$MSG(Trans) = S\_MSG(S\_Trans) \wedge$
$INDEX(Trans) = S\_INDEX(S\_Trans)\}$
...
End

Listing 6.2 – Un extrait du LTS_SYNC_CONTEXT

Propriétés requises pour la projection synchrone (voir Listing 6.2). La définition du système de transitions d'états correspondant à la projection synchrone est donnée par l'ensemble CPs_SYNC_B défini par l'axiome $axm1$ du Listing 6.2.

Des actions (send ! et receive ?) sont introduites. Ensuite, deux autres axiomes importants, $axm1.a$ et $axm1.a1$, sont donnés pour définir l'équivalence entre un *CP* et sa projection synchrone Sys_{sync}. La relation *EQUIV* est introduite. Elle caractérise l'ensemble des *CPs* équivalents à leur projection synchrone. L'axiom $axm1.a1$ formalise la définition 9 de la section 1.

Deuxième raffinement : modèle asynchrone. Le deuxième raffinement introduit la projection asynchrone avec les actions des entités émettrices et réceptrices. Les propriété système bien formé WF et synchronisabilité restent à prouver pour compléter la préservation de la réalisabilité.

La projection asynchrone (voir Listing 6.3). Les propriétés de synchronisabilité ($Sync(Sys_{sync}, Sys_{async})$), utilisée dans la Définition 10 et exprimée en axiomes $axm\_1.b$ et $axm\_1.b1$, et de la WF (WF (Sys_{async})) utilisée dans la Définition 11 et exprimées en axiomes $axm\_1.c$ et $axm\_1.c1$ sont introduites dans le contexte de ce niveau de raffinement. Ces deux propriétés complètent la preuve de réalisabilité.

Context $Lts\_Sync\_Context$ **Extends** $Lts\_Sync\_Context$
Sets $A\_STATES, \ldots,$
Constants $CPs\_ASYNC\_B, SYNCHRONISABILITY, WF, \ldots$
Axioms

$axm1 : CPs\_ASYNC\_B \in (A\_STATES \times ETIQ \times \mathbb{N}) \nrightarrow A\_STATES$

– – **Synchronisability property**
$axm\_1.b : SYNCHRONISABILITY \in CPs\_SYNC\_B \rightarrowtail R\_TRACE\_B$
$axm\_1.b1 : SYNCHRONISABILITY = \{S\_Trans \mapsto R\_Trans | S\_Trans \in CPs\_SYNC\_B \wedge$
$R\_Trans \in R\_TRACE\_B \wedge S\_INDEX(S\_Trans) = R\_INDEX(R\_Trans) \wedge$
$S\_SRC\_STATE(S\_Trans) = R\_SRC\_STATE(R\_Trans) \wedge$
$S\_P\_SRC(S\_Trans) = R\_P\_SRC(R\_Trans) \wedge$
$S\_MSG(S\_Trans) = R\_MSG(R\_Trans) \wedge$
$S\_P\_DST(S\_Trans) = R\_P\_DST(R\_Trans) \wedge$
$S\_DST\_STATE(S\_Trans) = R\_DST\_STATE(R\_Trans)\}$

– – **Well formedness property**
$axm\_1.c : WF \in A\_TRACES \rightarrow QUEUE$
$axm\_1.c1 : \forall A\_TR, queue \cdot (A\_TR \in A\_TRACES \wedge queue \in QUEUE \wedge queue = \varnothing)$
$\Rightarrow A\_TR \mapsto queue \in WF$
...
End

Listing 6.3 – Un extrait du LTS_ ASYNC_ CONTEXT

3.2 Partie dynamique : modélisation du comportement du *CP*

La machine racine. Ce modèle correspond à la définition du *CP*. Chaque opérateur correspond à un événement et contribue à la construction d'un *CP* donné représenté dans la variable d'état $BUILT\_CP$ qui doit définir uniquement le *CP* déterministe (voir invariant $inv1$ dans le Listing 6.4).

Invariants
$inv1 : BUILT\_CP \subseteq DC$
...
End

Listing 6.4 – Un extrait des invariants de la LTS_model.

Les définitions des événements B codant les opérateurs Définis (séquence de la Définition 13, branche de la Définition 14 et de la boucle de la Définition 15) sont données dans cette section. Chaque événement est protégé par les conditions suffisantes définies telles que, le *CP* et les entités projetées soient composées si et seulement si les conditions tiennent.

— **Initialisation.** Cet événement définit un $BUILT\_CP$ vide (dans l'action act1) comme *CP* initial.

Event $Initialisation \triangleq$
Any
Where
Then
$act1 : BUILT\_CP := \varnothing$
End

Listing 6.5 – Un extrait de l' :Ãľv :Ãľnement d'initialisation

— **Opérateur Add_Sequence.** L'événement *Add_Sequence* du Listing 6.6 correspond à l'opérateur de séquence (Définition 13). Son effet est d'ajouter un *CP* de base, donné en tant que paramètre, à savoir *Some_cp_b* au *CP* en construction (en utilisant l'opération union dans l'action *act*1). Il configure également le nouvel état final en action *act*2. Cet événement est déclenché uniquement si les conditions pertinentes sont respectées. En particulier, il est clairement indiqué, dans la condition *grd*3, que la propriété de séquence indépendante ISeqF doit être remplie avant d'ajouter un autre *CP* en séquence.

Event $Add\_Sequence \triangleq$
Any $Some\_cp\_b$
Where
$grd1 : Some\_cp\_b \in DC$
$grd2 : MSG(Some\_cp\_b) \neq End$
$grd3 : Some\_cp\_b \in \mathrm{ISeqF}$
$grd4 : SRC\_STATE(Some\_cp\_b) \in CP\_Final\_states$
...
Then
$act1 : BUILT\_CP := BUILT\_CP \cup \{Some\_cp\_b\}$
$act2 : CP\_Final\_states := (CP\_Final\_states \cup \{DST\_STATE(Some\_cp\_b)\}) \setminus \{SRC\_STATE(Some\_cp\_b)\}$
...
End

Listing 6.6 – Un extrait de l'op :Ãľrateur de composition de s :Ãľquence

— **Opérateur Add_Choice.** L'événement *Add_Choice* de Listing 6.7 modélise l'opérateur de choix (Définition 14). Il ajoute au *CP* en cours de composition un ensemble de CP_bs de base déterministes, donnés en paramètre, à savoir $Branches \subseteq DC$. De plus, tout CP_b de base, à savoir *branch* appartenant à *Branches* avec le même état source et la même entité émettrice, doit satisfaire les propriétés ISeqF et PCF (conditions *grd*3 et *grd*4). Les actions *act*1 et *act*2 mettent à jour le *CP* construit et les états finaux en conséquence.

Event $Add\_Choice \triangleq$
Any $Branches, branch$
Where
$grd1 : Branches \subseteq DC$
$grd2 : branch \in Branches$
$grd3 : BUILT\_CP \neq \varnothing \Rightarrow branch \in ISeqF$
$grd4 : Branches = PCF(SRC\_STATE(branch))$
$grd5 : SRC\_STATE(branch) \in CP\_Final\_states$
...
Then
$act1 : BUILT\_CP := BUILT\_CP \cup \{Branches\}$

$act2 : CP\_Final\_states := (CP\_Final\_states \cup$
$BR\_CP\_FINAL\_STATES(SRC\_STATE(branch))) \setminus \{SRC\_STATE(branch)\}$
...
End

Listing 6.7 – Un extrait de l'op :Ãľrateur de composition de choix

— **Opérateur Add_Loop.** L'événement *Add_Loop* du Listing 6.8 formalise l'opérateur de boucle (Definition 15) en ajoutant une transition de boucle (boucle auto ou boucle de cycle) donnée en paramètre, à savoir *Some_cp_b* au *BUIT_CP* dans l'action *act*1. De plus, il nécessite la propriété ISeqF dans *grd*2 et *grd*3. Notant que *grd*4 différencie l'événement *Add_Loop* et l'événement *Add_Sequence* en appliquant les états source et destination de la transition à ajouter *Some_cp_b* transition aux états déjà existants du *CP* construit.

Event *Add_Loop* ≙
Any *Some_cp_b*
Where
$grd1 : BUILT\_CP \neq \varnothing$
$grd2 : Some\_cp\_b \in DC \wedge (BUILT\_CP \neq \varnothing \Rightarrow Some\_cp\_b \in ISeqF)$
$grd3 : \exists Trans.(Trans \in BUILT\_CP) \wedge ((P\_SRC(Some\_cp\_b) = P\_SRC(Trans)) \vee$
$(P\_DST(Some\_cp\_b) = P\_SRC(Trans)))$
$grd4 : \exists Trans.(Trans \in BUILT\_CP) \wedge DST\_STATE(Some\_cp\_b) = SRC\_STATE(Trans)$
$grd5 : MSG(Some\_cp\_b) \neq End$
$grd6 : SRC\_STATE(Some\_cp\_b) \in CP\_Final\_states$
...
With
Then
$act1 : BUILT\_CP := BUILT\_CP \cup \{Some\_cp\_b\}$
...
End

Listing 6.8 – Un extrait de l'op :Ãľrateur de composition de boucle

La projection synchrone (cf. Listing 6.13). Le premier raffinement introduit la projection synchrone de *BUILT_CP* définie par la variable *BUILT_SYNC* dans le Listing 6.13. L'événement *Add_Sequence* ou l'opérateur de séquence (Listing 6.13) raffine le même événement du modèle racine. Il introduit l'ensemble *BUILT_SYNC* correspondant à la projection synchrone, comme indiqué dans Définition 7. Et, *Add_Sequence* s'applique que si les conditions sont valables. La clause *With* fournit un témoin pour coller la nouvelle transition *Some_cp_b* ajoutée au *CP*, avec sa version synchrone.

Event *Add_Sequence* **Refines** *Add_Sequence* ≙
Any $S\_Some\_cp\_b, Some\_cp\_sync\_b$
Where
$grd1 : Some\_cp\_sync\_b \in cps\_sync\_b$
$grd3 : S\_SOURCE\_STATE(Some\_cp\_sync\_b) \in CP\_Final\_states$ $grd4 : S\_Some\_cp\_b \in ISeqF$
$grd8 : MESSAGE(S\_Some\_cp\_b) \neq End$
$grd9 : MESSAGE(S\_Some\_cp\_b) = S\_MESSAGE(Some\_cp\_sync\_b)$
...
With $Some\_cp\_b : Some\_cp\_b = S\_Some\_cp\_b$
Then
$act1 : BUILT\_CP := BUILT\_CP \cup \{S\_Some\_cp\_b\}$
$act2 : BUILT\_SYNC := BUILT\_SYNC \cup \{Some\_cp\_sync\_b\}$
...
End

Listing 6.9 – Un extrait de l'op :Ãľrateur de composition de s :Ãľquence

Le Listing 6.10 présente l'invariant de la propriété d'équivalence ($\equiv$) $inv1.a$ correspondant à la Définition 9. L'invariant nécessite une équivalence entre un *CP* et sa projection synchrone Sys_{sync}.

L'invariant $inv2$ du Listing 6.10 décrit la propriété d'équivalence à l'aide de la relation $EQUIV$ définie dans le contexte synchrone (voir Listing 6.2). Ainsi, une partie de la propriété de réalisabilité (c'est-à-dire $CP \equiv Sys_{sync}$) de la Définition 9 est déjà prouvée à ce niveau de raffinement.

Invariants
$inv1 : BUILT\_SYNC \subseteq CPs\_SYNC\_B$
$inv\_1.a : \forall Trans \cdot \exists S\_Trans \cdot (Trans \in BUILT\_CP \land S\_Trans \in BUILT\_SYNC \land BUILT\_CP \neq \varnothing) \Rightarrow \mathbf{Trans} \mapsto \mathbf{S\_Trans} \in \mathbf{EQUIV}$
...
End

Listing 6.10 – Un extrait des invariants du mod :ÃÍle LTS_ Synchronous_ model.

La projection asynchrone (cf. Listing 6.12). Le deuxième raffinement introduit la projection asynchrone avec les actions (d'envoi et de réception) entre les entités émettrices et réceptrices. La WF et la synchronisabilité restent à prouver pour compléter la préservation de la réalisabilité.

Les invariants associés à ce modèle sont présentés dans le Listing 7.3. En particulier, les propriétés de synchronisabilité, exprimées en invariant $axm\_1.b$ utilisée dans la Définition 10 ($Sync(Sys_{sync}, Sys_{async})$) , et de système bien formé, exprimés en invariant $axm\_1.c$ utilisé dans la Définition 11 ($WF(Sys_{async})$) sont introduites dans les invariantes de ce niveau de raffinement. Ces deux propriétés complètent la preuve de réalisabilité.

Invariants
$inv1 : BUILT\_SYNC \subseteq CP\_SYNC\_B$
$inv2 : REDUCED\_TRACE \subseteq R\_TRACE\_B$
$inv3 : A\_TRACE \subseteq A\_TRACES$
$inv\_1.b : \forall S\_Trans \cdot \exists R\_Trans \cdot (S\_Trans \in BUILT\_SYNC \land R\_Trans \in REDUCED\_TRACE) \Rightarrow \mathbf{S\_Trans} \mapsto \mathbf{R\_Trans} \in \mathbf{SYNCHRONISABILITY}$
$inv\_1.c : \forall A\_Trans \cdot (A\_Trans \in A\_TRACES \land MESSAGE(Last\_cp\_trans) = End \land A\_TRACE \neq \varnothing) \Rightarrow \mathbf{A\_Trans} \mapsto \mathbf{queue} \in \mathbf{WF}$
$inv6 : BUILT\_ASYNC \subseteq CP\_ASYNC\_B$
...
End

Listing 6.11 – Un extrait des invariants du mod :ÃÍle LTS_ Asynchronous_ model

A ce niveau, chaque événement correspondant à un opérateur de composition est raffiné par trois événements : un pour gérer l'envoi de messages ($Add\_Sequence\_send$), un pour la réception de messages ($Add\_Sequence\_receive$) et un troisième ($Add\_Sequence\_send\_receive$) raffinant l'événement abstrait $Add\_sequence$.

Le Listing 6.12 définit ces événements. Les événements d'envoi et de réception sont entrelacés de manière asynchrone. Une fois qu'une paire d'événements d'envoi et de réception a été déclenchée, l'événement $Add\_Sequence\_send\_receive$ enregistre le fait que l'émission-réception est terminée. Cet événement augmente le nombre de messages reçus (action $act5$).

Les traces sont mises à jour en conséquence par les événements, elles servent à prouver les invariantes.

Event $Add\_Sequence\_Send \triangleq$
Any
$send, lts\_s, lts\_d, msg, index$

Where
$grd1 : \exists send\_st\_src, send\_st\_dest \cdot ((lts\_s \mapsto send\_st\_src) \in A\_GS \land ((send\_st\_src \mapsto (Send \mapsto msg \mapsto lts\_d) \mapsto index) \mapsto send\_st\_dest) \in CPs\_ASYNC\_B \land ...$
...

```
Then
 act1 : A_TRACE := A_TRACE ∪ {Reduces_Trace_states ↦ St_Num ↦
                    Send ↦ lts_s ↦ msg ↦ lts_d ↦ Reduces_Trace_states ↦
                    (St_Num + 1) ↦ A_Trace_index}
 act2 : queue, back := queue ∪ {lts_d ↦ msg ↦ back}, back + 1
 act3 : A_GS := A_Next_States({send} ↦ A_GS ↦ queue)
 ...
End
```

Un nouvel événement de terminaison *Add_End* est introduit au niveau asynchrone. *Add_End* ajoute une nouvelle transition échangeant le message *End* entre deux entités communicantes dans l'action *act*1.

```
Event Add_Sequence_Receive ≙
        Any
        send, receive, lts_s, lts_d, msg, index

        Where
        grd1 : queue ≠ ∅ ∧ lts_d ↦ msg ↦ front ∈ queue
        grd2 : ∃receive_st_src, receive_st_dest·(((lts_d ↦ receive_st_src) ∈ A_GS)∧
        ((receive_st_src ↦ (Receive ↦ msg ↦ lts_s) ↦ index) ↦ receive_st_dest)
        ∈ CPs_ASYNC_B ∧ ...
        ...

        Then
        act1 : A_TRACE := A_TRACE ∪ {Reduces_Trace_states ↦ St_Num ↦
                          Receive ↦ lts_s ↦ msg ↦ lts_d ↦ Reduces_Trace_states ↦ (St_Num + 1)
                          ↦ A_Trace_index}
        act2 : queue := queue \ {lts_d ↦ msg ↦ front}
        ...
    End

Event Add_Sequence_Send − Receive Refines Add_Sequence ≙
        Any
        A_Some_cp_b, A_Some_cp_sync_b, Send_cp_async_b, Receive_cp_async_b, R_trace_b

Where
        grd1 : A_MSG(Send_cp_async_b) = A_MSG(Receive_cp_async_b)
        grd2 : ACTION(Receive_cp_async_b) = Receive ∧ ACTION(Send_cp_async_b) = Send
        grd3 : A_Some_cp_b ∈ ISeqF
        grd4 : MSG(A_Some_cp_b) = A_MSG(Send_cp_async_b)
        ...

With S_Some_cp_b : S_Some_cp_b = A_Some_cp_b,
             Some_cp_sync_b : Some_cp_sync_b = A_Some_cp_sync_b

Then
        act1 : BUILT_CP := BUILT_CP ∪ {A_Some_cp_b}
        act2 : BUILT_SYNC := BUILT_SYNC ∪ {A_Some_cp_sync_b}
        act3 : BUILT_ASYNC := BUILT_ASYNC ∪ {Send_cp_async_b} ∪ {Receive_cp_async_b}
        act4 : REDUCED_TRACE := REDUCED_TRACE ∪ {R_trace_b}
        ...
   End
End
```

Listing 6.12 – Un extrait de LTS_Asynchronous_model

L'évènement *Add_End* est déclenché une seule fois lors de l'animation automatique du modèle de composition, c'est à la fin de la composition incrémentale, quand la variante exprimé dans la *grd*2 devient nul. La condition *grd*3 vérifie que le message échangé est égal a *End*. Une telle composition indique la fin de la composition.

Notant qu'un tel événement est introduit à fin de détecter la fin de la composition, le moment où la propriété WF doit être vérifié. La derniére transition ajoutée est stockée dans une variable *Last_cp_trans* dans l'action *act*2 utilisée pour écrire l'invariante de preuve du système bien formé (WF).

```
Event Add_End ≙
```

```
Any A_Some_cp_b
Where
    grd1 : Prophecy_of_Sent_Messages − Number_of_send = 0
    grd2 : SRC_STATE(A_Some_cp_b) ∈ CP_Final_states
    grd3 : MSG(A_Some_cp_b) = End
    ...
With Some_cp_b : Some_cp_b = S_Some_cp_b
Then
    act1 : BUILT_CP := BUILT_CP ∪ {A_Some_cp_b}
    act2 : Last_cp_trans := A_Some_cp_b
    ...
End
```

Listing 6.13 – Un extrait de l' :Ã'v :Ã'nement End

3.3 Une approche évolutive de construction des *CPs* réalisables

Notant que les raffinements des événements de composition permettent de valider le bon fonctionnement du système, tel que, l'équivalence, la synchronisabilité et la WF sont utilisés pour prouver la réalisabilité. Ici, seules les constructions permettant de définir les *CPs* réalisables corrects sont données. En effet, une fois les preuves effectuées, il suffit d'instancier le modèle pour construire des *CPs* réalisables. En procédant de cette manière, nous évitons de rejouer la preuve et on offre une approche évolutive pour la construction de *CPs* réalisables. De plus, il n'est pas nécessaire de calculer les compositions synchrones et asynchrones des entités projetées. Les théorèmes fournis de la section 2 définissent des conditions qui ne sont vérifiées qu'une fois sur les *CPs* construits. De plus, ils sont vérifiés sur les constructions à l'aide de propriétés syntaxiques dont les définitions impliquent une évolutivité due à la nature structurelle.

Le chapitre 9 présente un ensemble de points de repère qui permettent de valider notre proposition et de déterminer comment l'approche évolue en fonction d'un nombre arbitraire d'états et de transitions.

4 Instanciation du modèle B-événementiel par raffinement

Dans la section précédente, nous avons présenté un modèle générique de composition de système correspondant au modèle décrit à la Figure 6.2. Ce modèle est divisé en deux parties : une concerne la modélisation du système, états, variables, variantes et invariants ; et une deuxième modélise le comportement des systèmes et du mécanisme de composition réalisable. L'instanciation consiste à configurer le modèle générique obtenu pour des systèmes spécifiques. Il est obtenu après deux étapes, décrites ci-dessous, correspondant à l'instanciation de chaque partie de modélisation.

4.1 Étape1. L'instanciation du contexte

Premièrement, les valeurs spécifiques des ensembles abstraits définis dans le contexte *Lts_Contexte* présenté à la section 3.1 sont introduites. Un contexte d'instanciation *Lts_Instance*, qui étend le contexte *Lts_Contexte*, est défini avec des

valeurs concrètes pour tous les ensembles (PEERS, MESSAGES, CP_STATES et CPs_B) et pour les constantes formant ces mêmes ensembles.

Un deuxième contexte d'instanciation *Lts_Sync_Instance* raffinant l'instance abstraite est donné, tel que, nous définissons avec des valeurs concrètes l'ensemble CPs_SYNC_B du contexte synchrone *Lts_Sync_Contexte* qui raffine le contexte abstrait *Lts_Contexte*.

Un dernier contexte d'instanciation *Lts_Async_Instance* raffinant l'instance concrète synchrone est donné, où nous définissons avec des valeurs concrètes les ensembles (A_STATES, A_TRACES, S_Next_States, A_Next_States) relatives à la communication asynchrone (*Lts_Async_Contexte*).

4.2 Étape2. Raffinement et témoins pour l'instanciation

Pour utiliser les valeurs concrètes définies dans les trois contextes *Lts_Instance*, *Lts_Sync_Instance* et *Lts_Async_Instance*, les machines *Lts_Model*, *Lts_Sync_Model* et *Lts_Async_Model* modélisant notre système doivent voir (SEES) les trois contextes d'instanciation respectivement.

Notant que, la machine abstraite contient toutes les spécificités du système. Le comportement du système, précédemment modélisé de manière générique par la progression des événements, est maintenant détaillé par les événements *Add_Sequence*, *Add_Choice* et *Add_Loop* correspondant aux opérateurs de composition. La stratégie de raffinement est ajouté juste pour complété la preuve de réalisabilité des systèmes et valider la correction de notre approche de composition.

5 Application à une étude de cas

Instanciation et validation des axiomes. Pour illustrer notre approche, nous avons instancié notre modèle sur l'exemple concret correspondant à un processus d'approvisionnement simple au sein d'une entreprise virtuelle. Le *CP* décrivant ce processus est présenté sur la Figure 8.1. Les étiquettes des transitions de la forme $m^{p \longrightarrow p'}$ désignent un message m envoyé par l'entité p à l'entité p'.

Les contextes des Listings 6.14, 6.15 et 6.16 montrent l'instanciation du modèle générique pour le *CP* de la Figure 8.1. Il montre également que les axiomes définis dans le modèle sont habités. Le vérificateur de modèle ProB [70] associé à B-événementiel sur la plate-forme Rodin a été utilisé pour la validation automatique.

D'autres études de cas prisent du monde de la recherche sur la réalisabilité ont été utilisées pour instancier notre modèle. Ces études de cas utilisent les opérateurs de composition que nous avons définis.

Le contexte d'instanciation *Lts_Instance* du Listing 6.14 fournit des valeurs concrètes pour les ensembles différés du contexte *Lts_Contexte*. Tous les ensembles correspondant à la caractérisation statique des systèmes tels que les différents ensembles sont valorisés par l'ensemble d'instances possibles. Ils caractérisent des systèmes spécifiques correspondant à l'étude de cas. Tel que :

— $PEERS$ représente l'ensemble des entités communicantes formant le CPs
— $MESSAGES$ est l'ensemble des messages échangés entre entités.
— CP_STATES énumère les états globaux du CP.
— CPs_B est l'ensemble des transitions basiques, l'élément de base d'un CP.

Le deuxième contexte d'instanciation $Lts\_Sync\_Instance$ du Listing 6.15 fournit des valeurs concrètes pour l'ensemble des transitions synchrones basiques relatif au contexte $Lts\_Sync\_Contexte$, tel que :

— CPs_Sync_B est l'ensemble des transitions synchrones basiques, l'élément de base d'un Sys_{sync}.

Context $LTS\_CONTEXT\_Instance$ **Extends** $LTS\_CONTEXT$
Constants $A, B, L, OrderOp, deliverOp, deliver\_confOp,$
$deliverOp, deliveryOp, deliver\_confOp, terminateOp, terminateLOp, get\_statusOp,$
$get\_statusLOp, statusOp, s0, s1, s2, s3, s4, s5, s6, s7, \ldots, s11$

Axioms
$axm1 : partition(PEERS, \{A\}, \{B\}, \{L\}, \{Pend\})$
$axm2 : partition(MESSAGES, \{OrderOp\}, \{deliverOp\}, \{deliver\_confOp\}, \{deliveryOp\}, \{terminateOp\},$
$\{terminateLOp\}, \{get\_statusOp\}, \{get\_statusLOp\}, \{statusOp\}, \{End\})$
$axm3 : partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\}, \{s6\}, \{s7\}, \{s8\}, \{s9\}, \{s10\}, \{s11\})$
$axm4 : CPs\_B = \{s0 \mapsto B \mapsto OrderOp \mapsto A \mapsto s1 \mapsto 1,$
$s1 \mapsto A \mapsto deliverOp \mapsto L \mapsto s2 \mapsto 2,$
$s2 \mapsto L \mapsto deliver\_confOp \mapsto A \mapsto s3 \mapsto 3,$
$s3 \mapsto A \mapsto deliveryOp \mapsto B \mapsto s4 \mapsto 4,$
$s4 \mapsto B \mapsto terminateOp \mapsto A \mapsto s6 \mapsto 5,$
$s6 \mapsto A \mapsto terminateLOp \mapsto L \mapsto s8 \mapsto 6,$
$s4 \mapsto B \mapsto get\_statusOp \mapsto A \mapsto s5 \mapsto 7,$
$s5 \mapsto A \mapsto get\_statusOp \mapsto L \mapsto s7 \mapsto 8,$
$s7 \mapsto L \mapsto get\_statusLOp \mapsto A \mapsto s9 \mapsto 9,$
$s9 \mapsto A \mapsto statusOp \mapsto B \mapsto s4 \mapsto 10,$
$s8 \mapsto Pend \mapsto End \mapsto Pend \mapsto s10 \mapsto 11$
$\}$
End

Listing 6.14 – Un extrait de LTS_Instance.

Context $LTS\_SYNC\_CONTEXT\_Inst\_valide$ **Extends** $LTS\_SYNC\_CONTEXT$
Constants
Axioms

$axm1 : CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto OrderOp \mapsto A \mapsto B \mapsto Receive \mapsto OrderOp \mapsto s1 \mapsto 1,$
$s1 \mapsto Send \mapsto deliverOp \mapsto L \mapsto A \mapsto Receive \mapsto deliverOp \mapsto s2 \mapsto 2,$
$s2 \mapsto Send \mapsto deliver\_confOp \mapsto A \mapsto L \mapsto Receive \mapsto deliver\_confOp \mapsto s3 \mapsto 3,$
$s3 \mapsto Send \mapsto deliveryOp \mapsto B \mapsto A \mapsto Receive \mapsto deliveryOp \mapsto s4 \mapsto 4,$
$s4 \mapsto Send \mapsto terminateOp \mapsto A \mapsto B \mapsto Receive \mapsto terminateOp \mapsto s6 \mapsto 5,$
$s6 \mapsto Send \mapsto terminateLOp \mapsto L \mapsto A \mapsto Receive \mapsto terminateLOp \mapsto s8 \mapsto 6,$
$s4 \mapsto Send \mapsto get\_statusOp \mapsto A \mapsto B \mapsto Receive \mapsto get\_statusOp \mapsto s5 \mapsto 7,$
$s5 \mapsto Send \mapsto get\_statusOp \mapsto L \mapsto A \mapsto Receive \mapsto get\_statusOp \mapsto s7 \mapsto 8,$
$s7 \mapsto Send \mapsto get\_statusLOp \mapsto A \mapsto L \mapsto Receive \mapsto get\_statusLOp \mapsto s9 \mapsto 9,$
$s9 \mapsto Send \mapsto statusOp \mapsto B \mapsto A \mapsto Receive \mapsto statusOp \mapsto s4 \mapsto 10,$
$s8 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s10 \mapsto 11$
$\}$
End

Listing 6.15 – Un extrait de Lts_Sync_Instance.

Le dernier contexte d'instanciation $Lts\_Async\_Instance$ du Listing 6.16 fournit des valeurs concrètes pour les ensembles différés du contexte $Lts\_Async\_Contexte$. Tous les ensembles correspondant à la caractérisation statique du système asynchrone, tels que les différents ensembles sont valorisés par l'ensemble d'instances possibles. Ils caractérisent des systèmes de communication asynchrone spécifiques avec une file d'attente globale. Tel que :

— A_STATES est l'ensemble des états formant les entités projetées à partir du CP.
— CPs_ASYNC_B est l'ensemble des transitions asynchrone basiques, l'élément de base d'un Sys_{async}.
— R_TRACE_B est l'ensemble instanciant les traces asynchrones réduite du Sys_{async}.
— A_TRACES est l'ensemble instanciant les traces asynchrones complète du Sys_{async}.
— S_Next_States identifie les états synchrones globaux du système Sys_{sync}.
— A_Next_States énumère les états asynchrones globaux du système Sys_{async}.

Context $LTS\_Async\_Instance$ **Extends** $Lts\_Sync\_Context$
Constants $s0\_A, s1\_A, s2\_A, s3\_A, s4\_A, s5\_A, s6\_A, s7\_A, s8\_A, s9\_A, s0\_B, s1\_B, s2\_B, s3\_B, s4\_B, s0\_L, s1\_L, s2\_L, s3\_L, s4\_L, s5\_L, \ldots$

Axioms

$axm1 : partition(A\_STATES, \{s0\_A\}, \{s1\_A\}, \{s2\_A\}, \{s3\_A\}, \{s4\_A\}, \{s5\_A\}, \{s6\_A\}, \{s7\_A\}, \{s8\_A\}, \{s0\_B\}, \{s1\_B\}, \{s2\_B\}, \{s3\_B\}, \{s4\_B\}, \{s0\_L\}, \{s1\_L\}, \{s2\_L\}, \{s3\_L\}, \{s4\_L\}, \{s5\_L\})$

$axm2 : CPs\_ASYNC\_B = \{((s0\_B \mapsto (Send \mapsto OrderOp \mapsto A) \mapsto 1) \mapsto s1\_B),$
$((s0\_A \mapsto (Receive \mapsto OrderOp \mapsto B) \mapsto 1) \mapsto s1\_A),$
$((s1\_A \mapsto (Send \mapsto deliverOp \mapsto L) \mapsto 2) \mapsto s2\_A),$
$((s0\_L \mapsto (Receive \mapsto deliverOp \mapsto A) \mapsto 2) \mapsto s1\_L),$
$\ldots$
$\}$

$axm3 : R\_TRACE\_B = \{s0 \mapsto B \mapsto OrderOp \mapsto A \mapsto s1 \mapsto 1,$
$s1 \mapsto A \mapsto deliverOp \mapsto L \mapsto s2 \mapsto 2,$
$s2 \mapsto L \mapsto deliver\_confOp \mapsto A \mapsto s3 \mapsto 3,$
$s3 \mapsto A \mapsto deliveryOp \mapsto B \mapsto s4 \mapsto 4,$
$s4 \mapsto B \mapsto terminateOp \mapsto A \mapsto s6 \mapsto 5,$
$s6 \mapsto A \mapsto terminateLOp \mapsto L \mapsto s8 \mapsto 6,$
$s4 \mapsto B \mapsto get\_statusOp \mapsto A \mapsto s5 \mapsto 7,$
$\ldots\}$

$axm4 : A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto B \mapsto OrderOp \mapsto A \mapsto s \mapsto 1 \mapsto 1,$
$s \mapsto 1 \mapsto Receive \mapsto B \mapsto OrderOp \mapsto A \mapsto s \mapsto 2 \mapsto 2,$
$s \mapsto 2 \mapsto Send \mapsto A \mapsto deliverOp \mapsto L \mapsto s \mapsto 3 \mapsto 3,$
$s \mapsto 3 \mapsto Receive \mapsto A \mapsto deliverOp \mapsto L \mapsto s \mapsto 4 \mapsto 4,$
$s \mapsto 4 \mapsto Send \mapsto L \mapsto deliver\_confOp \mapsto A \mapsto s \mapsto 5 \mapsto 5,$
$s \mapsto 5 \mapsto Receive \mapsto L \mapsto deliver\_confOp \mapsto A \mapsto s \mapsto 6 \mapsto 6,$
$s \mapsto 6 \mapsto Send \mapsto A \mapsto deliveryOp \mapsto B \mapsto s \mapsto 7 \mapsto 7,$
$s \mapsto 7 \mapsto Receive \mapsto A \mapsto deliveryOp \mapsto B \mapsto s \mapsto 8 \mapsto 8,$
$s \mapsto 8 \mapsto Send \mapsto B \mapsto terminateOp \mapsto A \mapsto s \mapsto 9 \mapsto 9,$
$s \mapsto 9 \mapsto Receive \mapsto B \mapsto terminateOp \mapsto A \mapsto s \mapsto 10 \mapsto 10,$
$s \mapsto 10 \mapsto Send \mapsto A \mapsto terminateLOp \mapsto L \mapsto s \mapsto 11 \mapsto 11,$
$\ldots\}$

$axm6 : S\_Next\_States = \{$
$\{((s0\_B \mapsto (Send \mapsto OrderOp \mapsto A) \mapsto 1) \mapsto s1\_B)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s0\_B), (L \mapsto s0\_L)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s0\_A \mapsto (Receive \mapsto OrderOp \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto$
$\{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s1\_A \mapsto (Send \mapsto deliverOp \mapsto L) \mapsto 2) \mapsto s2\_A)\} \mapsto$
$\{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto$
$\{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s0\_L \mapsto (Receive \mapsto deliverOp \mapsto A) \mapsto 2) \mapsto s1\_L)\} \mapsto$
$\{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto$
$\{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s1\_L)\},$
$\ldots\}$

$axm7 : A\_Next\_States = \{$
$\{((s0\_B \mapsto (Send \mapsto OrderOp \mapsto A) \mapsto 1) \mapsto s1\_B)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s0\_B), (L \mapsto s0\_L)\} \mapsto \varnothing \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s0\_A \mapsto (Receive \mapsto OrderOp \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \{A \mapsto OrderOp \mapsto 0\} \mapsto$
$\{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s0\_A \mapsto (Receive \mapsto OrderOp \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto$

$\{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \varnothing \mapsto$
$\{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s1\_A \mapsto (Send \mapsto deliverOp \mapsto L) \mapsto 2) \mapsto s2\_A)\} \mapsto$
$\{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \varnothing \mapsto$
$\{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s0\_L \mapsto (Receive \mapsto deliverOp \mapsto A) \mapsto 2) \mapsto s1\_L)\} \mapsto$
$\{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \{L \mapsto deliverOp \mapsto 1\} \mapsto$
$\ldots\}$

End

Listing 6.16 – Un extrait de Lts_ Async_ Instance

6 Évaluation

Le principal avantage de notre proposition réside dans le fait que la preuve de correction de la stratégie de composition n'est effectuée qu'une fois. Cependant, cette preuve ainsi que la preuve de raffinement sont plus complexes car elles sont génériques.

6.1 Statistiques de preuves

La Table 6.1 présente les statistiques de preuve pour l'ensemble des développements B-événementiel. Notant que beaucoup d'efforts sont consacrés à la preuve interactive des modèles de composition. Toutes les obligations de preuve associées au développement formel B-événementiel présenté ici ont été prouvées, soit avec les prouveurs automatiques associés à la plate-forme Rodin, soit à l'aide des prouveurs interactives gérées par le développeur sur la plate-forme Rodin.

Le point clé lié à l'évolutivité concerne l'instanciation de systèmes spécifiques. En effet, le développement présenté ci-dessus est un développement générique, défini au niveau méta, où les obligations de preuve associées à l'exactitude de la composition de système obtenue agissent comme des méta-théorèmes.

L'utilisation des compositions généralisées (constructions Any) montre que le développement prend en compte tout système de transition décrit par un modèle *CP* ainsi que les invariants associés exprimés dans les modèles B-événementiel correspondants.

Cela semble très intéressant et prometteur car cela signifie que le schéma du mécanisme de composition n'est prouvé qu'une fois. Cependant, la preuve est plus difficile que le système concret seul. Par conséquent, le choix dépend de la possibilité de réutiliser un modèle de composition particulier dans plusieurs projets de développement.

Notons que des techniques de vérification de modèle peuvent être appliquées pour vérifier automatiquement l'exactitude de l'instanciation. L'exploration de tous les états possibles est possible car les ensembles sont définis avec un nombre fini de valeurs dans les contextes d'instanciation.

Cependant, ces techniques sont confrontées au problème de l'explosion d'état. Par exemple, la difficulté des preuves dans notre approche n'est pas affectée par la taille de système ni de la complexité ses échanges , alors qu'une méthode qui devrait énumérer explicitement toutes les valeurs possibles des composants du système serait sévèrement limitée par le nombre considérable de possibilités offertes par la

Modèle B-événementiel	Preuves interactives	Preuves automatiques	Obligations de Preuve
Contexte abstrait	06 (100%)	0 (0%)	06 (100%)
Contexte synchrone	02 (100%)	0 (0%)	02 (100%)
Contexte asynchrones	01 (33,33%)	02 (66,67%)	03 (100%)
Modèle abstrait	28 (58,33%)	20 (41,67%)	48 (100%)
Modèle synchrone	39 (39%)	61 (61%)	100 (100%)
Modèle asynchrone	73 (38,83%)	115 (61,17%)	188 (100%)
Totale	148 (100%)	198 (100%)	347 (100%)

TABLE *6.1 – Statistiques de preuves RODIN (modèle de composition)*

combinatoire. Les tailles des différentes épreuves pour les différentes machines et contextes sont disponibles à la Figure 6.3.

6.2 Méthodes formelles correctes-par-construction

L'approche proposée est générique. Les contextes abstrait, synchrone et asynchrone décrivent explicitement les concepts des systèmes manipulés (entités communicantes, messages, états du *CP*, *CP*, Sys_{sync}, Sys_{async}, traces asynchrone complètes, propriétés suffisantes, etc.). Ces concepts sont manipulés comme des objets de premier ordre dans les machines abstraite, synchrone et asynchrone afin de coder le modèle de comportement décrit avec les événements, initialisation, compositions séquentiel, en branche et en cycle. Notons que, le calcule des entités projetés a partir du *CP* ainsi que les traces de la recomposition synchrone et asynchrone du systèmes ne sont pas explicitement manipulées par le mécanisme de composition que nous avons introduit. Cela réduit considérablement la complexité du modèle générique car il repose sur les capacités de raffinement de B-événementiel pour gérer la modélisation du comportement de base du système et prouver sa réalisation. La méthode B-événementiel fournit une puissante technique de preuve inductive intégrée basée sur la préservation de l'invariante par les événements. Cela nous permet de scinder

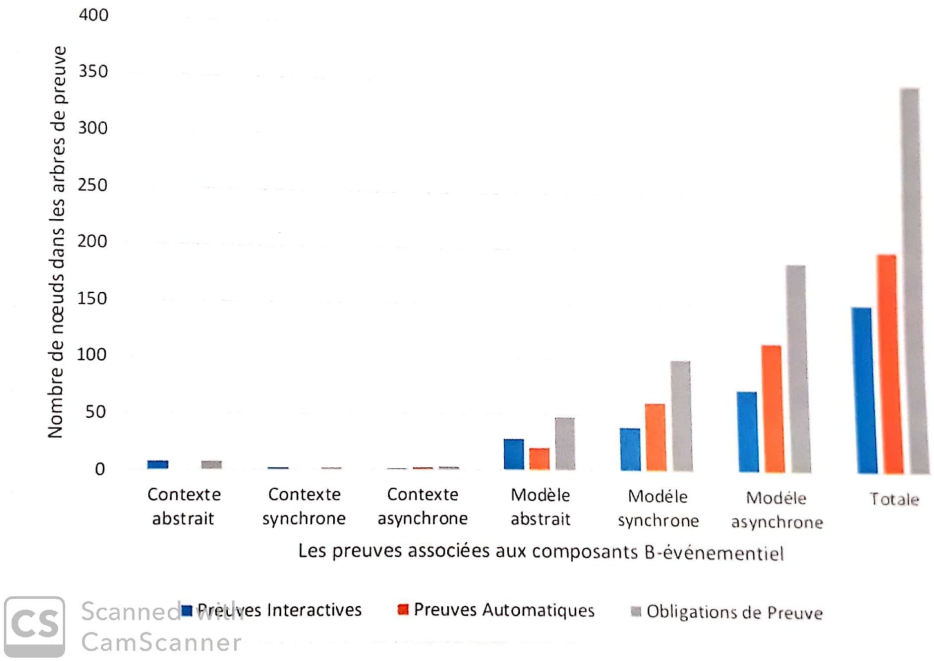


FIGURE *6.3 – Les preuves associées aux modèle de composition B-événementiel*

la preuve globale en preuves plus petites et plus faciles à gérer. Par conséquent, nous nous basons sur la définition des événements B-événementiel pour définir le *CP* générique. Les preuves de la préservation des invariantes et de la diminution de la variante sont obtenues au niveau abstrait de la machine *modèle abstrait*. Ils sont conservés par toute autre machine qui le raffine.

Un événement raffinant l'événement abstrait est introduit pour chaque événement concret. Le seul effort de preuve concerne le raffinement correct de l'événement.

Notons que dans d'autres techniques traditionnelles de correction par construction telles que Coq [9] ou Isabelle [76], des schémas classiques de preuve inductive sont proposés. Nous avons :

— d'abord pour décrire la structure inductive associée aux systèmes formalisés,
— puis de donner un schéma de preuve inductif spécifique pour cette structure inductive définie et,
— enfin pour prouver la bonne instanciation.

Dans la définition principale de ces techniques, le processus inductif associé a la composition réalisable des systèmes correspondant et la capacité de raffinement ne sont pas disponibles en tant que processus de preuve inductif intégré (comme dans B-événementiel où cette notion est disponible via les variables d'état et les événements). Le développeur devrait formaliser la notion de composition des *CPs* avec les principes de preuve inductive correspondants et l'instanciation des *CPs*, car aucun raffinement d'événement n'est disponible.

Par rapport à la méthode B-événementiel, il existe un besoin d'une autre spécification de niveau méta et d'un processus de preuve.

Chapitre 7

Réparation correcte-par-construction

Sommaire

L'analyse chorégraphique est un problème crucial dans le développement de systèmes simultanés et distribués. Une chorégraphie spécifie l'ordre souhaité pour l'échange de messages entre les composants d'un système. La réalisabilité d'une chorégraphie revient à déterminer l'existence de composants dont le comportement de communication est conforme à la chorégraphie donnée. Récemment, le problème de réalisabilité de la chorégraphie s'est révélé être décisif. Dans ce chapitre, nous étudions la possibilité de réparation de chorégraphies non réalisables, l'objectif étant d'identifier un ensemble de modifications d'une chorégraphie non réalisable donnée pour la rendre réalisable. Nous présentons une stratégie permettant de réparer automatiquement les chorégraphies non réalisables en se basant sur les propriétés nécessaires présentées précédemment et fournissons des garanties formelles d'exactitude et de validation de la proposition.

1 Opérateurs de réparation : théorèmes et preuves

Cette section présente l'ensemble des opérateurs de restauration des propriétés suffisantes proposées pour préserver la réalisabilité des *CP*s à savoir, la propriété d'absence des séquences dépendantes ISeqF et de choix parallèle PCF. Pour ce faire nous avons identifiés deux opérateurs pour réparer un *CP* identifier comme non

réalisable, par composition incrémentale de plusieurs CP_bs et $CP_{b_{Synch}}$s [1]. Ces opérateurs désignent : une réparation séquentielle dénoté par $\otimes_{(\text{ISeqF}_{Restore}, s^f_{CP})}$ et une réparation du choix parallèle dénoté par $\otimes_{(\text{PCF}_{Restore}, s^f_{CP})}$. Chacun des opérateurs est caractérisé par deux paramètres, le premier est un *CP réalisation* avec un état final $s^f_{CP} \in S^f_{CP}$ et le second est un ensemble de protocole de conversation de base former de CP_b et $CP_{b_{Synch}}$ (ou d'un ensemble de CP_bs et $CP_{b_{Synch}}$s). Chaque expression de la forme $\otimes_{(op, s^f_{CP})}(CP, \{CP_{b_{Synch}}, CP_b\})$ signifie que l'état initial de CP_b est fusionné avec l'état s^f_{CP}. En d'autres termes, CP_b est ajouté (collé) à CP à l'état s^f_{CP}.

Définition 16 *ISeqF restauration* $\otimes_{(ISeqF_{Restore}, s^f_{CP})}$.

Soit

- $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *est un protocole de conversation avec un état final* $s^f_{CP} \in S_{CP}$ *et*
- $CP_{b_{Synch}}$ *est un protocole de conversation synchronisation basique avec* $T_{CP_{b_{Synch}}} = \{s_{CP_{b_{Synch}}} \xrightarrow{l_{CP_{b_{Synch}}}} s'_{CP_{b_{Synch}}}\}$ *tel que* $\{s_{CP_{b_{Synch}}}, s'_{CP_{b_{Synch}}}\} \not\subset S_{CP}$.
- CP_b *est un protocole de conversation de base avec* $T_{CP_b} = \{s_{CP_b} \xrightarrow{l_{CP_b}} s'_{CP_b}\}$ *tel que* $\{s_{CP_b}, s'_{CP_b}\} \not\subset S_{CP}$.

Alors la restauration de la propriété ISeqF $\otimes_{(ISeqF_{Restore}, s^f_{CP})}(CP, \{CP_{b_{Synch}}, CP_b\})$ *est défini comme suit :*

- $S_{CP_{ISeqF_{Restore}}} = S_{CP} \cup \{s'_{CP_{b_{Synch}}}, s'_{CP_b} |$
  $s_{CP_{b_{Synch}}} \xrightarrow{l_{CP_{b_{Synch}}}} s'_{CP_{b_{Synch}}} \in T_{CP_{b_{Synch}}}, s_{CP_b} \xrightarrow{l_{CP_b}} s'_{CP_b} \in T_{CP_b}\}$
- $L_{CP_{ISeqF_{Restore}}} = L_{CP_{b_{Synch}}} \cup \{l_{CP_{b_{Synch}}}, l_{CP_b}\}$
- $T_{CP_{ISeqF_{Restore}}} = T_{CP} \cup \{s^f_{CP} \xrightarrow{l_{CP_{b_{Synch}}}} s'_{CP_{b_{Synch}}}, s^f_{CP} \xrightarrow{l_{CP_b}} s'_{CP_b}\}$
- $S^f_{CP_{ISeqF_{Restore}}} = (S^f_{CP} \setminus \{s^f_{CP}\}) \cup \{s'_{CP_b}\}$

Tel que, la restauration de la propriété ISeqF autrement dit, réparation séquentielle du protocole de conversation $CP_{\text{ISeqF}_{Restore}} = \otimes_{(\text{ISeqF}_{Restore}, s^f_{CP})}(CP, \{CP_{b_{Synch}}, CP_b\})$ ajoute une transition de synchronisation $CP_{b_{Synch}}$ entre CP et CP_b, et mis à jour l'ensemble des états $S_{CP_{\text{ISeqF}_{Restore}}}$, l'ensemble des messages échangés $L_{CP_{\text{ISeqF}_{Restore}}}$, l'ensemble des transitions $T_{CP_{\text{ISeqF}_{Restore}}}$ et l'ensemble des états finaux $S^f_{CP_{\text{ISeqF}_{Restore}}}$.

Définition 17 *PCF restauration* $\otimes_{(PCF_{Restore}, s^f_{CP})}$.

Soit

- $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *est un protocole de conversation avec un état final* $s^f_{CP} \in S_{CP}$ *et*
- $\{CP_{bi} \mid 1 \leq i \leq n, n \geq 2\}$ *est un ensemble fini de protocoles de conversation basique avec* $T_{CP_{bi}} = \{s_{CP_{bi}} \xrightarrow{l_{CP_{bi}}} s'_{CP_{bi}}\}$, $\{s_{CP_{bi}}, s'_{CP_{bi}}\} \not\subset S_{CP}$, *et les états finaux de* CP_{bi}, $1 \leq i \leq n, n \geq 2$.

1. $CP_{b_{Synch}}$s sont des transitions basiques de synchronisation, formant l'ensemble des scénario de réparation possibles

— $\{CP_{b_{Synch\,i}} \mid 1 \leq i \leq n, n \geq 2\}$ *est un ensemble de protocole de conversation basique de synchronisation avec* $T_{CP_{b_{Synch\,i}}} = \{s_{CP_{b_{Synch\,i}}} \xrightarrow{l_{CP_{b_{Synch\,i}}}} s'_{CP_{b_{Synch\,i}}}\}$ *tel que* $\{s_{CP_{b_{Synch}}}, s'_{CP_{b_{Synch}}}\} \not\subset S_{CP}$ *et* $\{\exists\ CP_{b_{Synch}}, CP_b \mid CP_{b_{Synch}} \in \{CP_{b_{Synch\,i}}\} \wedge CP_b \in \{CP_{bi}\}$ *tel que* $s_{CP_{b_{Synch}}} = s_{CP_b}$ *et* $s'_{CP_{b_{Synch}}} = s_{CP_b}$.

La restauration de la propriété PCF

$CP_{PCF_{Restore}} = \otimes_{(PCF_{Restore}, s^f_{CP})}(CP, \{CP_{b_{Synch}b}, CP_{bi}\})$ *est défini comme suit :*

— $S_{CP_{PCF_{Restore}}} = S_{CP} \cup \{s'_{CP_{b_{Synch}1}}, \ldots, s'_{CP_{b_{Synch}n}}, s'_{CP_{b1}}, \ldots, s'_{CP_{bn}} |$ $s_{CP_{b_{Synch\,i}}} \xrightarrow{l_{CP_{b_{Synch\,i}}}} s'_{CP_{b_{Synch\,i}}} \in T_{CP_{b_{Synch\,i}}}, s_{CP_{b_{Synch\,i}}} \xrightarrow{l_{CP_{b_{Synch\,i}}}} s'_{CP_{b_{Synch\,i}}} \in T_{CP_{b_{Synch\,i}}}\}$

— $L_{CP_{PCF_{Restore}}} = L_{CP} \cup \{l_{CP_{b_{Synch}1}}, \ldots, l_{CP_{b_{Synch}n}}, l_{CP_{b1}}, \ldots, l_{CP_{bn}}\}$

— $T_{CP_{PCF_{Restore}}} = T_{CP} \cup \{s^f_{CP_{b_{Synch}}} \xrightarrow{l_{CP_{b_{Synch}b1}}} s'_{CP_{b_{Synch}b1}}, \ldots, s^f_{CP_{b_{Synch}}} \xrightarrow{l_{CP_{b_{Synch}bn}}} s'_{CP_{b_{Synch}bn}} s^f_{CP} \xrightarrow{l_{CP_{b1}}} s'_{CP_{b1}}, \ldots, s^f_{CP} \xrightarrow{l_{CP_{bn}}} s'_{CP_{bn}}\}$

— $S^f_{CP_{PCF_{Restore}}} = (S^f_{CP} \setminus \{s^f_{CP}\}) \cup \{s'_{CP_{b1}}, \ldots, s'_{CP_{bn}}\}$

La composition en choix $CP_+ = \otimes_{(+, s^f_{CP})}(CP, \{CP_{bi}\})$ ajoute un ensemble de protocoles de conversation de base à l'état final s^f_{CP} ce qui forme un branchement (un choix) et mis à jour l'ensemble des états S_{CP_+}, l'ensemble des messages échangés L_{CP_+}, l'ensemble des transitions T_{CP_+} et l'ensemble des états finaux $S^f_{CP_+}$.

La table 7.1 introduit les théorèmes assurant la réalisabilité par la réparation incrémentale d'un *CP* caractérisé comme non réalisable, en utilisant l'ensemble de nos opérateurs de réparation. Un théorème est associé à chacun de ces opérateurs. Ces théorèmes reposent sur les conditions définies dans les chapitres précédents. La preuve de chaque théorème est également donnée.

Théorème 5	$CP \in R \wedge CP_b \in R \wedge CP_{b_{Synch}} \in R \wedge CP_{\gg} = \otimes_{(\gg, s^f_{CP})}(CP, CP_b) \notin \text{ISeqF}$ $\Rightarrow CP_{\text{ISeqF}_{Repair}} = \otimes_{(\text{ISeqF}_{Repair}, s^f_{CP})}(CP, \{CP_{b_{Synch}}, CP_b\}) \in \text{ISeqF}$ $\wedge CP_{\text{ISeqF}_{Repair}} \in R$
Théorème 6	$CP \in R \wedge \{CP_{b_{Synch\,i}}\} \subseteq R \wedge \{CP_{bi}\} \subseteq R$ $\wedge CP_+ = \otimes_{(+, s^f_{CP})}(CP, \{CP_{bi}\}) \notin \text{PCF}$ $\Rightarrow CP_{\text{PCF}_{Repair}} = \otimes_{(\text{PCF}_{Repair}, s^f_{CP})}(CP, \{CP_{b_{Synch\,i}}\}, \{CP_{bi}\}) \in \text{PCF}$ $\wedge\ CP_{\text{PCF}_{Repair}} \in R$

TABLE *7.1 – Théorèmes liés au CP réparé réalisables-par-construction*

Theorem 5 (Restauration de ISeqF).

Soit $CP = <S_{CP}, s^0_{CP}, L_{CP}, T_{CP}>$ *,le protocole de synchronisation basique* $CP_{b_{Synch}}$ *et* CP_b *et* $s^f_{CP} \in S^f_{CP}$ *qui est un état final dans CP, alors les formules suivantes*

sont satisfaites.

Si $CP \in R$ *et* $CP_{\gg} = \otimes_{(\gg, s^f_{CP})}(CP, CP_b) \notin ISeqF$ *alors*

$CP_{ISeqF_{Repair}} = \otimes_{(ISeqF_{Repair}, s^f_{CP})}(CP, \{CP_{b_{Synch}}, CP_b\}) \in ISeqF \wedge$
$CP_{ISeqF_{Repair}} \in R.$

Preuve 5. Voir Preuve 2, section 3, chapitre 5

Theorem 6 (Réparation du branchement) *Soit* $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$, $\{CP_{b_{Synch}i} \mid 2 \leq i \leq n\}$, $\{CP_{bi} \mid 2 \leq i \leq n\}$ *est l'ensemble de* n $(n \geq 2)$ *protocoles de conversation basique et l'ensemble des protocoles de synchronisation* $\{CP_{b_{Synch}i} \mid 2 \leq i \leq n\}$ *tel que* $s^f_{CP} \in S^f_{CP}$ *un état final dans* CP*, donc les formules suivantes sont valides.*

Si $CP \in R$, $CP_+ = \otimes_{(+, s^f_{CP})}(CP, \{CP_{b1}, \ldots, CP_{bn}\}) \notin PCF$ *alors*
$CP_{PCFRepair} = \otimes_{(PCF_{Repair}, s^f_{CP})}(CP, \{CP_{b_{Synch}1}, \ldots, CP_{b_{Synch}n}\}, \{CP_{b1}, \ldots, CP_{bn}\}) \in PCF$ *et*
$CP_{PCFRepair} = \otimes_{(PCF_{Repair}, s^f_{CP})}(CP, \{CP_{b_{Synch}i}\}, \{CP_{bi}\}) \in R.$

Preuve 6. Voir Preuve 3, section 3, chapitre 5

2 Étude de cas

Selon Basu et al. [13], l'étude de cas suivante décrit la chorégraphie illustrée à la Figure 8.2, il s'agit d'un protocole simple de transfert de fichier où P_1 est un client demandant le transfert de fichier, P_2 est un serveur de fichiers et P_3 initialise la communication entre le client et le serveur. Ce CP est décrit dans la Figure 8.2. Tout d'abord, le client envoie un message $(init)$ au serveur pour lui demander de démarrer le transfert (ms). Lorsque le transfert est terminé, le serveur envoie le message "Transfert terminé" (mf) et le protocole se termine. Toutefois, le client peut décider d'annuler le transfert avant de recevoir une réponse du serveur en envoyant un message "Annulation terminé" (mc), le cas auquel le serveur répond par le message "Transfert terminé" (mf), qui, à nouveau , termine le protocole.

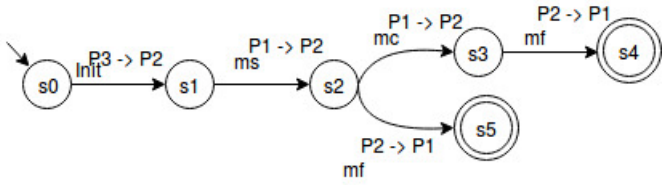


FIGURE *7.1 – Chorégraphie de protocole de transfert de fichier simple*

Lorsqu'il est projeté, le CP de Figure 8.2 produit les trois entités communicantes de la Figure 7.2.

La spécification de la chorégraphie de la Figure 8.2 n'est pas réalisable. Nous illustrons ceci en définissant une séquence d'applications d'opérateurs. Nous obtenons les ensembles suivantes.

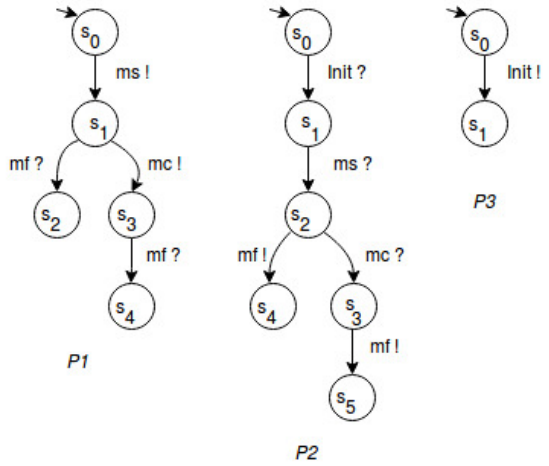


FIGURE 7.2 – *Entités projetées d'une chorégraphie de protocole simple de transfert de fichiers*

— $CP = \varnothing$

Transitions basiques.

$CP_{b0} = s_0 \xrightarrow{Init^{P3\to P2}} s_1,$
$CP_{b1} = s_1 \xrightarrow{ms^{P1\to P2}} s_2,$
$CP_{b2} = s_2 \xrightarrow{mc^{P1\to P2}} s_3,$
$CP_{b3} = s_3 \xrightarrow{mf^{P2\to P1}} s_4,$
$CP_{b4} = s_2 \xrightarrow{mf^{P2\to P1}} s_5.$

Composition. Les deux premières applications d'opérateurs suffisent à détecter les violations des conditions suffisantes pour les opérateurs de séquence et de choix.

— $CP_0 = \otimes_{(\gg, s^0_{CP})}(CP, CP_{b0})\checkmark$ $CP_1 \in$ ISeqF.
— $CP_1 = \otimes_{(\gg, s^1_{CP})}(CP_0, CP_{b1})\times,$ $CP_2 \notin$ ISeqF,
— $CP_2 = \otimes_{(+, s^2_{CP})}(CP_1, \{CP_{b2}, CP_{b3}\})\times,$ $CP_3 \notin$ PCF

Nous observons que CP_1 et CP_2 correspondent respectivement à l'application d'opérateur de séquence et de choix ne satisfont pas les conditions suffisantes correspondantes.

Le Listing 7.1 montre le contexte B-événementiel défini pour instancier le modèle générique de cette étude de cas.

```
Context
Constants
Axioms
 axm1 :partition(PEERS, {P1}, {P2}, {P3}, {Pend})
 axm2 :partition(MESSAGES, {Init}, {ms}, {mc}, {mf}, {End})
 axm3 :partition(CP_STATES, {s0}, {s1}, {s2}, {s3}, {s4}, {s5})
 axm4 : partition(A_STATES, {s0_P1}, {s1_P1}, ...)
 axm5 : CPs_B = {s0 ↦ P3 ↦ Init ↦ P2 ↦ s1 ↦ 1,
                 s1 ↦ P1 ↦ ms ↦ P2 ↦ s2 ↦ 2,
                 s2 ↦ P1 ↦ mc ↦ P2 ↦ s3 ↦ 3,
                 s1 ↦ P2 ↦ mf ↦ P1 ↦ s3 ↦ 3,
                 s2 ↦ P2 ↦ mf ↦ P1 ↦ s4 ↦ 4,
                 s4 ↦ Pend ↦ End ↦ Pend ↦ s5 ↦ 5
                     ...}
End
```

Listing 7.1 – Chor :Ãľgraphie de protocole de transfert de fichier simple

3 Scénarios de réparation

Nous observons que les conditions suffisantes ne sont pas satisfaites par le *CP* de la Figure 8.2 pour les opérateurs de séquence et de choix. L'instanciation du modèle générique par d'un tel *CP* confirme la violation de conditions suffisantes deux fois, aux étapes de la composition en séquence et de la composition en choix.

Grâce à l'instanciation fournie par le modèle formel B-événementiel que nous avons construit, il est possible de détecter automatiquement les violations de conditions suffisantes et donc de proposer une récupération ou une réparation permettant de rétablir l'ordre correcte d'échange de messages lors d'une communication. La réparation est basée sur l'introduction de messages de synchronisation. Dans ce cas, la réparation restaurera les propriétés ISeqF et PCF violées. Deux cas de réparation peuvent être distingués pour les opérateurs de séquence et de branche comme suit.

3.1 Restauration du ISeqF

Réparation de la propriété de séquence ISeqF. Deux réparations possibles sont identifiées sur les Figures 7.3 et 7.4. Suivant la définition de ISeqF , nous introduisons une transition de synchronisation avec le message *Sync*0 (en gras pointillé dans les Figures 7.3 et 7.4) établissant la condition ISeqF.

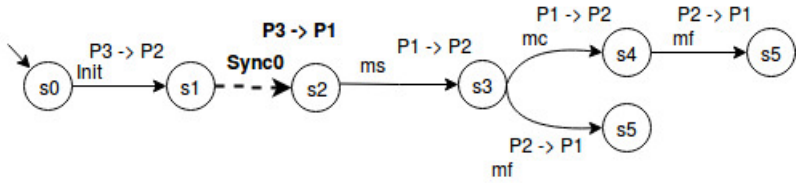


FIGURE *7.3 – Proposition 1 de la réparation de ISeqF.*

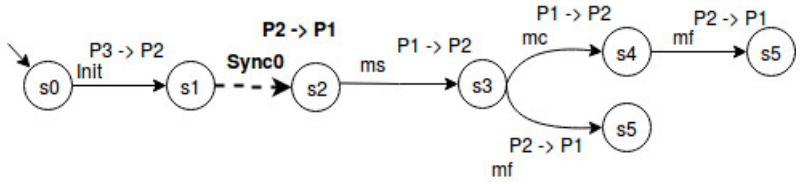


FIGURE *7.4 – Proposition 2 de la réparation de ISeqF.*

La spécification de la chorégraphie réparée de la Figure 7.3 est non réalisable. Nous illustrons ceci en définissant une séquence d'applications d'opérateurs. Nous obtenons les étapes suivantes de composition.

— $CP = \varnothing$

Transitions basiques.

$CP_{b0} = s_0 \xrightarrow{Init^{P3 \to P2}} s_1,$

$CP_{b1} = s_1 \xrightarrow{Sync0^{P3\to P1}} s_2,$
$CP_{b2} = s_2 \xrightarrow{ms^{P1\to P2}} s_3,$
$CP_{b3} = s_3 \xrightarrow{mc^{P1\to P2}} s_4,$
$CP_{b4} = s_3 \xrightarrow{mf^{P2\to P1}} s_5,$
$CP_{b5} = s_4 \xrightarrow{mf^{P2\to P1}} s_5.$

La prochaine étape consiste à appliquer une séquence d'opérateurs de composition prédéfinit, conduisant à la construction du CP de la Figure 7.3. Ces opérateurs ne sont appliqués que si les conditions suffisantes associées à chaque opérateur de composition sont vérifiées.

L'application d'opérateur de choix pour ajouter les deux transitions en branche CP_{b3} et CP_{b4} suffit pour détecter les violations des conditions suffisantes.

— $CP_0 = \otimes_{(\gg, s^0_{CP})}(CP, CP_{b0})\checkmark$ $CP_0 \in$ ISeqF.
— $CP_1 = \otimes_{(\gg, s^1_{CP})}(CP_0, CP_{b1})\checkmark,$ $CP_1 \in$ ISeqF,
— $CP_2 = \otimes_{(\gg, s(0)_{CP})}(CP_1, CP_{b2})\checkmark,$ $CP_2 \in$ ISeqF,
— $CP_3 = \otimes_{(+, s^2_{CP})}(CP_1, \{CP_{b3}, CP_{b4}\})\times,$ $CP_3 \notin$ PCF

Nous observons que CP_3 correspondant respectivement à l'application d'opérateur de choix ne satisfait pas les conditions suffisantes correspondantes.

3.2 Restauration du PCF

Réparation de la propriété de branchement PCF. De manière similaire à la réparation de séquence, deux scénarios de réparation sont possibles selon la définition de PCF . Ils sont décrits dans les Figures 7.5 et 7.6. La réparation de la transition de choix nécessite l'introduction d'un échange de messages de synchronisation $Sync1$ (en gras pointillé dans les Figures 7.5 et 7.6) qui précède les transitions en branches violant la condition PCF. Ces transitions échangent des messages de synchronisation entre la même entité émettrice que les autres branches et une entité réceptrice quelconque.

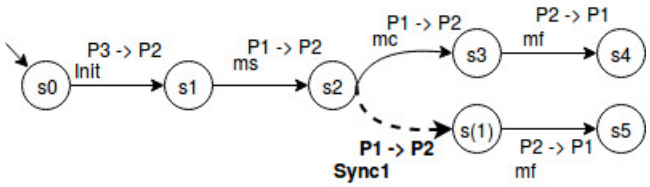


FIGURE 7.5 – *Proposition 1 de la réparation du PCF.*

La spécification de la chorégraphie réparée de la Figure 7.3 est réalisable. Nous illustrons ceci en définissant une séquence d'applications d'opérateurs. Nous obtenons les étapes de composition suivantes.

— $CP = \varnothing$

Transitions basiques.

$CP_{b0} = s_0 \xrightarrow{Init^{P3\to P2}} s_1,$
$CP_{b1} = s_1 \xrightarrow{ms^{P1\to P2}} s_2,$

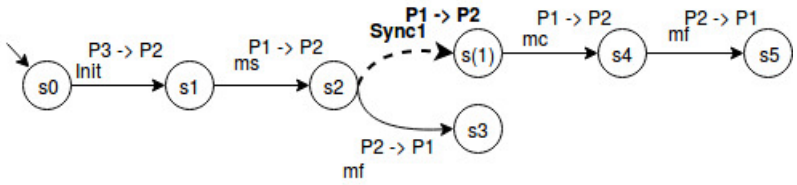


FIGURE *7.6 – Proposition 2 de la réparation du PCF.*

$CP_{b2} = s_2 \xrightarrow{Sync1^{P1\to P2}} s(1),$
$CP_{b3} = s_2 \xrightarrow{mf^{P2\to P1}} s_3,$
$CP_{b4} = s(1) \xrightarrow{mc^{P1\to P2}} s_4,$
$CP_{b5} = s_4 \xrightarrow{mf^{P2\to P1}} s_5.$

Composition. Les deux premières applications d'opérateurs de composition suffisent à détecter les violations des conditions suffisantes pour les opérateurs de séquence et de choix.

— $CP_0 = \otimes_{(\gg, s^0_{CP})}(CP, CP_{b0})\checkmark$ $CP_0 \in$ ISeqF.
— $CP_1 = \otimes_{(\gg, s^1_{CP})}(CP_0, CP_{b1})\times,$ $CP_1 \notin$ ISeqF,
— $CP_2 = \otimes_{(+, s^2_{CP})}(CP_1, \{CP_{b2}, CP_{b3}\})\checkmark,$ $CP_3 \in$ PCF
— $CP_3 = \otimes_{(\gg, s^4_{CP})}(CP_1, CP_{b4})\checkmark,$ $CP_3 \in$ ISeqF
— $CP_4 = \otimes_{(\gg, s^4_{CP})}(CP_1, CP_{b5})\checkmark,$ $CP_3 \in$ ISeqF

Nous observons que le protocole de conversation CP_1 correspond à l'application d'opérateur de séquence ne satisfait pas les conditions suffisantes correspondantes.

4 Formalisation avec B-événementiel

Cette section présente le modèle abstrait des opérateurs de réparation définis. Un événement B-événementiel est associé à chaque opérateur, permettant ainsi de réparer des CPs non-réalisables. Chaque événement ($ISeqF\_Restore$ et $PCF\_Restore$) est conditionné par nos propriétés suffisantes.

Le modèle de réparation B-événementiel est défini en deux parties. La partie contexte contient les axiomes pertinents définissant les entités communicantes, le protocole de conversation, les propositions de réparation, les conditions suffisantes, . . . , etc. Les différents événements correspondant aux opérateurs définis sont donnés par la liste des événements.

4.1 Partie statique : définitions requises

Le listing 7.2 définit le protocole de conversation, les propositions de reparation et les entités projetés dans les axiomes $axm1$, $axm2$ et $axm3$, respectivement. Ensuite, les différentes conditions suffisantes sont décrites.

— Le choix déterministe (DC) est défini à partir de l'ensemble des transitions formant un choix non déterministes (NDC) dans les axiomes $axm3\_Cond1$ et $axm4\_Cond1$. Ensuite, les transitions déterministes DC sont obtenues par soustraction dans l'axiome $axm5\_Cond1$.

— La définition de ISeqF est donnée par les axiomes $axm6\_Cond2$ et $axm7\_Cond2$. Il compare l'entité source $P\_SRC(cp\_b)$ a l'entité émettrice $LAST\_SDR\_Ps$ ou a l'entité réceptrice $LAST\_RCV\_Ps$ de la dernière transition du CP.

— De même, afin de définir la propriété PCF , dans les axiomes $axm8\_Cond3$ et $axm9\_Cond3$, l'entité émettrice $PEER\_SRC(Trans)$ des transitions formant le branchement.

L'axiome $axm14$ définit l'ensemble des transitions en séquence ($axm10$ et $axm11$) et en choix ($axm12$ et $axm13$) satisfaisant la propriété de réalisabilité.

Context
Sets PEERS, MESSAGES , CP_STATES
Constants CPs_B, Repare_propositions, PEERs_B, DC, ISeqF, NDC, ...
Axioms

$-- $ **Basic conversation protocols definition**

$axm1 : CPs\_B \subseteq CP\_STATES \times PEERS \times MESSAGES \times PEERS \times CP\_STATES \times \mathbb{N}$

$axm2 : Repare\_propositions \subseteq CP\_STATES \times PEERS \times MESSAGES \times PEERS \times CP\_STATES \times \mathbb{N}$

$axm3 : PEERs\_B \subseteq (A\_STATES \times ETIQ \times \mathbb{N}) \nrightarrow A\_STATES$

$axm4 : BUILT\_CP\_SET = CPs\_B \cup Repare\_propositions$

$-- $ **Determinstic CP definition DC**

$axm3\_Cond1 : NDC \subseteq BUILT\_CP\_SET$

$axm4\_Cond1 : \forall Trans2, Trans1 \cdot ($
$Trans1 \in BUILT\_CP\_SET \land Trans2 \in BUILT\_CP\_SET \land$
$SRC\_STATE(Trans1) = SRC\_STATE(Trans2) \land$
$LABEL(Trans1) = LABEL(Trans2) \land DST\_STATE(Trans1) \neq DST\_STATE(Trans2))$
$\Rightarrow \{Trans1, Trans2\} \subseteq NDC$

$axm5\_Cond1 : DC = BUILT\_CP\_SET \setminus NDC$

$-- $ **Independent sequence freeness definition ISEQF**

$axm6\_Cond2 : ISeqF \subseteq BUILT\_CP\_SET$

$axm7\_Cond2 : \forall cp\_b \cdot (cp\_b \in BUILT\_CP\_SET \land (P\_SRC(cp\_b) = LAST\_SDR\_Ps(SRC\_STATE(cp\_b)) \lor$
$P\_SRC(cp\_b) = LAST\_RCV\_Ps(SRC\_STATE(cp\_b))))$
$\Rightarrow \{cp\_b\} \subseteq ISeqF$

$--- $ **Parallel Choice freeness PCF**

$axm8\_Cond3 : PCF \in CP\_STATES \rightarrow \mathbb{P}(BUILT\_CP\_SET)$

$axm9\_Cond3 : \forall Trans1, Trans2 . (Trans1 \in BUILT\_CP\_SET \land Trans2 \in BUILT\_CP\_SET \land$
$SRC\_STATE(Trans1) = SRC\_STATE(Trans2) \land$
$P\_SRC(Trans1) = P\_SRC(Trans2) \land$
$DST\_STATE(Trans1) \neq DST\_STATE(Trans2)) \land$
$\Rightarrow PCF(SRC\_STATE(Trans1)) = \{Trans1, Trans2\}$

$axm10 : Sequence\_OK \subseteq BUILT\_CP\_SET$

$axm11 : \forall Trans . (Trans \in BUILT\_CP\_SET \land \{Trans\} \subseteq ISeqF)$
$\Rightarrow Trans \in Sequence\_OK$

$axm12 : Branch\_OK \subseteq BUILT\_CP\_SET$

$axm13 : \forall Trans1, Trans2 . (Trans1 \in BUILT\_CP\_SET \land Trans2 \in BUILT\_CP\_SET \land$
$\{Trans1, Trans2\} = PCF(SOURCE\_STATE(Trans1)))$
$\Rightarrow \{Trans1, Trans2\} \subseteq Branch\_OK$

$axm14 : OK = Sequence\_OK \cup Branch\_OK$

...
End

Listing 7.2 – Definition of sufficient conditions

4.2 Partie dynamique : modélisation de la réparation

Les définitions formelles des événements codant les opérateurs de réparation (restauration de ISeqF de la définition 16 ainsi que la restauration de PCF de la définition 17 sont donné dans cette section. Chaque événement est protégé par les conditions suffisantes définies telles que, le *CP* et les entités projetés soient composées si et seulement si les conditions des évènements sont satisfaites.

L'opérateur ISeqF_Restore. L'événement $ISeqF\_Repair$ de Listing 7.4 code la réparation de la propriété ISeqF (définition 2 chapitre 5) en cas de violation. Il ajoute dans l'action $act1$ un CP_b basique , à savoir $Some\_reparation$, donné en tant que paramètre du CP en cours de construction, ainsi qu'une transition $Some\_reparation$ appartenant à l'ensemble des transitions de réparation proposées, à savoir $Repare\_propositions$ ($grd1$). Cet événement de réparation est déclenché uniquement si la construction incrémentale est interrompue en raison de la condition ISeqF non satisfaite dans $grd2$. En particulier, la propriété ISeqF doit être satisfaite avant d'ajouter la transition de réparation en séquence ($grd3$).

Invariants
$inv1 : BUILT\_CP \subseteq DC$
$inv2 : \forall Trans.(Trans \in BUILT\_CP \wedge BUILT\_CP \neq \varnothing \wedge MSG(Trans) \neq End) \Rightarrow Trans \in OK$

...
End

Listing 7.3 – Un extrait des invariants du mod :Ãİle LTS_Asynchronous_model

Event $ISeqF\_Restore \triangleq$
Any $Some\_reparation, cp\_b\_Breaks\_ISeqF$
Where
$grd1$:$Some\_reparation \in Repare\_propositions$
$grd2$:$cp\_b\_Breaks\_ISeqF \in CPs\_B$
$grd3$:$BUILT\_CP \neq \varnothing \wedge Some\_reparation \in ISeqF$
$grd4$:$\exists Trans.(Trans \in CPs\_B \wedge Trans \notin ISeqF)$
$grd5$:$BUILT\_CP \neq \varnothing \Rightarrow cp\_b\_Breaks\_ISeqF \in ISeqF$
$grd6$:$Repair\_src\_state(Some\_reparation) \in CP\_Final\_states$

...

With
Then
$act1 : BUILT\_CP := BUILT\_CP \cup \{Some_reparation\} \cup \{cp\_b\_Breaks\_ISeqF\}$
$act2 : CP\_Final\_states := (CP\_Final\_states \cup \{DST\_STATE(cp\_b\_Breaks\_ISeqF)\}) \setminus$
$\{Repair\_src\_state(Some\_reparation)\}$
End

Listing 7.4 – An excerpt of the ISeqF_Restore composition operator

PCF_Restore operator. L'événement PCF_Repair du Listing 7.5 correspond à la réparation de la condition PCF (définition 3). Cet événement ajoute un ensemble de CP_b basique déterministes, définis comme paramètres de l'événement, à savoir PCF_Repair ($grd4$), et formé d'au moins deux CP_b basiques, à savoir $Branches\_reparation$ ($grd2$) avec même état et entité source. Ces transitions doivent satisfaire les conditions de séquence ISeqF et de branche PCF ($grd5$ et $grd6$ respectivement) liées à la composition du choix. $act1$ met à jour le CP en cours de construction.

Event $PCF\_Restore \triangleq$
Any $Branches\_reparation, branch\_reparation, cp\_b\_Breaks\_PCF$
Where

$grd1$:$\exists Branches, branch.(branch \in OK \wedge Branches \subseteq OK \wedge branch \in Branches \wedge$
$Branches \neq PCF\_Repair\_BRANCHES\_SET(SRC\_STATE(branch)))$
$grd2$:$Branches\_reparation \subseteq OK$
$grd3$:$branch\_reparation \in OK$
$grd4$:$Branches\_reparation \subseteq DC$
$grd5$:$BUILT\_CP \neq \varnothing \wedge branch\_reparation \in ISeqF$
$grd6$: $Branches\_reparation = PCF\_Repair\_BRANCHES\_SET(SRC\_STATE(branch\_reparation))$
$grd7$: $branch\_reparation \in Branches\_reparation$
$grd8$: $Branches\_reparation \subseteq Repare\_propositions$

$grd9 : BUILT\_CP \neq \varnothing \Rightarrow cp\_b\_Breaks\_PCF \in ISeqF$

With
Then
$act1 : BUILT\_CP := BUILT\_CP \cup Branches\_reparation \cup \{cp\_b\_Breaks\_PCF\}$
$act2 : CP\_Final\_states := (CP\_Final\_states \cup BR\_CP\_FINAL\_STATES(SRC\_STATE(branch\_reparation)) \cup \{DST\_STATE(cp\_b\_Breaks\_PCF)\}) \setminus (\{SRC\_STATE(branch\_reparation) \cup \{SRC\_STATE(cp\_b\_Breaks\_PCF)\})$
End

Listing 7.5 – An excerpt of the PCF_Repair composition operator

5 Restauration de la réalisabilité et instanciation

Les deux réparations proposées dans les Sections 3.1 et 3.2 donnent quatre scénarios de réparation. Parmi ces scénarios, nous sélectionnons le *CP* de la Figure 7.7 pour illustrer l'exactitude de la réparation. Concrètement, le *CP* défini à la Figure 8.2 est remplacé par celui de la Figure 7.7, en ajoutant les nouvelles transitions de synchronisation (transitions gras en pointillé). Le *CP* obtenu est réalisable.

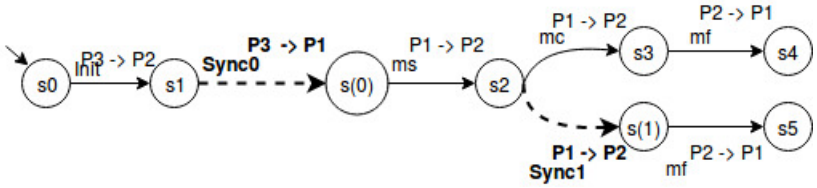


FIGURE 7.7 – *CP réparé du protocole de transfert de fichier*

La projection du *CP* réparé produit à partir des entités de la Figure 7.8.

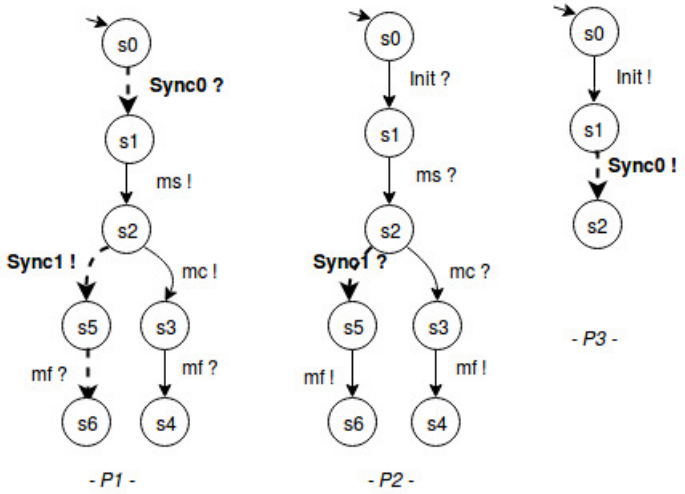


FIGURE 7.8 – *Entités projetées du CP réparé du protocole de transfert de fichier*

Ci-dessous, nous montrons la séquence des opérations de composition et de réparation permettant de construire le *CP* réparé de la Figure 7.7.

— $CP = \varnothing$

Transitions basiques.

$CP_{b0} = s_0 \xrightarrow{Init^{P3 \to P2}} s_1,$

$CP_{b1} = s_1 \xrightarrow{Sync0^{P3\to P1}} s(0)$,
$CP_{b2} = s(0) \xrightarrow{ms^{P1\to P2}} s_2$,
$CP_{b3} = s_2 \xrightarrow{mc^{P1\to P2}} s_3$,
$CP_{b4} = s_3 \xrightarrow{mf^{P2\to P1}} s_4$,
$CP_{b5} = s_2 \xrightarrow{Sync1^{P1\to P2}} s(1)$,
$CP_{b6} = s(1) \xrightarrow{mf^{P2\to P1}} s_5$.

Composition.

— $CP_1 = \otimes_{(\gg, s^1_{CP})}(CP, CP_{b0}), \checkmark$ $CP_1 \in$ ISeqF
— $CP_2 = \otimes_{(\gg, s(0)_{CP})}(CP_1, CP_{b1}), \checkmark$ $CP_2 \in$ ISeqF
— $CP_3 = \otimes_{(+, s^2_{CP})}(CP_1, \{CP_{b3}, CP_{b5}\}), \checkmark$ $CP_3 \in$ ISeqF $\wedge CP_3 \in$ DC $\wedge CP_3 \in$ PCF
— $CP_4 = \otimes_{(\gg, s^3_{CP})}(CP_3, CP_{b4}), \checkmark$ $CP_2 \in$ ISeqF
— $CP_5 = \otimes_{(\gg, s(1)_{CP})}(CP_4, CP_{b6}), \checkmark$ $CP_5 \in$ ISeqF

Le listing 7.6 décrit le contexte B-événementiel définissant les instances du modèle générique décrivant le CP de la figure 7.7.

Context
Constants $s0, s1, s2, s3, s4, s5, s(0), ms, mc, mf, Sync0, \dots$
Axioms
$axm1$:$partition(PEERS, \{P1\}, \{P2\}, \{P3\})$
$axm2$:$partition(MESSAGES, \{ms\}, \{mf\}, \{Sync0\}, \dots)$
$axm3$:$partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{ns(0)\}, \{s3\}, \dots)$
$axm4$: $partition(A\_STATES, \{s0\_P1\}, \{s1\_P1\}, \dots)$
$axm5$: $CPs\_B = \{s0 \mapsto P3 \mapsto Init \mapsto P2 \mapsto s1 \mapsto 1,$
$s2 \mapsto P1 \mapsto ms \mapsto P2 \mapsto s3 \mapsto 3,$
$s3 \mapsto P1 \mapsto mc \mapsto P2 \mapsto s5 \mapsto 5,$
$s4 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s6 \mapsto 6,$
$s5 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s7 \mapsto 7,$
$\}$

$axm6$: $Repare\_propositions = \{s1 \mapsto P1 \mapsto Sync0 \mapsto P3 \mapsto s2 \mapsto 2,$
$s1 \mapsto P2 \mapsto Sync0 \mapsto P3 \mapsto s2 \mapsto 2,$
$s2 \mapsto P1 \mapsto Sync1 \mapsto P2 \mapsto s3 \mapsto 4,$
$s2 \mapsto P2 \mapsto Sync1 \mapsto P3 \mapsto s4 \mapsto 4$
$\}$
End

Listing 7.6 – Repaired choreography of a simple file transfer protocol instantiation

6 Evaluation et statistique de preuves

La stratégie de réparation a conduit à un développement formel complètement prouvé disponible à l'annexe B. La table 7.2 montre les résultats des expériences que nous avons menées au sein de la plate-forme Rodin du B-événementiel. Le développement présenté a été entièrement encodé et prouvé. Une absence de blocage et un comportement correct ont toutes été prouvées. Les résultats montrent que peu d'obligations de preuve (POs) nécessitent des épreuves interactives (22 sur 84 POs générées). Les tailles des différentes preuves pour le machine et le contexte sont disponibles à la figure 7.9.

Modèle B-événementiel	**Preuves Automatiques**	**Preuves Interactive**	**Obligations de Preuves**
Contexte Abstrait	06 (75%)	02 (25%)	08 (100%)
Modèle Abstrait	56 (73,68%)	20 (26,31%)	76 (100%)
Total	62 (100%)	22 (100%)	84 (100%)

TABLE *7.2 – Statistiques des preuves RODIN (modèle de réparation)*

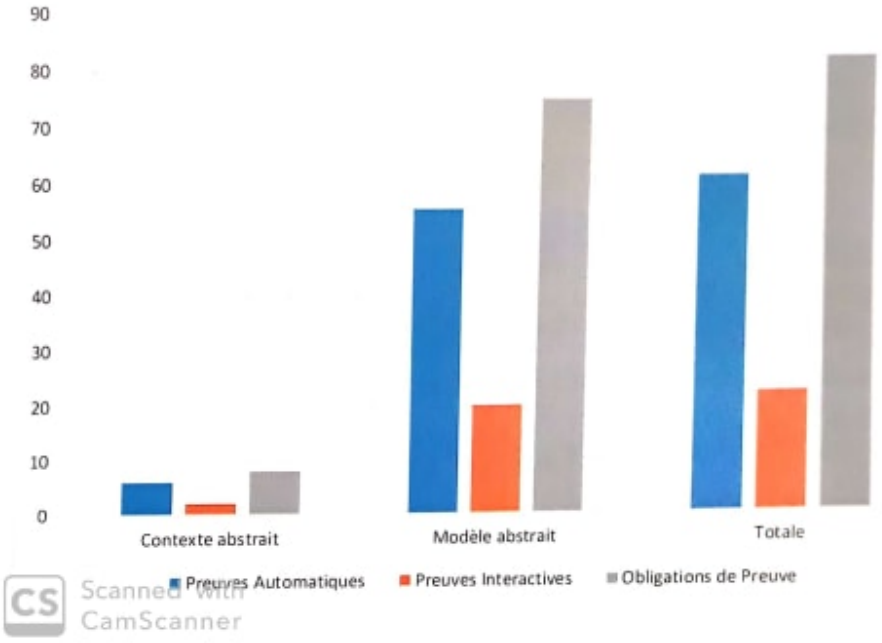


FIGURE *7.9 – Les preuves associées aux modèle de réparation B-événementiel*

Chapitre 8

Évaluation et mise en oeuvre

Sommaire

Dans ce chapitre, nous prouvons la validation de notre approche étendu dans les chapitres 5, 6 et 7, tel que, nous montrons comment notre approche s'applique et évolue à un ensemble de cas d'utilisation empruntés à la littérature et utilisés par la communauté des chercheurs. Nous montrons également que les approches de composition ainsi que de la réparation permettent de détecter les défaillances et que la reprise après défaillance n'est pas réalisable. Nous nous concentrons également sur l'évolutivité de l'approche dans le cas de *CPs* complexes.

1 Cas d'étude de composition réalisable

Dans nos précédents chapitres 5 et 6, nous avons prouvé formellement l'exactitude de notre approche de composition des *CPs* réalisables. Notre objectif dans ce chapitre est de montrer que l'approche est non seulement vérifiable, mais également valide. En effet, nous avons pris plusieurs repères dans la littérature consacrée aux domaines de recherche liés à la conception de systèmes répartis basés sur la propriété de réalisabilité. Tous ces points de référence ont été soumis à notre modèle

B-événementiel en tant qu'instances du modèle générique résumé dans le chapitre 6. Ces instances sont définies dans B-événementiel en décrivant les extensions d'ensembles différés (arbitraires) utilisés dans le modèle générique et en fournissant des témoins aux paramètres de la clause *ANY* de B-événementiel utilisée dans chaque événement définissant un opérateur de composition.

Cette section est divisée en deux parties. Tout d'abord, afin d'expliquer la configuration de l'instanciation du modèle générique, nous donnons une description détaillée du processus d'instanciation d'une première étude de cas réalisable dans la Section 1.1 et d'une seconde étude de cas non réalisable dans la Section 1.2. Ensuite, dans une deuxième partie, nous discutons des résultats obtenus pour d'autres études de cas composant nos indices de référence. Notant que l'instanciation du modèle générique est vérifiée à l'aide de la plate-forme RODIN et de l'animateur ProB.

1.1 CS1. Processus d'entreprise virtuelle

La chorégraphie décrit un *CP* d'entreprise virtuelle décrit à la Figure 8.1 et étudié à la section 4 du chapitre 5. Il est emprunté à [84] et décrit un protocole de conversation réalisable.

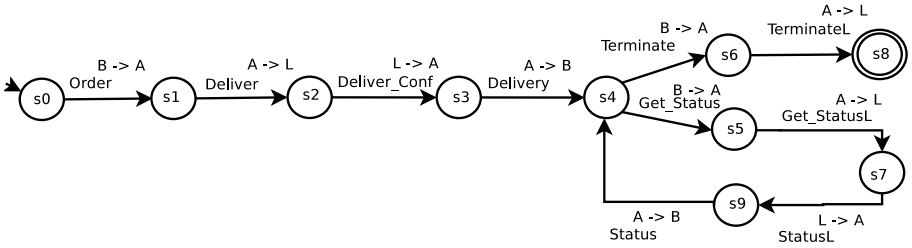


FIGURE *8.1 – Protocole de conversation d'un processus d'approvisionnement simple au sein d'une entreprise virtuelle*

1.2 CS2. Protocole de transfert de fichier

L'étude de cas présente une chorégraphie d'un protocole de transfert de fichiers non réalisable empruntée à [13]. Ce cas est décrit dans la section 1.2 du chapitre 7. Une réparation de cette chorégraphie est également proposée dans la section 2.

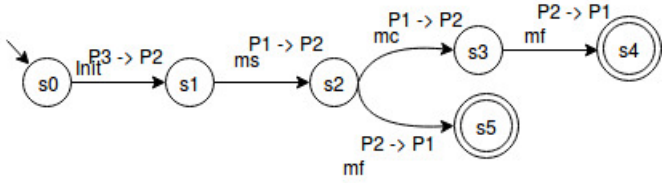


FIGURE *8.2 – Chorégraphie de protocole de transfert de fichier simple*

1.3 CS3. Accés à une application web

Le protocol d'accés à une application web est un exemple concret décrit dans la Figure 8.3 et emprunté à [16]. Il implique trois entités communicantes : un client (cl),

une interface Web (*int*) et une application logicielle (*appli*). Le *CP* commence par une interaction *login* (*connect*) entre le client et l'interface, suivie par la demande d'accès (*access*) déclenchée par le client. Cette demande peut être répétée autant que nécessaire. Enfin, le client décide de se déconnecté de l'interface (*logout*). Ce *CP* est réalisable.

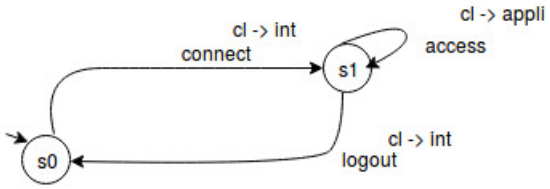


FIGURE *8.3 – Accès à une application web*

Le Listing 8.1 résume le comportement du système décrit en Figure 8.3 par instanciation du modèle générique de la composition réalisable des *CPs*.

Context
Constants
Axioms
$axm1 : partition(PEERS, \{cl\}, \{int\}, \{appli\}, \{Pend\})$
$axm2 : partition(MESSAGES, \{connect\}, \{access\}, \{logout\}, \{End\})$
$axm3 : partition(CP\_STATES, \{s0\}, \{s1\})$
$axm4 : partition(A\_STATES, \{s0\_cl\}, \{s1\_cl\}, \dots)$
$axm5 : CPs\_B = \{s0 \mapsto cl \mapsto connect \mapsto int \mapsto s1 \mapsto 1,$
$s1 \mapsto cl \mapsto access \mapsto appli \mapsto s1 \mapsto 2,$
$s1 \mapsto cl \mapsto logout \mapsto int \mapsto s0 \mapsto 3,$
$s0 \mapsto Pend \mapsto End \mapsto Pend \mapsto s2 \mapsto 4$
$\dots\}$
End

Listing 8.1 – Acc :ÃÍs :Ãă une application web

1.4 CS4. Accès à une base de données via une application web

Le protocol Accès à une base de données via une application web est une extension de CS1. Le *CP* est emprunté à [51]. Cela implique une entités supplémentaire avec une base de données *db*. Le *CP* introduit une connexion entre l'interface et l'application (*Setup*), un *accès* par le client à la base de données et un *log* par l'application. Le *CP* décrit dans la Figure 8.4 n'est pas réalisable, mais une réparation est donnée dans [51].

Le Listing 8.2 instantié le comportement du *CP* décrit dans la Figure 8.4.

Context
Constants
Axioms
$axm1 : partition(PEERS, \{cl\}, \{int\}, \{appli\}, \{access\}, \{db\}, \{Pend\})$
$axm2 : partition(MESSAGES, \{connect\}, \{setup\}, \{access\}, \{logout\}, \{log\}, \{End\})$
$axm3 : partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\})$
$axm4 : partition(A\_STATES, \{s0\_cl\}, \{s1\_cl\}, \dots)$
$axm5 : CPs\_B = \{s0 \mapsto cl \mapsto connect \mapsto int \mapsto s1 \mapsto 1,$
$s1 \mapsto int \mapsto setup \mapsto appli \mapsto s2 \mapsto 2,$

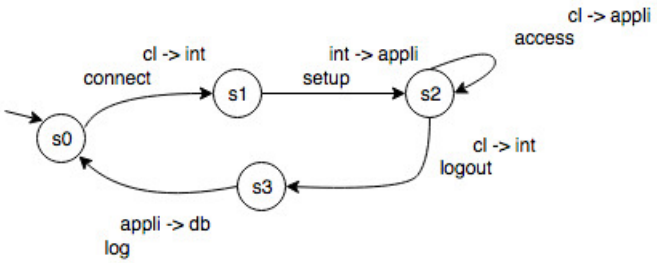


FIGURE *8.4 – Accès à une base de données via une application Web*

$s2 \mapsto cl \mapsto access \mapsto appli \mapsto s2 \mapsto 3,$
$s2 \mapsto cl \mapsto logout \mapsto int \mapsto s3 \mapsto 4,$
$s3 \mapsto appli \mapsto log \mapsto db \mapsto s0 \mapsto 5,$
$s0 \mapsto Pend \mapsto End \mapsto Pend \mapsto s4 \mapsto 6$
}

End

Listing 8.2 – Acc :ÃÍs :Ãă une application web

1.5 CS5. Application d'achat en ligne

L'application d'achat en ligne décrite en Figure 8.5 est non réalisable. Elle est empruntée à [80], l'application commence par demander le prix de certaines marchandises au vendeur et le client communique via un message *priceReq* avec le vendeur. Le vendeur calcule le prix du produit via un message *offer* et envoie le prix au client. Si l'offre est acceptée, le vendeur envoie à la banque les détails du paiement via le message *PayReq*. Ensuite, le client autorise le paiement via le message *pay*. Une fois le paiement est terminé avec succès, l'application se termine dés que, la banque confirme le paiement au vendeur et au client, ou le paiement est annulé. La description formelle du *CP* de la Figure 8.5 est donné en Listing 8.3.

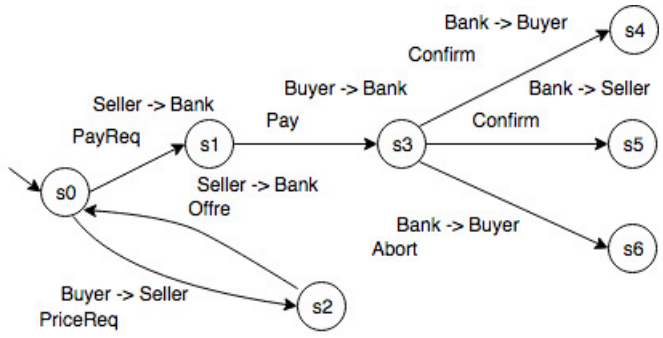


FIGURE *8.5 – Chorégraphie non réalisable d'achat en ligne*

Context
Constants
Axioms
$axm1$:$partition(PEERS, \{Seller\}, \{Buyer\}, \{Bank\}, \{access\}, \{Pend\})$
$axm2$:$partition(MESSAGES, \{PayReq\}, \{Pay\}, \{PriceReq\}, \{Offre\}, \{Confirm\}, \{Abort\}, \{End\})$
$axm3$:$partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\})$
$axm4$: $partition(A\_STATES, \{s0\_Seller\}, \{s1\_Seller\}, \dots)$
$axm5 : CPs\_B = \{s0 \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s1 \mapsto 1,$
$s0 \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s2 \mapsto 1,$
$s1 \mapsto Seller \mapsto Offre \mapsto Buyer \mapsto s0 \mapsto 2,$
$s2 \mapsto Buyer \mapsto Pay \mapsto Bank \mapsto s3 \mapsto 3,$

$s3 \mapsto Bank \mapsto Abort \mapsto Buyer \mapsto s4 \mapsto 4,$
$s3 \mapsto Bank \mapsto Confirm \mapsto Seller \mapsto s5 \mapsto 4,$
$s4 \mapsto Pend \mapsto End \mapsto Pend \mapsto s6 \mapsto 5,$
$s5 \mapsto Pend \mapsto End \mapsto Pend \mapsto s7 \mapsto 6$
}

End

Listing 8.3 – Acc :ÃÍs :Ãă une application web

1.6 CS6. protocole de visa de travail australien

La Figure 8.6 montre une représentation graphique d'un protocole pour un service de demande de visa de travail australien emprunté à [87]. Le service de demande de visa est initialement à l'état de départ s_0 et son utilisation commence lorsqu'un client envoie un message checkEligibility et le service passe à l'état s_1. En général, les clients souhaitant travailler en Australie peuvent arriver à l'état s_8 en remplissant le formulaire de demande de travail, en soumettant leur expérience professionnelle et en testant leur niveau d'anglais. Les clients qui souhaitent renouveler leurs visas de travail après expiration peuvent se rendre dans le même état, en remplissant la demande et en fournissant une lettre de recommandation de l'employeur.

Pour demander un visa d'étude australien, les étudiants étrangers remplissent le formulaire pour les étudiants étrangers et en soumettant le certificat d'obtention du diplôme et le passeport. Puis le système vérifie et approuve le statut de la demande. Le protocole arrive a sa fin a l'état s_9 ou s_{10}.

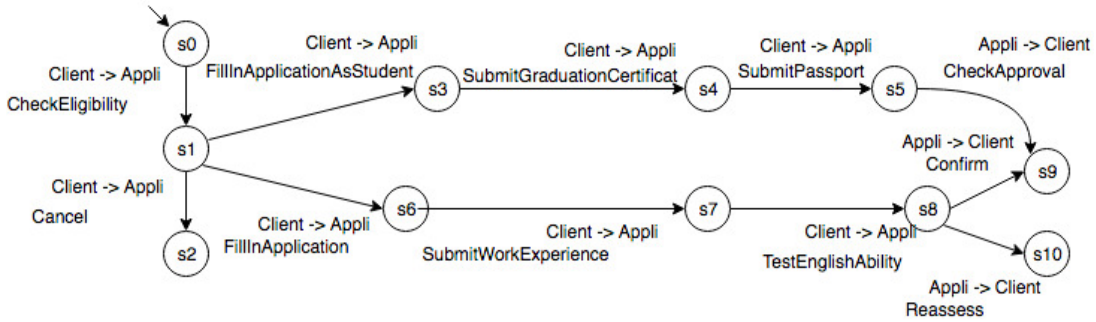


FIGURE *8.6 – Protocole de visa de travail australien*

Le Listing 8.4 représente l'instance du comportement du *CP* de la Figure 8.4.

Context
Constants
Axioms
$axm1 : partition(PEERS, \{Client\}, \{Appli\}, \{Pend\})$
$axm2 : partition(MESSAGES, \{CheckEligibility\}, \{Cancel\}, \{FillInApplication\}, \ldots, \{End\})$
$axm3 : partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \ldots)$
$axm4 : partition(A\_STATES, \{s0\_Client\}, \{s1\_Client\}, \ldots)$
$axm5 : CPs\_B = \{s0 \mapsto Client \mapsto CheckEligibility \mapsto Appli \mapsto s1 \mapsto 1,$
$s1 \mapsto Client \mapsto Cancel \mapsto Appli \mapsto s2 \mapsto 2,$
$s1 \mapsto Client \mapsto FillInApplicationAsStudent \mapsto B \mapsto s3 \mapsto 3,$
$s1 \mapsto Client \mapsto FillInApplication \mapsto Appli \mapsto s3 \mapsto 4,$
$s3 \mapsto Client \mapsto SubmitGraduationCertificat \mapsto Appli \mapsto s4 \mapsto 5,$
$s6 \mapsto Client \mapsto SubmitWorkExperience \mapsto Appli \mapsto s7 \mapsto 6,$
$s4 \mapsto Client \mapsto SubmitPassport \mapsto Appli \mapsto s5 \mapsto 7,$
$s7 \mapsto Client \mapsto TestEnglishAbility \mapsto Appli \mapsto s8 \mapsto 8,$
$s5 \mapsto Client \mapsto CheckApproval \mapsto Appli \mapsto s9 \mapsto 9,$
$s8 \mapsto Appli \mapsto Confirm \mapsto Client \mapsto s9 \mapsto 9,$

$s8 \mapsto Appli \mapsto Reassess \mapsto Client \mapsto s10 \mapsto 10,$
$\ldots\}$
End

Listing 8.4 – Acces a une application web

1.7 CS7. Protocole d'un jeu fictif

Le protocole d'un jeu fictif emprunté à [66] et représenté à la Figure 8.7 est composé de quatre entités communicantes où :

1. Alice (A) envoie *bwin* à Bob (B) ou *cwin* à Carol (C) pour décider qui gagne la partie. Dans le premier cas, A déclenche la transition *AB ! Bwin* dans laquelle le message *bwin* est placé dans la queue FIFO AB de A à B, et de même dans le second cas.
2. Si B gagne (le message *bwin* se trouve en haut de la file d'attente AB et que B l'utilise en prenant la transition *AB ? Bwin*), il envoie une notification *(close)* à C pour l'informer qu'elle a perdu. Symétriquement, C informe B de sa victoire *(blose)*.
3. Pendant le jeu, C informe Dave (D) qu'elle est occupée.
4. Une fois que B et C ont été informés du résultat du jeu, B envoie un signal (sig) à A, tandis que C envoie un message (msg) à A.
5. Une fois le résultat envoyé, A notifie D que C est maintenant libre *message free* et un nouveau tour commence.

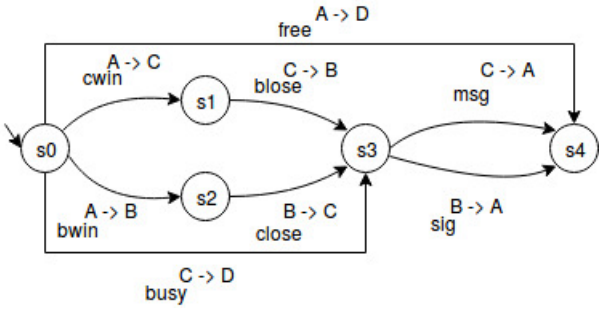


FIGURE *8.7 – Protocole d'un jeu fictif*

Le Listing 8.5 montre le contexte B-événementiel défini pour instancier le modèle générique de cette étude de cas.

Context
Constants
Axioms
$axm1$:$partition(PEERS, \{A\}, \{B\}, \{C\}, \{D\}, \{Pend\})$
$axm2$:$partition(MESSAGES, \{bwin\}, \{cwin\}, \{sig\}, \{msg\}, \{blose\}, \{close\}, \{busy\}, \{free\}, \{End\})$
$axm3$:$partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\})$
$axm4$: $partition(A\_STATES, \{s0\_A\}, \{s1\_A\}, \ldots)$
$axm5 : CPs\_B = \{s0 \mapsto A \mapsto cwin \mapsto C \mapsto s1 \mapsto 1,$
$s0 \mapsto A \mapsto bwin \mapsto B \mapsto s2 \mapsto 2,$
$s1 \mapsto C \mapsto blose \mapsto B \mapsto s3 \mapsto 3,$
$s2 \mapsto B \mapsto close \mapsto C \mapsto s3 \mapsto 4,$
$s3 \mapsto C \mapsto msg \mapsto A \mapsto s4 \mapsto 5,$
$s3 \mapsto B \mapsto sig \mapsto A \mapsto s4 \mapsto 6,$

$s0 \mapsto C \mapsto busy \mapsto D \mapsto s3 \mapsto 7,$
$s0 \mapsto A \mapsto free \mapsto D \mapsto s4 \mapsto 8,$
$\ldots\}$
End

Listing 8.5 – Acces a une application web

2 Cas d'étude de réparation de CPs non réalisables

Dans notre précédent Chapitre 7, nous avons expliqué et prouvé formellement notre stratégie de réparation et restauration de la propriété de réalisabilité. Dans cette section, notre objectif est de montrer que la réparation est non seulement vérifiable, mais également valide. En effet, nous avons pris plusieurs études de cas dans la littérature. Tous ces études de cas ont été soumis à notre modèle de réparation B-événementiel en tant qu'instances du modèle générique résumé dans le chapitre 7. Ces instances définissent les extensions d'ensembles arbitraires utilisés dans le modèle générique et en fournissant des témoins aux paramètres de la clause *ANY* de B-événementiel utilisée dans chaque événement définissant un opérateur.

Notant que l'instanciation du modèle générique est vérifiée à l'aide de la plateforme RODIN et de l'animateur ProB.

2.1 CS8. Réparation de CS4

Le protocole de conversation décrit dans la Figure 8.8 represente une réparation possible du *CP* non-réalisable de la Figure 8.4.

Le *CP* non-réalisable est emprunté à [51]. Cela implique une introduction de nouvelles transitions de synchronisation entre l'entité *appli* et le client *cl*. Et une deuxième synchronisation entre l'entité interface *int* et l'aplication *appli*. Le *CP* décrit restaure sa réalisabilité, l'illustration du bon fonctionnement du *CP* après réparation est détaillé dans le Listing 8.6.

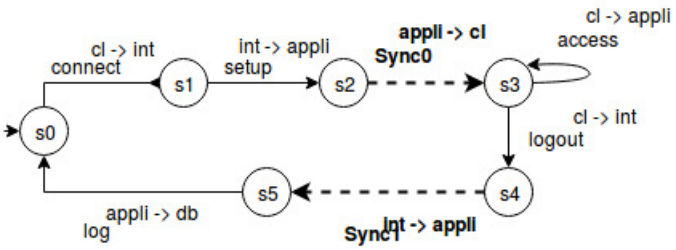


FIGURE *8.8 – Accès à une application web, CP réparé*

Context
Constants
Axioms
$axm1$:$partition(PEERS, \{cl\}, \{int\}, \{appli\}, \{Pend\})$
$axm2$:$partition(MESSAGES, \{connect\}, \{access\}, \{logout\}, \{End\})$
$axm3$:$partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\})$
$axm4$: $partition(A\_STATES, \{s0\_cl\}, \{s1\_cl\}, \ldots)$
$axm5$: $CPs\_B = \{s0 \mapsto cl \mapsto connect \mapsto int \mapsto s1 \mapsto 1,$
$s1 \mapsto int \mapsto setup \mapsto appli \mapsto s1 \mapsto 3,$
$s3 \mapsto cl \mapsto access \mapsto appli \mapsto s3 \mapsto 4,$
$s3 \mapsto cl \mapsto logout \mapsto int \mapsto s4 \mapsto 3,$
$s5 \mapsto appli \mapsto log \mapsto db \mapsto s0 \mapsto 4$

$\ldots\}$
$axm6 : Repare\_propositions = \{s0 \mapsto appli \mapsto Sync0 \mapsto cl \mapsto s1 \mapsto 1,$
$s0 \mapsto int \mapsto Sync0 \mapsto cl \mapsto s1 \mapsto 1,$
$s1 \mapsto cl \mapsto Sync1 \mapsto appli \mapsto s3 \mapsto 3,$
$s1 \mapsto int \mapsto Sync1 \mapsto appli \mapsto s3 \mapsto 3,$
$\}$
End

Listing 8.6 – Acces a une application web

2.2 CS9. Réparation de CS5

La chorégraphie d'achat en ligne de la Figure 8.5 est réparée à l'aide de notre stratégie de réparation afin de rétablir l'ensemble des conditions suffisantes. La réparation proposée est donnée à la Figure 8.9.

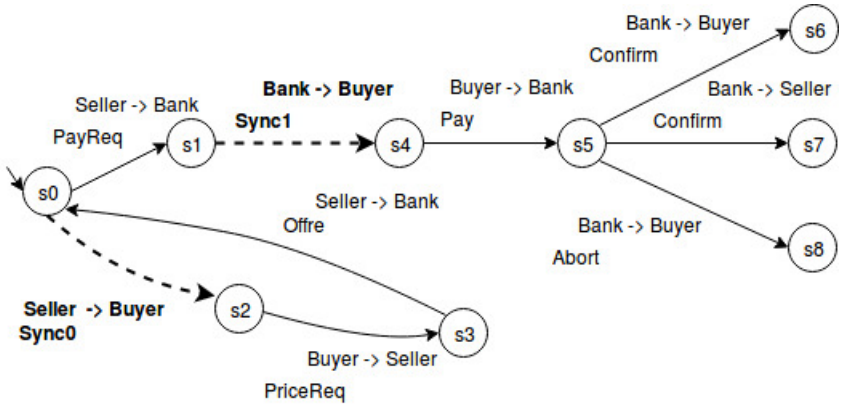


FIGURE *8.9 – Chorégraphie d'un achat en ligne*

Le listing 8.7 formalise le *CP* réparé de la Figure 8.9

Context
Constants
Axioms
$axm1 : partition(PEERS, \{Seller\}, \{Buyer\}, \{Bank\}, \{access\}, \{Pend\})$
$axm2 : partition(MESSAGES, \{PayReq\}, \{Pay\}, \{PriceReq\}, \{Offre\}, \{Confirm\}, \{Abort\}, \{End\})$
$axm3 : partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\})$
$axm4 : partition(A\_STATES, \{s0\_Seller\}, \{s1\_Seller\}, \ldots)$
$axm5 : CPs\_B = \{s0 \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s1 \mapsto 2,$
$s2 \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s3 \mapsto 3,$
$s1 \mapsto Seller \mapsto Offre \mapsto Bank \mapsto s0 \mapsto 4,$
$s2 \mapsto Buyer \mapsto Pay \mapsto Bank \mapsto s3 \mapsto 6,$
$s3 \mapsto Bank \mapsto Abort \mapsto Buyer \mapsto s4 \mapsto 7,$
$s3 \mapsto Bank \mapsto Confirm \mapsto Seller \mapsto s5 \mapsto 8,$
$\ldots\}$
$axm6 : Repare\_propositions = \{s0 \mapsto Seller \mapsto Sync0 \mapsto Buyer \mapsto s1 \mapsto 1,$
$s0 \mapsto Buyer \mapsto Sync0 \mapsto Seller \mapsto s1 \mapsto 1,$
$s1 \mapsto Seller \mapsto Sync1 \mapsto Buyer \mapsto s3 \mapsto 3,$
$s1 \mapsto Bank \mapsto Sync1 \mapsto Buyer \mapsto s3 \mapsto 3,$
$\}$
End

Listing 8.7 – Acces a une application web

2.3 CS10. Réparation de CS7

Un scénario de réparation possible est représenté dans la Figure 8.10. Le *CP* avant réparation décrit dans la Figure 8.7 est caractérisé comme non-réalisable, une

transition de synchronisation entre les entité A et C est ajouté à l'état initiale du *CP* afin de rétablir la réalisabilité du système.

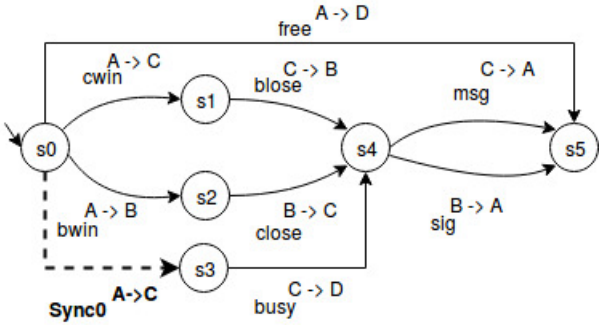


FIGURE *8.10 – Accès à une application web, CP réparé*

Le listing 8.8 illustre les scénarios de réparation possibles du protocole de jeu fictif non-réalisable.

Context
Constants
Axioms
$axm1$:$partition(PEERS, \{A\}, \{B\}, \{C\}, \{D\}, \{Pend\})$
$axm2$:$partition(MESSAGES, \{bwin\}, \{cwin\}, \{sig\}, \{msg\}, \{blose\}, \{close\}, \{busy\}, \{free\}, \{End\})$
$axm3$:$partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\})$
$axm4$: $partition(A\_STATES, \{s0\_A\}, \{s1\_A\}, \dots)$
$axm5$: $CPs\_B = \{s0 \mapsto A \mapsto cwin \mapsto C \mapsto s1 \mapsto 2,$
$s0 \mapsto A \mapsto bwin \mapsto B \mapsto s2 \mapsto 3,$
$s1 \mapsto C \mapsto blose \mapsto B \mapsto s4 \mapsto 4,$
$s2 \mapsto B \mapsto close \mapsto C \mapsto s4 \mapsto 5,$
$s4 \mapsto C \mapsto msg \mapsto A \mapsto s5 \mapsto 6,$
$s4 \mapsto B \mapsto sig \mapsto A \mapsto s5 \mapsto 7,$
$s3 \mapsto C \mapsto busy \mapsto D \mapsto s4 \mapsto 8,$
$s0 \mapsto A \mapsto free \mapsto D \mapsto s5 \mapsto 9,$
$\dots\}$
$axm6$: $Repare\_propositions = \{s0 \mapsto A \mapsto Sync0 \mapsto C \mapsto s1 \mapsto 1,$
$\}$
End

Listing 8.8 – Acces a une application web

2.4 CS11. Réparation de CS2

La Figure 8.11 décrit la réparation du *CP* du protocole de transfert de fichier décrit à la Figure 7.7 et étudié à la Section 1.

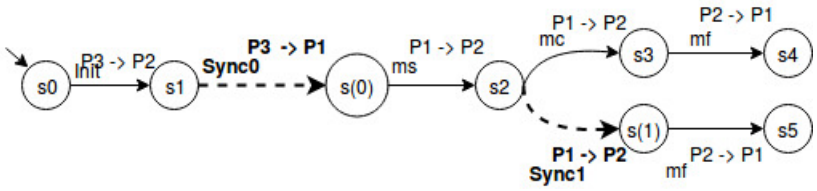


FIGURE *8.11 – CP réparé du protocole de transfert de fichier*

3 Rapport technique des travaux connexes

La majorité des techniques de vérification et validation de la réalisabilité dans la littérature sont basé sur le processus de model-checking, ce qui présente des limites à savoir, le passage à l'échelle et le problème d'explosion combinatoire des états du système a vérifié. Ce qui empêche le teste des système complexe c.à.d des systèmes avec nombre et complexité des entités très élevé.

Nous avons bien pris en considération dans nos contributions les deux problèmes majeurs cités au auparavant, tel que notre approche est guidée par les preuves mathématiques ainsi que celle de l'outil Rodin B-événementiel. Notre construction des CPs est correcte-par-construction. Nous arrivons à la conversation asynchrone bien formée par construction en suivant les étapes de raffinement qui garantissent la préservation du même comportement du *CP* dans les différents niveaux d'abstraction, tout en évitant le passage par la construction des systèmes synchrone et asynchrone comme dans [10] ce qui permet de réduire la complexité de la vérification et donc de résoudre le problème de l'explosion combinatoire.
En résumé l'approche proposée supporte le passage a l'échelle ainsi que les modèles complexe, et la démarche correcte-par-construction permet d'evité l'explosion combinatoire.

Plus généralement, nous discutons ici nos expériences réalisées sur des exemples des travaux connexes.

Pour valider notre approche et montrer l'efficacité et l'exactitude de notre modèle B-événementiel, nous avons choisi d'utiliser des points de repère issus de la littérature. La Table 8.1 résume les résultats des différentes études de cas présentées ci-dessus. Il donne une évaluation quantitative de différents critères. Pour chaque étude de cas, cette table indique le nombre d'états, des entités communicantes, de messages échangés et de transitions d'échanges. Il indique si le *CP* est réalisable ou non-réalisable et donne les détails des opérateurs de composition utilisés pour construire le *CP*. Il montre également quelles propriétés sont violées en cas de *CP* non réalisable. Toutes les études de cas incluent des séquences, des choix, des boucles. Nous avons également abordé le cas d'un opérateur parallèle en interprétant cet composition comme un entrelacement utilisant les opérateurs de séquence et de choix.

Tel que :
— $\gg$: décrire l'opérateur de séquence.
— $+$: décrire l'opérateur de choix.
— $\circlearrowleft$: décrire l'opérateur de boucle.
— $-$: signifie qu'il n'y a pas de propriétés violées. *ç.a.d.* le *CP* est caractérisé comme réalisable.

Pour donner une idée du nombre d'états construits pendant l'instanciation à l'aide de l'animateur ProB, considérons l'étude de cas *CS1. Processus d'entreprise virtuelle* de la Figure. 8.1. Comme indiqué dans le tableau 8.1, pour un CP composé de 10 états et de 10 transitions, jusqu'à 298 états et 322 transitions sont générés par ProB sur la plate-forme Rodin. Ils sont utilisés pour vérifier l'exactitude de l'instanciation (témoins).

Le modèle B-événementiel complet et toutes les instanciations B-événementiel correspondant aux études de cas présentées sont disponibles en ligne à l'adresse suivante : https ://www.irit.fr/∼Sarah.Benyagoub/models.php

Références	États	Entités	Messages	Transitions	Réalisable/ Non-Réalisable	Opérateurs	Propriétés violé	Automatique vérification	Total des États	Total des Transitions
CS1. Processus d'entreprise virtuelle [84]	10	3	9	10	Réalisable	$\gg, \circlearrowleft$	-	190(63%)	298	322
CS2. Protocole de transfert de fichier [13] web application [51]	5	2	3	4	Non-réalisable	$\gg, +$	ISeqF, PCF	90(63%)	98	32
CS3. Accès application web [16]	2	3	3	3	Réalisable	$\gg, +, \circlearrowleft$	-	9 (34%)	24	23
CS4. Accés à une base de données [51]	4	4	5	5	Non-réalisable	$\gg, +, \circlearrowleft$	ISeqF	7(28%)	22	21
CS5. Application d'achat en ligne [80]	7	3	6	7	Non-réalisable	$\gg, +, \circlearrowleft$	ISeqF, PCF	8(30%)	24	23
CS6. Protocole de visa Australien [87]	20	2	21	21	Réalisable	$\gg, +, \circlearrowleft$	-	90(52%)	29	32
CS7. Protocole d'un jeu fictif [66]	5	4	8	8	Non-réalizable	$\gg, +, \circlearrowleft$	PCF	6(62%)	9	8
CS8. Accès à un bdd réparé [51]	6	4	7	7	Réalisable	$\gg, +, \circlearrowleft$	-	110(79%)	108	39
CS9. Application réparé d'achat en ligne [80]	9	3	9	9	Réalisable	$\gg, +, \circlearrowleft$	-	18(48%)	38	29
CS10.Protocole réparé d'un jeu fictif [66]	6	4	9	9	Réalisable	$\gg, +, \circlearrowleft$	-	10(72%)	306	341
CS11. Protocole réparé de transfert [13]	6	4	7	7	Réalisable	$\gg, +, \circlearrowleft$	-	110(79%)	108	39

TABLE *8.1 – Rapport technique des travaux connexes*

Troisième partie

Conclusion

Chapitre 9

Conclusion et perspective

L'objectif fondamental de cette thèse est la formalisation et la validation de la composition des systèmes distribués. Cette problématique a été abordé au niveau protocole de conversation. La modélisation des protocoles de conversation nous a permis de définir et analyser le comportement des entités communicantes formant le système.

L'état de l'art étudié nous a aidé à classer les algorithmes d'analyse de réalisabilité proposés.

Nous avons distingué trois catégories principales qui sont : Les approche de vérification et validation de la réalisabilité, les approches d'évolution des *CPs* et les approches correcte-par-construction.

Ce classement a été effectuer selon les limites des propositions face au problème d'explosion combinatoire vu la complexité élevé des systèmes actuels.

Afin de minimiser les problèmes majeurs des approches de validation de la réalisabilité des systèmes. nous avons proposé un ensemble de conditions nécessaires pour la préservation de la réalisation des CPs tout en évitant le passage par les étapes de recomposition des CPs en mode de communication synchrone et asynchrone. L'élimination de ces étapes réduit la complexité de l'approche proposée ce qui marque la différence par rapport aux proposition des travaux connexes.

Notre approche permet la construction des *CPs* réalisable. La méthode de composition est correcte par construction, elle utilise un ensemble d'opérateurs de composition en séquence, en choix, en boucle/cycle et en parallèle qui respectent un ensemble de conditions prédéfini à savoir, la propriétés d'absence des séquences indépendantes et la propriété d'absence des choix parallèle, ce qui garanti la préservation de la réalisation des CPs résultats. Sachant que la construction d'un *CP* donné est incrémentale.

L'approche de composition a été complètement formalisée dans un outil de développement formel qui est B-événementiel platforme Rodin afin de montrer que chaque opérateur de composition qui prend en entrée un *CP* réalisable produit en sortie un *CP* qui satisfait la même propriété, c.à.d, chacun des opérateurs de composition garanti la préservation de la réalisabilité.

Afin d'atteindre cet objectif nous avons mis en place un développement par étapes en utilisant la stratégie de raffinement afin d'introduire les différents conditions de la réalisabilité à savoir, l'équivalence, la synchronisabilité et la WF comme des invariants de notre modèle B-événementiel, sachant que le raffinement préserve les invariants prouvés.

Pour se faire nous avons définit un langage permettant la création incrémentale des *CPs* réalisables simple ou complexe à partir d'un ensemble de transitions basiques. De plus, nous avons montré que le développement proposé est générique et peut être instancié sur n'importe quel ensemble de *CPs* composites et basiques.

La solution donnée dans cette thèse est optimale et présente plusieurs avantage par rapport aux propositions du domaine tel que. Notre approche supporte le passage a l'échelle.

C'est une solution formelle basée sur la stratégie de raffinement et les preuves, ce qui offre une puissance de définition des opérateurs de composition ainsi qu'au propriétés de préservation de la réalisabilité vérifié par les opérateurs de séquence, de branche, de boucle/cycle et du parallèle. Le raffinement nous a permis aussi de casser les preuves sur les différents niveau d'abstraction (le niveau CP, le niveau synchrone et le niveau asynchrone), ce qui facilite la formalisation et la preuve du modèle.

Proposition d'une approche (plus l'implémentation de son modèle) qui suggère des corrections pour les *CPs* non réalisable en se basant sur nos conditions nécessaires définis pour la préservation de la propriété de réalisabilité.

L'idée est d'introduire des transitions de synchronisation au cœur du *CP* non réalisable jusqu'à' la satisfaction des conditions de réalisation du *CP*. Nous avons aussi formalisé et prouvé l'exactitude de la stratégie de réparation en B-événementiel. L'approche a été validé par plusieurs étude de cas utilisé par les travaux connexes.

Ce travail a plusieurs perspectives. À titre de perspective à court terme :

— Étendre le modèle par introduction de l'opérateur de composition parallèle. Ce qui n'est pas supporté dans le modèle actuel.
— Nous étendons notre modèle avec de nouveaux opérateurs qui nous permet de composer des CPs entiers au lieu d'exiger une composition incrémentale en utilisant le CP_b basique.
— Nous souhaitons aussi définir l'ensemble des modèles pour les CPs réalisables en étudiant l'exhaustivité de notre langage afin d'identifier la classe des systèmes de communication asynchrones du monde réel que nous pouvons spécifier.
— Enfin, nous visons à fournir aux concepteurs un moteur pour l'instanciation automatique des CPs réalisables.

À titre de perspective à long terme :

— Nous prévoyons de gérer d'autres systèmes asynchrones car nous avons utilisé un modèle causale dans lequel l'envoi et la réception sont dans le même ordre.
— Amélioration de l'approche proposé par la définition des conditions néces-

saires et suffisantes pour la composition des *CPs* réalisables.

Bibliographie

[1] Jean-Raymond Abrial et Jean-Raymond Abrial : *The B-book : assigning programs to meanings*. Cambridge University Press, 2005.

[2] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta et Laurent Voisin : Rodin : an open toolset for modelling and reasoning in event-b. *International journal on software tools for technology transfer*, 12(6):447–466, 2010.

[3] Jean-Raymond Abrial et Stefan Hallerstede : Refinement, decomposition, and instantiation of discrete models : Application to event-b. *Fundamenta Informaticae*, 77(1-2):1–28, 2007.

[4] JR Abrial : Modeling in event-b : System and softwareengineer, 2010.

[5] Lakhdar Akroun et Gwen Salaün : Automated verification of automata communicating via fifo and bag buffers. *Formal Methods in System Design*, 52(3):260–276, 2018.

[6] Lakhdar Akroun, Gwen Salaün et Lina Ye : Automated analysis of asynchronously communicating systems. *In International Symposium on Model Checking Software*, pages 1–18. Springer, 2016.

[7] Marco Autili, Paola Inverardi et Massimo Tivoli : Choreography realizability enforcement through the automatic synthesis of distributed coordination delegates. *Science of Computer Programming*, 160:3–29, 2018.

[8] Marco Autili et Massimo Tivoli : Distributed enforcement of service choreographies. *arXiv preprint arXiv :1502.03512*, 2015.

[9] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy *et al.* : *The Coq proof assistant reference manual : Version 6.1*. Thèse de doctorat, Inria, 1997.

[10] S. Basu, T. Bultan et M. Ouederni : Deciding Choreography Realizability. *In Proc. of POPL'12*, pages 191–202. ACM, 2012.

[11] Samik Basu et Tevfik Bultan : Choreography conformance via synchronizability. *In Proceedings of the 20th international conference on World wide web*, pages 795–804. ACM, 2011.

[12] Samik Basu et Tevfik Bultan : Automated choreography repair. *In Fundamental Approaches to Software Engineering - 19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and*

Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings, pages 13–30, 2016.

[13] Samik Basu et Tevfik Bultan : Automated choreography repair. *In* Perdita Stevens et Andrzej Wasowski, éditeurs : *Fundamental Approaches to Software Engineering - 19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9633 de *Lecture Notes in Computer Science*, pages 13–30. Springer, 2016.

[14] Samik Basu et Tevfik Bultan : On deciding synchronizability for asynchronously communicating systems. *Theoretical Computer Science*, 656:60–75, 2016.

[15] Samik Basu, Tevfik Bultan et Meriem Ouederni : Synchronizability for verification of asynchronously communicating systems. *In International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 56–71. Springer, 2012.

[16] S. Benyagoub, M. Ouederni et Y. Aït Ameur : Towards correct Evolution of Conversation Protocols. *In Proc. of VECOS'16.*, volume 1689 de *CEUR Workshop Proceedings*, pages 193–201. CEUR-WS.org, 2016.

[17] S. Benyagoub, M. Ouederni, N.K. Singh et Y. Aït Ameur : Correct-by-Construction Evolution of Realisable Conversation Protocols. *In Proc. of MEDI'16* , volume 9893 de *LNCS*, pages 260–273. Springer, 2016.

[18] Sarah Benyagoub, Meriem Ouederni, Yamine Aït-Ameur et Atif Mashkoor : Handling Reparation in Incremental Construction of Realizable Conversation Protocols. New Trends in Model and Data Engineering - MEDI International Workshops, DETECT, MEDI4SG, IWCFS, REMEDY, pages 159–166, 2018.

[19] Sarah Benyagoub, Meriem Ouederni, Yamine Aït-Ameur et Atif Mashkoor : Incremental construction of realizable choreographies. *In NASA Formal Methods Symposium*, pages 1–19. Springer, 2018.

[20] Sarah Benyagoub, Meriem Ouederni, Yamine Aït-Ameur et Atif Mashkoor : Scalable Correct-by-Construction Conversation Protocols with Event-B : Validation, Experiments and Benchmarks. 23rd International Conference on Engineering of Complex Computer Systems (ICECCS), pages 209–212, 2018.

[21] Bernard Berthomieu et Francois Vernadat : Time petri nets analysis with tina. *In null*, pages 123–124. IEEE, 2006.

[22] Yves Bertot et Pierre Castéran : *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[23] Dines Bjørner et Cliff B Jones : He vienna development method : The meta-language. *Lecture Notes in Computer Science 61*, 1978.

[24] Laura Bocchi, Nobuko Yoshida et Julien Lange : Meeting deadlines together. 2015.

[25] Ahmed Bouajjani, Constantin Enea, Kailiang Ji et Shaz Qadeer : On the completeness of verifying message passing programs under bounded asynchrony. *In International Conference on Computer Aided Verification*, pages 372–391. Springer, 2018.

[26] D. Brand et P. Zafiropulo : On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, 1983.

[27] Daniel Brand et Pitro Zafiropulo : On communicating finite-state machines. *Journal of the ACM (JACM)*, 30(2):323–342, 1983.

[28] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill et Lain-Jinn Hwang : Symbolic model checking : 1020 states and beyond. *Information and computation*, 98(2):142–170, 1992.

[29] C. Tinelli C. Barrett : *CVC3*. International Conference on Computer Aided Verification (CAV '07), Berlin, Germany, 19th édition, 2007.

[30] Carlos Canal, Ernesto Pimentel et José M Troya : Compatibility and inheritance in software architectures. *Science of Computer Programming*, 41(2): 105–138, 2001.

[31] Pierre Chambart et Ph Schnoebelen : Mixing lossy and perfect fifo channels. *In International Conference on Concurrency Theory*, pages 340–355. Springer, 2008.

[32] Bernadette Charron-Bost, Friedemann Mattern et Gerard Tel : Synchronous, asynchronous, and causally ordered communication. *Distributed Computing*, 9(4):173–191, 1996.

[33] E. Clarke, O. Grumberg, S. Jha, Y. Lu et H. Veith : Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50, 2003.

[34] Lorenzo Clemente, Frédéric Herbreteau et Grégoire Sutre : Decidable topologies for communicating automata with fifo and bag channels. *In International Conference on Concurrency Theory*, pages 281–296. Springer, 2014.

[35] Luca De Alfaro et Thomas A Henzinger : Interface automata. *In ACM SIGSOFT Software Engineering Notes*, volume 26, pages 109–120. ACM, 2001.

[36] L. M. de Moura et N. Bjorner : *Z3 : An Efficient SMT Solver*. Lecture Notes in Computer Science, 2008.

[37] Pierre-Malo Deniélou et Nobuko Yoshida : Multiparty session types meet communicating automata. *In European Symposium on Programming*, pages 194–213. Springer, 2012.

[38] Pierre-Malo Deniélou et Nobuko Yoshida : Multiparty compatibility in communicating automata : Characterisation and synthesis of global session types. *In International Colloquium on Automata, Languages, and Programming*, pages 174–186. Springer, 2013.

[39] Edsger Wybe Dijkstra, Edsger Wybe Dijkstra, Edsger Wybe Dijkstra, Etats-Unis Informaticien et Edsger Wybe Dijkstra : *A discipline of programming*, volume 1. prentice-hall Englewood Cliffs, 1976.

[40] P. Van Eijk et Michel Diaz, éditeurs. *Formal Description Technique Lotos : Results of the Esprit Sedos Project*. Elsevier Science Inc., New York, NY, USA, 1989.

[41] E. Contejean F. Bobot, S. Conchon et S. Lescuyer : *Implementing Polymorphism in SMT solvers*. International Workshop on Satisfiability Modulo, 6th édition, 2008.

[42] Z. Farah, Y. Ait-Ameur, M. Ouederni et K. Tari : A Correct-by-Construction Model for Asynchronously Communicating Systems. *International Journal on Software Tools for Technology Transfer*, pages 1–21, 2016.

[43] Z. Farah, Y. Ait-Ameur, M. Ouederni et K. Tari : A Correct-by-Construction Model for Asynchronously Communicating Systems. *International Journal on Software Tools for Technology Transfer*, pages 1–21, 2016.

[44] Walid Fdhila, Conrad Indiono, Stefanie Rinderle-Ma et Manfred Reichert : Dealing with change in process choreographies : Design and implementation of propagation algorithms. *Information systems*, 49:1–24, 2015.

[45] Alain Finkel et Etienne Lozes : Synchronizability of communicating finite state machines is not decidable. *arXiv preprint arXiv :1702.07213*, 2017.

[46] X. Fu, T. Bultan et J. Su : Conversation Protocols : A Formalism for Specification and Verification of Reactive Electronic Services. *Theoretical Computer Science*, 328(1-2):19–37, 2004.

[47] Xiang Fu, Tevfik Bultan et Jianwen Su : Synchronizability of conversations among web services. *IEEE Transactions on Software Engineering*, 31(12):1042–1055, 2005.

[48] Hania Gadouche, Zoubeyr Farah et Abdelkamel Tari : A correct-by-construction model for attribute-based access control. *In International Conference on Model and Data Engineering*, pages 233–247. Springer, 2018.

[49] Hubert Garavel, Frédéric Lang, Radu Mateescu et Wendelin Serwe : Cadp 2011 : a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, 15(2):89–107, 2013.

[50] Blaise Genest, Dietrich Kuske et Anca Muscholl : A kleene theorem and model checking algorithms for existentially bounded communicating automata. *Information and Computation*, 204(6):920–956, 2006.

[51] M. Güdemann, G. Salaün et M. Ouederni : Counterexample Guided Synthesis of Monitors for Realizability Enforcement. *In Proc. of ATVA'12*, volume 7561 de *LNCS*, pages 238–253. Springer, 2012.

[52] Matthias Güdemann, Pascal Poizat, Gwen Salaün et Alexandre Dumont : Verchor : A framework for verifying choreographies. *In International Conference on Fundamental Approaches to Software Engineering*, pages 226–230. Springer, 2013.

[53] David Harel : Statecharts : A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[54] Alexander Heussner, Tristan Le Gall et Grégoire Sutre : Mcscm : a general framework for the verification of communicating machines. *In International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 478–484. Springer, 2012.

[55] Alexander Heussner, Jérôme Leroux, Anca Muscholl et Grégoire Sutre : Reachability analysis of communicating pushdown systems. *In International Conference on Foundations of Software Science and Computational Structures*, pages 267–281. Springer, 2010.

[56] Thai Son HOANG et Jean-Raymond ABRIAL : Reasoning about liveness properties in event-b. *In International Conference on Formal Engineering Methods*, pages 456–471. Springer, 2011.

[57] Charles Antony Richard HOARE : An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[58] Gerard J HOLZMANN : *The SPIN model checker : Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

[59] Kohei HONDA, Vasco T VASCONCELOS et Makoto KUBO : Language primitives and type discipline for structured communication-based programming. *In European Symposium on Programming*, pages 122–138. Springer, 1998.

[60] J. E. HOPCROFT et J. D. ULLMAN : *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[61] Ivan J JURETA, Stéphane FAULKNER et Philippe THIRAN : *Dynamic Requirements Specification for adaptable and open Service-oriented Systems*. Springer, 2007.

[62] Pistore M. KAZHAMIAKIN, R. : Analysis of realizability conditions for web service choreographies. volume 4052 de *LNCS*, page 61–76. Springer, 2003.

[63] Ajay D KSHEMKALYANI et Mukesh SINGHAL : *Distributed computing : principles, algorithms, and systems.* Cambridge University Press, 2011.

[64] Salvatore LA TORRE, Parthasarathy MADHUSUDAN et Gennaro PARLATO : Context-bounded analysis of concurrent queue systems. *In International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 299–314. Springer, 2008.

[65] Leslie LAMPORT : *Specifying systems : the TLA+ language and tools for hardware and software engineers.* Addison-Wesley Longman Publishing Co., Inc., 2002.

[66] Julien LANGE, Emilio TUOSTO et Nobuko YOSHIDA : From Communicating Machines to Graphical Choreographies. Proceedings of the 42nd Annual ACM POPL, pages 221–232, 2015.

[67] Julien LANGE et Nobuko YOSHIDA : Verifying asynchronous interactions via communicating session automata. *arXiv preprint arXiv :1901.09606*, 2019.

[68] Leonardo AF LEITE, Gustavo Ansaldi OLIVA, Guilherme M NOGUEIRA, Marco Aurélio GEROSA, Fabio KON et Dejan S MILOJICIC : A Systematic Literature Review of Service Choreography Adaptation. *Service Oriented Computing and Applications*, 7(3):199–216, 2013.

[69] Michael LEUSCHEL et Michael BUTLER : Prob : A model checker for b. *In International Symposium of Formal Methods Europe*, pages 855–874. Springer, 2003.

[70] Michael LEUSCHEL et Michael J. BUTLER : Prob : A model checker for B. *In* Keijiro ARAKI, Stefania GNESI et Dino MANDRIOLI, éditeurs : *FME 2003 : Formal Methods, International Symposium of Formal Methods Europe, Pisa, Italy, September 8-14, 2003, Proceedings*, volume 2805 de *Lecture Notes in Computer Science*, pages 855–874. Springer, 2003.

[71] N. Lohmann et K. Wolf : Realizability Is Controllability. *In Proc. of WS-FM'09*, volume 6194 de *LNCS*. Springer, 2010.

[72] ISO Lotos : A formal description technique based on the temporal ordering of observational behaviour. *ISO8807, 1XS989*, 1989.

[73] Rajit Manohar et Alain J. Martin : Slack elasticity in concurrent computing. *In Mathematics of Program Construction, (MPC)*, pages 272–285, 1998.

[74] Nenad Medvidovic : ADLs and dynamic Architecture Changes. *In Proc. of SIGSOFT'96 workshops*, pages 24–27. ACM, 1996.

[75] Robin Milner : *A Calculus of Communicating Systems*, volume 92 de *Lecture Notes in Computer Science*. Springer, 1980.

[76] Tobias Nipkow, Lawrence C Paulson et Markus Wenzel : *Isabelle/HOL : a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[77] Sam Owre, John M Rushby et Natarajan Shankar : Pvs : A prototype verification system. *In International Conference on Automated Deduction*, pages 748–752. Springer, 1992.

[78] C. Peltz : Web services orchestration and choreography. volume 4052 de *LNCS*, page 46–52. Springer, 2003.

[79] Daniel Plagge et Michael Leuschel : Seven at one stroke : Ltl model checking for high-level specifications in b, z, csp, and more. *International journal on software tools for technology transfer*, 12(1):9–21, 2010.

[80] Mila Dalla Preda, Maurizio Gabbrielli, Saverio Giallorenzo, Ivan Lanese et Jacopo Mauro : Dynamic Choreographies - Safe Runtime Updates of Distributed Applications. *In Proc. of COORDINATION'15*, volume 9037 de *LNCS*, pages 67–82. Springer, 2015.

[81] D. Delahaye R. Bonichon et D. Doligez. Zenon : *An Extensible Automated Theorem Prover Producing Checkable Proofs*. In Dershowitz and Voronkov.

[82] A. Riazanov et A. Voronkov : *The design and implementation of Vampire*. Journal of AI Communicationse, 2002.

[83] S. Rinderle, A. Wombacher et M. Reichert : On the Controlled Evolution of Process Choreographies. *In Proc. of ICDE'06*, pages 124–124. IEEE, 2006.

[84] S. Rinderle, A. Wombacher et M. Reichert : On the Controlled Evolution of Process Choreographies. Proc. of ICDE, pages 100–124, 2006.

[85] Stefanie Rinderle, Andreas Wombacher et Manfred Reichert : Evolution of process choreographies in DYCHOR. *In Proc. of CoopIS'06*, volume 4275 de *LNCS*, pages 273–290. Springer, 2006.

[86] N. Roohi et G. Salaün : Realizability and Dynamic Reconfiguration of Chor Specifications. *Informatica (Slovenia)*, 35(1):39–49, 2011.

[87] S. H. Ryu, F. Casati, H. Skogsrud, B. Benatallah et R. Saint-Paul : Supporting the Dynamic Evolution of Web Service Protocols in Service-oriented Architectures. *ACM Transactions on the Web (TWEB)*, 2(2):13, 2008.

[88] Alceste Scalas et Nobuko Yoshida : Less is more : multiparty session types revisited. *Proceedings of the ACM on Programming Languages*, 3(POPL):30, 2019.

[89] Stephen F Siegel : Efficient verification of halting properties for mpi programs with wildcard receives. *In International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 413–429. Springer, 2005.

[90] Oleg Sokolsky, Sampath Kannan et Insup Lee : Simulation-based graph similarity. *In International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 426–440. Springer, 2006.

[91] J Michael Spivey et JR Abrial : *The Z notation*. Prentice Hall Hemel Hempstead, 1992.

[92] Sarvani Vakkalanka, Anh Vo, Ganesh Gopalakrishnan et Robert M Kirby : Precise dynamic analysis for slack elasticity : Adding buffering without adding bugs. *In European MPI Users' Group Meeting*, pages 152–159. Springer, 2010.

[93] RESULT OF VOTING : Information technology–z formal specification notation–syntax, type system and semantics. 2002.

[94] Andreas Wombacher : Alignment of Choreography Changes in BPEL Processes. *In Proc. of SCC'09*, pages 1–8. IEEE, 2009.

[95] Zhaohui Wu, Shuiguang Deng, Ying Li et Jian Wu : Computing compatibility in dynamic service composition. *Knowledge and information systems*, 19(1): 107–129, 2009.

[96] Daniel M Yellin et Robert E Strom : Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2):292–333, 1997.

Quatrième partie

Annexes

Annexe A

Modèle B-événementiel de composition

Modéle abstrait

CONTEXT LTS_CONTEXT

SETS

PEERS

MESSAGES

CP_STATES

CONSTANTS

CPs_B Set of CP basic transitions

PEER_SOURCE Function returning source peer

INDEX Fuction returning the transition index

SOURCE_STATE Fuction returning the transition source state

DESTINATION_STATE Fuction returning the transition destination state

MESSAGE Fuction returning the transition message

LABEL Fuction returning the transition label

PEER_DESTINATION Fuction returning the transition destination peer

NDC Non desterminist property

DC Determinist property

BR_CP_FINAL_STATES Fuction returning Branches final states

ISeqF Independent sequence freedom property

PCF Parallel choice property

Initial_state_value Value of initial state

CP_Final_states_value Value of the set of final states

End_message value of the termination message

LAST_SENDER_RECEIVER_PEERS Fuction returning the last sender and receiver peers

Prophecy_value Value of prophecy variable

AXIOMS

axm1_CP: $CPs\_B \subseteq CP\_STATES \times PEERS \times MESSAGES \times PEERS \times CP\_STATES \times \mathbb{N}$

axm1: $finite(CPs\_B)$

axm2: $finite(PEERS) \land card(PEERS) \geq 2$

axm3: $finite(MESSAGES) \land card(MESSAGES) \geq 1$

axm4: $Initial\_state\_value \in CP\_STATES$

axm5: $CP\_Final\_states\_value \subseteq CP\_STATES$

axm6: $End\_message \in MESSAGES$

axm7: $Prophecy\_value \in \mathbb{N}$

INDEX: $INDEX \in CPs\_B \leftrightarrow \mathbb{N}$

FCT_INDEX: $INDEX = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto idx \mid state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in CPs\_B \land idx \in \mathbb{N} \land idx = Index\}$

SOURCE_STATE: $SOURCE\_STATE \in CPs\_B \leftrightarrow CP\_STATES$

FCT_SOURCE_STATE: $SOURCE\_STATE = \{st\_so \mapsto Peer\_so \mapsto msg \mapsto Peer\_de \mapsto st\_de \mapsto Index \mapsto state\_src \mid st\_so \mapsto Peer\_so \mapsto msg \mapsto Peer\_de \mapsto st\_de \mapsto Index \in CPs\_B \land state\_src \in CP\_STATES \land state\_src = st\_so\}$

DESTINATION_STATE: $DESTINATION\_STATE \in CPs\_B \leftrightarrow CP\_STATES$

FCT_DESTINATION_STATE: $DESTINATION\_STATE = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto state\_dest \mid state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in CPs\_B \land state\_dest \in CP\_STATES \land state\_dest = state\_dst\}$

PEER_SOURCE: $PEER\_SOURCE \in CPs\_B \leftrightarrow PEERS$

FCT_PEER_SOURCE: $PEER\_SOURCE = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto peer\_source \mid state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in CPs\_B \land peer\_source \in PEERS \land peer\_source = Peer\_src\}$

PEER_DESTINATION: $PEER\_DESTINATION \in CPs\_B \leftrightarrow PEERS$

FCT_PEER_DESTINATION: $PEER\_DESTINATION = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto peer\_destination | state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in CPs\_B \land peer\_destination \in PEERS \land peer\_destination = Peer\_dst\}$

MESSAGE: $MESSAGE \in CPs\_B \leftrightarrow MESSAGES$

FCT_MESSAGE: $MESSAGE = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto message | state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in CPs\_B \land message \in MESSAGES \land message = msg\}$

LABEL: $LABEL \in CPs\_B \leftrightarrow PEERS \times MESSAGES \times PEERS$

FCT_LABEL: $LABEL = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto Label | state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in CPs\_B \land Label \in PEERS \times MESSAGES \times PEERS \land Label = Peer\_src \mapsto msg \mapsto Peer\_dst\}$

NDC: $NDC \subseteq CPs\_B$

FCT_NDC: $\forall Trans1, Trans2 \cdot (Trans1 \in CPs\_B \land Trans2 \in CPs\_B \land SOURCE\_STATE(Trans1) = SOURCE\_STATE(Trans2) \land LABEL(Trans1) = LABEL(Trans2) \land DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2)) \Rightarrow \{Trans1, Trans2\} \subseteq NDC$

NDC_Largest-set: $\forall Trans1 \cdot (Trans1 \in NDC) \Rightarrow (\exists Trans2 \cdot (Trans2 \in NDC \land SOURCE\_STATE(Trans1) = SOURCE\_STATE(Trans2) \land LABEL(Trans1) = LABEL(Trans2) \land DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2)))$

DETERMINIST_CP: $DC = CPs\_B \setminus NDC$

BR_CP_FINAL_STATES: $BR\_CP\_FINAL\_STATES \in CP\_STATES \rightarrow \mathbb{P}(CP\_STATES)$

FCT_BR_CP_FINAL_STATES: $\forall Trans1, Trans2 \cdot (Trans1 \in CPs\_B \land Trans2 \in CPs\_B \land SOURCE\_STATE(Trans1) = SOURCE\_STATE(Trans2) \land DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2)) \Rightarrow \{DESTINATION\_STATE(Trans1), DESTINATION\_STATE(Trans2)\} \subseteq BR\_CP\_FINAL\_STATES(SOURCE\_STATE(Trans1))$

LAST_SENDER_RECEIVER_PEERS: $LAST\_SENDER\_RECEIVER\_PEERS \in CP\_STATES \rightarrow \mathbb{P}(PEERS)$

FCT_LAST_SENDER_RECEIVER_PEERS: $\forall state, Trans1, Trans2 \cdot (state \in CP\_STATES \land Trans1 \in CPs\_B \land Trans2 \in CPs\_B \land state = SOURCE\_STATE(Trans2) \land state = DESTINATION\_STATE(Trans1) \land DESTINATION\_STATE(Trans1) = SOURCE\_STATE(Trans2)) \Rightarrow \{PEER\_SOURCE(Trans1), PEER\_DESTINATION(Trans1)\} \subseteq LAST\_SENDER\_RECEIVER\_PEERS(state)$

ISeqF: $ISeqF \subseteq CPs\_B$

FCT_ISeqF: $\forall cp\_b \cdot (cp\_b \in CPs\_B \land PEER\_SOURCE(cp\_b) \in LAST\_SENDER\_RECEIVER\_PEERS(SOURCE\_STATE(cp\_b))) \Rightarrow cp\_b \in ISeqF$

ISeqF_Largest-set: $\forall cp\_b \cdot (cp\_b \in ISeqF) \Rightarrow PEER\_SOURCE(cp\_b) \in LAST\_SENDER\_RECEIVER\_PEERS(SOURCE\_STATE(cp\_b))$

PCF_BRANCHES_SET: $PCF \in CP\_STATES \rightarrow \mathbb{P}(CPs\_B)$

FCT_PCF_BRANCHES_SET: $\forall Trans1, Trans2 \cdot (Trans1 \in CPs\_B \wedge Trans2 \in CPs\_B \wedge SOURCE\_STATE(Trans1) = SOURCE\_STATE(Trans2) \wedge PEER\_SOURCE(Trans1) = PEER\_SOURCE(Trans2) \wedge DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2)) \Rightarrow \{Trans1, Trans2\} \subseteq PCF(SOURCE\_STATE(Trans1))$

PCF_Largest-set: $\forall Trans1, Trans2, s \cdot (s \in CP\_STATES \wedge Trans1 \in PCF(s) \wedge Trans2 \in PCF(s) \wedge Trans1 \neq Trans2 \wedge s = SOURCE\_STATE(Trans1) \wedge s = SOURCE\_STATE(Trans2)) \Rightarrow PEER\_SOURCE(Trans1) = PEER\_SOURCE(Trans2) \wedge DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2)$

END

MACHINE LTS_model
SEES LTS_CONTEXT
VARIABLES

BUILT_CP Built Conversation Protocol which is empty at the initialisation
CP_Initial_state CP Initial state
CP_Final_states The set of CP final states
end Variable that manages events
Number_of_send Number of sent messages
Prophecy_of_Sent_Messages Prophecy of sent messages

INVARIANTS

inv1: $BUILT\_CP \subseteq DC$
inv2: $finite(BUILT\_CP)$
inv3: $CP\_Initial\_state \in CP\_STATES$
inv4: $CP\_Final\_states \subseteq CP\_STATES$
inv7: $end \in \{0, 1\}$
inv8: $Number\_of\_send \in \mathbb{N}$
inv9: $Prophecy\_of\_Sent\_Messages \in \mathbb{N}$

VARIANT

$Prophecy\_of\_Sent\_Messages - Number\_of\_send$

EVENTS

Initialisation

begin

act1: $BUILT\_CP := \varnothing$
act2: $CP\_Initial\_state := Initial\_state\_value$
act3: $CP\_Final\_states := CP\_Final\_states\_value$
act4: $end := 0$
act5: $Number\_of\_send := 0$
act6: $Prophecy\_of\_Sent\_Messages := Prophecy\_value$

end

Event Add-Sequence ⟨convergent⟩

any

Some_cp_b

where

grd1: $Some\_cp\_b \in dom(MESSAGE)$
grd2: $Some\_cp\_b \in DC$
grd3: $MESSAGE(Some\_cp\_b) \neq End\_message$
grd4: $SOURCE\_STATE(Some\_cp\_b) \in CP\_Final\_states$
grd5: $\forall Trans \cdot (Trans \in BUILT\_CP \wedge Trans \in dom(SOURCE\_STATE)) \Rightarrow DESTINATION\_STATE(Some\_cp\_b) \neq SOURCE\_STATE(Trans)$
grd6: $SOURCE\_STATE(Some\_cp\_b) \neq DESTINATION\_STATE(Some\_cp\_b)$
grd7: $SOURCE\_STATE(Some\_cp\_b) \in CP\_Final\_states$
grd8: $BUILT\_CP \neq \varnothing \Rightarrow Some\_cp\_b \in ISeqF$
grd9: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{Some\_cp\_b\}$
act2: $CP\_Final\_states := (CP\_Final\_states \cup \{DESTINATION\_STATE(Some\_cp\_b)\}) \setminus \{SOURCE\_STATE(Some\_cp\_b)\}$
act3: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_Choice ⟨convergent⟩

any

Branches
branch

where

grd1: $branch \in dom(SOURCE\_STATE)$

grd2: $branch \in dom(PEER\_SOURCE)$
grd3: $finite(PCF(SOURCE\_STATE(branch)))$
grd4: $Branches \subseteq DC$
grd5: $Branches = PCF(SOURCE\_STATE(branch))$
grd6: $branch \in Branches$
grd7: $BUILT\_CP \neq \varnothing \Rightarrow branch \in ISeqF$
grd8: $MESSAGE(branch) \neq End\_message$
grd9: $finite(PCF)$
grd10: $SOURCE\_STATE(branch) \in CP\_Final\_states$
grd11: $Prophecy\_of\_Sent\_Messages - Number\_of\_send \in \mathbb{N}$

then

act1: $BUILT\_CP := BUILT\_CP \cup Branches$
act2: $CP\_Final\_states := (CP\_Final\_states \cup BR\_CP\_FINAL\_STATES(SOURCE\_STATE(branch))) \setminus \{SOURCE\_STATE(branch)\}$
act3: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_Self-Loop ⟨convergent⟩

any

Some_cp_b

where

grd1: $Some\_cp\_b \in DC \wedge (BUILT\_CP \neq \varnothing \Rightarrow Some\_cp\_b \in ISeqF)$
grd2: $MESSAGE(Some\_cp\_b) \neq End\_message$
grd3: $SOURCE\_STATE(Some\_cp\_b) \in CP\_Final\_states$
grd4: $SOURCE\_STATE(Some\_cp\_b) = DESTINATION\_STATE(Some\_cp\_b)$
grd5: $DESTINATION\_STATE(Some\_cp\_b) \in CP\_Final\_states$
grd6: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{Some\_cp\_b\}$
act2: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_Loop ⟨convergent⟩

any

Some_cp_b

where

grd1: $Some\_cp\_b \in DC$
grd2: $BUILT\_CP \neq \varnothing \Rightarrow Some\_cp\_b \in ISeqF$
grd3: $\exists Trans \cdot (Trans \in BUILT\_CP \wedge Trans \in dom(SOURCE\_STATE) \wedge DESTINATION\_STATE(Some\_cp\_b) = SOURCE\_STATE(Trans))$
grd4: $Some\_cp\_b \in dom(MESSAGE)$
grd5: $MESSAGE(Some\_cp\_b) \neq End\_message$
grd6: $SOURCE\_STATE(Some\_cp\_b) \in CP\_Final\_states$
grd7: $\exists Trans \cdot (Trans \in BUILT\_CP \wedge Trans \in dom(PEER\_SOURCE)) \wedge ((PEER\_SOURCE(Some\_cp\_b) = PEER\_SOURCE(Trans)) \vee (PEER\_DESTINATION(Some\_cp\_b) = PEER\_SOURCE(Trans)))$
grd8: $\exists Trans \cdot (Trans \in BUILT\_CP \wedge Trans \in dom(SOURCE\_STATE) \wedge DESTINATION\_STATE(Some\_cp\_b) = SOURCE\_STATE(Trans) \wedge SOURCE\_STATE(Some\_cp\_b) \neq DESTINATION\_STATE(Some\_cp\_b) \wedge SOURCE\_STATE(Some\_cp\_b) \in CP\_Final\_states)$
grd9: $Prophecy\_of\_Sent\_Messages - Number\_of\_send \in \mathbb{N}$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{Some\_cp\_b\}$
act2: $CP\_Final\_states := (CP\_Final\_states \cup \{DESTINATION\_STATE(Some\_cp\_b)\}) \setminus \{SOURCE\_STATE(Some\_cp\_b)\}$
act3: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_End ⟨ordinary⟩

any

Some_cp_b

where

grd1: $Some\_cp\_b \in dom(MESSAGE)$

grd2: $Some\_cp\_b \in dom(SOURCE\_STATE)$

grd3: $Some\_cp\_b \in DC$

grd4: $Prophecy\_of\_Sent\_Messages - Number\_of\_send = 0$

grd5: $SOURCE\_STATE(Some\_cp\_b) \in CP\_Final\_states$

grd6: $end = 0$

grd7: $MESSAGE(Some\_cp\_b) = End\_message$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{Some\_cp\_b\}$

act2: $end := 1$

end

END

Premier raffinement

CONTEXT LTS_SYNC_CONTEXT

EXTENDS LTS_CONTEXT

SETS

ACTIONS

CONSTANTS

CPs_SYNC_B Set of CP_SYNCHRONE basic transitions

Send Send action

Receive Receive action

Internal Internal action

S_MESSAGE Fuction returning the Synchronous-transition message

S_SOURCE_STATE Fuction returning the Synchronous-transition source state

S_DESTINATION_STATE Fuction returning the Synchronous-transition destination state

S_INDEX Fuction returning the Synchronous-transition index

S_PEER_SOURCE Function returning Synchronous-transition source peer

S_PEER_DESTINATION Fuction returning the Synchronous-transition destination peer

EQUIVALENCE EQUIVALENCE property

S_PCF Fuction returning the synchronous parallel choise freeness set

S_LABEL Fuction returning the synchronus label

AXIOMS

CPs_SYNC_B: $CPs\_SYNC\_B \subseteq CP\_STATES \times ACTIONS \times MESSAGES \times PEERS \times PEERS \times ACTIONS \times MESSAGES \times CP\_STATES \times \mathbb{N}$

axm1: $finite(PEERS) \land card(PEERS) \geq 2$

axm2: $finite(MESSAGES) \land card(MESSAGES) \geq 1$

axm3: $ACTIONS \neq \varnothing$

axm4: $partition(ACTIONS, \{Send\}, \{Receive\}, \{Internal\})$

S_INDEX: $S\_INDEX \in CPs\_SYNC\_B \leftrightarrow \mathbb{N}$

FCT_S_INDEX: $S\_INDEX = \{S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \mapsto S\_idx | S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \in CPs\_SYNC\_B \land S\_idx \in \mathbb{N} \land S\_idx = S\_Index\}$

S_MESSAGE: $S\_MESSAGE \in CPs\_SYNC\_B \leftrightarrow MESSAGES$

FCT_S_MESSAGE: $S\_MESSAGE = \{S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \mapsto S\_message | S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \in CPs\_SYNC\_B \land S\_message \in MESSAGES \land S\_message = S\_msg\}$

S_DESTINATION_STATE: $S\_DESTINATION\_STATE \in CPs\_SYNC\_B \leftrightarrow CP\_STATES$

FCT_S_DESTINATION_STATE: $S\_DESTINATION\_STATE = \{S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \mapsto S\_state\_de | S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \in CPs\_SYNC\_B \land S\_state\_de \in CP\_STATES \land S\_state\_de = S\_state\_dst\}$

S_SOURCE_STATE: $S\_SOURCE\_STATE \in CPs\_SYNC\_B \leftrightarrow CP\_STATES$

FCT_S_SOURCE_STATE: $S\_SOURCE\_STATE = \{S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \mapsto S\_state\_so | S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \in CPs\_SYNC\_B \land S\_state\_src \in CP\_STATES \land S\_state\_so = S\_state\_src\}$

S_PEER_SOURCE: $S\_PEER\_SOURCE \in CPs\_SYNC\_B \leftrightarrow PEERS$

FCT_S_PEER_SOURCE: $S\_PEER\_SOURCE = \{S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \mapsto S\_Peer\_so | S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \in CPs\_SYNC\_B \land S\_Peer\_so \in PEERS \land S\_Peer\_so = S\_Peer\_src\}$

S_PEER_DESTINATION: $S\_PEER\_DESTINATION \in CPs\_SYNC\_B \leftrightarrow PEERS$

FCT_S_PEER_DESTINATION: $S\_PEER\_DESTINATION = \{S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \mapsto S\_Peer\_de | S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \in CPs\_SYNC\_B \land S\_Peer\_de \in PEERS \land S\_Peer\_de = S\_Peer\_dst\}$

S_BRANCHES_SET: $S\_PCF \subseteq CPs\_SYNC\_B$

FCT_S_BRANCHES_SET: $\forall Trans1, Trans2 \cdot (Trans1 \in CPs\_SYNC\_B \land Trans2 \in CPs\_SYNC\_B \land Trans1 \in dom(S\_SOURCE\_STATE) \land Trans1 \in dom(S\_DESTINATION\_STATE) \land Trans2 \in dom(S\_SOURCE\_STATE) \land Trans2 \in dom(S\_DESTINATION\_STATE) \land S\_SOURCE\_STATE(Trans1) = S\_SOURCE\_STATE(Trans2) \land S\_DESTINATION\_STATE(Trans1) \neq S\_DESTINATION\_STATE(Trans2) \land S\_PEER\_SOURCE(Trans1) = S\_PEER\_SOURCE(Trans2)) \Rightarrow \{Trans1, Trans2\} \subseteq S\_PCF$

S_LABEL: $S\_LABEL \in CPs\_SYNC\_B \leftrightarrow PEERS \times MESSAGES \times PEERS$

FCT_S_LABEL: $S\_LABEL = \{S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \mapsto Label | S\_state\_src \mapsto send \mapsto S\_msg \mapsto S\_Peer\_dst \mapsto S\_Peer\_src \mapsto receive \mapsto S\_msg \mapsto S\_state\_dst \mapsto S\_Index \in CPs\_SYNC\_B \land Label \in PEERS \times MESSAGES \times PEERS \land Label = S\_Peer\_src \mapsto S\_msg \mapsto S\_Peer\_dst\}$

EQUIVALENCE: $EQUIVALENCE \in CPs\_B \rightarrowtail CPs\_SYNC\_B$

FCT_EQUIVALENCE: $EQUIVALENCE = \{Trans \mapsto S\_Trans | Trans \in CPs\_B \land S\_Trans \in CPs\_SYNC\_B \land Trans \in dom(SOURCE\_STATE) \land Trans \in dom(DESTINATION\_STATE) \land Trans \in dom(PEER\_SOURCE) \land S\_Trans \in dom(S\_SOURCE\_STATE) \land Trans \in dom(PEER\_DESTINATION) \land Trans \in dom(MESSAGE) \land Trans \in dom(INDEX) \land SOURCE\_STATE(Trans) = S\_SOURCE\_STATE(S\_Trans) \land DESTINATION\_STATE(Trans) = S\_DESTINATION\_STATE(S\_Trans) \land PEER\_SOURCE(Trans) = S\_PEER\_SOURCE(S\_Trans) \land PEER\_DESTINATION(Trans) = S\_PEER\_DESTINATION(S\_Trans) \land MESSAGE(Trans) = S\_MESSAGE(S\_Trans) \land INDEX(Trans) = S\_INDEX(S\_Trans)\}$

END

MACHINE LTS_Synchronous_model
REFINES LTS_model
SEES LTS_SYNC_CONTEXT
VARIABLES

BUILT_CP
BUILT_SYNCHRONE
CP_Initial_state
CP_Final_states
end
Number_of_send
Prophecy_of_Sent_Messages

INVARIANTS

inv1: $BUILT\_CP \subseteq DC$
inv2: $BUILT\_SYNCHRONE \subseteq CPs\_SYNC\_B$
inv3: $CP\_Initial\_state \in CP\_STATES$
inv4: $CP\_Final\_states \subseteq CP\_STATES$
inv5: $end \in \{0, 1\}$
EQUIVALENCE CPCP_SYNC: $\forall Trans \cdot \exists S\_Trans \cdot (Trans \in BUILT\_CP \land S\_Trans \in BUILT\_SYNCHRONE \land BUILT\_CP \neq \varnothing) \Rightarrow Trans \mapsto S\_Trans \in EQUIVALENCE$

EVENTS

Initialisation

begin

act1: $BUILT\_CP := \varnothing$
act2: $BUILT\_SYNCHRONE := \varnothing$
act3: $CP\_Initial\_state := Initial\_state\_value$
act4: $CP\_Final\_states := CP\_Final\_states\_value$
act5: $end := 0$
act6: $Number\_of\_send := 0$
act7: $Prophecy\_of\_Sent\_Messages := Prophecy\_value$

end

Event Add-Sequence ⟨convergent⟩
refines Add-Sequence

any

S_Some_cp_b
Some_cp_sync_b Some basic transition of the set CPs_SYNC_B

where

grd1: $S\_Some\_cp\_b \in dom(MESSAGE)$
grd2: $S\_Some\_cp\_b \mapsto Some\_cp\_sync\_b \in EQUIVALENCE$
grd3: $\forall S\_Trans \cdot (S\_Trans \in BUILT\_SYNCHRONE \land Some\_cp\_sync\_b \in dom(S\_DESTINATION\_STATE) \land S\_Trans \in dom(S\_SOURCE\_STATE)) \Rightarrow S\_DESTINATION\_STATE(Some\_cp\_sync\_b) \neq S\_SOURCE\_STATE(S\_Trans)$
grd4: $S\_Some\_cp\_b \mapsto Some\_cp\_sync\_b \in EQUIVALENCE$
grd5: $S\_Some\_cp\_b \in DC$
grd6: $Some\_cp\_sync\_b \in CPs\_SYNC\_B$
grd7: $SOURCE\_STATE(S\_Some\_cp\_b) \in CP\_Final\_states$
grd8: $S\_SOURCE\_STATE(Some\_cp\_sync\_b) \in CP\_Final\_states$
grd9: $\forall Trans \cdot (Trans \in BUILT\_CP \land Trans \in dom(SOURCE\_STATE)) \Rightarrow DESTINATION\_STATE(S\_Some\_cp\_b) \neq SOURCE\_STATE(Trans)$
grd10: $MESSAGE(S\_Some\_cp\_b) \neq End\_message$
grd11: $MESSAGE(S\_Some\_cp\_b) = S\_MESSAGE(Some\_cp\_sync\_b)$

grd12: $SOURCE\_STATE(S\_Some\_cp\_b) \neq DESTINATION\_STATE(S\_Some\_cp\_b)$
grd13: $SOURCE\_STATE(S\_Some\_cp\_b) \in CP\_Final\_states$
grd14: $S\_SOURCE\_STATE(Some\_cp\_sync\_b) \neq S\_DESTINATION\_STATE(Some\_cp\_sync\_b)$
grd15: $S\_SOURCE\_STATE(Some\_cp\_sync\_b) \in CP\_Final\_states$
grd16: $BUILT\_CP \neq \varnothing \Rightarrow S\_Some\_cp\_b \in ISeqF$
grd17: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$

with

Some_cp_b: $Some\_cp\_b = S\_Some\_cp\_b$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{S\_Some\_cp\_b\}$
act2: $BUILT\_SYNCHRONE := BUILT\_SYNCHRONE \cup \{Some\_cp\_sync\_b\}$
act3: $CP\_Final\_states := (CP\_Final\_states \cup \{DESTINATION\_STATE(S\_Some\_cp\_b)\}) \setminus \{SOURCE\_STATE(S\_Some\_cp\_b)\}$
act4: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_Choice ⟨convergent⟩

refines Add_Choice

any

S_Branches
S_branch
S_Sync_Branches
S_Sync_branch

where

grd1: $S\_branch \in dom(SOURCE\_STATE)$
grd2: $S\_branch \in dom(PEER\_SOURCE)$
grd3: $S\_branch \in dom(MESSAGE)$
grd4: $S\_Sync\_branch \in dom(S\_MESSAGE)$
grd5: $S\_branch \mapsto S\_Sync\_branch \in EQUIVALENCE$
grd6: $finite(PCF(SOURCE\_STATE(S\_branch)))$
grd7: $S\_Branches \subseteq DC$
grd8: $S\_branch \in S\_Branches$
grd9: $S\_Sync\_Branches \subseteq CPs\_SYNC\_B$
grd10: $S\_Sync\_branch \in S\_Sync\_Branches$
grd11: $SOURCE\_STATE(S\_branch) \in CP\_Final\_states$
grd12: $BUILT\_CP \neq \varnothing \Rightarrow S\_branch \in ISeqF$
grd13: $S\_Branches = PCF(SOURCE\_STATE(S\_branch))$
grd14: $S\_Sync\_Branches = S\_PCF$
grd15: $S\_SOURCE\_STATE(S\_Sync\_branch) \in CP\_Final\_states$
grd16: $MESSAGE(S\_branch) = S\_MESSAGE(S\_Sync\_branch)$
grd17: $MESSAGE(S\_branch) \neq End\_message$
grd18: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$

with

Branches: $Branches = S\_Branches$
branch: $branch = S\_branch$

then

act1: $BUILT\_CP := BUILT\_CP \cup S\_Branches$
act2: $BUILT\_SYNCHRONE := BUILT\_SYNCHRONE \cup S\_Sync\_Branches$
act3: $CP\_Final\_states := (CP\_Final\_states \cup BR\_CP\_FINAL\_STATES(SOURCE\_STATE(S\_branch))) \setminus \{SOURCE\_STATE(S\_branch)\}$
act4: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_Self-Loop ⟨convergent⟩

refines Add_Self-Loop

any

S_Some_cp_b
Some_cp_sync_b

where

grd1: $S\_Some\_cp\_b \in dom(PEER\_SOURCE)$

grd2: $S\_Some\_cp\_b \in dom(SOURCE\_STATE)$
grd3: $S\_Some\_cp\_b \in dom(DESTINATION\_STATE)$
grd4: $S\_Some\_cp\_b \in dom(MESSAGE)$
grd5: $S\_Some\_cp\_b \mapsto Some\_cp\_sync\_b \in EQUIVALENCE$
grd6: $S\_Some\_cp\_b \in DC \wedge (BUILT\_CP \neq \varnothing \Rightarrow S\_Some\_cp\_b \in ISeqF)$
grd7: $Some\_cp\_sync\_b \in CPs\_SYNC\_B$
grd8: $SOURCE\_STATE(S\_Some\_cp\_b) \in CP\_Final\_states$
grd9: $S\_SOURCE\_STATE(Some\_cp\_sync\_b) \in CP\_Final\_states$
grd10: $SOURCE\_STATE(S\_Some\_cp\_b) = DESTINATION\_STATE(S\_Some\_cp\_b)$
grd11: $S\_SOURCE\_STATE(Some\_cp\_sync\_b) = S\_DESTINATION\_STATE(Some\_cp\_sync\_b)$
grd12: $MESSAGE(S\_Some\_cp\_b) = S\_MESSAGE(Some\_cp\_sync\_b)$
grd13: $MESSAGE(S\_Some\_cp\_b) \neq End\_message$
grd14: $DESTINATION\_STATE(S\_Some\_cp\_b) \in CP\_Final\_states$
grd15: $S\_DESTINATION\_STATE(Some\_cp\_sync\_b) \in CP\_Final\_states$
grd16: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$

with

Some_cp_b: $Some\_cp\_b = S\_Some\_cp\_b$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{S\_Some\_cp\_b\}$
act2: $BUILT\_SYNCHRONE := BUILT\_SYNCHRONE \cup \{Some\_cp\_sync\_b\}$
act3: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_Loop ⟨convergent⟩

refines Add_Loop

any

S_Some_cp_b
Some_cp_sync_b

where

grd1: $BUILT\_CP \neq \varnothing$
grd2: $BUILT\_SYNCHRONE \neq \varnothing$
grd3: $S\_Some\_cp\_b \in dom(PEER\_SOURCE)$
grd4: $S\_Some\_cp\_b \in dom(SOURCE\_STATE)$
grd5: $S\_Some\_cp\_b \in dom(DESTINATION\_STATE)$
grd6: $S\_Some\_cp\_b \in dom(MESSAGE)$
grd7: $Some\_cp\_sync\_b \in CPs\_SYNC\_B$
grd8: $S\_Some\_cp\_b \mapsto Some\_cp\_sync\_b \in EQUIVALENCE$
grd9: $S\_Some\_cp\_b \in DC \wedge (BUILT\_CP \neq \varnothing \Rightarrow S\_Some\_cp\_b \in ISeqF)$
grd10: $SOURCE\_STATE(S\_Some\_cp\_b) \in CP\_Final\_states$
grd11: $S\_SOURCE\_STATE(Some\_cp\_sync\_b) \in CP\_Final\_states$
grd12: $\exists S\_Trans \cdot (S\_Trans \in BUILT\_SYNCHRONE) \wedge$
$S\_DESTINATION\_STATE(Some\_cp\_sync\_b) = S\_SOURCE\_STATE(S\_Trans)$
grd13: $\exists Trans \cdot (Trans \in BUILT\_CP \wedge Trans \in dom(SOURCE\_STATE)) \wedge$
$DESTINATION\_STATE(S\_Some\_cp\_b) = SOURCE\_STATE(Trans)$
grd14: $MESSAGE(S\_Some\_cp\_b) = S\_MESSAGE(Some\_cp\_sync\_b)$
grd15: $MESSAGE(S\_Some\_cp\_b) \neq End\_message$
grd16: $\exists Trans \cdot (Trans \in BUILT\_CP \wedge Trans \in dom(PEER\_SOURCE) \wedge$
$S\_Some\_cp\_b \in dom(PEER\_DESTINATION)) \wedge$
$((PEER\_SOURCE(S\_Some\_cp\_b) = PEER\_SOURCE(Trans))$
$\vee$
$(PEER\_DESTINATION(S\_Some\_cp\_b) = PEER\_SOURCE(Trans)))$
grd17: $SOURCE\_STATE(S\_Some\_cp\_b) \neq DESTINATION\_STATE(S\_Some\_cp\_b)$
grd18: $S\_SOURCE\_STATE(Some\_cp\_sync\_b) \neq S\_DESTINATION\_STATE(Some\_cp\_sync\_b)$
grd19: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$

with

Some_cp_b: $Some\_cp\_b = S\_Some\_cp\_b$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{S\_Some\_cp\_b\}$
act2: $BUILT\_SYNCHRONE := BUILT\_SYNCHRONE \cup \{Some\_cp\_sync\_b\}$
act3: $CP\_Final\_states := (CP\_Final\_states \cup \{DESTINATION\_STATE(S\_Some\_cp\_b)\}) \setminus$
$\{SOURCE\_STATE(S\_Some\_cp\_b)\}$

act4: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_End ⟨ordinary⟩

refines Add_End

any

S_Some_cp_b

where

grd1: $S\_Some\_cp\_b \in dom(SOURCE\_STATE)$

grd2: $S\_Some\_cp\_b \in dom(MESSAGE)$

grd3: $S\_Some\_cp\_b \in DC$

grd4: $Prophecy\_of\_Sent\_Messages - Number\_of\_send = 0$

grd5: $SOURCE\_STATE(S\_Some\_cp\_b) \in CP\_Final\_states$

grd6: $end = 0$

grd7: $MESSAGE(S\_Some\_cp\_b) = End\_message$

with

Some_cp_b: $Some\_cp\_b = S\_Some\_cp\_b$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{S\_Some\_cp\_b\}$

act2: $end := 1$

end

END

Deuxième raffinement

CONTEXT LTS_ASYNC_CONTEXT
EXTENDS LTS_SYNC_CONTEXT
SETS

A_STATES
A_TRACE_STATES A_TRACE_STATES

CONSTANTS

A_Next_States
QUEUE
PEERs_B
A_SOURCE_STATE
A_DESTINATION_STATE
A_MESSAGE
SYNCHRONISABILITY
ACTION
A_GLOBAL_STATES
Tr_MESSAGE
S_Next_States
Next_States
TR_INDEX
R_TRACE_B
R_INDEX
R_MESSAGE
A_TRACES
A_LABEL
ETIQ
WF
R_PEER_SOURCE
R_PEER_DESTINATION
R_SOURCE_STATE
R_DESTINATION_STATE
A_INDEX
A_GS_value
S_GS_value
Last_cp_trans

AXIOMS

QUEUE: $QUEUE \subseteq \mathbb{P}(PEERS \times MESSAGES \times \mathbb{N})$
ETIQ: $ETIQ \subseteq ACTIONS \times MESSAGES \times PEERS$
PEERs_B: $PEERs\_B \in (A\_STATES \times ETIQ \times \mathbb{N}) \rightarrow A\_STATES$
A_TRACE: $A\_TRACES \subseteq A\_TRACE\_STATES \times \mathbb{N} \times ACTIONS \times PEERS \times MESSAGES \times PEERS \times A\_TRACE\_STATES \times \mathbb{N} \times \mathbb{N}$
R_TRACE_B: $R\_TRACE\_B \subseteq CP\_STATES \times PEERS \times MESSAGES \times PEERS \times CP\_STATES \times \mathbb{N}$
S_Next_States: $S\_Next\_States \in \mathbb{P}(PEERs\_B) \times \mathbb{P}(PEERS \times A\_STATES) \rightarrow \mathbb{P}(PEERS \times A\_STATES)$

Next_States: $Next\_States \subseteq PEERs\_B \times \mathbb{P}(PEERS \times A\_STATES)$
A_Next_States: $A\_Next\_States \in \mathbb{P}(PEERs\_B) \times \mathbb{P}(PEERS \times A\_STATES) \times \mathbb{P}(PEERS \times MESSAGES \times \mathbb{N}) \rightarrow \mathbb{P}(PEERS \times A\_STATES)$
A_GLOBAL_STATE: $A\_GLOBAL\_STATES \in \mathbb{P}(PEERS \times A\_STATES)$
A_GS_value: $A\_GS\_value \subseteq A\_GLOBAL\_STATES$
S_GS_value: $S\_GS\_value \subseteq A\_GLOBAL\_STATES$
ACTIONS: $ACTIONS \neq \varnothing$
ACTIONS_partition: $partition(ACTIONS, \{Send\}, \{Receive\}, \{Internal\})$
A_MESSAGE: $A\_MESSAGE \in PEERs\_B \leftrightarrow MESSAGES$

FCT_A_MESSAGE: $A\_MESSAGE = \{((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) \mapsto message | ((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) \in PEERs\_B \land message \in MESSAGES \land message = A\_msg\}$

A_SOURCE_STATE: $A\_SOURCE\_STATE \in PEERs\_B \leftrightarrow A\_STATES$

FCT_A_SOURCE_STATE: $A\_SOURCE\_STATE = \{((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) \mapsto state\_src | ((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) \in PEERs\_B \land state\_src \in A\_STATES \land state\_src = A\_state\_src\}$

A_DESTINATION_STATE: $A\_DESTINATION\_STATE \in PEERs\_B \leftrightarrow A\_STATES$

FCT_A_DESTINATION_STATE: $A\_DESTINATION\_STATE = \{((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) \mapsto state\_dst | ((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) \in PEERs\_B \land state\_dst \in A\_STATES \land state\_dst = A\_state\_dst\}$

A_LABEL: $A\_LABEL \in PEERs\_B \leftrightarrow ACTIONS \times MESSAGES \times PEERS$

FCT_A_LABEL: $A\_LABEL = \{((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) \mapsto A\_label | ((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) \in PEERs\_B \land A\_label \in ACTIONS \times MESSAGES \times PEERS \land A\_label = (action \mapsto A\_msg \mapsto peer)\}$

ACTION: $ACTION \in PEERs\_B \leftrightarrow ACTIONS$

FCT_ACTION: $ACTION = \{((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) \mapsto Action | ((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) \in PEERs\_B \land Action \in ACTIONS \land Action = action\}$

R_MESSAGE: $R\_MESSAGE \in R\_TRACE\_B \leftrightarrow MESSAGES$

FCT_R_MESSAGE: $R\_MESSAGE = \{state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \mapsto message | state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \in R\_TRACE\_B \land message \in MESSAGES \land message = A\_msg\}$

R_INDEX: $R\_INDEX \in R\_TRACE\_B \leftrightarrow \mathbb{N}$

FCT_R_INDEX: $R\_INDEX = \{state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \mapsto Index | state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \in R\_TRACE\_B \land Index \in \mathbb{N} \land Index = index\}$

R_PEER_SOURCE: $R\_PEER\_SOURCE \in R\_TRACE\_B \leftrightarrow PEERS$

FCT_R_PEER_SOURCE: $R\_PEER\_SOURCE = \{state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \mapsto peer\_source | state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \in R\_TRACE\_B \land peer\_source \in PEERS \land peer\_source = peer\_src\}$

R_PEER_DESTINATION: $R\_PEER\_DESTINATION \in R\_TRACE\_B \leftrightarrow PEERS$

FCT_R_PEER_DESTINATION: $R\_PEER\_DESTINATION = \{state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \mapsto peer\_destination | state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \in R\_TRACE\_B \land peer\_destination \in PEERS \land peer\_destination = peer\_dst\}$

R_SOURCE_STATE: $R\_SOURCE\_STATE \in R\_TRACE\_B \leftrightarrow CP\_STATES$

FCT_R_SOURCE_STATE: $R\_SOURCE\_STATE = \{state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \mapsto state\_src | state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \in R\_TRACE\_B \land state\_src = state\_src\}$

R_DESTINATION_STATE: $R\_DESTINATION\_STATE \in R\_TRACE\_B \leftrightarrow CP\_STATES$

FCT_R_DESTINATION_STATE: $R\_DESTINATION\_STATE = \{state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \mapsto state\_destination | state\_src \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \in R\_TRACE\_B \land state\_destination = state\_dst\}$

TR_DESTINATION_STATE: $TR\_INDEX \in A\_TRACES \leftrightarrow \mathbb{N}$

FCT_TR_DESTINATION_STATE: $TR\_INDEX = \{state\_src \mapsto action \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \mapsto idx | state\_src \mapsto action \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto index \in A\_TRACES \land idx \in \mathbb{N} \land idx = index\}$

Tr_MESSAGE: $Tr\_MESSAGE \in A\_TRACES \leftrightarrow MESSAGES$

FCT_Tr_MESSAGE: $Tr\_MESSAGE = \{state\_src \mapsto i \mapsto action \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto i \mapsto index \mapsto msg | state\_src \mapsto i \mapsto action \mapsto peer\_src \mapsto A\_msg \mapsto peer\_dst \mapsto state\_dst \mapsto i \mapsto index \in A\_TRACES \land msg \in MESSAGES \land msg = A\_msg\}$

A_INDEX: $A\_INDEX \in PEERs\_B \rightarrow \mathbb{N}$

FCT_A_INDEX: $\forall A\_state\_src, action, A\_msg, peer, index, A\_state\_dst \cdot ((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst \in PEERs\_B) \Rightarrow A\_INDEX((A\_state\_src \mapsto (action \mapsto A\_msg \mapsto peer) \mapsto index) \mapsto A\_state\_dst) = index$

SYNCHRONISABILITY: $SYNCHRONISABILITY \in CPs\_SYNC\_B \rightarrowtail R\_TRACE\_B$

FCT_SYNCHRONISABILITY: $SYNCHRONISABILITY = \{S\_Trans \mapsto R\_Trans | S\_Trans \in CPs\_SYNC\_B \land R\_Trans \in R\_TRACE\_B \land S\_SOURCE\_STATE(S\_Trans) = R\_SOURCE\_STATE(R\_Trans) \land S\_PEER\_SOURCE(S\_Trans) = R\_PEER\_SOURCE(R\_Trans) \land S\_MESSAGE(S\_Trans) = R\_MESSAGE(R\_Trans) \land S\_PEER\_DESTINATION(S\_Trans) = R\_PEER\_DESTINATION(R\_Trans) \land S\_DESTINATION\_STATE(S\_Trans) = R\_DESTINATION\_STATE(R\_Trans) \land S\_INDEX(S\_Trans) = R\_INDEX(R\_Trans)\}$

WF: $WF \in A\_TRACES \rightarrow QUEUE$

FCT_WF: $\forall A\_TR, queue \cdot (A\_TR \in A\_TRACES \land queue \in QUEUE \land queue = \varnothing) \Rightarrow A\_TR \mapsto queue \in WF$

Last_cp_trans_value: $Last\_cp\_trans\_value \in CPs\_B$

END

MACHINE LTS_Asynchronous_model
REFINES LTS_Synchronous_model
SEES LTS_ASYNC_CONTEXT
VARIABLES

BUILT_CP Built conversation protocol
CP_Initial_state The initial state of CP
CP_Final_states The set of CP final states
BUILT_SYNCHRONE Built synchronous conversation protocol
BUILT_ASYNCHRONE Built asynchronous conversation protocol
A_TRACE Built asynchronous conversation protocol traces
REDUCED_TRACE Built the reduced asynchronous trace
Last_cp_trans The last CP transition
queue Queue
back index queue
front index queue
A_GS Asynchronous global states
S_GS Synchronous global states
Prophecy_of_Sent_Messages Prophecy of sent messages
Number_of_send Number of sent messages
Interaction_Completed Variable used to manage events "Add-send, Add-receive and Add_send-receive"
Reduces_Trace_states States of reduced trace
Number_of_Reduced_Trace_states number of states of the reduced trace
Reduced_Trace_index Index of the REDUCED_TRACE
A_Trace_index Index of A_TRACE
end
A_Final_states
A_index
Exchanged_message

INVARIANTS

inv1: $BUILT\_CP \subseteq DC$
inv2: $CP\_Initial\_state \in CP\_STATES$
inv3: $CP\_Final\_states \subseteq CP\_STATES$
inv4: $BUILT\_SYNCHRONE \subseteq CPs\_SYNC\_B$
inv5: $BUILT\_ASYNCHRONE \subseteq PEERs\_B$
inv6: $queue \subseteq PEERS \times MESSAGES \times \mathbb{N}$
inv7: $back \in \mathbb{N}$
inv8: $front \in \mathbb{N}$
inv9: $A\_Trace\_index \in \mathbb{N}$
inv10: $Reduced\_Trace\_index \in \mathbb{N}$
inv11: $A\_TRACE \subseteq A\_TRACES$
inv12: $Last\_cp\_trans \in CPs\_B$
inv13: $REDUCED\_TRACE \subseteq R\_TRACE\_B$
inv14: $A\_GS \in PEERS \leftrightarrow A\_STATES$
inv15: $S\_GS \in PEERS \leftrightarrow A\_STATES$
inv16: $Number\_of\_Reduced\_Trace\_states \in \mathbb{N}$
inv17: $Reduces\_Trace\_states \in A\_TRACE\_STATES$
inv18: $Number\_of\_send \in \mathbb{N}$
inv19: $Prophecy\_of\_Sent\_Messages \in \mathbb{N}$
inv20: $Interaction\_Completed \in \{0, 1\}$
inv22: $end \in \{0, 1\}$
inv23: $A\_Final\_states \subseteq CP\_STATES$
inv24: $A\_index \in \mathbb{N}$

SYNCHRONISABILITY BUILT_SYNCHRONE_REDUCED_TRACE: $\forall S\_Trans \cdot \exists R\_Trans \cdot ($
$S\_Trans \in BUILT\_SYNCHRONE \land R\_Trans \in REDUCED\_TRACE)$
$\Rightarrow$
$S\_Trans \mapsto R\_Trans \in SYNCHRONISABILITY$

WF: $\forall A\_Trans \cdot (A\_Trans \in A\_TRACES \land\ MESSAGE(Last\_cp\_trans) = End\_message \land$
$A\_TRACE \neq \varnothing)$
$\Rightarrow$
$A\_Trans \mapsto queue \in WF$

inv25: $Number\_of\_send \in \mathbb{N}$

inv26: $Prophecy\_of\_Sent\_Messages \in \mathbb{N}$

inv27: $Exchanged\_message \in MESSAGES$

EVENTS

Initialisation

begin

act1: $BUILT\_CP := \varnothing$
act2: $CP\_Initial\_state := Initial\_state\_value$
act3: $CP\_Final\_states := CP\_Final\_states\_value$
act4: $BUILT\_SYNCHRONE := \varnothing$
act5: $BUILT\_ASYNCHRONE := \varnothing$
act6: $queue := \varnothing$
act7: $front := 0$
act8: $back := 0$
act9: $Interaction\_Completed := 0$
act10: $A\_GS := A\_GS\_value$
act11: $S\_GS := S\_GS\_value$
act12: $A\_Trace\_index := 1$
act13: $REDUCED\_TRACE := \varnothing$
act14: $Reduced\_Trace\_index := 1$
act15: $Number\_of\_send := 0$
act16: $Prophecy\_of\_Sent\_Messages := Prophecy\_value$
act17: $Number\_of\_Reduced\_Trace\_states := 0$
act18: $Reduces\_Trace\_states :\in A\_TRACE\_STATES$
act19: $A\_TRACE := \varnothing$
act20: $end := 0$
act21: $A\_Final\_states := CP\_Final\_states\_value$
act22: $A\_index := 1$
act23: $Last\_cp\_trans := Last\_cp\_trans\_value$
act24: $Exchanged\_message :\in MESSAGES$

end

Event Add-Send ⟨ordinary⟩

any

send
lts_s
lts_d
msg
index

where

grd1: $\{send\} \mapsto A\_GS \mapsto queue \in dom(A\_Next\_States)$
grd2: $\exists send\_st\_src, send\_st\_dest \cdot ((lts\_s \mapsto send\_st\_src) \in A\_GS \land$
$((send\_st\_src \mapsto (Send \mapsto msg \mapsto lts\_d) \mapsto index) \mapsto send\_st\_dest) \in PEERs\_B \land$
$(send = (send\_st\_src \mapsto (Send \mapsto msg \mapsto lts\_d) \mapsto index) \mapsto send\_st\_dest))$
grd3: $Interaction\_Completed = 0$
grd4: $Reduces\_Trace\_states \mapsto Number\_of\_Reduced\_Trace\_states \mapsto Send \mapsto lts\_s \mapsto msg \mapsto$
$lts\_d \mapsto Reduces\_Trace\_states \mapsto Number\_of\_Reduced\_Trace\_states{+}1 \mapsto A\_Trace\_index$
$\in A\_TRACES$
grd5: $MESSAGE(Last\_cp\_trans) \neq End\_message$
grd6: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$

then

act1: $A\_TRACE := A\_TRACE \cup \{Reduces\_Trace\_states \mapsto Number\_of\_Reduced\_Trace\_states \mapsto Send \mapsto lts\_s \mapsto msg \mapsto lts\_d \mapsto Reduces\_Trace\_states \mapsto (Number\_of\_Reduced\_Trace\_states+1) \mapsto A\_Trace\_index\}$

act2: $queue, back := queue \cup \{lts\_d \mapsto msg \mapsto back\}, back + 1$

act3: $A\_GS := A\_Next\_States(\{send\} \mapsto A\_GS \mapsto queue)$

act4: $A\_Trace\_index := A\_Trace\_index + 1$

act5: $Number\_of\_Reduced\_Trace\_states := Number\_of\_Reduced\_Trace\_states + 1$

end

Event Add-Receive ⟨ordinary⟩

any

send
receive
lts_s
lts_d
msg
index

where

grd1: $\{receive\} \mapsto A\_GS \mapsto queue \in dom(A\_Next\_States)$

grd2: $queue \neq \varnothing$

grd3: $lts\_d \mapsto msg \mapsto front \in queue$

grd4: $\exists receive\_st\_src, receive\_st\_dest \cdot (((lts\_d \mapsto receive\_st\_src) \in A\_GS) \wedge ((receive\_st\_src \mapsto (Receive \mapsto msg \mapsto lts\_s) \mapsto index) \mapsto receive\_st\_dest) \in PEERs\_B \wedge (receive = (receive\_st\_src \mapsto (Receive \mapsto msg \mapsto lts\_s) \mapsto index) \mapsto receive\_st\_dest))$

grd5: $Interaction\_Completed = 0$

grd6: $back > front$

grd7: $queue \setminus \{lts\_d \mapsto msg \mapsto front + 1\} = \varnothing \Rightarrow back = front + 1$

grd8: $\exists send\_st\_src, send\_st\_dest \cdot (((send\_st\_src \mapsto (Send \mapsto msg \mapsto lts\_d) \mapsto index) \mapsto send\_st\_dest) \in PEERs\_B \wedge (send = (send\_st\_src \mapsto (Send \mapsto msg \mapsto lts\_d) \mapsto index) \mapsto send\_st\_dest))$

grd9: $\exists receive\_st\_src, receive\_st\_dest \cdot (((receive\_st\_src \mapsto (Receive \mapsto msg \mapsto lts\_s) \mapsto index) \mapsto receive\_st\_dest) \in PEERs\_B \wedge (receive = (receive\_st\_src \mapsto (Receive \mapsto msg \mapsto lts\_s) \mapsto index) \mapsto receive\_st\_dest))$

grd10: $Reduces\_Trace\_states \mapsto Number\_of\_Reduced\_Trace\_states \mapsto Receive \mapsto lts\_s \mapsto msg \mapsto lts\_d \mapsto Reduces\_Trace\_states \mapsto (Number\_of\_Reduced\_Trace\_states+1) \mapsto A\_Trace\_index \in A\_TRACES$

grd11: $MESSAGE(Last\_cp\_trans) \neq End\_message$

grd12: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$

then

act1: $A\_TRACE := A\_TRACE \cup \{Reduces\_Trace\_states \mapsto Number\_of\_Reduced\_Trace\_states \mapsto Receive \mapsto lts\_s \mapsto msg \mapsto lts\_d \mapsto Reduces\_Trace\_states \mapsto (Number\_of\_Reduced\_Trace\_states + 1) \mapsto A\_Trace\_index\}$

act2: $A\_Trace\_index := A\_Trace\_index + 1$

act3: $A\_GS := A\_Next\_States(\{receive\} \mapsto A\_GS \mapsto queue)$

act4: $front := front + 1$

act5: $queue := queue \setminus \{lts\_d \mapsto msg \mapsto front\}$

act6: $Interaction\_Completed := 1$

act7: $Number\_of\_Reduced\_Trace\_states := Number\_of\_Reduced\_Trace\_states + 1$

act8: $Exchanged\_message := msg$

end

Event Add-Sequence_Send-Receive ⟨convergent⟩

refines Add-Sequence

any

A_Some_cp_b
A_Some_cp_sync_b
Send_cp_async_b
Receive_cp_async_b
R_trace_b

where

grd1: $Send\_cp\_async\_b \in dom(A\_MESSAGE)$

grd2: $Receive\_cp\_async\_b \in dom(A\_MESSAGE)$

grd3: $Receive\_cp\_async\_b \in dom(ACTION)$
grd4: $Send\_cp\_async\_b \in dom(ACTION)$
grd5: $A\_Some\_cp\_b \in DC$
grd6: $A\_Some\_cp\_sync\_b \in CPs\_SYNC\_B$
grd7: $Send\_cp\_async\_b \in PEERs\_B$
grd8: $Receive\_cp\_async\_b \in PEERs\_B$
grd9: $R\_trace\_b \in R\_TRACE\_B$
grd10: $SOURCE\_STATE(A\_Some\_cp\_b) \in CP\_Final\_states$
grd11: $S\_SOURCE\_STATE(A\_Some\_cp\_sync\_b) \in CP\_Final\_states$
grd12: $A\_MESSAGE(Send\_cp\_async\_b) = A\_MESSAGE(Receive\_cp\_async\_b)$
grd13: $ACTION(Receive\_cp\_async\_b) = Receive$
grd14: $ACTION(Send\_cp\_async\_b) = Send$
grd15: $\forall Trans \cdot (Trans \in BUILT\_CP \land Trans \in dom(SOURCE\_STATE)) \Rightarrow DESTINATION\_STATE(A\_Some\_cp\_b) \neq SOURCE\_STATE(Trans)$
grd16: $\forall S\_Trans \cdot (S\_Trans \in BUILT\_SYNCHRONE \land S\_Trans \in dom(S\_SOURCE\_STATE)) \Rightarrow S\_DESTINATION\_STATE(A\_Some\_cp\_sync\_b) \neq S\_SOURCE\_STATE(S\_Trans)$
grd17: $S\_MESSAGE(A\_Some\_cp\_sync\_b) = A\_MESSAGE(Send\_cp\_async\_b)$
grd18: $A\_MESSAGE(Send\_cp\_async\_b) = R\_MESSAGE(R\_trace\_b)$
grd19: $MESSAGE(A\_Some\_cp\_b) = S\_MESSAGE(A\_Some\_cp\_sync\_b)$
grd20: $MESSAGE(A\_Some\_cp\_b) \neq End\_message$
grd21: $SOURCE\_STATE(A\_Some\_cp\_b) \in CP\_Final\_states$
grd22: $Interaction\_Completed = 1$
grd23: $A\_Some\_cp\_sync\_b \mapsto R\_trace\_b \in SYNCHRONISABILITY$
grd24: $BUILT\_CP \neq \varnothing \Rightarrow A\_Some\_cp\_b \in ISeqF$
grd25: $A\_Some\_cp\_sync\_b \in CPs\_SYNC\_B$
grd26: $Send\_cp\_async\_b \in PEERs\_B$
grd27: $Receive\_cp\_async\_b \in PEERs\_B$
grd28: $R\_trace\_b \in R\_TRACE\_B$
grd29: $SOURCE\_STATE(A\_Some\_cp\_b) \neq DESTINATION\_STATE(A\_Some\_cp\_b)$
grd30: $S\_SOURCE\_STATE(A\_Some\_cp\_sync\_b) \neq S\_DESTINATION\_STATE(A\_Some\_cp\_sync\_b)$
grd31: $MESSAGE(A\_Some\_cp\_b) = Exchanged\_message$
grd32: $S\_MESSAGE(A\_Some\_cp\_sync\_b) = Exchanged\_message$
grd33: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$
grd34: $A\_Some\_cp\_b \mapsto A\_Some\_cp\_sync\_b \in EQUIVALENCE$

with

S_Some_cp_b: $S\_Some\_cp\_b = A\_Some\_cp\_b$
Some_cp_sync_b: $Some\_cp\_sync\_b = A\_Some\_cp\_sync\_b$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{A\_Some\_cp\_b\}$
act2: $BUILT\_SYNCHRONE := BUILT\_SYNCHRONE \cup \{A\_Some\_cp\_sync\_b\}$
act3: $BUILT\_ASYNCHRONE := BUILT\_ASYNCHRONE \cup \{Send\_cp\_async\_b\} \cup \{Receive\_cp\_async\_b\}$
act4: $REDUCED\_TRACE := REDUCED\_TRACE \cup \{R\_trace\_b\}$
act5: $Reduced\_Trace\_index := Reduced\_Trace\_index + 1$
act6: $CP\_Final\_states := (CP\_Final\_states \cup \{DESTINATION\_STATE(A\_Some\_cp\_b)\}) \setminus \{SOURCE\_STATE(A\_Some\_cp\_b)\}$
act7: $Interaction\_Completed := 0$
act8: $A\_index := A\_index + 1$
act9: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_Choice_Send-Receive ⟨convergent⟩

refines Add_Choice

any

A_Branches
A_branch
A_Sync_Branches
A_Sync_branch
A_Async_Branches

A_Async_branch
R_Traces
R_trace

where

grd1: $A\_branch \in dom(SOURCE\_STATE)$
grd2: $A\_branch \in dom(PEER\_SOURCE)$
grd3: $A\_Sync\_branch \in dom(S\_SOURCE\_STATE)$
grd4: $A\_Sync\_branch \in dom(S\_MESSAGE)$
grd5: $A\_Async\_branch \in dom(A\_MESSAGE)$
grd6: $R\_trace \in dom(R\_SOURCE\_STATE)$
grd7: $A\_branch \mapsto A\_Sync\_branch \in EQUIVALENCE$
grd8: $\forall S\_Trans \cdot \exists R\_Trans \cdot (S\_Trans \in BUILT\_SYNCHRONE \cup A\_Sync\_Branches \wedge R\_Trans \in REDUCED\_TRACE \cup R\_Traces) \Rightarrow S\_Trans \mapsto R\_Trans \in SYNCHRONISABILITY$
grd9: $A\_branch \in dom(MESSAGE)$
grd10: $A\_Branches \subseteq DC$
grd11: $A\_branch \in A\_Branches$
grd12: $BUILT\_CP \neq \varnothing \Rightarrow A\_branch \in ISeqF$
grd13: $A\_Sync\_Branches \subseteq CPs\_SYNC\_B$
grd14: $A\_Sync\_branch \in A\_Sync\_Branches$
grd15: $A\_Async\_Branches \subseteq PEERs\_B$
grd16: $A\_Async\_branch \in A\_Async\_Branches$
grd17: $R\_Traces \subseteq R\_TRACE\_B$
grd18: $R\_trace \in R\_Traces$
grd19: $A\_Branches = PCF(SOURCE\_STATE(A\_branch))$
grd20: $A\_Sync\_Branches = S\_PCF$
grd21: $R\_Traces = PCF(R\_SOURCE\_STATE(R\_trace))$
grd22: $SOURCE\_STATE(A\_branch) \in CP\_Final\_states$
grd23: $S\_SOURCE\_STATE(A\_Sync\_branch) \in CP\_Final\_states$
grd24: $\forall Trans \cdot (Trans \in PEERs\_B \wedge A\_INDEX(Trans) = A\_index) \Rightarrow \{Trans\} \subseteq A\_Async\_Branches$
grd25: $Interaction\_Completed = 1$
grd26: $MESSAGE(A\_branch) = S\_MESSAGE(A\_Sync\_branch)$
grd27: $S\_MESSAGE(A\_Sync\_branch) = A\_MESSAGE(A\_Async\_branch)$
grd28: $MESSAGE(A\_branch) \neq End\_message$
grd29: $MESSAGE(A\_branch) = Exchanged\_message$
grd30: $S\_MESSAGE(A\_Sync\_branch) = Exchanged\_message$
grd31: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$
grd32: $finite(PCF(R\_SOURCE\_STATE(R\_trace)))$

with

S_Branches: $S\_Branches = A\_Branches$
S_branch: $S\_branch = A\_branch$
S_Sync_Branches: $S\_Sync\_Branches = A\_Sync\_Branches$
S_Sync_branch: $S\_Sync\_branch = A\_Sync\_branch$

then

act1: $BUILT\_CP := BUILT\_CP \cup A\_Branches$
act2: $BUILT\_SYNCHRONE := BUILT\_SYNCHRONE \cup A\_Sync\_Branches$
act3: $BUILT\_ASYNCHRONE := BUILT\_ASYNCHRONE \cup A\_Async\_Branches$
act4: $REDUCED\_TRACE := REDUCED\_TRACE \cup R\_Traces$
act5: $Reduced\_Trace\_index := Reduced\_Trace\_index + card(R\_Traces)$
act6: $CP\_Final\_states := (CP\_Final\_states \cup BR\_CP\_FINAL\_STATES(SOURCE\_STATE(A\_branch))) \setminus \{SOURCE\_STATE(A\_branch)\}$
act7: $Interaction\_Completed := 0$
act8: $A\_index := A\_index + 1$
act9: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_Self-Loop_Send-Receive ⟨convergent⟩
refines Add_Self-Loop

any

A_Some_cp_b
A_Some_cp_sync_b
Send_cp_async_b
Receive_cp_async_b
R_trace_b

where

grd1: $A\_Some\_cp\_b \in dom(DESTINATION\_STATE)$
grd2: $A\_Some\_cp\_b \in dom(PEER\_SOURCE)$
grd3: $A\_Some\_cp\_b \in dom(SOURCE\_STATE)$
grd4: $A\_Some\_cp\_b \in dom(DESTINATION\_STATE)$
grd5: $A\_Some\_cp\_b \in dom(MESSAGE)$
grd6: $Receive\_cp\_async\_b \in dom(A\_SOURCE\_STATE)$
grd7: $Receive\_cp\_async\_b \in dom(A\_LABEL)$
grd8: $Receive\_cp\_async\_b \in dom(A\_DESTINATION\_STATE)$
grd9: $Send\_cp\_async\_b \in dom(A\_SOURCE\_STATE)$
grd10: $Send\_cp\_async\_b \in dom(A\_LABEL)$
grd11: $Send\_cp\_async\_b \in dom(A\_DESTINATION\_STATE)$
grd12: $Send\_cp\_async\_b \in dom(ACTION)$
grd13: $Receive\_cp\_async\_b \in dom(ACTION)$
grd14: $A\_Some\_cp\_b \in dom(MESSAGE)$
grd15: $Send\_cp\_async\_b \in dom(A\_MESSAGE)$
grd16: $Receive\_cp\_async\_b \in dom(A\_MESSAGE)$
grd17: $A\_Some\_cp\_sync\_b \in dom(S\_DESTINATION\_STATE)$
grd18: $A\_Some\_cp\_sync\_b \in dom(S\_MESSAGE)$
grd19: $A\_Some\_cp\_sync\_b \in dom(S\_MESSAGE)$
grd20: $A\_Some\_cp\_b \mapsto A\_Some\_cp\_sync\_b \in EQUIVALENCE$
grd21: $A\_Some\_cp\_sync\_b \mapsto R\_trace\_b \in SYNCHRONISABILITY$
grd22: $A\_Some\_cp\_b \in DC$
grd23: $BUILT\_CP \neq \varnothing \Rightarrow A\_Some\_cp\_b \in ISeqF$
grd24: $A\_Some\_cp\_sync\_b \in CPs\_SYNC\_B$
grd25: $Send\_cp\_async\_b \in PEERs\_B$
grd26: $Receive\_cp\_async\_b \in PEERs\_B$
grd27: $R\_trace\_b \in R\_TRACE\_B$
grd28: $SOURCE\_STATE(A\_Some\_cp\_b) \in CP\_Final\_states$
grd29: $S\_SOURCE\_STATE(A\_Some\_cp\_sync\_b) \in CP\_Final\_states$
grd30: $A\_MESSAGE(Send\_cp\_async\_b) = A\_MESSAGE(Receive\_cp\_async\_b)$
grd31: $ACTION(Receive\_cp\_async\_b) = Receive$
grd32: $ACTION(Send\_cp\_async\_b) = Send$
grd33: $Interaction\_Completed = 1$
grd34: $S\_MESSAGE(A\_Some\_cp\_sync\_b) = A\_MESSAGE(Send\_cp\_async\_b)$
grd35: $R\_trace\_b \in dom(R\_MESSAGE)$
grd36: $A\_MESSAGE(Send\_cp\_async\_b) = R\_MESSAGE(R\_trace\_b)$
grd37: $MESSAGE(A\_Some\_cp\_b) = S\_MESSAGE(A\_Some\_cp\_sync\_b)$
grd38: $MESSAGE(A\_Some\_cp\_b) \neq End\_message$
grd39: $SOURCE\_STATE(A\_Some\_cp\_b) = DESTINATION\_STATE(A\_Some\_cp\_b) \wedge DESTINATION\_STATE(A\_Some\_cp\_b) \in CP\_Final\_states$
grd40: $S\_SOURCE\_STATE(A\_Some\_cp\_sync\_b) = S\_DESTINATION\_STATE(A\_Some\_cp\_sync\_b) \wedge S\_DESTINATION\_STATE(A\_Some\_cp\_sync\_b) \in CP\_Final\_states$
grd41: $MESSAGE(A\_Some\_cp\_b) = Exchanged\_message$
grd42: $S\_MESSAGE(A\_Some\_cp\_sync\_b) = Exchanged\_message$
grd43: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$

with

S_Some_cp_b: $S\_Some\_cp\_b = A\_Some\_cp\_b$
Some_cp_sync_b: $Some\_cp\_sync\_b = A\_Some\_cp\_sync\_b$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{A\_Some\_cp\_b\}$
act2: $BUILT\_SYNCHRONE := BUILT\_SYNCHRONE \cup \{A\_Some\_cp\_sync\_b\}$
act3: $BUILT\_ASYNCHRONE := BUILT\_ASYNCHRONE \cup \{Send\_cp\_async\_b\} \cup \{Receive\_cp\_async\_b\}$

act4: $REDUCED\_TRACE := REDUCED\_TRACE \cup \{R\_trace\_b\}$
act5: $Reduced\_Trace\_index := Reduced\_Trace\_index + 1$
act6: $Interaction\_Completed := 0$
act7: $A\_index := A\_index + 1$
act8: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_Loop_Send-Receive ⟨convergent⟩
refines Add_Loop

any

A_Some_cp_b
A_Some_cp_sync_b
Send_cp_async_b
Receive_cp_async_b
R_trace_b

where

grd1: $Send\_cp\_async\_b \in dom(ACTION)$
grd2: $Receive\_cp\_async\_b \in dom(ACTION)$
grd3: $Send\_cp\_async\_b \in dom(A\_MESSAGE)$
grd4: $Receive\_cp\_async\_b \in dom(A\_MESSAGE)$
grd5: $A\_Some\_cp\_b \mapsto A\_Some\_cp\_sync\_b \in EQUIVALENCE$
grd6: $A\_Some\_cp\_sync\_b \mapsto R\_trace\_b \in SYNCHRONISABILITY$
grd7: $A\_Some\_cp\_b \in DC$
grd8: $(BUILT\_CP \neq \varnothing \Rightarrow A\_Some\_cp\_b \in ISeqF)$
grd9: $A\_Some\_cp\_sync\_b \in CPs\_SYNC\_B$
grd10: $Send\_cp\_async\_b \in PEERs\_B$
grd11: $Receive\_cp\_async\_b \in PEERs\_B$
grd12: $R\_trace\_b \in R\_TRACE\_B$
grd13: $SOURCE\_STATE(A\_Some\_cp\_b) \in CP\_Final\_states$
grd14: $S\_SOURCE\_STATE(A\_Some\_cp\_sync\_b) \in CP\_Final\_states$
grd15: $A\_MESSAGE(Send\_cp\_async\_b) = A\_MESSAGE(Receive\_cp\_async\_b)$
grd16: $ACTION(Receive\_cp\_async\_b) = Receive$
grd17: $ACTION(Send\_cp\_async\_b) = Send$
grd18: $Interaction\_Completed = 1$
grd19: $\exists Trans \cdot (Trans \in BUILT\_CP \land Trans \in dom(PEER\_SOURCE) \land Trans \in dom(PEER\_DESTINATION)) \land ((PEER\_SOURCE(A\_Some\_cp\_b) = PEER\_SOURCE(Trans)) \lor (PEER\_DESTINATION(A\_Some\_cp\_b) = PEER\_SOURCE(Trans)))$
grd20: $S\_MESSAGE(A\_Some\_cp\_sync\_b) = A\_MESSAGE(Send\_cp\_async\_b)$
grd21: $A\_MESSAGE(Send\_cp\_async\_b) = R\_MESSAGE(R\_trace\_b)$
grd22: $MESSAGE(A\_Some\_cp\_b) = S\_MESSAGE(A\_Some\_cp\_sync\_b)$
grd23: $BUILT\_SYNCHRONE \neq \varnothing$
grd24: $MESSAGE(A\_Some\_cp\_b) \neq End\_message$
grd25: $\exists Trans \cdot (Trans \in BUILT\_CP \land Trans \in dom(SOURCE\_STATE)) \land DESTINATION\_STATE(A\_Some\_cp\_b) = SOURCE\_STATE(Trans) \land SOURCE\_STATE(A\_Some\_cp\_b) \neq DESTINATION\_STATE(A\_Some\_cp\_b)$
grd26: $\exists S\_Trans \cdot (S\_Trans \in BUILT\_SYNCHRONE) \land S\_DESTINATION\_STATE(A\_Some\_cp\_sync\_b) = S\_SOURCE\_STATE(S\_Trans) \land S\_SOURCE\_STATE(A\_Some\_cp\_sync\_b) \neq S\_DESTINATION\_STATE(A\_Some\_cp\_sync\_b)$
grd27: $MESSAGE(A\_Some\_cp\_b) = Exchanged\_message$
grd28: $S\_MESSAGE(A\_Some\_cp\_sync\_b) = Exchanged\_message$
grd29: $Prophecy\_of\_Sent\_Messages - Number\_of\_send > 0$

with

S_Some_cp_b: $S\_Some\_cp\_b = A\_Some\_cp\_b$
Some_cp_sync_b: $Some\_cp\_sync\_b = A\_Some\_cp\_sync\_b$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{A\_Some\_cp\_b\}$
act2: $BUILT\_SYNCHRONE := BUILT\_SYNCHRONE \cup \{A\_Some\_cp\_sync\_b\}$
act3: $BUILT\_ASYNCHRONE := BUILT\_ASYNCHRONE \cup \{Send\_cp\_async\_b\} \cup \{Receive\_cp\_async\_b\}$

act4: $REDUCED\_TRACE := REDUCED\_TRACE \cup \{R\_trace\_b\}$
act5: $Reduced\_Trace\_index := Reduced\_Trace\_index + 1$
act6: $Interaction\_Completed := 0$
act7: $A\_index := A\_index + 1$
act8: $CP\_Final\_states := (CP\_Final\_states \cup \{DESTINATION\_STATE(A\_Some\_cp\_b)\}) \setminus \{SOURCE\_STATE(A\_Some\_cp\_b)\}$
act9: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_End ⟨ordinary⟩
refines Add_End

any
A_Some_cp_b

where
grd1: $A\_Some\_cp\_b \in dom(MESSAGE)$
grd2: $A\_Some\_cp\_b \in dom(SOURCE\_STATE)$
grd3: $A\_Some\_cp\_b \in DC$
grd4: $Prophecy\_of\_Sent\_Messages - Number\_of\_send = 0$
grd5: $SOURCE\_STATE(A\_Some\_cp\_b) \in CP\_Final\_states$
grd6: $end = 0$
grd7: $\forall A\_Trans \cdot (A\_Trans \in A\_TRACES \wedge MESSAGE(A\_Some\_cp\_b) = End\_message \wedge A\_TRACE \neq \varnothing) \Rightarrow A\_Trans \mapsto queue \in WF$
grd8: $MESSAGE(A\_Some\_cp\_b) = End\_message$

with
S_Some_cp_b: $S\_Some\_cp\_b = A\_Some\_cp\_b$

then
act1: $BUILT\_CP := BUILT\_CP \cup \{A\_Some\_cp\_b\}$
act2: $Last\_cp\_trans := A\_Some\_cp\_b$
act3: $end := 1$

end

END

Annexe B

Modèle B-événementiel de réparation

CONTEXT LTS_CONTEXT

SETS

PEERS

MESSAGES

CP_STATES

CONSTANTS

CPs_B Set of CP basic transitions

LAST_SENDER_PEERS Fuction that returns the last sender peers

PEER_SOURCE Function returns source peer

LAST_RECEIVER_PEERS Fuction that returns the last receiver peers

INDEX Fuction that returns the transition index

SOURCE_STATE Fuction returns the transition source state

DESTINATION_STATE Fuction returns the transition destination state

MESSAGE Fuction returns the transition message

LABEL Fuction returns the transition label

PEER_DESTINATION Fuction returns the transition destination peer

NON_DETERMINIST_CP Non desterministic property

DC Determinist property

BR_CP_FINAL_STATES Branches final states

ISeqF Independent sequence freedom property

PCF_BRANCHES_SET Parallel choice property

Repare_propositions

Repair_src_state

Repair_dst_state

BUILT_CP_SET

PCF_Repair_BRANCHES_SET

Sequence_OK

Branch_OK

OK

AXIOMS

CPs_B: $CPs\_B \subseteq CP\_STATES \times PEERS \times MESSAGES \times PEERS \times CP\_STATES \times \mathbb{N}$

Repare_propositions: $Repare\_propositions \subseteq CP\_STATES \times PEERS \times MESSAGES \times PEERS \times CP\_STATES \times \mathbb{N}$

axm1: $BUILT\_CP\_SET = CPs\_B \cup Repare\_propositions$

axm2: $finite(CPs\_B)$

axm3: $finite(PEERS) \land card(PEERS) \geq 2$

axm4: $finite(MESSAGES) \land card(MESSAGES) \geq 1$

INDEX: $INDEX \in BUILT\_CP\_SET \leftrightarrow \mathbb{N}$

FCT_INDEX: $INDEX = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto idx | state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in BUILT\_CP\_SET \land idx \in \mathbb{N} \land idx = Index\}$

SOURCE_STATE: $SOURCE\_STATE \in BUILT\_CP\_SET \leftrightarrow CP\_STATES$

FCT_SOURCE_STATE: $SOURCE\_STATE = \{st\_so \mapsto Peer\_so \mapsto msg \mapsto Peer\_de \mapsto st\_de \mapsto Index \mapsto state\_src | st\_so \mapsto Peer\_so \mapsto msg \mapsto Peer\_de \mapsto st\_de \mapsto Index \in BUILT\_CP\_SET \land state\_src \in CP\_STATES \land state\_src = st\_so\}$

DESTINATION_STATE: $DESTINATION\_STATE \in BUILT\_CP\_SET \leftrightarrow CP\_STATES$

FCT_DESTINATION_STATE: $DESTINATION\_STATE = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto state\_dest | state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in BUILT\_CP\_SET \land state\_dest \in CP\_STATES \land state\_dest = state\_dst\}$

PEER_SOURCE: $PEER\_SOURCE \in BUILT\_CP\_SET \leftrightarrow PEERS$

FCT_PEER_SOURCE: $PEER\_SOURCE = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto peer\_source | state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in BUILT\_CP\_SET \land peer\_source \in PEERS \land peer\_source = Peer\_src\}$

LAST_SENDER_PEERS: $LAST\_SENDER\_PEERS \in CP\_STATES \rightarrow PEERS$

FCT_LAST_SENDER_PEERS: $\forall state, Trans1, Trans2 \cdot ($
$state \in CP\_STATES \land Trans1 \in dom(DESTINATION\_STATE) \land$
$Trans2 \in dom(SOURCE\_STATE) \land Trans1 \in dom(PEER\_SOURCE) \land$
$state = SOURCE\_STATE(Trans2) \land state = DESTINATION\_STATE(Trans1) \land$
$\{Trans1, Trans2\} \subseteq CPs\_B \land DESTINATION\_STATE(Trans1) = SOURCE\_STATE(Trans2))$
$\Rightarrow$
$LAST\_SENDER\_PEERS(state) = PEER\_SOURCE(Trans1)$

PEER_DESTINATION: $PEER\_DESTINATION \in BUILT\_CP\_SET \rightarrow PEERS$

FCT_PEER_DESTINATION: $PEER\_DESTINATION = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto peer\_destination | state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in BUILT\_CP\_SET \land peer\_destination \in PEERS \land peer\_destination = Peer\_dst\}$

LAST_RECEIVER_PEERS: $LAST\_RECEIVER\_PEERS \in CP\_STATES \rightarrow PEERS$

FCT_LAST_RECEIVER_PEERS: $\forall state, Trans1, Trans2 \cdot ($
$state \in CP\_STATES \land Trans1 \in dom(DESTINATION\_STATE)$
$\land Trans2 \in dom(SOURCE\_STATE) \land Trans1 \in dom(PEER\_DESTINATION) \land$
$state = SOURCE\_STATE(Trans2) \land state = DESTINATION\_STATE(Trans1) \land$
$\{Trans1, Trans2\} \subseteq BUILT\_CP\_SET \land$
$DESTINATION\_STATE(Trans1) = SOURCE\_STATE(Trans2))$
$\Rightarrow$
$LAST\_RECEIVER\_PEERS(state) = PEER\_DESTINATION(Trans1)$

MESSAGE: $MESSAGE \in BUILT\_CP\_SET \leftrightarrow MESSAGES$

FCT_MESSAGE: $MESSAGE = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto message | state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in BUILT\_CP\_SET \land message \in MESSAGES \land message = msg\}$

LABEL: $LABEL \in BUILT\_CP\_SET \leftrightarrow PEERS \times MESSAGES \times PEERS$

FCT_LABEL: $LABEL = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto Label | state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in BUILT\_CP\_SET \land Label \in PEERS \times MESSAGES \times PEERS \land Label = Peer\_src \mapsto msg \mapsto Peer\_dst\}$

NON_DETERMINIST_CP: $NON\_DETERMINIST\_CP \subseteq BUILT\_CP\_SET$

FCT_NON_DETERMINIST_CP: $\forall Trans2, Trans1 \cdot (Trans1 \in BUILT\_CP\_SET \land Trans2 \in BUILT\_CP\_SET \land SOURCE\_STATE(Trans1) = SOURCE\_STATE(Trans2) \land LABEL(Trans1) = LABEL(Trans2) \land DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2))$
$\Rightarrow$
$\{Trans1, Trans2\} \subseteq NON\_DETERMINIST\_CP$

DETERMINIST_CP: $DC = BUILT\_CP\_SET \setminus NON\_DETERMINIST\_CP$

BR_CP_FINAL_STATES: $BR\_CP\_FINAL\_STATES \in CP\_STATES \rightarrow \mathbb{P}(CP\_STATES)$

FCT_BR_CP_FINAL_STATES: $\forall Trans1, Trans2 \cdot (Trans1 \in BUILT\_CP\_SET \land Trans2 \in BUILT\_CP\_SET \land SOURCE\_STATE(Trans1) = SOURCE\_STATE(Trans2) \land DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2))$
$\Rightarrow$
$BR\_CP\_FINAL\_STATES(SOURCE\_STATE(Trans1)) =$
$\{DESTINATION\_STATE(Trans1), DESTINATION\_STATE(Trans2)\}$

ISeqF: $ISeqF \subseteq BUILT\_CP\_SET$

FCT_ISeqF: $\forall cp\_b \cdot ($
$cp\_b \in BUILT\_CP\_SET \land$
$(PEER\_SOURCE(cp\_b) = LAST\_SENDER\_PEERS(SOURCE\_STATE(cp\_b))$
$\lor$
$PEER\_SOURCE(cp\_b) = LAST\_RECEIVER\_PEERS(SOURCE\_STATE(cp\_b))))$
$\Rightarrow$
$\{cp\_b\} \subseteq ISeqF$

PCF_Repair_BRANCHES_SET: $PCF\_Repair\_BRANCHES\_SET \in CP\_STATES \rightarrow \mathbb{P}(BUILT\_CP\_SET)$

FCT_PCF_Repair_BRANCHES_SET: $\forall Trans1, Trans2 \cdot ($
$Trans1 \in BUILT\_CP\_SET \land Trans2 \in BUILT\_CP\_SET \land$
$SOURCE\_STATE(Trans1) = SOURCE\_STATE(Trans2) \land$
$PEER\_SOURCE(Trans1) = PEER\_SOURCE(Trans2) \land$
$DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2))$
$\Rightarrow$
$PCF\_Repair\_BRANCHES\_SET(SOURCE\_STATE(Trans1)) = \{Trans1, Trans2\}$

Repair_src_state: $Repair\_src\_state \in Repare\_propositions \leftrightarrow CP\_STATES$
On peut utilise SOURCE8STATE vu qu'on a changé part tout le CPs_B

FCT_Repair_src_state: $Repair\_src\_state = \{st\_so \mapsto Peer\_so \mapsto msg \mapsto Peer\_de \mapsto st\_de \mapsto Index \mapsto state\_src | st\_so \mapsto Peer\_so \mapsto msg \mapsto Peer\_de \mapsto st\_de \mapsto Index \in Repare\_propositions \wedge state\_src \in CP\_STATES \wedge state\_src = st\_so\}$

Repair_dst_state: $Repair\_dst\_state \in Repare\_propositions \leftrightarrow CP\_STATES$

FCT_Repair_dst_state: $Repair\_dst\_state = \{state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \mapsto state\_dest | state\_src \mapsto Peer\_src \mapsto msg \mapsto Peer\_dst \mapsto state\_dst \mapsto Index \in Repare\_propositions \wedge state\_dest \in CP\_STATES \wedge state\_dest = state\_dst\}$

PCF_BRANCHES_SET: $PCF\_BRANCHES\_SET \in CP\_STATES \rightarrow \mathbb{P}(CPs\_B)$

FCT_PCF_BRANCHES_SET: $\forall Trans1, Trans2 \cdot ($
$Trans1 \in CPs\_B \wedge Trans2 \in CPs\_B \wedge$
$SOURCE\_STATE(Trans1) = SOURCE\_STATE(Trans2) \wedge$
$PEER\_SOURCE(Trans1) = PEER\_SOURCE(Trans2) \wedge$
$DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2))$
$\Rightarrow$
$PCF\_BRANCHES\_SET(SOURCE\_STATE(Trans1)) = \{Trans1, Trans2\}$

Sequence_OK: $Sequence\_OK \subseteq BUILT\_CP\_SET$

FCT_Sequence_OK: $\forall Trans \cdot (Trans \in BUILT\_CP\_SET \wedge \{Trans\} \subseteq ISeqF) \Rightarrow Trans \in Sequence\_OK$

Branch_OK: $Branch\_OK \subseteq BUILT\_CP\_SET$

FCT_Branch_OK: $\forall Trans1, Trans2 \cdot (Trans1 \in BUILT\_CP\_SET \wedge Trans2 \in BUILT\_CP\_SET \wedge \{Trans1, Trans2\} = PCF\_BRANCHES\_SET(SOURCE\_STATE(Trans1)))$
$\Rightarrow$
$\{Trans1, Trans2\} \subseteq Branch\_OK$

OK: $OK = Sequence\_OK \cup Branch\_OK$

END

MACHINE LTS_model
SEES LTS_CONTEXT
VARIABLES

BUILT_CP Built Conversation Protocol which is empty at the initialisation

CP_Initial_state The initial state of CP

CP_Final_states The set of CP final states

Prophecy_of_Sent_Messages Prophecy variable of sent messages

Number_of_send The number of messages that we want to send

end Variable that manages events

INVARIANTS

inv1: $BUILT\_CP \subseteq BUILT\_CP\_SET$

inv7: $finite(BUILT\_CP)$

inv2: $CP\_Initial\_state \in CP\_STATES$

inv3: $CP\_Final\_states \subseteq CP\_STATES$

inv5: $Prophecy\_of\_Sent\_Messages \in \mathbb{N}$

inv6: $Number\_of\_send \in \mathbb{N}$

inv8: $end \in \{0, 1\}$

ISeqF-PCF_OK:

$\forall Trans \cdot (Trans \in BUILT\_CP \land Trans \in dom(MESSAGE) \land MESSAGE(Trans) \neq End)$

$\Rightarrow$

$Trans \in OK$

∀Trans·(Trans ∈ BUILT_CP ∧ BUILT_CP ≠ ∅ ∧ Trans∈ dom(MESSAGE) ∧ MESSAGE(Trans) ≠ End)

⇒

Trans ∈ OK

VARIANT

$Prophecy\_of\_Sent\_Messages - Number\_of\_send$

EVENTS

Initialisation

begin

act1: $BUILT\_CP := \varnothing$

act2: $CP\_Initial\_state :\in CP\_STATES$

act3: $CP\_Final\_states :\in \{CP\_STATES\}$

act4: $Number\_of\_send := 0$

act5: $Prophecy\_of\_Sent\_Messages :\in \mathbb{N}$

act6: $end := 0$

end

Event Add-Sequence ⟨convergent⟩

any

Some_cp_b

where

grd1: $Some\_cp\_b \in dom(DESTINATION\_STATE)$

grd2: $Some\_cp\_b \in dom(SOURCE\_STATE)$

grd3: $Some\_cp\_b \in CPs\_B$

grd4: $\forall Trans \cdot (Trans \in BUILT\_CP \land BUILT\_CP \neq \varnothing \land Trans \in dom(SOURCE\_STATE))$

$\Rightarrow$

$DESTINATION\_STATE(Some\_cp\_b) \neq SOURCE\_STATE(Trans)$

grd5: $Some\_cp\_b \in OK$

grd6: $BUILT\_CP \neq \varnothing \Rightarrow Some\_cp\_b \in ISeqF$

grd7: $Prophecy\_of\_Sent\_Messages - Number\_of\_send \in \mathbb{N}$

grd8: $SOURCE\_STATE(Some\_cp\_b) \in CP\_Final\_states$

grd9: $BUILT\_CP \neq \varnothing \Rightarrow Some\_cp\_b \in OK$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{Some\_cp\_b\}$

act2: $CP\_Final\_states := (CP\_Final\_states \cup$

$\{DESTINATION\_STATE(Some\_cp\_b)\}) \setminus \{SOURCE\_STATE(Some\_cp\_b)\}$

act3: $Number\_of\_send := Number\_of\_send + 1$

end

Event ISeqF_Restore ⟨ordinary⟩

any

Some_reparation
cp_b_Breaks_ISeqF

where

grd1: $Some\_reparation \in dom(Repair\_src\_state)$
grd2: $cp\_b\_Breaks\_ISeqF \in dom(SOURCE\_STATE)$
grd3: $Some\_reparation \in dom(Repair\_dst\_state)$
grd4: $cp\_b\_Breaks\_ISeqF \in dom(DESTINATION\_STATE)$
grd5: $Some\_reparation \in Repare\_propositions$
grd6: $BUILT\_CP \neq \varnothing \wedge Some\_reparation \in ISeqF$
grd7: $Repair\_src\_state(Some\_reparation) \in CP\_Final\_states$
grd8: $\exists Trans \cdot (Trans \notin ISeqF)$
grd9: $cp\_b\_Breaks\_ISeqF \in CPs\_B$
grd10: $BUILT\_CP \neq \varnothing \Rightarrow cp\_b\_Breaks\_ISeqF \in ISeqF$
grd11: $SOURCE\_STATE(cp\_b\_Breaks\_ISeqF) = Repair\_dst\_state(Some\_reparation)$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{Some\_reparation\} \cup \{cp\_b\_Breaks\_ISeqF\}$
act2: $CP\_Final\_states := (CP\_Final\_states \cup \{DESTINATION\_STATE(cp\_b\_Breaks\_ISeqF)\}) \setminus \{Repair\_src\_state(Some\_reparation)\}$

end

Event Add_Choice ⟨convergent⟩

any

Branches Some transition of the set of basic deterministic transitions
branch

where

grd1: $branch \in dom(SOURCE\_STATE)$
grd2: $branch \in dom(MESSAGE)$
grd3: $Branches \subseteq DC$
grd4: $branch \in Branches$
ISeq: $BUILT\_CP \neq \varnothing \Rightarrow branch \in ISeqF$
PC: $Branches = PCF\_BRANCHES\_SET(SOURCE\_STATE(branch))$
grd5: $MESSAGE(branch) \neq End$
grd6: $Prophecy\_of\_Sent\_Messages - Number\_of\_send \in \mathbb{N}$
grd7: $finite(PCF\_BRANCHES\_SET(SOURCE\_STATE(branch)))$
grd8: $SOURCE\_STATE(branch) \in CP\_Final\_states$

then

act1: $BUILT\_CP := BUILT\_CP \cup Branches$
act2: $CP\_Final\_states := (CP\_Final\_states \cup BR\_CP\_FINAL\_STATES(SOURCE\_STATE(branch))) \setminus \{SOURCE\_STATE(branch)\}$
act3: $Number\_of\_send := Number\_of\_send + card(Branches)$

end

Event PCF_Restore ⟨ordinary⟩

any

Branches_reparation
branch_reparation
cp_b_Breaks_PCF

where

grd1: $branch\_reparation \in dom(DESTINATION\_STATE)$
grd2: $branch\_reparation \in dom(SOURCE\_STATE)$
grd3: $branch\_reparation \in dom(Repair\_dst\_state)$
grd4: $cp\_b\_Breaks\_PCF \in dom(DESTINATION\_STATE)$
grd5: $finite(PCF\_Repair\_BRANCHES\_SET(SOURCE\_STATE(branch\_reparation)))$
grd6: $\exists Branches, branch \cdot (branch \in OK \wedge Branches \subseteq OK \wedge branch \in dom(SOURCE\_STATE) \wedge branch \in Branches \wedge Branches \neq PCF\_Repair\_BRANCHES\_SET(SOURCE\_STATE(branch)))$

grd7: $Branches\_reparation \subseteq OK$

grd8: $branch\_reparation \in OK$
grd9: $cp\_b\_Breaks\_PCF \in OK$
grd10: $Branches\_reparation \subseteq DC$
grd11: $BUILT\_CP \neq \varnothing \land branch\_reparation \in ISeqF$
grd12: $Branches\_reparation = PCF\_Repair\_BRANCHES\_SET(SOURCE\_STATE(branch\_reparation))$

grd13: $branch\_reparation \in Branches\_reparation$
grd14: $Branches\_reparation \subseteq Repare\_propositions$
grd15: $Repair\_src\_state(branch\_reparation) \in CP\_Final\_states$
grd16: $SOURCE\_STATE(branch\_reparation) \in CP\_Final\_states$
grd17: $MESSAGE(branch\_reparation) \neq End$
grd18: $cp\_b\_Breaks\_PCF \in CPs\_B$
grd19: $BUILT\_CP \neq \varnothing \Rightarrow cp\_b\_Breaks\_PCF \in ISeqF$
grd20: $SOURCE\_STATE(cp\_b\_Breaks\_PCF) = Repair\_dst\_state(branch\_reparation)$
grd21: $SOURCE\_STATE(cp\_b\_Breaks\_PCF) = DESTINATION\_STATE(branch\_reparation)$

then

act1: $BUILT\_CP := BUILT\_CP \cup Branches\_reparation \cup \{cp\_b\_Breaks\_PCF\}$
act2: $CP\_Final\_states := (CP\_Final\_states \cup$
$BR\_CP\_FINAL\_STATES(SOURCE\_STATE(branch\_reparation)) \cup$
$\{DESTINATION\_STATE(cp\_b\_Breaks\_PCF)\})\backslash(\{SOURCE\_STATE(branch\_reparation)\} \cup$
$\{SOURCE\_STATE(cp\_b\_Breaks\_PCF)\})$

end

Event Add_Self-Loop ⟨convergent⟩

any

Some_cp_b

where

grd1: $Some\_cp\_b \in dom(SOURCE\_STATE)$
grd2: $Some\_cp\_b \in DC \land (BUILT\_CP \neq \varnothing \Rightarrow Some\_cp\_b \in ISeqF)$
grd3: $Prophecy\_of\_Sent\_Messages - Number\_of\_send \in \mathbb{N}$
grd4: $Some\_cp\_b \in dom(MESSAGE)$
grd5: $MESSAGE(Some\_cp\_b) \neq End$
grd6: $SOURCE\_STATE(Some\_cp\_b) \in CP\_Final\_states$
grd7: $SOURCE\_STATE(Some\_cp\_b) = DESTINATION\_STATE(Some\_cp\_b)$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{Some\_cp\_b\}$
act2: $Number\_of\_send := Number\_of\_send + 1$

end

Event Add_Loop ⟨convergent⟩

any

Some_cp_b

where

grd1: $BUILT\_CP \neq \varnothing$
grd2: $Some\_cp\_b \in DC \land (BUILT\_CP \neq \varnothing \Rightarrow Some\_cp\_b \in ISeqF)$
grd3: $Some\_cp\_b \in dom(DESTINATION\_STATE)$
grd4: $\forall Trans \cdot (Trans \in BUILT\_CP \land Trans \in dom(SOURCE\_STATE))$
$\Rightarrow$
$DESTINATION\_STATE(Some\_cp\_b) = SOURCE\_STATE(Trans)$
grd5: $Some\_cp\_b \in dom(PEER\_SOURCE)$
grd6: $Some\_cp\_b \in dom(SOURCE\_STATE)$
grd7: $Some\_cp\_b \in dom(DESTINATION\_STATE)$
grd8: $SOURCE\_STATE(Some\_cp\_b) \in dom(LAST\_RECEIVER\_PEERS)$
grd9: $Some\_cp\_b \in dom(MESSAGE)$
grd10: $Prophecy\_of\_Sent\_Messages - Number\_of\_send \in \mathbb{N}$
grd11: $MESSAGE(Some\_cp\_b) \neq End$
grd12: $SOURCE\_STATE(Some\_cp\_b) \in CP\_Final\_states$
grd13: $\forall Trans \cdot (Trans \in BUILT\_CP \land Trans \in dom(PEER\_SOURCE))$
$\Rightarrow$
$((PEER\_SOURCE(Some\_cp\_b) = PEER\_SOURCE(Trans))$

$\vee$

$(PEER\_DESTINATION(Some\_cp\_b) = PEER\_SOURCE(Trans)))$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{Some\_cp\_b\}$

act2: $Number\_of\_send := Number\_of\_send + 1$

act3: $CP\_Final\_states := CP\_Final\_states \setminus \{SOURCE\_STATE(Some\_cp\_b)\}$

end

Event Add_End ⟨ordinary⟩

any

Some_cp_b

where

grd1: $Some\_cp\_b \in dom(MESSAGE)$

grd2: $Some\_cp\_b \in dom(SOURCE\_STATE)$

grd3: $Some\_cp\_b \in DC$

grd4: $Prophecy\_of\_Sent\_Messages - Number\_of\_send = 0$

grd5: $SOURCE\_STATE(Some\_cp\_b) \in CP\_Final\_states$

grd6: $end = 0$

grd7: $MESSAGE(Some\_cp\_b) = End$

then

act1: $BUILT\_CP := BUILT\_CP \cup \{Some\_cp\_b\}$

act2: $end := 1$

end

END

Annexe C

Exemples d'instanciation

CONTEXT CS1
EXTENDS LTS_ASYNC_CONTEXT
CONSTANTS

connect
access
logout
End
cl
int
appli
s0
s1
s2
Pend
s0_cl
s1_cl
s0_int
s1_int
s0_appli
s

AXIOMS

axm1: $partition(PEERS, \{cl\}, \{appli\}, \{int\}, \{Pend\})$

axm2: $partition(MESSAGES, \{connect\}, \{access\}, \{logout\}, \{End\})$

axm3: $partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\})$

axm4: $CPs\_B = \{s0 \mapsto cl \mapsto connect \mapsto int \mapsto s1 \mapsto 1, s1 \mapsto cl \mapsto access \mapsto appli \mapsto s1 \mapsto 2, s1 \mapsto cl \mapsto logout \mapsto int \mapsto s0 \mapsto 3, s0 \mapsto Pend \mapsto End \mapsto Pend \mapsto s2 \mapsto 4\}$

axm5: $CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto connect \mapsto int \mapsto cl \mapsto Receive \mapsto connect \mapsto s1 \mapsto 1, s1 \mapsto Send \mapsto access \mapsto appli \mapsto cl \mapsto Receive \mapsto access \mapsto s1 \mapsto 2, s1 \mapsto Send \mapsto logout \mapsto int \mapsto cl \mapsto Receive \mapsto logout \mapsto s0 \mapsto 3, s0 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s2 \mapsto 4\}$

axm6: $partition(A\_STATES, \{s0\_cl\}, \{s1\_cl\}, \{s0\_int\}, \{s1\_int\}, \{s0\_appli\})$

axm7: $partition(A\_TRACE\_STATES, \{s\})$

axm8: $PEERs\_B = \{((s0\_cl \mapsto (Send \mapsto connect \mapsto int) \mapsto 1) \mapsto s1\_cl), ((s0\_int \mapsto (Receive \mapsto connect \mapsto cl) \mapsto 1) \mapsto s1\_int), ((s1\_cl \mapsto (Send \mapsto access \mapsto appli) \mapsto 2) \mapsto s1\_cl), ((s0\_appli \mapsto (Receive \mapsto access \mapsto cl) \mapsto 2) \mapsto s0\_appli), ((s1\_cl \mapsto (Send \mapsto logout \mapsto int) \mapsto 3) \mapsto s0\_cl), ((s1\_int \mapsto (Receive \mapsto logout \mapsto cl) \mapsto 3) \mapsto s0\_int)\}$

axm9: $R\_TRACE\_B = \{s0 \mapsto cl \mapsto connect \mapsto int \mapsto s1 \mapsto 1, s1 \mapsto cl \mapsto access \mapsto appli \mapsto s1 \mapsto 2, s1 \mapsto cl \mapsto logout \mapsto int \mapsto s0 \mapsto 3, s0 \mapsto Pend \mapsto End \mapsto Pend \mapsto s2 \mapsto 4\}$

axm10: $A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto cl \mapsto connect \mapsto int \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto cl \mapsto connect \mapsto int \mapsto s \mapsto 2 \mapsto 2, s \mapsto 2 \mapsto Send \mapsto cl \mapsto access \mapsto appli \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto cl \mapsto access \mapsto appli \mapsto s \mapsto 4 \mapsto 4, s \mapsto 2 \mapsto Send \mapsto cl \mapsto logout \mapsto int \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto cl \mapsto logout \mapsto int \mapsto s \mapsto 4 \mapsto 4, s \mapsto 4 \mapsto Send \mapsto cl \mapsto logout \mapsto int \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto cl \mapsto logout \mapsto int \mapsto s \mapsto 6 \mapsto 6, s \mapsto 4 \mapsto Send \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 6 \mapsto 6, s \mapsto 6 \mapsto Send \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 8 \mapsto 8\}$

axm11: $S\_Next\_States = \{\{((s0\_cl \mapsto (Send \mapsto connect \mapsto int) \mapsto 1) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\}, \{((s0\_int \mapsto (Receive \mapsto connect \mapsto cl) \mapsto 1) \mapsto s1\_int)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_cl \mapsto (Send \mapsto access \mapsto appli) \mapsto 2) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s0\_appli \mapsto (Receive \mapsto access \mapsto cl) \mapsto 2) \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_cl \mapsto (Send \mapsto logout \mapsto int) \mapsto 3) \mapsto s0\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_int \mapsto (Receive \mapsto logout \mapsto cl) \mapsto 3) \mapsto s0\_int)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\}\}$

axm12: $A\_Next\_States = \{\{((s0\_cl \mapsto (Send \mapsto connect \mapsto int) \mapsto 1) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\} \mapsto \varnothing \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\}, \{((s1\_cl \mapsto (Send \mapsto access \mapsto appli) \mapsto 2) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{int \mapsto connect \mapsto 0\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_cl \mapsto (Send \mapsto access \mapsto appli) \mapsto 2) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \varnothing \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_cl \mapsto (Send \mapsto access \mapsto appli) \mapsto 2) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{appli \mapsto access \mapsto 1\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_cl \mapsto (Send \mapsto access \mapsto appli) \mapsto 2) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{int \mapsto connect \mapsto 0, appli \mapsto access \mapsto 1\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s0\_int \mapsto (Receive \mapsto connect \mapsto cl) \mapsto 1) \mapsto s1\_int)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\} \mapsto \{appli \mapsto access \mapsto 1, int \mapsto connect \mapsto 0\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s0\_int \mapsto (Receive \mapsto connect \mapsto cl) \mapsto 1) \mapsto s1\_int)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\} \mapsto \{int \mapsto connect \mapsto 0\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s0\_appli \mapsto (Receive \mapsto access \mapsto cl) \mapsto 2) \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{appli \mapsto access \mapsto 1\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s0\_appli \mapsto (Receive \mapsto access \mapsto cl) \mapsto 2) \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{appli \mapsto access \mapsto 1\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_cl \mapsto (Send \mapsto logout \mapsto int) \mapsto 3) \mapsto s0\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \varnothing \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_cl \mapsto (Send \mapsto logout \mapsto int) \mapsto 3) \mapsto s0\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \varnothing \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_int \mapsto (Receive \mapsto logout \mapsto cl) \mapsto 3) \mapsto s0\_int)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{int \mapsto logout \mapsto 1\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\}, \{((s1\_int \mapsto (Receive \mapsto logout \mapsto cl) \mapsto 3) \mapsto s0\_int)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \varnothing \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\}, \{((s1\_cl \mapsto (Send \mapsto logout \mapsto int) \mapsto 3) \mapsto s0\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \varnothing \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_cl \mapsto (Send \mapsto logout \mapsto int) \mapsto 3) \mapsto s0\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \varnothing \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\}, \{((s1\_int \mapsto (Receive \mapsto logout \mapsto cl) \mapsto 3) \mapsto s0\_int)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \{int \mapsto logout \mapsto 2\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\}, \{((s1\_int \mapsto (Receive \mapsto logout \mapsto cl) \mapsto 3) \mapsto s0\_int)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli)\} \mapsto \varnothing \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli)\}\}$

axm13: $Initial\_state\_value = s0$

axm14: $CP\_Final\_states\_value = \{s0\}$

axm15: $Prophecy\_value = 3$

axm16: $End\_message = End$

axm17: $A\_GS\_value = \{cl \mapsto s0\_cl, int \mapsto s0\_int, appli \mapsto s0\_appli\}$

axm18: $S\_GS\_value = \{cl \mapsto s0\_cl, int \mapsto s0\_int, appli \mapsto s0\_appli\}$

axm19: $Last\_cp\_trans\_value = s0 \mapsto cl \mapsto connect \mapsto int \mapsto s1 \mapsto 1$

description

END

CONTEXT CS2
EXTENDS LTS_ASYNC_CONTEXT
CONSTANTS

connect
access
logout
End
cl
int
appli
s0
s1
s2
Pend
setup
log
db
s3
s4
s0_cl
s1_cl
s0_int
s1_int
s2_int
s0_appli
s1_appli
s0_db
s

AXIOMS

axm1: $partition(PEERS, \{cl\}, \{appli\}, \{int\}, \{db\}, \{Pend\})$

axm2: $partition(MESSAGES, \{connect\}, \{setup\}, \{access\}, \{logout\}, \{log\}, \{End\})$

axm3: $partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\})$

axm4: $CPs\_B = \{s0 \mapsto cl \mapsto connect \mapsto int \mapsto s1 \mapsto 1, s1 \mapsto int \mapsto setup \mapsto appli \mapsto s2 \mapsto 2, s2 \mapsto cl \mapsto logout \mapsto int \mapsto s3 \mapsto 3, s3 \mapsto appli \mapsto log \mapsto db \mapsto s0 \mapsto 4, s0 \mapsto Pend \mapsto End \mapsto Pend \mapsto s4 \mapsto 5\}$

axm5: $CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto connect \mapsto int \mapsto cl \mapsto Receive \mapsto connect \mapsto s1 \mapsto 1, s1 \mapsto Send \mapsto setup \mapsto appli \mapsto int \mapsto Receive \mapsto setup \mapsto s2 \mapsto 2, s2 \mapsto Send \mapsto logout \mapsto int \mapsto cl \mapsto Receive \mapsto logout \mapsto s3 \mapsto 3, s3 \mapsto Send \mapsto log \mapsto db \mapsto appli \mapsto Receive \mapsto log \mapsto s0 \mapsto 4, s0 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s4 \mapsto 5\}$

axm6: $partition(A\_STATES, \{s0\_cl\}, \{s1\_cl\}, \{s0\_int\}, \{s1\_int\}, \{s2\_int\}, \{s0\_appli\}, \{s1\_appli\}, \{s0\_db\})$

axm7: $partition(A\_TRACE\_STATES, \{s\})$

axm8: $PEERs\_B = \{((s0\_cl \mapsto (Send \mapsto connect \mapsto int) \mapsto 1) \mapsto s1\_cl), ((s0\_int \mapsto (Receive \mapsto connect \mapsto cl) \mapsto 1) \mapsto s1\_int), ((s1\_int \mapsto (Send \mapsto setup \mapsto appli) \mapsto 2) \mapsto s2\_int), ((s0\_appli \mapsto (Receive \mapsto setup \mapsto int) \mapsto 2) \mapsto s1\_appli), ((s1\_cl \mapsto (Send \mapsto access \mapsto appli) \mapsto 2) \mapsto s1\_cl), ((s1\_appli \mapsto (Receive \mapsto access \mapsto cl) \mapsto 2) \mapsto s1\_appli), ((s1\_cl \mapsto (Send \mapsto logout \mapsto int) \mapsto 3) \mapsto s0\_cl), ((s2\_int \mapsto (Receive \mapsto logout \mapsto cl) \mapsto 3) \mapsto s0\_int), ((s1\_appli \mapsto (Send \mapsto log \mapsto db) \mapsto 4) \mapsto s0\_appli), ((s0\_db \mapsto (Receive \mapsto log \mapsto appli) \mapsto 4) \mapsto s0\_db)\}$

axm9: $R\_TRACE\_B = \{s0 \mapsto cl \mapsto connect \mapsto int \mapsto s1 \mapsto 1, s1 \mapsto int \mapsto setup \mapsto appli \mapsto s2 \mapsto 2, s2 \mapsto cl \mapsto logout \mapsto int \mapsto s3 \mapsto 3, s3 \mapsto appli \mapsto log \mapsto db \mapsto s0 \mapsto 4, s0 \mapsto Pend \mapsto End \mapsto Pend \mapsto s4 \mapsto 5\}$

axm10: $A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto cl \mapsto connect \mapsto int \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto cl \mapsto connect \mapsto int \mapsto s \mapsto 2 \mapsto 2, s \mapsto 2 \mapsto Send \mapsto int \mapsto setup \mapsto appli \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto int \mapsto setup \mapsto appli \mapsto s \mapsto 4 \mapsto 4, s \mapsto 4 \mapsto Send \mapsto cl \mapsto access \mapsto appli \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto cl \mapsto access \mapsto appli \mapsto s \mapsto 6 \mapsto 6, s \mapsto 4 \mapsto Send \mapsto cl \mapsto logout \mapsto int \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto cl \mapsto logout \mapsto int \mapsto s \mapsto 6 \mapsto 6, s \mapsto 6 \mapsto Send \mapsto appli \mapsto log \mapsto db \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto appli \mapsto log \mapsto db \mapsto s \mapsto 8 \mapsto 8\}$

axm11: $S\_Next\_States = \{\{((s0\_cl \mapsto (Send \mapsto connect \mapsto int) \mapsto 1) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}, \{((s0\_int \mapsto (Receive \mapsto connect \mapsto cl) \mapsto 1) \mapsto s1\_int)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}, \{((s1\_int \mapsto (Send \mapsto setup \mapsto appli) \mapsto 2) \mapsto s2\_int)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}, \{((s0\_appli \mapsto (Receive \mapsto setup \mapsto int) \mapsto 2) \mapsto s1\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s1\_cl \mapsto (Send \mapsto access \mapsto appli) \mapsto 2) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s1\_appli \mapsto (Receive \mapsto access \mapsto cl) \mapsto 2) \mapsto s1\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s1\_cl \mapsto (Send \mapsto logout \mapsto int) \mapsto 3) \mapsto s0\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s2\_int \mapsto (Receive \mapsto logout \mapsto cl) \mapsto 3) \mapsto s0\_int)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s1\_appli \mapsto (Send \mapsto log \mapsto db) \mapsto 4) \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}, \{((s0\_db \mapsto (Receive \mapsto log \mapsto appli) \mapsto 4) \mapsto s0\_db)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}\}$

axm12: $A\_Next\_States = \{\{((s0\_cl \mapsto (Send \mapsto connect \mapsto int) \mapsto 1) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \varnothing \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}, \{((s0\_int \mapsto (Receive \mapsto connect \mapsto cl) \mapsto 1) \mapsto s1\_int)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \{int \mapsto connect \mapsto 0\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}, \{((s0\_int \mapsto (Receive \mapsto connect \mapsto cl) \mapsto 1) \mapsto s1\_int)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \varnothing \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}, \{((s1\_int \mapsto (Send \mapsto setup \mapsto appli) \mapsto 2) \mapsto s2\_int)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s1\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \varnothing \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}, \{((s0\_appli \mapsto (Receive \mapsto setup \mapsto int) \mapsto 2) \mapsto s1\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \{appli \mapsto setup \mapsto 1\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s0\_appli \mapsto (Receive \mapsto setup \mapsto int) \mapsto 2) \mapsto s1\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \varnothing \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s1\_cl \mapsto (Send \mapsto access \mapsto appli) \mapsto 2) \mapsto s1\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \varnothing \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s1\_appli \mapsto (Receive \mapsto access \mapsto cl) \mapsto 2) \mapsto s1\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \{appli \mapsto access \mapsto 2\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s1\_appli \mapsto (Receive \mapsto access \mapsto cl) \mapsto 2) \mapsto s1\_appli)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \varnothing \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s1\_cl \mapsto (Send \mapsto logout \mapsto int) \mapsto 3) \mapsto s0\_cl)\} \mapsto \{(cl \mapsto s1\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \varnothing \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s2\_int \mapsto (Receive \mapsto logout \mapsto cl) \mapsto 3) \mapsto s0\_int)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \{int \mapsto logout \mapsto 2\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s2\_int \mapsto (Receive \mapsto logout \mapsto cl) \mapsto 3) \mapsto s0\_int)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s2\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \varnothing \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\}, \{((s1\_appli \mapsto (Send \mapsto log \mapsto db) \mapsto 4) \mapsto s0\_appli)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s1\_appli), (db \mapsto s0\_db)\} \mapsto \varnothing \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}, \{((s0\_db \mapsto (Receive \mapsto log \mapsto appli) \mapsto 4) \mapsto s0\_db)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \{db \mapsto log \mapsto 3\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}, \{((s0\_db \mapsto (Receive \mapsto log \mapsto appli) \mapsto 4) \mapsto s0\_db)\} \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\} \mapsto \varnothing \mapsto \{(cl \mapsto s0\_cl), (int \mapsto s0\_int), (appli \mapsto s0\_appli), (db \mapsto s0\_db)\}\}$

axm13: $Initial\_state\_value = s0$

axm14: $CP\_Final\_states\_value = \{s0\}$

axm15: $Prophecy\_value = 5$

axm16: $End\_message = End$

axm17: $A\_GS\_value = \{cl \mapsto s0\_cl, int \mapsto s0\_int, appli \mapsto s0\_appli, db \mapsto s0\_db\}$

axm18: $S\_GS\_value = \{cl \mapsto s0\_cl, int \mapsto s0\_int, appli \mapsto s0\_appli, db \mapsto s0\_db\}$

axm19: $Last\_cp\_trans\_value = s0 \mapsto cl \mapsto connect \mapsto int \mapsto s1 \mapsto 1$

END

CONTEXT CS3_realizable
EXTENDS LTS_ASYNC_CONTEXT
CONSTANTS

Buyer
Seller
Bank
Pend
PriceReq
Offre
PayReq
Sync0
Pay
Abort
Confirm
End
s0
s1
s2
s3
s4
s5
s6
s7
s8
s9
Sync1
s10
s11
s0_Buyer
s1_Buyer
s2_Buyer
s3_Buyer
s4_Buyer
s5_Buyer
s0_Seller
s1_Seller
s2_Seller
s3_Seller
s4_Seller
s0_Bank
s
s1_Bank
s2_Bank
s3_Bank
s4_Bank
s5_Bank
s6_Buyer
s6_Bank

AXIOMS

axm1: $partition(PEERS, \{Buyer\}, \{Seller\}, \{Bank\}, \{Pend\})$

axm2: $partition(MESSAGES, \{PriceReq\}, \{Sync0\}, \{Offre\}, \{PayReq\}, \{Sync1\}, \{Pay\}, \{Abort\}, \{Confirm\}, \{End\})$

axm3: $partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\}, \{s6\}, \{s7\}, \{s8\}, \{s9\}, \{s10\}, \{s11\})$

axm4: $CPs\_B = \{s0 \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s1 \mapsto 1, s0 \mapsto Seller \mapsto Sync0 \mapsto Buyer \mapsto s2 \mapsto 1, s2 \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s3 \mapsto 2, s3 \mapsto Seller \mapsto Offre \mapsto Buyer \mapsto s0 \mapsto 3, s1 \mapsto Bank \mapsto Sync1 \mapsto Buyer \mapsto s4 \mapsto 4, s4 \mapsto Buyer \mapsto Pay \mapsto Bank \mapsto s5 \mapsto 5, s5 \mapsto Bank \mapsto Abort \mapsto Buyer \mapsto s8 \mapsto 6, s5 \mapsto Bank \mapsto Confirm \mapsto Seller \mapsto s7 \mapsto 6, s5 \mapsto Bank \mapsto Confirm \mapsto Buyer \mapsto s6 \mapsto 6, s6 \mapsto Pend \mapsto End \mapsto Pend \mapsto s9 \mapsto 7, s7 \mapsto Pend \mapsto End \mapsto Pend \mapsto s10 \mapsto 8, s8 \mapsto Pend \mapsto End \mapsto Pend \mapsto s11 \mapsto 9\}$

axm5: $CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto PayReq \mapsto Bank \mapsto Seller \mapsto Receive \mapsto PayReq \mapsto s1 \mapsto 1, s0 \mapsto Send \mapsto Sync0 \mapsto Buyer \mapsto Seller \mapsto Receive \mapsto Sync0 \mapsto s2 \mapsto 1, s2 \mapsto Send \mapsto PriceReq \mapsto Seller \mapsto Buyer \mapsto Receive \mapsto PriceReq \mapsto s3 \mapsto 2, s3 \mapsto Send \mapsto Offre \mapsto Buyer \mapsto Seller \mapsto Receive \mapsto Offre \mapsto s0 \mapsto 3, s1 \mapsto Send \mapsto Sync1 \mapsto Buyer \mapsto Bank \mapsto Receive \mapsto Sync1 \mapsto s4 \mapsto 4, s4 \mapsto Send \mapsto Pay \mapsto Bank \mapsto Buyer \mapsto Receive \mapsto Pay \mapsto s5 \mapsto 5, s5 \mapsto Send \mapsto Abort \mapsto Buyer \mapsto Bank \mapsto Receive \mapsto Abort \mapsto s8 \mapsto 6, s5 \mapsto Send \mapsto Confirm \mapsto Seller \mapsto Bank \mapsto Receive \mapsto Confirm \mapsto s7 \mapsto 6, s5 \mapsto Send \mapsto Confirm \mapsto Buyer \mapsto Bank \mapsto Receive \mapsto Confirm \mapsto s6 \mapsto 6, s6 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s9 \mapsto 7, s7 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s10 \mapsto 8, s8 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s11 \mapsto 9\}$

axm6: $partition(A\_STATES, \{s0\_Buyer\}, \{s1\_Buyer\}, \{s2\_Buyer\}, \{s3\_Buyer\}, \{s4\_Buyer\}, \{s5\_Buyer\}, \{s6\_Buyer\}, \{s0\_Seller\}, \{s1\_Seller\}, \{s2\_Seller\}, \{s3\_Seller\}, \{s4\_Seller\}, \{s0\_Bank\}, \{s1\_Bank\}, \{s2\_Bank\}, \{s3\_Bank\}, \{s4\_Bank\}, \{s5\_Bank\}, \{s6\_Bank\})$

axm7: $partition(A\_TRACE\_STATES, \{s\})$

axm8: $PEERs\_B = \{((s0\_Seller \mapsto (Send \mapsto PayReq \mapsto Bank) \mapsto 1) \mapsto s2\_Seller), ((s0\_Bank \mapsto (Receive \mapsto PayReq \mapsto Seller) \mapsto 1) \mapsto s1\_Bank), ((s0\_Seller \mapsto (Send \mapsto Sync0 \mapsto Buyer) \mapsto 1) \mapsto s1\_Seller), ((s0\_Buyer \mapsto (Receive \mapsto Sync0 \mapsto Seller) \mapsto 1) \mapsto s1\_Buyer), ((s1\_Buyer \mapsto (Send \mapsto PriceReq \mapsto Seller) \mapsto 2) \mapsto s2\_Buyer), ((s1\_Seller \mapsto (Receive \mapsto PriceReq \mapsto Buyer) \mapsto 2) \mapsto s3\_Seller), ((s3\_Seller \mapsto (Send \mapsto Offre \mapsto Buyer) \mapsto 3) \mapsto s0\_Seller), ((s2\_Buyer \mapsto (Receive \mapsto Offre \mapsto Seller) \mapsto 3) \mapsto s0\_Buyer), ((s1\_Bank \mapsto (Send \mapsto Sync1 \mapsto Buyer) \mapsto 4) \mapsto s2\_Bank), ((s0\_Buyer \mapsto (Receive \mapsto Sync1 \mapsto Bank) \mapsto 4) \mapsto s3\_Buyer), ((s3\_Buyer \mapsto (Send \mapsto Pay \mapsto Bank) \mapsto 5) \mapsto s4\_Buyer), ((s2\_Bank \mapsto (Receive \mapsto Pay \mapsto Buyer) \mapsto 5) \mapsto s3\_Bank), ((s3\_Bank \mapsto (Send \mapsto Abort \mapsto Buyer) \mapsto 6) \mapsto s4\_Bank), ((s4\_Buyer \mapsto (Receive \mapsto Abort \mapsto Bank) \mapsto 6) \mapsto s5\_Buyer), ((s3\_Bank \mapsto (Send \mapsto Confirm \mapsto Seller) \mapsto 6) \mapsto s5\_Bank), ((s2\_Seller \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 6) \mapsto s4\_Seller), ((s3\_Bank \mapsto (Send \mapsto Confirm \mapsto Buyer) \mapsto 6) \mapsto s6\_Bank), ((s4\_Buyer \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 6) \mapsto s6\_Buyer)\}$

axm9: $R\_TRACE\_B = \{s0 \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s1 \mapsto 1, s0 \mapsto Seller \mapsto Sync0 \mapsto Buyer \mapsto s2 \mapsto 1, s2 \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s3 \mapsto 2, s3 \mapsto Seller \mapsto Offre \mapsto Buyer \mapsto s0 \mapsto 3, s1 \mapsto Bank \mapsto Sync1 \mapsto Buyer \mapsto s4 \mapsto 4, s4 \mapsto Buyer \mapsto Pay \mapsto Bank \mapsto s5 \mapsto 5, s5 \mapsto Bank \mapsto Abort \mapsto Buyer \mapsto s8 \mapsto 6, s5 \mapsto Bank \mapsto Confirm \mapsto Seller \mapsto s7 \mapsto 6, s5 \mapsto Bank \mapsto Confirm \mapsto Buyer \mapsto s6 \mapsto 6, s6 \mapsto Pend \mapsto End \mapsto Pend \mapsto s9 \mapsto 7, s7 \mapsto Pend \mapsto End \mapsto Pend \mapsto s10 \mapsto 8, s8 \mapsto Pend \mapsto End \mapsto Pend \mapsto s11 \mapsto 9\}$

axm10: $A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s \mapsto 2 \mapsto 2, s \mapsto 0 \mapsto Send \mapsto Seller \mapsto Sync0 \mapsto Buyer \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto Seller \mapsto Sync0 \mapsto Buyer \mapsto s \mapsto 2 \mapsto 2, s \mapsto 2 \mapsto Send \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s \mapsto 4 \mapsto 4, s \mapsto 4 \mapsto Send \mapsto Seller \mapsto Offre \mapsto Buyer \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto Seller \mapsto Offre \mapsto Buyer \mapsto s \mapsto 6 \mapsto 6, s \mapsto 6 \mapsto Send \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s \mapsto 8 \mapsto 8, s \mapsto 8 \mapsto Send \mapsto Bank \mapsto Sync1 \mapsto Buyer \mapsto s \mapsto 9 \mapsto 9, s \mapsto 9 \mapsto Receive \mapsto Bank \mapsto Sync1 \mapsto Buyer \mapsto s \mapsto 10 \mapsto 10, s \mapsto 10 \mapsto Send \mapsto Buyer \mapsto Pay \mapsto Bank \mapsto s \mapsto 11 \mapsto 11, s \mapsto 11 \mapsto Receive \mapsto Buyer \mapsto Pay \mapsto Bank \mapsto s \mapsto 12 \mapsto 12, s \mapsto 12 \mapsto Send \mapsto Bank \mapsto Abort \mapsto Buyer \mapsto s \mapsto 13 \mapsto 13, s \mapsto 13 \mapsto Receive \mapsto Bank \mapsto Abort \mapsto Buyer \mapsto s \mapsto 14 \mapsto 14, s \mapsto 12 \mapsto Send \mapsto Bank \mapsto Confirm \mapsto Seller \mapsto s \mapsto 13 \mapsto 13, s \mapsto 13 \mapsto Receive \mapsto Bank \mapsto Confirm \mapsto Seller \mapsto s \mapsto 14 \mapsto 14, s \mapsto 12 \mapsto Send \mapsto Bank \mapsto Confirm \mapsto Buyer \mapsto s \mapsto 13 \mapsto 13, s \mapsto 13 \mapsto Receive \mapsto Bank \mapsto Confirm \mapsto Buyer \mapsto s \mapsto 14 \mapsto 14, s \mapsto 14 \mapsto Send \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 15 \mapsto 15, s \mapsto 15 \mapsto Receive \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 16 \mapsto 16\}$

axm11: $S\_Next\_States = \{\{((s0\_Seller \mapsto (Send \mapsto PayReq \mapsto Bank) \mapsto 1) \mapsto s2\_Seller)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Bank \mapsto (Receive \mapsto PayReq \mapsto Seller) \mapsto 1) \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto$

$s2\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s0\_Seller \mapsto (Send \mapsto Sync0 \mapsto Buyer) \mapsto 1) \mapsto s1\_Seller)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Buyer \mapsto (Receive \mapsto Sync0 \mapsto Seller) \mapsto 1) \mapsto s1\_Buyer)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s1\_Buyer \mapsto (Send \mapsto PriceReq \mapsto Seller) \mapsto 2) \mapsto s2\_Buyer)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s1\_Seller \mapsto (Receive \mapsto PriceReq \mapsto Buyer) \mapsto 2) \mapsto s3\_Seller)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s3\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s3\_Seller \mapsto (Send \mapsto Offre \mapsto Buyer) \mapsto 3) \mapsto s0\_Seller)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s3\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s2\_Buyer \mapsto (Receive \mapsto Offre \mapsto Seller) \mapsto 3) \mapsto s0\_Buyer)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s1\_Bank \mapsto (Send \mapsto Sync1 \mapsto Buyer) \mapsto 4) \mapsto s2\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\}, \{((s0\_Buyer \mapsto (Receive \mapsto Sync1 \mapsto Bank) \mapsto 4) \mapsto s3\_Buyer)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\}, \{((s3\_Buyer \mapsto (Send \mapsto Pay \mapsto Bank) \mapsto 5) \mapsto s4\_Buyer)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\}, \{((s2\_Bank \mapsto (Receive \mapsto Pay \mapsto Buyer) \mapsto 5) \mapsto s3\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s3\_Bank)\}, \{((s3\_Bank \mapsto (Send \mapsto Abort \mapsto Buyer) \mapsto 6) \mapsto s4\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s3\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\}, \{((s4\_Buyer \mapsto (Receive \mapsto Abort \mapsto Bank) \mapsto 6) \mapsto s5\_Buyer)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\} \mapsto \{(Buyer \mapsto s5\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\}, \{((s3\_Bank \mapsto (Send \mapsto Confirm \mapsto Seller) \mapsto 6) \mapsto s5\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s3\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s5\_Bank)\}, \{((s2\_Seller \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 6) \mapsto s4\_Seller)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s5\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s4\_Seller), (Bank \mapsto s5\_Bank)\}, \{((s3\_Bank \mapsto (Send \mapsto Confirm \mapsto Buyer) \mapsto 6) \mapsto s6\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s3\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s6\_Bank)\}, \{((s4\_Buyer \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 6) \mapsto s6\_Buyer)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s6\_Bank)\}\}$

axm12: $A\_Next\_States = \{\{((s0\_Seller \mapsto (Send \mapsto PayReq \mapsto Bank) \mapsto 1) \mapsto s2\_Seller)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Bank \mapsto (Receive \mapsto PayReq \mapsto Seller) \mapsto 1) \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{Bank \mapsto PayReq \mapsto 0\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s0\_Bank \mapsto (Receive \mapsto PayReq \mapsto Seller) \mapsto 1) \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s0\_Seller \mapsto (Send \mapsto Sync0 \mapsto Buyer) \mapsto 1) \mapsto s1\_Seller)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Buyer \mapsto (Receive \mapsto Sync0 \mapsto Seller) \mapsto 1) \mapsto s1\_Buyer)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{Buyer \mapsto Sync0 \mapsto 0\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Buyer \mapsto (Receive \mapsto Sync0 \mapsto Seller) \mapsto 1) \mapsto s1\_Buyer)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s1\_Buyer \mapsto (Send \mapsto PriceReq \mapsto Seller) \mapsto 2) \mapsto s2\_Buyer)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s1\_Seller \mapsto (Receive \mapsto PriceReq \mapsto Buyer) \mapsto 2) \mapsto s3\_Seller)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{Seller \mapsto PriceReq \mapsto 1\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s3\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s1\_Seller \mapsto (Receive \mapsto PriceReq \mapsto Buyer) \mapsto 2) \mapsto s3\_Seller)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s3\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s3\_Seller \mapsto (Send \mapsto Offre \mapsto Buyer) \mapsto 3) \mapsto s0\_Seller)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s3\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s2\_Buyer \mapsto (Receive \mapsto Offre \mapsto Seller) \mapsto 3) \mapsto s0\_Buyer)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{Buyer \mapsto Offre \mapsto 2\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s2\_Buyer \mapsto (Receive \mapsto Offre \mapsto Seller) \mapsto 3) \mapsto s0\_Buyer)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller),$

$(Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}$,
$\{((s0\_Seller \mapsto (Send \mapsto PayReq \mapsto Bank) \mapsto 1) \mapsto s2\_Seller)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Bank \mapsto (Receive \mapsto PayReq \mapsto Seller) \mapsto 1) \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{Bank \mapsto PayReq \mapsto 3\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s1\_Bank)\}$,
$\{((s0\_Bank \mapsto (Receive \mapsto PayReq \mapsto Seller) \mapsto 1) \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s1\_Bank \mapsto (Send \mapsto Sync1 \mapsto Buyer) \mapsto 4) \mapsto s2\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\}, \{((s0\_Buyer \mapsto (Receive \mapsto Sync1 \mapsto Bank) \mapsto 4) \mapsto s3\_Buyer)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \{Buyer \mapsto Sync1 \mapsto 4\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\}, \{((s0\_Buyer \mapsto (Receive \mapsto Sync1 \mapsto Bank) \mapsto 4) \mapsto s3\_Buyer)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\}, \{((s3\_Buyer \mapsto (Send \mapsto Pay \mapsto Bank) \mapsto 5) \mapsto s4\_Buyer)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\}, \{((s2\_Bank \mapsto (Receive \mapsto Pay \mapsto Buyer) \mapsto 5) \mapsto s3\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \{Bank \mapsto Pay \mapsto 5\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s3\_Bank)\}, \{((s2\_Bank \mapsto (Receive \mapsto Pay \mapsto Buyer) \mapsto 5) \mapsto s3\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s3\_Bank)\}, \{((s3\_Bank \mapsto (Send \mapsto Abort \mapsto Buyer) \mapsto 6) \mapsto s4\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s3\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\}, \{((s4\_Buyer \mapsto (Receive \mapsto Abort \mapsto Bank) \mapsto 6) \mapsto s5\_Buyer)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\} \mapsto \{Buyer \mapsto Abort \mapsto 6\} \mapsto \{(Buyer \mapsto s5\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\}, \{((s4\_Buyer \mapsto (Receive \mapsto Abort \mapsto Bank) \mapsto 6) \mapsto s5\_Buyer)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s5\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\}, \{((s3\_Bank \mapsto (Send \mapsto Confirm \mapsto Seller) \mapsto 6) \mapsto s5\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s3\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s5\_Bank)\}$,
$\{((s2\_Seller \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 6) \mapsto s4\_Seller)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s5\_Bank)\} \mapsto \{Seller \mapsto Confirm \mapsto 6\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s4\_Seller), (Bank \mapsto s5\_Bank)\}, \{((s2\_Seller \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 6) \mapsto s4\_Seller)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s5\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s4\_Seller), (Bank \mapsto s5\_Bank)\}, \{((s3\_Bank \mapsto (Send \mapsto Confirm \mapsto Buyer) \mapsto 6) \mapsto s6\_Bank)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s3\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s6\_Bank)\}, \{((s4\_Buyer \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 6) \mapsto s6\_Buyer)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s6\_Bank)\} \mapsto \{Buyer \mapsto Confirm \mapsto 6\} \mapsto \{(Buyer \mapsto s6\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s6\_Bank)\}$,
$\{((s4\_Buyer \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 6) \mapsto s6\_Buyer)\} \mapsto \{(Buyer \mapsto s4\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s6\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s6\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s6\_Bank)\}\}$

axm13: $Initial\_state\_value = s0$

axm14: $CP\_Final\_states\_value = \{s0\}$

axm15: $Prophecy\_value = 7$

axm16: $End\_message = End$

axm17: $A\_GS\_value = \{Buyer \mapsto s0\_Buyer, Seller \mapsto s0\_Seller, Bank \mapsto s0\_Bank\}$

axm18: $S\_GS\_value = \{Buyer \mapsto s0\_Buyer, Seller \mapsto s0\_Seller, Bank \mapsto s0\_Bank\}$

axm19: $Last\_cp\_trans\_value = s0 \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s1 \mapsto 1$

END

CONTEXT CS3_un-realizable
EXTENDS LTS_ASYNC_CONTEXT
CONSTANTS

Buyer
Seller
Bank
Pend
PriceReq
Offre
PayReq
Pay
Abort
Confirm
End
s0
s1
s2
s3
s4
s5
s6
s7
s0_Buyer
s1_Buyer
s2_Buyer
s3_Buyer
s0_Seller
s1_Seller
s2_Seller
s3_Seller
s0_Bank
s1_Bank
s2_Bank
s3_Bank
s4_Bank
s
s8
s9

AXIOMS

axm1: $partition(PEERS, \{Buyer\}, \{Seller\}, \{Bank\}, \{Pend\})$

axm2: $partition(MESSAGES, \{PriceReq\}, \{Offre\}, \{PayReq\}, \{Pay\}, \{Abort\}, \{Confirm\}, \{End\})$

axm3: $partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\}, \{s6\}, \{s7\}, \{s8\}, \{s9\})$

axm4: $CPs\_B = \{s0 \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s1 \mapsto 1, s0 \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s2 \mapsto 1, s2 \mapsto Seller \mapsto Offre \mapsto Buyer \mapsto s0 \mapsto 2, s1 \mapsto Buyer \mapsto Pay \mapsto Bank \mapsto s3 \mapsto 3, s3 \mapsto Bank \mapsto Confirm \mapsto Buyer \mapsto s4 \mapsto 4, s3 \mapsto Bank \mapsto Confirm \mapsto Seller \mapsto s5 \mapsto 4, s3 \mapsto Bank \mapsto Abort \mapsto Buyer \mapsto s6 \mapsto 4, s4 \mapsto Pend \mapsto End \mapsto Pend \mapsto s7 \mapsto 5, s5 \mapsto Pend \mapsto End \mapsto Pend \mapsto s8 \mapsto 6, s6 \mapsto Pend \mapsto End \mapsto Pend \mapsto s9 \mapsto 7\}$

axm5: $CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto PayReq \mapsto Bank \mapsto Seller \mapsto Receive \mapsto PayReq \mapsto s1 \mapsto 1, s0 \mapsto Send \mapsto PriceReq \mapsto Seller \mapsto Buyer \mapsto Receive \mapsto PriceReq \mapsto s2 \mapsto 1, s2 \mapsto Send \mapsto Offre \mapsto Buyer \mapsto Seller \mapsto Receive \mapsto Offre \mapsto s0 \mapsto 2, s1 \mapsto Send \mapsto Pay \mapsto Bank \mapsto Buyer \mapsto Receive \mapsto Pay \mapsto s3 \mapsto 3, s3 \mapsto Send \mapsto Confirm \mapsto Buyer \mapsto Bank \mapsto Receive \mapsto Confirm \mapsto s4 \mapsto 4, s3 \mapsto Send \mapsto Confirm \mapsto Seller \mapsto Bank \mapsto Receive \mapsto Confirm \mapsto s5 \mapsto 4, s3 \mapsto Send \mapsto Abort \mapsto Buyer \mapsto Bank \mapsto Receive \mapsto Abort \mapsto s6 \mapsto 4, s4 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s7 \mapsto 5, s5 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s8 \mapsto 6, s6 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s9 \mapsto 7\}$

axm6: $partition(A\_STATES, \{s0\_Buyer\}, \{s1\_Buyer\}, \{s2\_Buyer\}, \{s3\_Buyer\}, \{s0\_Seller\}, \{s1\_Seller\}, \{s2\_Seller\}, \{s3\_Seller\}, \{s0\_Bank\}, \{s1\_Bank\}, \{s2\_Bank\}, \{s3\_Bank\}, \{s4\_Bank\})$

axm7: $partition(A\_TRACE\_STATES, \{s\})$

axm8: $PEERs\_B = \{((s0\_Seller \mapsto (Send \mapsto PayReq \mapsto Bank) \mapsto 1) \mapsto s1\_Seller), ((s0\_Bank \mapsto (Receive \mapsto PayReq \mapsto Seller) \mapsto 1) \mapsto s1\_Bank), ((s0\_Buyer \mapsto (Send \mapsto PriceReq \mapsto Seller) \mapsto 1) \mapsto s1\_Buyer), ((s0\_Seller \mapsto (Receive \mapsto PriceReq \mapsto Buyer) \mapsto 1) \mapsto s2\_Seller), ((s2\_Seller \mapsto (Send \mapsto Offre \mapsto Buyer) \mapsto 2) \mapsto s0\_Seller), ((s1\_Buyer \mapsto (Receive \mapsto Offre \mapsto Seller) \mapsto 2) \mapsto s0\_Buyer), ((s1\_Buyer \mapsto (Send \mapsto Pay \mapsto Bank) \mapsto 3) \mapsto s2\_Buyer), ((s1\_Bank \mapsto (Receive \mapsto Pay \mapsto Buyer) \mapsto 3) \mapsto s2\_Bank), ((s2\_Bank \mapsto (Send \mapsto Abort \mapsto Buyer) \mapsto 4) \mapsto s3\_Bank), ((s2\_Buyer \mapsto (Receive \mapsto Abort \mapsto Bank) \mapsto 4) \mapsto s3\_Buyer), ((s2\_Bank \mapsto (Send \mapsto Confirm \mapsto Seller) \mapsto 4) \mapsto s4\_Bank), ((s2\_Seller \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 4) \mapsto s3\_Seller)\}$

axm9: $R\_TRACE\_B = \{s0 \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s1 \mapsto 1, s0 \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s2 \mapsto 1, s2 \mapsto Seller \mapsto Offre \mapsto Buyer \mapsto s0 \mapsto 2, s1 \mapsto Buyer \mapsto Pay \mapsto Bank \mapsto s3 \mapsto 3, s3 \mapsto Bank \mapsto Confirm \mapsto Buyer \mapsto s4 \mapsto 4, s3 \mapsto Bank \mapsto Confirm \mapsto Seller \mapsto s5 \mapsto 4, s3 \mapsto Bank \mapsto Abort \mapsto Buyer \mapsto s6 \mapsto 4, s4 \mapsto Pend \mapsto End \mapsto Pend \mapsto s7 \mapsto 5, s5 \mapsto Pend \mapsto End \mapsto Pend \mapsto s8 \mapsto 6, s6 \mapsto Pend \mapsto End \mapsto Pend \mapsto s9 \mapsto 7\}$

axm10: $S\_Next\_States = \{\{((s0\_Seller \mapsto (Send \mapsto PayReq \mapsto Bank) \mapsto 1) \mapsto s1\_Seller)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Bank \mapsto (Receive \mapsto PayReq \mapsto Seller) \mapsto 1) \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s0\_Buyer \mapsto (Send \mapsto PriceReq \mapsto Seller) \mapsto 1) \mapsto s1\_Buyer)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Seller \mapsto (Receive \mapsto PriceReq \mapsto Buyer) \mapsto 1) \mapsto s2\_Seller)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s2\_Seller \mapsto (Send \mapsto Offre \mapsto Buyer) \mapsto 2) \mapsto s0\_Seller)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s1\_Buyer \mapsto (Receive \mapsto Offre \mapsto Seller) \mapsto 2) \mapsto s0\_Buyer)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s1\_Buyer \mapsto (Send \mapsto Pay \mapsto Bank) \mapsto 3) \mapsto s2\_Buyer)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s1\_Bank \mapsto (Receive \mapsto Pay \mapsto Buyer) \mapsto 3) \mapsto s2\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s2\_Bank)\}, \{((s2\_Bank \mapsto (Send \mapsto Abort \mapsto Buyer) \mapsto 4) \mapsto s3\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s3\_Bank)\}, \{((s2\_Buyer \mapsto (Receive \mapsto Abort \mapsto Bank) \mapsto 4) \mapsto s3\_Buyer)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s3\_Bank)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s3\_Bank)\}, \{((s2\_Bank \mapsto (Send \mapsto Confirm \mapsto Seller) \mapsto 4) \mapsto s4\_Bank)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s4\_Bank)\}, \{((s2\_Seller \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 4) \mapsto s3\_Seller)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s3\_Seller), (Bank \mapsto s4\_Bank)\}\}$

axm11: $A\_Next\_States = \{\{((s0\_Seller \mapsto (Send \mapsto PayReq \mapsto Bank) \mapsto 1) \mapsto s1\_Seller)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Bank \mapsto (Receive \mapsto PayReq \mapsto Seller) \mapsto 1) \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{Bank \mapsto PayReq \mapsto 0\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s0\_Bank \mapsto (Receive \mapsto PayReq \mapsto Seller) \mapsto 1) \mapsto s1\_Bank)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s0\_Buyer \mapsto (Send \mapsto PriceReq \mapsto Seller) \mapsto 1) \mapsto s1\_Buyer)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Seller \mapsto (Receive \mapsto PriceReq \mapsto Buyer) \mapsto 1) \mapsto s2\_Seller)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{Seller \mapsto PriceReq \mapsto 0\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Seller \mapsto (Receive \mapsto PriceReq \mapsto Buyer) \mapsto 1) \mapsto s2\_Seller)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s2\_Seller \mapsto (Send \mapsto Offre \mapsto Buyer) \mapsto 2) \mapsto s0\_Seller)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s1\_Buyer \mapsto (Receive \mapsto Offre \mapsto Seller) \mapsto 2) \mapsto s0\_Buyer)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \{Buyer \mapsto Offre \mapsto 1\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller),$

$(Bank \mapsto s0\_Bank)\}, \{((s1\_Buyer \mapsto (Receive \mapsto Offre \mapsto Seller) \mapsto 2) \mapsto s0\_Buyer)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s0\_Bank)\}, \{((s0\_Buyer \mapsto (Send \mapsto PriceReq \mapsto Seller) \mapsto 1) \mapsto s1\_Buyer)\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s0\_Seller \mapsto (Receive \mapsto PriceReq \mapsto Buyer) \mapsto 1) \mapsto s2\_Seller)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \{Seller \mapsto PriceReq \mapsto 1\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s0\_Seller \mapsto (Receive \mapsto PriceReq \mapsto Buyer) \mapsto 1) \mapsto s2\_Seller)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s1\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s2\_Seller \mapsto (Send \mapsto Offre \mapsto Buyer) \mapsto 2) \mapsto s0\_Seller)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s1\_Buyer \mapsto (Receive \mapsto Offre \mapsto Seller) \mapsto 2) \mapsto s0\_Buyer)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \{Buyer \mapsto Offre \mapsto 2\} \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s1\_Buyer \mapsto (Receive \mapsto Offre \mapsto Seller) \mapsto 2) \mapsto s0\_Buyer)\} \mapsto \{(Buyer \mapsto s1\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s0\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s1\_Bank)\}, \{((s1\_Bank \mapsto (Receive \mapsto Pay \mapsto Buyer) \mapsto 3) \mapsto s2\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \{Bank \mapsto Pay \mapsto 3\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s2\_Bank)\}, \{((s1\_Bank \mapsto (Receive \mapsto Pay \mapsto Buyer) \mapsto 3) \mapsto s2\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s1\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s2\_Bank)\}, \{((s2\_Bank \mapsto (Send \mapsto Abort \mapsto Buyer) \mapsto 4) \mapsto s3\_Bank)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s3\_Bank)\}, \{((s2\_Buyer \mapsto (Receive \mapsto Abort \mapsto Bank) \mapsto 4) \mapsto s3\_Buyer)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s3\_Bank)\} \mapsto \{Buyer \mapsto Abort \mapsto 4\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s3\_Bank)\}, \{((s2\_Buyer \mapsto (Receive \mapsto Abort \mapsto Bank) \mapsto 4) \mapsto s3\_Buyer)\} \mapsto \{(Buyer \mapsto s2\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s3\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s3\_Bank)\}, \{((s2\_Bank \mapsto (Send \mapsto Confirm \mapsto Seller) \mapsto 4) \mapsto s4\_Bank)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s2\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s0\_Seller), (Bank \mapsto s4\_Bank)\}, \{((s2\_Seller \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 4) \mapsto s3\_Seller)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\} \mapsto \{Seller \mapsto Confirm \mapsto 5\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s3\_Seller), (Bank \mapsto s4\_Bank)\}, \{((s2\_Seller \mapsto (Receive \mapsto Confirm \mapsto Bank) \mapsto 4) \mapsto s3\_Seller)\} \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s2\_Seller), (Bank \mapsto s4\_Bank)\} \mapsto \varnothing \mapsto \{(Buyer \mapsto s3\_Buyer), (Seller \mapsto s3\_Seller), (Bank \mapsto s4\_Bank)\}\}$

axm12: $A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s \mapsto 2 \mapsto 2, s \mapsto 0 \mapsto Send \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s \mapsto 2 \mapsto 2, s \mapsto 2 \mapsto Send \mapsto Seller \mapsto Offre \mapsto Buyer \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto Seller \mapsto Offre \mapsto Buyer \mapsto s \mapsto 4 \mapsto 4, s \mapsto 4 \mapsto Send \mapsto Seller \mapsto Pay \mapsto Buyer \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto Seller \mapsto Pay \mapsto Buyer \mapsto s \mapsto 6 \mapsto 6, s \mapsto 6 \mapsto Send \mapsto Seller \mapsto Abort \mapsto Buyer \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto Seller \mapsto Abort \mapsto Buyer \mapsto s \mapsto 8 \mapsto 8, s \mapsto 8 \mapsto Send \mapsto Seller \mapsto Confirm \mapsto Buyer \mapsto s \mapsto 9 \mapsto 9, s \mapsto 9 \mapsto Receive \mapsto Seller \mapsto Confirm \mapsto Buyer \mapsto s \mapsto 10 \mapsto 10\}$

axm13: $Initial\_state\_value = s0$

axm14: $CP\_Final\_states\_value = \{s0\}$

axm15: $Prophecy\_value = 7$

axm16: $End\_message = End$

axm17: $S\_GS\_value = \{Buyer \mapsto s0\_Buyer, Seller \mapsto s0\_Seller, Bank \mapsto s0\_Bank\}$

axm18: $A\_GS\_value = \{Buyer \mapsto s0\_Buyer, Seller \mapsto s0\_Seller, Bank \mapsto s0\_Bank\}$

axm19: $Last\_cp\_trans\_value = s0 \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s1 \mapsto 1$

END

CONTEXT CS4_realizable
EXTENDS LTS_ASYNC_CONTEXT
CONSTANTS

P1
P2
Pend
ms
mf
mc
m0
End
s0
s1
s2
s3
s4
s5
s6
P3
Init
Sync0
Sync1
s7
s8
s9
s0_P1
s1_P1
s2_P1
s3_P1
s4_P1
s5_P1
s0_P2
s1_P2
s2_P2
s3_P2
s4_P2
s5_P2
s0_P3
s1_P3
s
s2_P3
s6_P1
s6_P2

AXIOMS

axm1: $partition(PEERS, \{P1\}, \{P2\}, \{P3\}, \{Pend\})$

axm2: $partition(MESSAGES, \{Init\}, \{ms\}, \{mf\}, \{mc\}, \{m0\}, \{Sync0\}, \{Sync1\}, \{End\})$

axm3: $partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\}, \{s6\}, \{s7\}, \{s8\}, \{s9\})$

axm4: $CPs\_B = \{s0 \mapsto P3 \mapsto Init \mapsto P2 \mapsto s1 \mapsto 1, s1 \mapsto P3 \mapsto Sync0 \mapsto P1 \mapsto s2 \mapsto 2, s2 \mapsto P1 \mapsto ms \mapsto P2 \mapsto s3 \mapsto 3, s3 \mapsto P1 \mapsto mc \mapsto P2 \mapsto s4 \mapsto 4, s3 \mapsto P1 \mapsto Sync1 \mapsto P2 \mapsto s5 \mapsto 4, s4 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s6 \mapsto 5, s5 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s7 \mapsto 6, s6 \mapsto Pend \mapsto End \mapsto Pend \mapsto s8 \mapsto 7, s7 \mapsto Pend \mapsto End \mapsto Pend \mapsto s9 \mapsto 8\}$

axm5: $CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto Init \mapsto P2 \mapsto P3 \mapsto Receive \mapsto Init \mapsto s1 \mapsto 1, s1 \mapsto Send \mapsto Sync0 \mapsto P1 \mapsto P3 \mapsto Receive \mapsto Sync0 \mapsto s2 \mapsto 2, s2 \mapsto Send \mapsto ms \mapsto P2 \mapsto P1 \mapsto Receive \mapsto ms \mapsto s3 \mapsto 3, s3 \mapsto Send \mapsto mc \mapsto P2 \mapsto P1 \mapsto Receive \mapsto mc \mapsto s4 \mapsto 4, s3 \mapsto Send \mapsto Sync1 \mapsto P2 \mapsto P1 \mapsto Receive \mapsto Sync1 \mapsto s5 \mapsto 4, s4 \mapsto Send \mapsto mf \mapsto P1 \mapsto P2 \mapsto Receive \mapsto mf \mapsto s6 \mapsto 5, s5 \mapsto Send \mapsto mf \mapsto P1 \mapsto P2 \mapsto Receive \mapsto mf \mapsto s7 \mapsto 6, s6 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s8 \mapsto 7, s7 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s9 \mapsto 8\}$

axm6: $partition(A\_STATES, \{s0\_P1\}, \{s1\_P1\}, \{s2\_P1\}, \{s3\_P1\}, \{s4\_P1\}, \{s5\_P1\}, \{s6\_P1\}, \{s0\_P2\}, \{s1\_P2\}, \{s2\_P2\}, \{s3\_P2\}, \{s4\_P2\}, \{s5\_P2\}, \{s6\_P2\}, \{s0\_P3\}, \{s1\_P3\}, \{s2\_P3\})$

axm7: $partition(A\_TRACE\_STATES, \{s\})$

axm8: $PEERs\_B = \{((s0\_P3 \mapsto (Send \mapsto Init \mapsto P2) \mapsto 1) \mapsto s1\_P3), ((s0\_P2 \mapsto (Receive \mapsto Init \mapsto P3) \mapsto 1) \mapsto s1\_P2), ((s1\_P3 \mapsto (Send \mapsto Sync0 \mapsto P1) \mapsto 2) \mapsto s2\_P3), ((s0\_P1 \mapsto (Receive \mapsto Sync0 \mapsto P3) \mapsto 2) \mapsto s1\_P1), ((s1\_P1 \mapsto (Send \mapsto ms \mapsto P2) \mapsto 3) \mapsto s2\_P1), ((s1\_P2 \mapsto (Receive \mapsto ms \mapsto P1) \mapsto 3) \mapsto s2\_P2), ((s2\_P1 \mapsto (Send \mapsto mc \mapsto P2) \mapsto 4) \mapsto s3\_P1), ((s2\_P2 \mapsto (Receive \mapsto mc \mapsto P1) \mapsto 4) \mapsto s3\_P2), ((s2\_P1 \mapsto (Send \mapsto Sync1 \mapsto P2) \mapsto 4) \mapsto s5\_P1), ((s2\_P2 \mapsto (Receive \mapsto Sync1 \mapsto P1) \mapsto 4) \mapsto s5\_P2), ((s3\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 5) \mapsto s4\_P2), ((s3\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 5) \mapsto s4\_P1), ((s5\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 6) \mapsto s6\_P2), ((s5\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 6) \mapsto s6\_P1)\}$

axm9: $R\_TRACE\_B = \{s0 \mapsto P3 \mapsto Init \mapsto P2 \mapsto s1 \mapsto 1, s1 \mapsto P3 \mapsto Sync0 \mapsto P1 \mapsto s2 \mapsto 2, s2 \mapsto P1 \mapsto ms \mapsto P2 \mapsto s3 \mapsto 3, s3 \mapsto P1 \mapsto mc \mapsto P2 \mapsto s4 \mapsto 4, s3 \mapsto P1 \mapsto Sync1 \mapsto P2 \mapsto s5 \mapsto 4, s4 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s6 \mapsto 5, s5 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s7 \mapsto 6, s6 \mapsto Pend \mapsto End \mapsto Pend \mapsto s8 \mapsto 7, s7 \mapsto Pend \mapsto End \mapsto Pend \mapsto s9 \mapsto 8\}$

axm10: $A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto P3 \mapsto Init \mapsto P2 \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto P3 \mapsto Init \mapsto P2 \mapsto s \mapsto 2 \mapsto 2, s \mapsto 2 \mapsto Send \mapsto P3 \mapsto Sync0 \mapsto P1 \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto P3 \mapsto Sync0 \mapsto P1 \mapsto s \mapsto 4 \mapsto 4, s \mapsto 4 \mapsto Send \mapsto P1 \mapsto ms \mapsto P2 \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto P1 \mapsto ms \mapsto P2 \mapsto s \mapsto 6 \mapsto 6, s \mapsto 6 \mapsto Send \mapsto P1 \mapsto mc \mapsto P2 \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto P1 \mapsto mc \mapsto P2 \mapsto s \mapsto 8 \mapsto 8, s \mapsto 6 \mapsto Send \mapsto P1 \mapsto Sync1 \mapsto P2 \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto P1 \mapsto Sync1 \mapsto P2 \mapsto s \mapsto 8 \mapsto 8, s \mapsto 8 \mapsto Send \mapsto P2 \mapsto mf \mapsto P1 \mapsto s \mapsto 9 \mapsto 9, s \mapsto 9 \mapsto Receive \mapsto P2 \mapsto mf \mapsto P1 \mapsto s \mapsto 10 \mapsto 10, s \mapsto 8 \mapsto Send \mapsto P2 \mapsto mf \mapsto P1 \mapsto s \mapsto 9 \mapsto 9, s \mapsto 9 \mapsto Receive \mapsto P2 \mapsto mf \mapsto P1 \mapsto s \mapsto 10 \mapsto 10, s \mapsto 10 \mapsto Send \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 11 \mapsto 11, s \mapsto 11 \mapsto Receive \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 12 \mapsto 12, s \mapsto 10 \mapsto Send \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 11 \mapsto 11, s \mapsto 11 \mapsto Receive \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 12 \mapsto 12\}$

axm11: $S\_Next\_States = \{\{((s0\_P3 \mapsto (Send \mapsto Init \mapsto P2) \mapsto 1) \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s0\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s1\_P3)\}, \{((s0\_P2 \mapsto (Receive \mapsto Init \mapsto P3) \mapsto 1) \mapsto s1\_P2)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P3 \mapsto (Send \mapsto Sync0 \mapsto P1) \mapsto 2) \mapsto s2\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\}, \{((s0\_P1 \mapsto (Receive \mapsto Sync0 \mapsto P3) \mapsto 2) \mapsto s1\_P1)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\}, \{((s1\_P1 \mapsto (Send \mapsto ms \mapsto P2) \mapsto 3) \mapsto s2\_P1)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\}, \{((s1\_P2 \mapsto (Receive \mapsto ms \mapsto P1) \mapsto 3) \mapsto s2\_P2)\} \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\}, \{((s2\_P1 \mapsto (Send \mapsto mc \mapsto P2) \mapsto 4) \mapsto s3\_P1)\} \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\}, \{((s2\_P2 \mapsto (Receive \mapsto mc \mapsto P1) \mapsto 4) \mapsto s3\_P2)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s3\_P2), (P3 \mapsto s2\_P3)\}, \{((s2\_P1 \mapsto (Send \mapsto Sync1 \mapsto P2) \mapsto 4) \mapsto s5\_P1)\} \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\}, \{((s2\_P2 \mapsto (Receive \mapsto Sync1 \mapsto P1) \mapsto 4) \mapsto s5\_P2)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s5\_P2), (P3 \mapsto s2\_P3)\}, \{((s3\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 5) \mapsto s4\_P2)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s3\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s4\_P2), (P3 \mapsto s2\_P3)\}, \{((s3\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 5) \mapsto s4\_P1)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s4\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P2 \mapsto s4\_P1), (P2 \mapsto s4\_P2), (P3 \mapsto s2\_P3)\}, \{((s5\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 6) \mapsto s6\_P2)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s5\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s6\_P2), (P3 \mapsto s2\_P3)\}, \{((s5\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 6) \mapsto s6\_P1)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s6\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{(P2 \mapsto s6\_P1), (P2 \mapsto s6\_P2), (P3 \mapsto s2\_P3)\}\}$

axm12: $A\_Next\_States = \{\{((s0\_P3 \mapsto (Send \mapsto Init \mapsto P2) \mapsto 1) \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s0\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s1\_P3)\}, \{((s0\_P2 \mapsto (Receive \mapsto Init \mapsto P3) \mapsto 1) \mapsto s1\_P2)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s1\_P3)\} \mapsto$

$\{P2 \mapsto Init \mapsto 0\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s0\_P2 \mapsto (Receive \mapsto Init \mapsto P3) \mapsto 1) \mapsto s1\_P2)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P3 \mapsto (Send \mapsto Sync0 \mapsto P1) \mapsto 2) \mapsto s2\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\}, \{((s0\_P1 \mapsto (Receive \mapsto Sync0 \mapsto P3) \mapsto 2) \mapsto s1\_P1)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{P1 \mapsto Sync0 \mapsto 1\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\}, \{((s0\_P1 \mapsto (Receive \mapsto Sync0 \mapsto P3) \mapsto 2) \mapsto s1\_P1)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\}, \{((s1\_P1 \mapsto (Send \mapsto ms \mapsto P2) \mapsto 3) \mapsto s2\_P1)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\}, \{((s1\_P2 \mapsto (Receive \mapsto ms \mapsto P1) \mapsto 3) \mapsto s2\_P2)\} \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{P2 \mapsto ms \mapsto 2\} \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\}, \{((s1\_P2 \mapsto (Receive \mapsto ms \mapsto P1) \mapsto 3) \mapsto s2\_P2)\} \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\}, \{((s2\_P1 \mapsto (Send \mapsto mc \mapsto P2) \mapsto 4) \mapsto s3\_P1)\} \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\}, \{((s2\_P2 \mapsto (Receive \mapsto mc \mapsto P1) \mapsto 4) \mapsto s3\_P2)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{P2 \mapsto mc \mapsto 3\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s3\_P2), (P3 \mapsto s2\_P3)\}, \{((s2\_P2 \mapsto (Receive \mapsto mc \mapsto P1) \mapsto 4) \mapsto s3\_P2)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s3\_P2), (P3 \mapsto s2\_P3)\}, \{((s2\_P1 \mapsto (Send \mapsto Sync1 \mapsto P2) \mapsto 4) \mapsto s5\_P1)\} \mapsto \{(P2 \mapsto s2\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\}, \{((s2\_P2 \mapsto (Receive \mapsto Sync1 \mapsto P1) \mapsto 4) \mapsto s5\_P2)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{P2 \mapsto Sync1 \mapsto 3\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s5\_P2), (P3 \mapsto s2\_P3)\}, \{((s2\_P2 \mapsto (Receive \mapsto Sync1 \mapsto P1) \mapsto 4) \mapsto s5\_P2)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s5\_P2), (P3 \mapsto s2\_P3)\}, \{((s3\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 5) \mapsto s4\_P2)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s3\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s4\_P2), (P3 \mapsto s2\_P3)\}, \{((s3\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 5) \mapsto s4\_P1)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s4\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{P1 \mapsto mf \mapsto 4\} \mapsto \{(P2 \mapsto s4\_P1), (P2 \mapsto s4\_P2), (P3 \mapsto s2\_P3)\}, \{((s3\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 5) \mapsto s4\_P1)\} \mapsto \{(P2 \mapsto s3\_P1), (P2 \mapsto s4\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P2 \mapsto s4\_P1), (P2 \mapsto s4\_P2), (P3 \mapsto s2\_P3)\}, \{((s5\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 6) \mapsto s6\_P2)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s5\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s6\_P2), (P3 \mapsto s2\_P3)\}, \{((s5\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 6) \mapsto s6\_P1)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s6\_P2), (P3 \mapsto s2\_P3)\} \mapsto \{P1 \mapsto mf \mapsto 4\} \mapsto \{(P2 \mapsto s6\_P1), (P2 \mapsto s6\_P2), (P3 \mapsto s2\_P3)\}, \{((s5\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 6) \mapsto s6\_P1)\} \mapsto \{(P2 \mapsto s5\_P1), (P2 \mapsto s6\_P2), (P3 \mapsto s2\_P3)\} \mapsto \varnothing \mapsto \{(P2 \mapsto s6\_P1), (P2 \mapsto s6\_P2), (P3 \mapsto s2\_P3)\}\}$

axm13: $Initial\_state\_value = s0$

axm14: $CP\_Final\_states\_value = \{s0\}$

axm15: $Prophecy\_value = 5$

axm16: $End\_message = End$

axm17: $S\_GS\_value = \{P1 \mapsto s0\_P1, P2 \mapsto s0\_P2, P3 \mapsto s0\_P3\}$

axm18: $A\_GS\_value = \{P1 \mapsto s0\_P1, P2 \mapsto s0\_P2, P3 \mapsto s0\_P3\}$

axm19: $Last\_cp\_trans\_value = s0 \mapsto P3 \mapsto Init \mapsto P2 \mapsto s1 \mapsto 1$

END

CONTEXT CS4_un-realizable
EXTENDS LTS_ASYNC_CONTEXT
CONSTANTS

P1
P2
Pend
ms
mf
mc
m0
End
s0
s1
s2
s3
s4
s5
s6
P3
Init
s
s0_P1
s1_P1
s2_P1
s3_P1
s4_P1
s5_P1
s0_P2
s1_P2
s2_P2
s3_P2
s4_P2
s5_P2
s0_P3
s1_P3

AXIOMS

axm1: $partition(PEERS, \{P1\}, \{P2\}, \{P3\}, \{Pend\})$

axm2: $partition(MESSAGES, \{Init\}, \{ms\}, \{mf\}, \{mc\}, \{m0\}, \{End\})$

axm3: $partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\}, \{s6\})$

axm4: $CPs\_B = \{s0 \mapsto P3 \mapsto Init \mapsto P2 \mapsto s1 \mapsto 1, s1 \mapsto P1 \mapsto ms \mapsto P2 \mapsto s2 \mapsto 2, s2 \mapsto P1 \mapsto mc \mapsto P2 \mapsto s3 \mapsto 3, s2 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s5 \mapsto 3, s3 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s4 \mapsto 4, s4 \mapsto Pend \mapsto End \mapsto Pend \mapsto s5 \mapsto 5, s5 \mapsto Pend \mapsto End \mapsto Pend \mapsto s6 \mapsto 6\}$

axm5: $CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto Init \mapsto P2 \mapsto P3 \mapsto Receive \mapsto Init \mapsto s1 \mapsto 1, s1 \mapsto Send \mapsto ms \mapsto P2 \mapsto P1 \mapsto Receive \mapsto ms \mapsto s2 \mapsto 2, s2 \mapsto Send \mapsto mc \mapsto P2 \mapsto P1 \mapsto Receive \mapsto mc \mapsto s3 \mapsto 3, s2 \mapsto Send \mapsto mf \mapsto P1 \mapsto P2 \mapsto Receive \mapsto mf \mapsto s5 \mapsto 3, s3 \mapsto Send \mapsto mf \mapsto P1 \mapsto P2 \mapsto Receive \mapsto mf \mapsto s4 \mapsto 4, s4 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s5 \mapsto 5, s5 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s6 \mapsto 6\}$

axm6: $partition(A\_STATES, \{s0\_P1\}, \{s1\_P1\}, \{s2\_P1\}, \{s3\_P1\}, \{s4\_P1\}, \{s5\_P1\}, \{s0\_P2\}, \{s1\_P2\}, \{s2\_P2\}, \{s3\_P2\}, \{s4\_P2\}, \{s5\_P2\}, \{s0\_P3\}, \{s1\_P3\})$

axm7: $partition(A\_TRACE\_STATES, \{s\})$

axm8: $PEERs\_B = \{((s0\_P3 \mapsto (Send \mapsto Init \mapsto P2) \mapsto 1) \mapsto s1\_P3), ((s0\_P2 \mapsto (Receive \mapsto Init \mapsto P3) \mapsto 1) \mapsto s1\_P2), ((s0\_P1 \mapsto (Send \mapsto ms \mapsto P2) \mapsto 2) \mapsto s1\_P1), ((s1\_P2 \mapsto (Receive \mapsto ms \mapsto P1) \mapsto 2) \mapsto s2\_P2), ((s1\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 3) \mapsto s3\_P2), ((s1\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 3) \mapsto s3\_P1), ((s1\_P1 \mapsto (Send \mapsto mc \mapsto P2) \mapsto 3) \mapsto s2\_P1), ((s1\_P2 \mapsto (Receive \mapsto mc \mapsto P1) \mapsto 3) \mapsto s2\_P2), ((s2\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 4) \mapsto s4\_P2), ((s2\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 4) \mapsto s4\_P1)\}$

axm9: $R\_TRACE\_B = \{s0 \mapsto P3 \mapsto Init \mapsto P2 \mapsto s1 \mapsto 1, s1 \mapsto P1 \mapsto ms \mapsto P2 \mapsto s2 \mapsto 2, s2 \mapsto P1 \mapsto mc \mapsto P2 \mapsto s3 \mapsto 3, s2 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s5 \mapsto 3, s3 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s4 \mapsto 4, s4 \mapsto Pend \mapsto End \mapsto Pend \mapsto s5 \mapsto 5, s5 \mapsto Pend \mapsto End \mapsto Pend \mapsto s6 \mapsto 6\}$

axm10: $A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto P3 \mapsto Init \mapsto P2 \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto P3 \mapsto Init \mapsto P2 \mapsto s \mapsto 2 \mapsto 2, s \mapsto 2 \mapsto Send \mapsto P1 \mapsto ms \mapsto P2 \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto P1 \mapsto ms \mapsto P2 \mapsto s \mapsto 4 \mapsto 4, s \mapsto 4 \mapsto Send \mapsto P1 \mapsto mc \mapsto P2 \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto P1 \mapsto mc \mapsto P2 \mapsto s \mapsto 6 \mapsto 6, s \mapsto 4 \mapsto Send \mapsto P2 \mapsto mf \mapsto P1 \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto P2 \mapsto mf \mapsto P1 \mapsto s \mapsto 6 \mapsto 6, s \mapsto 6 \mapsto Send \mapsto P2 \mapsto mf \mapsto P1 \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto P2 \mapsto mf \mapsto P1 \mapsto s \mapsto 8 \mapsto 8\}$

axm11: $S\_Next\_States = \{\{((s0\_P3 \mapsto (Send \mapsto Init \mapsto P2) \mapsto 1) \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s0\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s1\_P3)\}, \{((s0\_P2 \mapsto (Receive \mapsto Init \mapsto P3) \mapsto 1) \mapsto s1\_P2)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s0\_P1 \mapsto (Send \mapsto ms \mapsto P2) \mapsto 2) \mapsto s1\_P1)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P2 \mapsto (Receive \mapsto ms \mapsto P1) \mapsto 2) \mapsto s2\_P2)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 3) \mapsto s3\_P2)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 3) \mapsto s3\_P1)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P1 \mapsto (Send \mapsto mc \mapsto P2) \mapsto 3) \mapsto s2\_P1)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P2 \mapsto (Receive \mapsto mc \mapsto P1) \mapsto 3) \mapsto s2\_P2)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}, \{((s2\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 4) \mapsto s4\_P2)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s2\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 4) \mapsto s4\_P1)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}\}$

axm12: $A\_Next\_States = \{\{((s0\_P3 \mapsto (Send \mapsto Init \mapsto P2) \mapsto 1) \mapsto s1\_P3)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s0\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s1\_P3)\}, \{((s0\_P2 \mapsto (Receive \mapsto Init \mapsto P3) \mapsto 1) \mapsto s1\_P2)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{P2 \mapsto Init \mapsto 0\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s0\_P2 \mapsto (Receive \mapsto Init \mapsto P3) \mapsto 1) \mapsto s1\_P2)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s0\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s0\_P1 \mapsto (Send \mapsto ms \mapsto P2) \mapsto 2) \mapsto s1\_P1)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P2 \mapsto (Receive \mapsto ms \mapsto P1) \mapsto 2) \mapsto s2\_P2)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{P2 \mapsto ms \mapsto 1\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P2 \mapsto (Receive \mapsto ms \mapsto P1) \mapsto 2) \mapsto s2\_P2)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 3) \mapsto s3\_P2)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 3) \mapsto s3\_P1)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{P1 \mapsto mf \mapsto 2\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 3) \mapsto s3\_P1)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P1 \mapsto (Send \mapsto mc \mapsto P2) \mapsto 3) \mapsto s2\_P1)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P2 \mapsto (Receive \mapsto mc \mapsto P1) \mapsto 3) \mapsto s2\_P2)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{P2 \mapsto mc \mapsto 3\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}, \{((s1\_P2 \mapsto (Receive \mapsto mc \mapsto P1) \mapsto 3) \mapsto s2\_P2)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}, \{((s2\_P2 \mapsto (Send \mapsto mf \mapsto P1) \mapsto 4) \mapsto s4\_P2)\} \mapsto \{(P1 \mapsto s0\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\}, \{((s2\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 4) \mapsto s4\_P1)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \{P1 \mapsto mf \mapsto 4\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}, \{((s2\_P1 \mapsto (Receive \mapsto mf \mapsto P2) \mapsto 4) \mapsto s4\_P1)\} \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s1\_P2), (P3 \mapsto s1\_P3)\} \mapsto \varnothing \mapsto \{(P1 \mapsto s1\_P1), (P2 \mapsto s2\_P2), (P3 \mapsto s1\_P3)\}\}$

axm13: $Initial\_state\_value = s0$

axm14: $CP\_Final\_states\_value = \{s0\}$

axm15: $Prophecy\_value = 5$

axm16: $End\_message = End$

axm17: $S\_GS\_value = \{P1 \mapsto s0\_P1, P2 \mapsto s0\_P2, P3 \mapsto s0\_P3\}$

axm18: $A\_GS\_value = \{P1 \mapsto s0\_P1, P2 \mapsto s0\_P2, P3 \mapsto s0\_P3\}$

axm19: $Last\_cp\_trans\_value = s0 \mapsto P3 \mapsto Init \mapsto P2 \mapsto s1 \mapsto 1$

END

CONTEXT CS5
EXTENDS LTS_ASYNC_CONTEXT
CONSTANTS

s0
s1
s2
s3
s4
s5
s6
s7
s8
s9
s10
s11
s12
Client
Appli
Pend
CheckEligibility
Cancel
FillInApplicationAsStudent
FillInApplication
SubmitGraduationCertificat
SubmitPassport
CheckApproval
SubmitWorkExperience
TestEnglishAbility
Confirm
Reassess
End
s13
s0_Client
s1_Client
s2_Client
s3_Client
s4_Client
s5_Client
s6_Client
s7_Client
s8_Client
s9_Client
s10_Client
s0_Appli
s1_Appli
s2_Appli
s3_Appli
s4_Appli
s5_Appli
s6_Appli
s7_Appli
s8_Appli

s9_Appli

s10_Appli

s

AXIOMS

axm1: $partition(PEERS, \{Client\}, \{Appli\}, \{Pend\})$

axm2: $partition(MESSAGES, \{CheckEligibility\}, \{Cancel\}, \{FillInApplicationAsStudent\}, \{FillInApplication\}, \{SubmitGraduationCertificat\}, \{SubmitPassport\}, \{CheckApproval\}, \{SubmitWorkExperience\}, \{TestEnglishAbility\}, \{Confirm\}, \{Reassess\}, \{End\})$

axm3: $partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\}, \{s6\}, \{s7\}, \{s8\}, \{s9\}, \{s10\}, \{s11\}, \{s12\}, \{s13\})$

axm4: $CPs\_B = \{s0 \mapsto Client \mapsto CheckEligibility \mapsto Appli \mapsto s1 \mapsto 1, s1 \mapsto Client \mapsto Cancel \mapsto Appli \mapsto s2 \mapsto 2, s1 \mapsto Client \mapsto FillInApplicationAsStudent \mapsto Appli \mapsto s3 \mapsto 2, s1 \mapsto Client \mapsto FillInApplication \mapsto Appli \mapsto s6 \mapsto 2, s3 \mapsto Client \mapsto SubmitGraduationCertificat \mapsto Appli \mapsto s4 \mapsto 3, s4 \mapsto Client \mapsto SubmitPassport \mapsto Appli \mapsto s5 \mapsto 4, s5 \mapsto Appli \mapsto CheckApproval \mapsto Client \mapsto s9 \mapsto 5, s6 \mapsto Client \mapsto SubmitWorkExperience \mapsto Appli \mapsto s7 \mapsto 6, s7 \mapsto Client \mapsto TestEnglishAbility \mapsto Appli \mapsto s8 \mapsto 7, s8 \mapsto Appli \mapsto Confirm \mapsto Client \mapsto s9 \mapsto 8, s8 \mapsto Appli \mapsto Reassess \mapsto Client \mapsto s10 \mapsto 8, s9 \mapsto Pend \mapsto End \mapsto Pend \mapsto s11 \mapsto 9, s10 \mapsto Pend \mapsto End \mapsto Pend \mapsto s12 \mapsto 10, s2 \mapsto Pend \mapsto End \mapsto Pend \mapsto s13 \mapsto 11\}$

axm5: $CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto CheckEligibility \mapsto Appli \mapsto Client \mapsto Receive \mapsto CheckEligibility \mapsto s1 \mapsto 1, s1 \mapsto Send \mapsto Cancel \mapsto Appli \mapsto Client \mapsto Receive \mapsto Cancel \mapsto s2 \mapsto 2, s1 \mapsto Send \mapsto FillInApplicationAsStudent \mapsto Appli \mapsto Client \mapsto Receive \mapsto FillInApplicationAsStudent \mapsto s3 \mapsto 2, s1 \mapsto Send \mapsto FillInApplication \mapsto Appli \mapsto Client \mapsto Receive \mapsto FillInApplication \mapsto s6 \mapsto 2, s3 \mapsto Send \mapsto SubmitGraduationCertificat \mapsto Appli \mapsto Client \mapsto Receive \mapsto SubmitGraduationCertificat \mapsto s4 \mapsto 3, s4 \mapsto Send \mapsto SubmitPassport \mapsto Appli \mapsto Client \mapsto Receive \mapsto SubmitPassport \mapsto s5 \mapsto 4, s5 \mapsto Send \mapsto CheckApproval \mapsto Client \mapsto Appli \mapsto Receive \mapsto CheckApproval \mapsto s9 \mapsto 5, s6 \mapsto Send \mapsto SubmitWorkExperience \mapsto Appli \mapsto Client \mapsto Receive \mapsto SubmitWorkExperience \mapsto s7 \mapsto 6, s7 \mapsto Send \mapsto TestEnglishAbility \mapsto Appli \mapsto Client \mapsto Receive \mapsto TestEnglishAbility \mapsto s8 \mapsto 7, s8 \mapsto Send \mapsto Confirm \mapsto Client \mapsto Appli \mapsto Receive \mapsto Confirm \mapsto s9 \mapsto 8, s8 \mapsto Send \mapsto Reassess \mapsto Client \mapsto Appli \mapsto Receive \mapsto Reassess \mapsto s10 \mapsto 8, s9 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s11 \mapsto 9, s10 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s12 \mapsto 10, s2 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s13 \mapsto 11\}$

axm6: $partition(A\_STATES, \{s0\_Client\}, \{s1\_Client\}, \{s2\_Client\}, \{s3\_Client\}, \{s4\_Client\}, \{s5\_Client\}, \{s6\_Client\}, \{s7\_Client\}, \{s8\_Client\}, \{s9\_Client\}, \{s10\_Client\}, \{s0\_Appli\}, \{s1\_Appli\}, \{s2\_Appli\}, \{s3\_Appli\}, \{s4\_Appli\}, \{s5\_Appli\}, \{s6\_Appli\}, \{s7\_Appli\}, \{s8\_Appli\}, \{s9\_Appli\}, \{s10\_Appli\})$

axm7: $partition(A\_TRACE\_STATES, \{s\})$

axm8: $PEERs\_B = \{((s0\_Client \mapsto (Send \mapsto CheckEligibility \mapsto Appli) \mapsto 1) \mapsto s1\_Client), ((s0\_Appli \mapsto (Receive \mapsto CheckEligibility \mapsto Client) \mapsto 1) \mapsto s1\_Appli), ((s1\_Client \mapsto (Send \mapsto Cancel \mapsto Appli) \mapsto 2) \mapsto s2\_Client), ((s1\_Appli \mapsto (Receive \mapsto Cancel \mapsto Client) \mapsto 2) \mapsto s2\_Appli), ((s1\_Client \mapsto (Send \mapsto FillInApplicationAsStudent \mapsto Appli) \mapsto 2) \mapsto s3\_Client), ((s1\_Appli \mapsto (Receive \mapsto FillInApplicationAsStudent \mapsto Client) \mapsto 2) \mapsto s3\_Appli), ((s1\_Client \mapsto (Send \mapsto FillInApplication \mapsto Appli) \mapsto 2) \mapsto s6\_Client), ((s1\_Appli \mapsto (Receive \mapsto FillInApplication \mapsto Client) \mapsto 2) \mapsto s6\_Appli), ((s3\_Client \mapsto (Send \mapsto SubmitGraduationCertificat \mapsto Appli) \mapsto 3) \mapsto s4\_Client), ((s3\_Appli \mapsto (Receive \mapsto SubmitGraduationCertificat \mapsto Client) \mapsto 3) \mapsto s4\_Appli), ((s4\_Client \mapsto (Send \mapsto SubmitPassport \mapsto Appli) \mapsto 4) \mapsto s5\_Client), ((s4\_Appli \mapsto (Receive \mapsto SubmitPassport \mapsto Client) \mapsto 4) \mapsto s5\_Appli), ((s5\_Appli \mapsto (Send \mapsto CheckApproval \mapsto Client) \mapsto 5) \mapsto s9\_Appli), ((s5\_Client \mapsto (Receive \mapsto CheckApproval \mapsto Appli) \mapsto 5) \mapsto s9\_Client), ((s6\_Client \mapsto (Send \mapsto SubmitWorkExperience \mapsto Appli) \mapsto 3) \mapsto s7\_Client), ((s6\_Appli \mapsto (Receive \mapsto SubmitWorkExperience \mapsto Client) \mapsto 3) \mapsto s7\_Appli), ((s7\_Client \mapsto (Send \mapsto TestEnglishAbility \mapsto Appli) \mapsto 4) \mapsto s8\_Client), ((s7\_Appli \mapsto (Receive \mapsto TestEnglishAbility \mapsto Client) \mapsto 4) \mapsto s8\_Appli), ((s8\_Appli \mapsto (Send \mapsto Confirm \mapsto Client) \mapsto 5) \mapsto s9\_Appli), ((s8\_Client \mapsto (Receive \mapsto Confirm \mapsto Appli) \mapsto 5) \mapsto s9\_Client), ((s8\_Appli \mapsto (Send \mapsto Reassess \mapsto Client) \mapsto 5) \mapsto s10\_Appli), ((s8\_Client \mapsto (Receive \mapsto Reassess \mapsto Appli) \mapsto 5) \mapsto s10\_Client)\}$

axm9: $R\_TRACE\_B = \{s0 \mapsto Client \mapsto CheckEligibility \mapsto Appli \mapsto s1 \mapsto 1, s1 \mapsto Client \mapsto Cancel \mapsto Appli \mapsto s2 \mapsto 2, s1 \mapsto Client \mapsto FillInApplicationAsStudent \mapsto Appli \mapsto s3 \mapsto 2, s1 \mapsto Client \mapsto FillInApplication \mapsto Appli \mapsto s6 \mapsto 2, s3 \mapsto Client \mapsto SubmitGraduationCertificat \mapsto Appli \mapsto s4 \mapsto 3, s4 \mapsto Client \mapsto SubmitPassport \mapsto Appli \mapsto s5 \mapsto 4, s5 \mapsto Appli \mapsto CheckApproval \mapsto$

$Client \mapsto s9 \mapsto 5, s6 \mapsto Client \mapsto SubmitWorkExperience \mapsto Appli \mapsto s7 \mapsto 6, s7 \mapsto Client \mapsto TestEnglishAbility \mapsto Appli \mapsto s8 \mapsto 7, s8 \mapsto Appli \mapsto Confirm \mapsto Client \mapsto s9 \mapsto 8, s8 \mapsto Appli \mapsto Reassess \mapsto Client \mapsto s10 \mapsto 8, s9 \mapsto Pend \mapsto End \mapsto Pend \mapsto s11 \mapsto 9, s10 \mapsto Pend \mapsto End \mapsto Pend \mapsto s12 \mapsto 10, s2 \mapsto Pend \mapsto End \mapsto Pend \mapsto s13 \mapsto 11\}$

axm10: $A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto Client \mapsto CheckEligibility \mapsto Appli \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto Client \mapsto CheckEligibility \mapsto Appli \mapsto s \mapsto 2 \mapsto 2, s \mapsto 2 \mapsto Send \mapsto Client \mapsto Cancel \mapsto Appli \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto Client \mapsto Cancel \mapsto Appli \mapsto s \mapsto 4 \mapsto 4, s \mapsto 2 \mapsto Send \mapsto Client \mapsto FillInApplicationAsStudent \mapsto Appli \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto Client \mapsto FillInApplicationAsStudent \mapsto Appli \mapsto s \mapsto 4 \mapsto 4, s \mapsto 4 \mapsto Send \mapsto Client \mapsto SubmitGraduationCertificat \mapsto Appli \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto Client \mapsto SubmitGraduationCertificat \mapsto Appli \mapsto s \mapsto 6 \mapsto 6, s \mapsto 6 \mapsto Send \mapsto Client \mapsto SubmitPassport \mapsto Appli \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto Client \mapsto SubmitPassport \mapsto Appli \mapsto s \mapsto 8 \mapsto 8, s \mapsto 8 \mapsto Send \mapsto Appli \mapsto CheckApproval \mapsto Client \mapsto s \mapsto 9 \mapsto 9, s \mapsto 9 \mapsto Receive \mapsto Appli \mapsto CheckApproval \mapsto Client \mapsto s \mapsto 10 \mapsto 10, s \mapsto 2 \mapsto Send \mapsto Client \mapsto FillInApplication \mapsto Appli \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto Client \mapsto FillInApplication \mapsto Appli \mapsto s \mapsto 4 \mapsto 4, s \mapsto 4 \mapsto Send \mapsto Client \mapsto SubmitWorkExperience \mapsto Appli \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto Client \mapsto SubmitWorkExperience \mapsto Appli \mapsto s \mapsto 6 \mapsto 6, s \mapsto 6 \mapsto Send \mapsto Client \mapsto TestEnglishAbility \mapsto Appli \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto Client \mapsto TestEnglishAbility \mapsto Appli \mapsto s \mapsto 8 \mapsto 8, s \mapsto 8 \mapsto Send \mapsto Appli \mapsto Confirm \mapsto Client \mapsto s \mapsto 9 \mapsto 9, s \mapsto 9 \mapsto Receive \mapsto Appli \mapsto Confirm \mapsto Client \mapsto s \mapsto 10 \mapsto 10, s \mapsto 8 \mapsto Send \mapsto Appli \mapsto Reassess \mapsto Client \mapsto s \mapsto 9 \mapsto 9, s \mapsto 9 \mapsto Receive \mapsto Appli \mapsto Reassess \mapsto Client \mapsto s \mapsto 10 \mapsto 10\}$

axm11: $S\_Next\_States = \{\{((s0\_Client \mapsto (Send \mapsto CheckEligibility \mapsto Appli) \mapsto 1) \mapsto s1\_Client)\} \mapsto \{(Client \mapsto s0\_Client), (Appli \mapsto s0\_Appli)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s0\_Appli)\}, \{((s0\_Appli \mapsto (Receive \mapsto CheckEligibility \mapsto Client) \mapsto 1) \mapsto s1\_Appli)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s0\_Appli)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s1\_Appli)\}, \{((s1\_Client \mapsto (Send \mapsto Cancel \mapsto Appli) \mapsto 2) \mapsto s2\_Client)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \{(Client \mapsto s2\_Client), (Appli \mapsto s1\_Appli)\}, \{((s1\_Appli \mapsto (Receive \mapsto Cancel \mapsto Client) \mapsto 2) \mapsto s2\_Appli)\} \mapsto \{(Client \mapsto s2\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \{(Client \mapsto s2\_Client), (Appli \mapsto s2\_Appli)\}, \{((s1\_Client \mapsto (Send \mapsto FillInApplicationAsStudent \mapsto Appli) \mapsto 2) \mapsto s3\_Client)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \{(Client \mapsto s3\_Client), (Appli \mapsto s1\_Appli)\}, \{((s1\_Appli \mapsto (Receive \mapsto FillInApplicationAsStudent \mapsto Client) \mapsto 2) \mapsto s3\_Appli)\} \mapsto \{(Client \mapsto s3\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \{(Client \mapsto s3\_Client), (Appli \mapsto s3\_Appli)\}, \{((s3\_Client \mapsto (Send \mapsto SubmitGraduationCertificat \mapsto Appli) \mapsto 3) \mapsto s4\_Client)\} \mapsto \{(Client \mapsto s3\_Client), (Appli \mapsto s3\_Appli)\} \mapsto \{(Client \mapsto s4\_Client), (Appli \mapsto s3\_Appli)\}, \{((s3\_Appli \mapsto (Receive \mapsto SubmitGraduationCertificat \mapsto Client) \mapsto 3) \mapsto s4\_Appli)\} \mapsto \{(Client \mapsto s4\_Client), (Appli \mapsto s3\_Appli)\} \mapsto \{(Client \mapsto s4\_Client), (Appli \mapsto s4\_Appli)\}, \{((s4\_Client \mapsto (Send \mapsto SubmitPassport \mapsto Appli) \mapsto 4) \mapsto s5\_Client)\} \mapsto \{(Client \mapsto s4\_Client), (Appli \mapsto s4\_Appli)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s4\_Appli)\}, \{((s4\_Appli \mapsto (Receive \mapsto SubmitPassport \mapsto Client) \mapsto 4) \mapsto s5\_Appli)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s4\_Appli)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s5\_Appli)\}, \{((s5\_Appli \mapsto (Send \mapsto CheckApproval \mapsto Client) \mapsto 5) \mapsto s9\_Appli)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s5\_Appli)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s9\_Appli)\}, \{((s5\_Client \mapsto (Receive \mapsto CheckApproval \mapsto Appli) \mapsto 5) \mapsto s9\_Client)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s9\_Appli)\} \mapsto \{(Client \mapsto s9\_Client), (Appli \mapsto s9\_Appli)\}, \{((s1\_Client \mapsto (Send \mapsto FillInApplication \mapsto Appli) \mapsto 2) \mapsto s6\_Client)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \{(Client \mapsto s6\_Client), (Appli \mapsto s1\_Appli)\}, \{((s1\_Appli \mapsto (Receive \mapsto FillInApplication \mapsto Client) \mapsto 2) \mapsto s6\_Appli)\} \mapsto \{(Client \mapsto s6\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \{(Client \mapsto s6\_Client), (Appli \mapsto s6\_Appli)\}, \{((s6\_Client \mapsto (Send \mapsto SubmitWorkExperience \mapsto Appli) \mapsto 3) \mapsto s7\_Client)\} \mapsto \{(Client \mapsto s6\_Client), (Appli \mapsto s6\_Appli)\} \mapsto \{(Client \mapsto s7\_Client), (Appli \mapsto s6\_Appli)\}, \{((s6\_Appli \mapsto (Receive \mapsto SubmitWorkExperience \mapsto Client) \mapsto 3) \mapsto s7\_Appli)\} \mapsto \{(Client \mapsto s7\_Client), (Appli \mapsto s6\_Appli)\} \mapsto \{(Client \mapsto s7\_Client), (Appli \mapsto s7\_Appli)\}, \{((s7\_Client \mapsto (Send \mapsto TestEnglishAbility \mapsto Appli) \mapsto 4) \mapsto s8\_Client)\} \mapsto \{(Client \mapsto s7\_Client), (Appli \mapsto s7\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s7\_Appli)\}, \{((s7\_Appli \mapsto (Receive \mapsto TestEnglishAbility \mapsto Client) \mapsto 4) \mapsto s8\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s7\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s8\_Appli)\}, \{((s8\_Appli \mapsto (Send \mapsto Confirm \mapsto Client) \mapsto 5) \mapsto s9\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s8\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s9\_Appli)\}, \{((s8\_Client \mapsto (Receive \mapsto Confirm \mapsto Appli) \mapsto 5) \mapsto s9\_Client)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s9\_Appli)\} \mapsto \{(Client \mapsto s9\_Client), (Appli \mapsto s9\_Appli)\}, \{((s8\_Appli \mapsto (Send \mapsto Reassess \mapsto Client) \mapsto 5) \mapsto s10\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto$

$s8\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s10\_Appli)\}, \{((s8\_Client \mapsto (Receive \mapsto Reassess \mapsto Appli) \mapsto 5) \mapsto s10\_Client)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s10\_Appli)\} \mapsto \{(Client \mapsto s10\_Client), (Appli \mapsto s10\_Appli)\}\}$

axm12: $A\_Next\_States = \{\{((s0\_Client \mapsto (Send \mapsto CheckEligibility \mapsto Appli) \mapsto 1) \mapsto s1\_Client)\} \mapsto \{(Client \mapsto s0\_Client), (Appli \mapsto s0\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s0\_Appli)\}, \{((s0\_Appli \mapsto (Receive \mapsto CheckEligibility \mapsto Client) \mapsto 1) \mapsto s1\_Appli)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s0\_Appli)\} \mapsto \{Appli \mapsto CheckEligibility \mapsto 0\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s1\_Appli)\}, \{((s0\_Appli \mapsto (Receive \mapsto CheckEligibility \mapsto Client) \mapsto 1) \mapsto s1\_Appli)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s0\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s1\_Appli)\}, \{((s1\_Client \mapsto (Send \mapsto Cancel \mapsto Appli) \mapsto 2) \mapsto s2\_Client)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s2\_Client), (Appli \mapsto s1\_Appli)\}, \{((s1\_Appli \mapsto (Receive \mapsto Cancel \mapsto Client) \mapsto 2) \mapsto s2\_Appli)\} \mapsto \{(Client \mapsto s2\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \{Appli \mapsto Cancel \mapsto 1\} \mapsto \{(Client \mapsto s2\_Client), (Appli \mapsto s2\_Appli)\}, \{((s1\_Appli \mapsto (Receive \mapsto Cancel \mapsto Client) \mapsto 2) \mapsto s2\_Appli)\} \mapsto \{(Client \mapsto s2\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s2\_Client), (Appli \mapsto s2\_Appli)\}, \{((s1\_Client \mapsto (Send \mapsto FillInApplicationAsStudent \mapsto Appli) \mapsto 2) \mapsto s3\_Client)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s3\_Client), (Appli \mapsto s1\_Appli)\}, \{((s1\_Appli \mapsto (Receive \mapsto FillInApplicationAsStudent \mapsto Client) \mapsto 2) \mapsto s3\_Appli)\} \mapsto \{(Client \mapsto s3\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \{Appli \mapsto FillInApplicationAsStudent \mapsto 1\} \mapsto \{(Client \mapsto s3\_Client), (Appli \mapsto s3\_Appli)\}, \{((s1\_Appli \mapsto (Receive \mapsto FillInApplicationAsStudent \mapsto Client) \mapsto 2) \mapsto s3\_Appli)\} \mapsto \{(Client \mapsto s3\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s3\_Client), (Appli \mapsto s3\_Appli)\}, \{((s3\_Client \mapsto (Send \mapsto SubmitGraduationCertificat \mapsto Appli) \mapsto 3) \mapsto s4\_Client)\} \mapsto \{(Client \mapsto s3\_Client), (Appli \mapsto s3\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s4\_Client), (Appli \mapsto s3\_Appli)\}, \{((s3\_Appli \mapsto (Receive \mapsto SubmitGraduationCertificat \mapsto Client) \mapsto 3) \mapsto s4\_Appli)\} \mapsto \{(Client \mapsto s4\_Client), (Appli \mapsto s3\_Appli)\} \mapsto \{Appli \mapsto SubmitGraduationCertificat \mapsto 2\} \mapsto \{(Client \mapsto s4\_Client), (Appli \mapsto s4\_Appli)\}, \{((s3\_Appli \mapsto (Receive \mapsto SubmitGraduationCertificat \mapsto Client) \mapsto 3) \mapsto s4\_Appli)\} \mapsto \{(Client \mapsto s4\_Client), (Appli \mapsto s3\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s4\_Client), (Appli \mapsto s4\_Appli)\}, \{((s4\_Client \mapsto (Send \mapsto SubmitPassport \mapsto Appli) \mapsto 4) \mapsto s5\_Client)\} \mapsto \{(Client \mapsto s4\_Client), (Appli \mapsto s4\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s4\_Appli)\}, \{((s4\_Appli \mapsto (Receive \mapsto SubmitPassport \mapsto Client) \mapsto 4) \mapsto s5\_Appli)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s4\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s5\_Appli)\}, \{((s4\_Appli \mapsto (Receive \mapsto SubmitPassport \mapsto Client) \mapsto 4) \mapsto s5\_Appli)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s4\_Appli)\} \mapsto \{Appli \mapsto SubmitPassport \mapsto 3\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s5\_Appli)\}, \{((s5\_Appli \mapsto (Send \mapsto CheckApproval \mapsto Client) \mapsto 5) \mapsto s9\_Appli)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s5\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s9\_Appli)\}, \{((s5\_Client \mapsto (Receive \mapsto CheckApproval \mapsto Appli) \mapsto 5) \mapsto s9\_Client)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s9\_Appli)\} \mapsto \{Client \mapsto CheckApproval \mapsto 4\} \mapsto \{(Client \mapsto s9\_Client), (Appli \mapsto s9\_Appli)\}, \{((s5\_Client \mapsto (Receive \mapsto CheckApproval \mapsto Appli) \mapsto 5) \mapsto s9\_Client)\} \mapsto \{(Client \mapsto s5\_Client), (Appli \mapsto s9\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s9\_Client), (Appli \mapsto s9\_Appli)\}, \{((s1\_Client \mapsto (Send \mapsto FillInApplication \mapsto Appli) \mapsto 2) \mapsto s6\_Client)\} \mapsto \{(Client \mapsto s1\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s6\_Client), (Appli \mapsto s1\_Appli)\}, \{((s1\_Appli \mapsto (Receive \mapsto FillInApplication \mapsto Client) \mapsto 2) \mapsto s6\_Appli)\} \mapsto \{(Client \mapsto s6\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \{Appli \mapsto FillInApplication \mapsto 1\} \mapsto \{(Client \mapsto s6\_Client), (Appli \mapsto s6\_Appli)\}, \{((s1\_Appli \mapsto (Receive \mapsto FillInApplication \mapsto Client) \mapsto 2) \mapsto s6\_Appli)\} \mapsto \{(Client \mapsto s6\_Client), (Appli \mapsto s1\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s6\_Client), (Appli \mapsto s6\_Appli)\}, \{((s6\_Client \mapsto (Send \mapsto SubmitWorkExperience \mapsto Appli) \mapsto 3) \mapsto s7\_Client)\} \mapsto \{(Client \mapsto s6\_Client), (Appli \mapsto s6\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s7\_Client), (Appli \mapsto s6\_Appli)\}, \{((s6\_Appli \mapsto (Receive \mapsto SubmitWorkExperience \mapsto Client) \mapsto 3) \mapsto s7\_Appli)\} \mapsto \{(Client \mapsto s7\_Client), (Appli \mapsto s6\_Appli)\} \mapsto \{Appli \mapsto SubmitWorkExperience \mapsto 2\} \mapsto \{(Client \mapsto s7\_Client), (Appli \mapsto s7\_Appli)\}, \{((s6\_Appli \mapsto (Receive \mapsto SubmitWorkExperience \mapsto Client) \mapsto 3) \mapsto s7\_Appli)\} \mapsto \{(Client \mapsto s7\_Client), (Appli \mapsto s6\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s7\_Client), (Appli \mapsto s7\_Appli)\}, \{((s7\_Client \mapsto (Send \mapsto TestEnglishAbility \mapsto Appli) \mapsto 4) \mapsto s8\_Client)\} \mapsto \{(Client \mapsto s7\_Client), (Appli \mapsto s7\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s7\_Appli)\}, \{((s7\_Appli \mapsto (Receive \mapsto TestEnglishAbility \mapsto Client) \mapsto 4) \mapsto s8\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s7\_Appli)\} \mapsto \{Appli \mapsto TestEnglishAbility \mapsto 3\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s8\_Appli)\}, \{((s7\_Appli \mapsto (Receive \mapsto TestEnglishAbility \mapsto Client) \mapsto 4) \mapsto s8\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s7\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s8\_Appli)\}, \{((s8\_Appli \mapsto (Send \mapsto Confirm \mapsto Client) \mapsto 5) \mapsto s9\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s8\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s9\_Appli)\}, \{((s8\_Client \mapsto (Receive \mapsto Confirm \mapsto Appli) \mapsto 5) \mapsto s9\_Client)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s9\_Appli)\} \mapsto \{Client \mapsto Confirm \mapsto 4\} \mapsto$

$\{(Client \mapsto s9\_Client), (Appli \mapsto s9\_Appli)\}, \{((s8\_Client \mapsto (Receive \mapsto Confirm \mapsto Appli) \mapsto 5) \mapsto s9\_Client)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s9\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s9\_Client), (Appli \mapsto s9\_Appli)\}, \{((s8\_Appli \mapsto (Send \mapsto Reassess \mapsto Client) \mapsto 5) \mapsto s10\_Appli)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s8\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s10\_Appli)\}, \{((s8\_Client \mapsto (Receive \mapsto Reassess \mapsto Appli) \mapsto 5) \mapsto s10\_Client)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s10\_Appli)\} \mapsto \{Client \mapsto Reassess \mapsto 4\} \mapsto \{(Client \mapsto s10\_Client), (Appli \mapsto s10\_Appli)\}, \{((s8\_Client \mapsto (Receive \mapsto Reassess \mapsto Appli) \mapsto 5) \mapsto s10\_Client)\} \mapsto \{(Client \mapsto s8\_Client), (Appli \mapsto s10\_Appli)\} \mapsto \varnothing \mapsto \{(Client \mapsto s10\_Client), (Appli \mapsto s10\_Appli)\}\}$

axm13: $Initial\_state\_value = s0$

axm14: $CP\_Final\_states\_value = \{s0\}$

axm15: $Prophecy\_value = 5$

axm16: $End\_message = End$

axm17: $S\_GS\_value = \{Client \mapsto s0\_Client, Appli \mapsto s0\_Appli\}$

axm18: $A\_GS\_value = \{Client \mapsto s0\_Client, Appli \mapsto s0\_Appli\}$

axm19: $Last\_cp\_trans\_value = s0 \mapsto Client \mapsto CheckEligibility \mapsto Appli \mapsto s1 \mapsto 1$

END

CONTEXT CS6
EXTENDS LTS_ASYNC_CONTEXT
CONSTANTS

A
B
C
D
bwin
cwin
sig
message
blose
close
busy
free
End
s0
s1
s2
s3
s4
s5
Pend
s0_A
s1_A
s2_A
s0_B
s1_B
s2_B
s3_B
s0_C
s1_C
s2_C
s3_C
s0_D
s1_D
s

AXIOMS

axm1: $partition(PEERS, \{A\}, \{B\}, \{C\}, \{D\}, \{Pend\})$

axm2: $partition(MESSAGES, \{bwin\}, \{cwin\}, \{sig\}, \{message\}, \{blose\}, \{close\}, \{busy\}, \{free\}, \{End\})$

axm3: $partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\})$

axm4: $CPs\_B = \{s0 \mapsto A \mapsto cwin \mapsto C \mapsto s1 \mapsto 1, s0 \mapsto A \mapsto bwin \mapsto B \mapsto s2 \mapsto 1, s0 \mapsto C \mapsto busy \mapsto D \mapsto s3 \mapsto 1, s0 \mapsto A \mapsto free \mapsto D \mapsto s4 \mapsto 1, s1 \mapsto C \mapsto blose \mapsto B \mapsto s3 \mapsto 2, s2 \mapsto B \mapsto close \mapsto C \mapsto s3 \mapsto 3, s3 \mapsto C \mapsto message \mapsto A \mapsto s4 \mapsto 4, s3 \mapsto B \mapsto sig \mapsto A \mapsto s4 \mapsto 5, s4 \mapsto Pend \mapsto End \mapsto Pend \mapsto s5 \mapsto 6\}$

axm5: $CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto cwin \mapsto C \mapsto A \mapsto Receive \mapsto cwin \mapsto s1 \mapsto 1, s0 \mapsto Send \mapsto bwin \mapsto B \mapsto A \mapsto Receive \mapsto bwin \mapsto s2 \mapsto 1, s0 \mapsto Send \mapsto busy \mapsto D \mapsto C \mapsto Receive \mapsto busy \mapsto s3 \mapsto 1, s0 \mapsto Send \mapsto free \mapsto D \mapsto A \mapsto Receive \mapsto free \mapsto s4 \mapsto 1, s1 \mapsto Send \mapsto blose \mapsto B \mapsto C \mapsto Receive \mapsto blose \mapsto s3 \mapsto 2, s2 \mapsto Send \mapsto close \mapsto C \mapsto B \mapsto Receive \mapsto close \mapsto s3 \mapsto 3, s3 \mapsto Send \mapsto message \mapsto A \mapsto C \mapsto Receive \mapsto message \mapsto s4 \mapsto 4, s3 \mapsto Send \mapsto sig \mapsto A \mapsto B \mapsto Receive \mapsto sig \mapsto s4 \mapsto 5, s4 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s5 \mapsto 6\}$

axm6: $partition(A\_STATES, \{s0\_A\}, \{s1\_A\}, \{s2\_A\}, \{s0\_B\}, \{s1\_B\}, \{s2\_B\}, \{s3\_B\}, \{s0\_C\}, \{s1\_C\}, \{s2\_C\}, \{s3\_C\}, \{s0\_D\}, \{s1\_D\})$

axm7: $partition(A\_TRACE\_STATES, \{s\})$

axm8: $PEERs\_B = \{((s0\_A \mapsto (Send \mapsto cwin \mapsto C) \mapsto 1) \mapsto s1\_A), ((s0\_C \mapsto (Receive \mapsto cwin \mapsto A) \mapsto 1) \mapsto s1\_C), ((s0\_A \mapsto (Send \mapsto bwin \mapsto B) \mapsto 1) \mapsto s1\_A), ((s0\_B \mapsto (Receive \mapsto bwin \mapsto A) \mapsto 1) \mapsto s1\_B), ((s0\_A \mapsto (Send \mapsto free \mapsto D) \mapsto 1) \mapsto s2\_A), ((s0\_D \mapsto (Receive \mapsto free \mapsto A) \mapsto 1) \mapsto s1\_D), ((s0\_C \mapsto (Send \mapsto busy \mapsto D) \mapsto 1) \mapsto s3\_C), ((s0\_D \mapsto (Receive \mapsto busy \mapsto C) \mapsto 1) \mapsto s1\_D), ((s1\_C \mapsto (Send \mapsto blose \mapsto B) \mapsto 2) \mapsto s2\_C), ((s0\_B \mapsto (Receive \mapsto blose \mapsto C) \mapsto 2) \mapsto s2\_B), ((s1\_B \mapsto (Send \mapsto close \mapsto C) \mapsto 3) \mapsto s2\_B), ((s0\_C \mapsto (Receive \mapsto close \mapsto B) \mapsto 3) \mapsto s2\_C), ((s2\_C \mapsto (Send \mapsto message \mapsto A) \mapsto 4) \mapsto s3\_C), ((s1\_A \mapsto (Receive \mapsto message \mapsto C) \mapsto 4) \mapsto s2\_A), ((s2\_B \mapsto (Send \mapsto sig \mapsto A) \mapsto 5) \mapsto s3\_B), ((s1\_A \mapsto (Receive \mapsto sig \mapsto B) \mapsto 5) \mapsto s2\_A)\}$

axm9: $R\_TRACE\_B = \{s0 \mapsto A \mapsto cwin \mapsto C \mapsto s1 \mapsto 1, s0 \mapsto A \mapsto bwin \mapsto B \mapsto s2 \mapsto 1, s0 \mapsto C \mapsto busy \mapsto D \mapsto s3 \mapsto 1, s0 \mapsto A \mapsto free \mapsto D \mapsto s4 \mapsto 1, s1 \mapsto C \mapsto blose \mapsto B \mapsto s3 \mapsto 2, s2 \mapsto B \mapsto close \mapsto C \mapsto s3 \mapsto 3, s3 \mapsto C \mapsto message \mapsto A \mapsto s4 \mapsto 4, s3 \mapsto B \mapsto sig \mapsto A \mapsto s4 \mapsto 5, s4 \mapsto Pend \mapsto End \mapsto Pend \mapsto s5 \mapsto 6\}$

axm10: $A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto A \mapsto cwin \mapsto C \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto A \mapsto cwin \mapsto C \mapsto s \mapsto 2 \mapsto 2, s \mapsto 2 \mapsto Send \mapsto C \mapsto blose \mapsto B \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto C \mapsto blose \mapsto B \mapsto s \mapsto 4 \mapsto 4, s \mapsto 4 \mapsto Send \mapsto C \mapsto message \mapsto A \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto C \mapsto message \mapsto A \mapsto s \mapsto 6 \mapsto 6, s \mapsto 4 \mapsto Send \mapsto B \mapsto sig \mapsto A \mapsto s \mapsto 5 \mapsto 5, s \mapsto 6 \mapsto Receive \mapsto B \mapsto sig \mapsto A \mapsto s \mapsto 6 \mapsto 6, s \mapsto 0 \mapsto Send \mapsto A \mapsto bwin \mapsto B \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto A \mapsto bwin \mapsto B \mapsto s \mapsto 2 \mapsto 2, s \mapsto 2 \mapsto Send \mapsto B \mapsto close \mapsto C \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto B \mapsto close \mapsto C \mapsto s \mapsto 4 \mapsto 4, s \mapsto 0 \mapsto Send \mapsto C \mapsto busy \mapsto D \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto C \mapsto busy \mapsto D \mapsto s \mapsto 2 \mapsto 2, s \mapsto 4 \mapsto Send \mapsto B \mapsto sig \mapsto A \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto B \mapsto sig \mapsto A \mapsto s \mapsto 6 \mapsto 6, s \mapsto 0 \mapsto Send \mapsto A \mapsto free \mapsto D \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto A \mapsto free \mapsto D \mapsto s \mapsto 2 \mapsto 2, s \mapsto 6 \mapsto Send \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 8 \mapsto 8\}$

axm11: $S\_Next\_States = \{\{((s0\_A \mapsto (Send \mapsto cwin \mapsto C) \mapsto 1) \mapsto s1\_A)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_C \mapsto (Receive \mapsto cwin \mapsto A) \mapsto 1) \mapsto s1\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s1\_C), (D \mapsto s0\_D)\}, \{((s0\_A \mapsto (Send \mapsto bwin \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_B \mapsto (Receive \mapsto bwin \mapsto A) \mapsto 1) \mapsto s1\_B)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_A \mapsto (Send \mapsto free \mapsto D) \mapsto 1) \mapsto s2\_A)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_D \mapsto (Receive \mapsto free \mapsto A) \mapsto 1) \mapsto s1\_D)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s1\_D)\}, \{((s0\_C \mapsto (Send \mapsto busy \mapsto D) \mapsto 1) \mapsto s3\_C)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s0\_D \mapsto (Receive \mapsto busy \mapsto C) \mapsto 1) \mapsto s1\_D)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s3\_C), (D \mapsto s1\_D)\}, \{((s1\_C \mapsto (Send \mapsto blose \mapsto B) \mapsto 2) \mapsto s2\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s1\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s0\_B \mapsto (Receive \mapsto blose \mapsto C) \mapsto 2) \mapsto s2\_B)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s1\_B \mapsto (Send \mapsto close \mapsto C) \mapsto 3) \mapsto s2\_B)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_C \mapsto (Receive \mapsto close \mapsto B) \mapsto 3) \mapsto s2\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s2\_C \mapsto (Send \mapsto message \mapsto A) \mapsto 4) \mapsto s3\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s1\_A \mapsto (Receive \mapsto message \mapsto C) \mapsto 4) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s2\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s2\_C \mapsto (Send \mapsto message \mapsto A) \mapsto 4) \mapsto s3\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s1\_A \mapsto (Receive \mapsto message \mapsto C) \mapsto 4) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s2\_B \mapsto (Send \mapsto sig \mapsto A) \mapsto 5) \mapsto s3\_B)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s3\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s1\_A \mapsto (Receive \mapsto sig \mapsto B) \mapsto 5) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s3\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s3\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}\}$

axm12: $A\_Next\_States = \{\{((s0\_A \mapsto (Send \mapsto cwin \mapsto C) \mapsto 1) \mapsto s1\_A)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_C \mapsto (Receive \mapsto cwin \mapsto A) \mapsto 1) \mapsto s1\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{C \mapsto cwin \mapsto 0\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s1\_C), (D \mapsto s0\_D)\}, \{((s0\_C \mapsto (Receive \mapsto cwin \mapsto A) \mapsto 1) \mapsto s1\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s1\_C), (D \mapsto s0\_D)\}, \{((s0\_A \mapsto (Send \mapsto bwin \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_B \mapsto (Receive \mapsto bwin \mapsto A) \mapsto 1) \mapsto s1\_B)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{B \mapsto bwin \mapsto 0\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_B \mapsto (Receive \mapsto bwin \mapsto A) \mapsto 1) \mapsto s1\_B)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_A \mapsto (Send \mapsto free \mapsto D) \mapsto 1) \mapsto s2\_A)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s2\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_D \mapsto (Receive \mapsto free \mapsto A) \mapsto 1) \mapsto s1\_D)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{D \mapsto free \mapsto 0\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s1\_D)\}, \{((s0\_D \mapsto (Receive \mapsto free \mapsto A) \mapsto 1) \mapsto s1\_D)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s2\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s1\_D)\}, \{((s0\_C \mapsto (Send \mapsto busy \mapsto D) \mapsto 1) \mapsto s3\_C)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s0\_D \mapsto (Receive \mapsto busy \mapsto C) \mapsto 1) \mapsto s1\_D)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\} \mapsto \{D \mapsto busy \mapsto 0\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s3\_C), (D \mapsto s1\_D)\}, \{((s0\_D \mapsto (Receive \mapsto busy \mapsto C) \mapsto 1) \mapsto s1\_D)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (C \mapsto s3\_C), (D \mapsto s1\_D)\}, \{((s1\_C \mapsto (Send \mapsto blose \mapsto B) \mapsto 2) \mapsto s2\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s1\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s0\_B \mapsto (Receive \mapsto blose \mapsto C) \mapsto 2) \mapsto s2\_B)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \{B \mapsto blose \mapsto 1\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s0\_B \mapsto (Receive \mapsto blose \mapsto C) \mapsto 2) \mapsto s2\_B)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s0\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s1\_B \mapsto (Send \mapsto close \mapsto C) \mapsto 3) \mapsto s2\_B)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\}, \{((s0\_C \mapsto (Receive \mapsto close \mapsto B) \mapsto 3) \mapsto s2\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \{C \mapsto close \mapsto 1\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s0\_C \mapsto (Receive \mapsto close \mapsto B) \mapsto 3) \mapsto s2\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s0\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s2\_C \mapsto (Send \mapsto message \mapsto A) \mapsto 4) \mapsto s3\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s1\_A \mapsto (Receive \mapsto message \mapsto C) \mapsto 4) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\} \mapsto \{A \mapsto message \mapsto 2\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s2\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s1\_A \mapsto (Receive \mapsto message \mapsto C) \mapsto 4) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s2\_A), (B \mapsto s2\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s2\_C \mapsto (Send \mapsto message \mapsto A) \mapsto 4) \mapsto s3\_C)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s1\_A \mapsto (Receive \mapsto message \mapsto C) \mapsto 4) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\} \mapsto \{A \mapsto message \mapsto 2\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s1\_A \mapsto (Receive \mapsto message \mapsto C) \mapsto 4) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (C \mapsto s3\_C), (D \mapsto s0\_D)\}, \{((s2\_B \mapsto (Send \mapsto sig \mapsto A) \mapsto 5) \mapsto s3\_B)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s2\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s3\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s1\_A \mapsto (Receive \mapsto sig \mapsto B) \mapsto 5) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s3\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \{A \mapsto sig \mapsto 2\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s3\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}, \{((s1\_A \mapsto (Receive \mapsto sig \mapsto B) \mapsto 5) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s3\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\} \mapsto \varnothing \mapsto \{(A \mapsto s2\_A), (B \mapsto s3\_B), (C \mapsto s2\_C), (D \mapsto s0\_D)\}\}$

axm13: $Initial\_state\_value = s0$

axm14: $CP\_Final\_states\_value = \{s0\}$

axm15: $Prophecy\_value = 3$

axm16: $End\_message = End$

axm17: $S\_GS\_value = \{A \mapsto s0\_A, B \mapsto s0\_B, C \mapsto s0\_C, D \mapsto s0\_D\}$

axm18: $A\_GS\_value = \{A \mapsto s0\_A, B \mapsto s0\_B, C \mapsto s0\_C, D \mapsto s0\_D\}$

axm19: $Last\_cp\_trans\_value = s0 \mapsto A \mapsto cwin \mapsto C \mapsto s1 \mapsto 1$

END

CONTEXT CS7
EXTENDS LTS_ASYNC_CONTEXT
CONSTANTS

A
B
L
Pend
OrderOp
deliverOp
End
deliver_confOp
deliveryOp
terminateOp
terminateLOp
get_statusOp
get_statusLOp
statusOp
s0
s1
s2
s3
s4
s5
s6
s7
s8
s9
s10
s11
s0_A
s1_A
s2_A
s3_A
s4_A
s5_A
s6_A
s7_A
s8_A
s9_A
s0_B
s1_B
s2_B
s3_B
s4_B
s0_L
s1_L
s2_L
s3_L
s4_L
s5_L
s

AXIOMS

axm1: $partition(PEERS, \{A\}, \{B\}, \{L\}, \{Pend\})$

axm2: $partition(MESSAGES, \{OrderOp\}, \{deliverOp\}, \{deliver\_confOp\}, \{deliveryOp\}, \{terminateOp\}, \{terminateLOp\}, \{get\_statusOp\}, \{get\_statusLOp\}, \{statusOp\}, \{End\})$

axm3: $partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\}, \{s6\}, \{s7\}, \{s8\}, \{s9\}, \{s10\}, \{s11\})$

axm4: $CPs\_B = \{s0 \mapsto B \mapsto OrderOp \mapsto A \mapsto s1 \mapsto 1, s1 \mapsto A \mapsto deliverOp \mapsto L \mapsto s2 \mapsto 2, s2 \mapsto L \mapsto deliver\_confOp \mapsto A \mapsto s3 \mapsto 3, s3 \mapsto A \mapsto deliveryOp \mapsto B \mapsto s4 \mapsto 4, s4 \mapsto B \mapsto terminateOp \mapsto A \mapsto s6 \mapsto 5, s6 \mapsto A \mapsto terminateLOp \mapsto L \mapsto s8 \mapsto 6, s4 \mapsto B \mapsto get\_statusOp \mapsto A \mapsto s5 \mapsto 7, s5 \mapsto A \mapsto get\_statusOp \mapsto L \mapsto s7 \mapsto 8, s7 \mapsto L \mapsto get\_statusLOp \mapsto A \mapsto s9 \mapsto 9, s9 \mapsto A \mapsto statusOp \mapsto B \mapsto s4 \mapsto 10, s8 \mapsto Pend \mapsto End \mapsto Pend \mapsto s10 \mapsto 11\}$

axm5: $CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto OrderOp \mapsto A \mapsto B \mapsto Receive \mapsto OrderOp \mapsto s1 \mapsto 1, s1 \mapsto Send \mapsto deliverOp \mapsto L \mapsto A \mapsto Receive \mapsto deliverOp \mapsto s2 \mapsto 2, s2 \mapsto Send \mapsto deliver\_confOp \mapsto A \mapsto L \mapsto Receive \mapsto deliver\_confOp \mapsto s3 \mapsto 3, s3 \mapsto Send \mapsto deliveryOp \mapsto B \mapsto A \mapsto Receive \mapsto deliveryOp \mapsto s4 \mapsto 4, s4 \mapsto Send \mapsto terminateOp \mapsto A \mapsto B \mapsto Receive \mapsto terminateOp \mapsto s6 \mapsto 5, s6 \mapsto Send \mapsto terminateLOp \mapsto L \mapsto A \mapsto Receive \mapsto terminateLOp \mapsto s8 \mapsto 6, s4 \mapsto Send \mapsto get\_statusOp \mapsto A \mapsto B \mapsto Receive \mapsto get\_statusOp \mapsto s5 \mapsto 7, s5 \mapsto Send \mapsto get\_statusOp \mapsto L \mapsto A \mapsto Receive \mapsto get\_statusOp \mapsto s7 \mapsto 8, s7 \mapsto Send \mapsto get\_statusLOp \mapsto A \mapsto L \mapsto Receive \mapsto get\_statusLOp \mapsto s9 \mapsto 9, s9 \mapsto Send \mapsto statusOp \mapsto B \mapsto A \mapsto Receive \mapsto statusOp \mapsto s4 \mapsto 10, s8 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s10 \mapsto 11\}$

axm6: $partition(A\_STATES, \{s0\_A\}, \{s1\_A\}, \{s2\_A\}, \{s3\_A\}, \{s4\_A\}, \{s5\_A\}, \{s6\_A\}, \{s7\_A\}, \{s8\_A\}, \{s9\_A\}, \{s0\_B\}, \{s1\_B\}, \{s2\_B\}, \{s3\_B\}, \{s4\_B\}, \{s0\_L\}, \{s1\_L\}, \{s2\_L\}, \{s3\_L\}, \{s4\_L\}, \{s5\_L\})$

axm7: $partition(A\_TRACE\_STATES, \{s\})$

axm8: $PEERs\_B = \{((s0\_B \mapsto (Send \mapsto OrderOp \mapsto A) \mapsto 1) \mapsto s1\_B), ((s0\_A \mapsto (Receive \mapsto OrderOp \mapsto B) \mapsto 1) \mapsto s1\_A), ((s1\_A \mapsto (Send \mapsto deliverOp \mapsto L) \mapsto 2) \mapsto s2\_A), ((s0\_L \mapsto (Receive \mapsto deliverOp \mapsto A) \mapsto 2) \mapsto s1\_L), ((s1\_L \mapsto (Send \mapsto deliver\_confOp \mapsto A) \mapsto 3) \mapsto s2\_L), ((s2\_A \mapsto (Receive \mapsto deliver\_confOp \mapsto L) \mapsto 3) \mapsto s3\_A), ((s3\_A \mapsto (Send \mapsto deliveryOp \mapsto B) \mapsto 4) \mapsto s4\_A), ((s1\_B \mapsto (Receive \mapsto deliveryOp \mapsto A) \mapsto 4) \mapsto s2\_B), ((s2\_B \mapsto (Send \mapsto terminateOp \mapsto A) \mapsto 5) \mapsto s4\_B), ((s4\_A \mapsto (Receive \mapsto terminateOp \mapsto B) \mapsto 5) \mapsto s5\_A), ((s2\_B \mapsto (Send \mapsto get\_statusOp \mapsto A) \mapsto 5) \mapsto s3\_B), ((s4\_A \mapsto (Receive \mapsto get\_statusOp \mapsto B) \mapsto 5) \mapsto s6\_A), ((s5\_A \mapsto (Send \mapsto terminateLOp \mapsto L) \mapsto 6) \mapsto s7\_A), ((s2\_L \mapsto (Receive \mapsto terminateLOp \mapsto A) \mapsto 6) \mapsto s4\_L), ((s6\_A \mapsto (Send \mapsto get\_statusOp \mapsto L) \mapsto 7) \mapsto s8\_A), ((s2\_L \mapsto (Receive \mapsto get\_statusOp \mapsto A) \mapsto 7) \mapsto s3\_L), ((s3\_L \mapsto (Send \mapsto get\_statusLOp \mapsto A) \mapsto 8) \mapsto s2\_L), ((s8\_A \mapsto (Receive \mapsto get\_statusLOp \mapsto L) \mapsto 8) \mapsto s9\_A), ((s9\_A \mapsto (Send \mapsto statusOp \mapsto B) \mapsto 9) \mapsto s4\_A), ((s3\_B \mapsto (Receive \mapsto statusOp \mapsto A) \mapsto 9) \mapsto s2\_B)\}$

axm9: $R\_TRACE\_B = \{s0 \mapsto B \mapsto OrderOp \mapsto A \mapsto s1 \mapsto 1, s1 \mapsto A \mapsto deliverOp \mapsto L \mapsto s2 \mapsto 2, s2 \mapsto L \mapsto deliver\_confOp \mapsto A \mapsto s3 \mapsto 3, s3 \mapsto A \mapsto deliveryOp \mapsto B \mapsto s4 \mapsto 4, s4 \mapsto B \mapsto terminateOp \mapsto A \mapsto s6 \mapsto 5, s6 \mapsto A \mapsto terminateLOp \mapsto L \mapsto s8 \mapsto 6, s4 \mapsto B \mapsto get\_statusOp \mapsto A \mapsto s5 \mapsto 7, s5 \mapsto A \mapsto get\_statusOp \mapsto L \mapsto s7 \mapsto 8, s7 \mapsto L \mapsto get\_statusLOp \mapsto A \mapsto s9 \mapsto 9, s9 \mapsto A \mapsto statusOp \mapsto B \mapsto s4 \mapsto 10, s8 \mapsto Pend \mapsto End \mapsto Pend \mapsto s10 \mapsto 11\}$

axm10: $A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto B \mapsto OrderOp \mapsto A \mapsto s \mapsto 1 \mapsto 1, s \mapsto 1 \mapsto Receive \mapsto B \mapsto OrderOp \mapsto A \mapsto s \mapsto 2 \mapsto 2, s \mapsto 2 \mapsto Send \mapsto A \mapsto deliverOp \mapsto L \mapsto s \mapsto 3 \mapsto 3, s \mapsto 3 \mapsto Receive \mapsto A \mapsto deliverOp \mapsto L \mapsto s \mapsto 4 \mapsto 4, s \mapsto 4 \mapsto Send \mapsto L \mapsto deliver\_confOp \mapsto A \mapsto s \mapsto 5 \mapsto 5, s \mapsto 5 \mapsto Receive \mapsto L \mapsto deliver\_confOp \mapsto A \mapsto s \mapsto 6 \mapsto 6, s \mapsto 6 \mapsto Send \mapsto A \mapsto deliveryOp \mapsto B \mapsto s \mapsto 7 \mapsto 7, s \mapsto 7 \mapsto Receive \mapsto A \mapsto deliveryOp \mapsto B \mapsto s \mapsto 8 \mapsto 8, s \mapsto 8 \mapsto Send \mapsto B \mapsto terminateOp \mapsto A \mapsto s \mapsto 9 \mapsto 9, s \mapsto 9 \mapsto Receive \mapsto B \mapsto terminateOp \mapsto A \mapsto s \mapsto 10 \mapsto 10, s \mapsto 10 \mapsto Send \mapsto A \mapsto terminateLOp \mapsto L \mapsto s \mapsto 11 \mapsto 11, s \mapsto 11 \mapsto Receive \mapsto A \mapsto terminateLOp \mapsto L \mapsto s \mapsto 12 \mapsto 12, s \mapsto 8 \mapsto Send \mapsto B \mapsto get\_statusOp \mapsto A \mapsto s \mapsto 9 \mapsto 9, s \mapsto 9 \mapsto Receive \mapsto B \mapsto get\_statusOp \mapsto A \mapsto s \mapsto 10 \mapsto 10, s \mapsto 10 \mapsto Send \mapsto A \mapsto get\_statusOp \mapsto L \mapsto s \mapsto 11 \mapsto 11, s \mapsto 11 \mapsto Receive \mapsto A \mapsto get\_statusOp \mapsto L \mapsto s \mapsto 12 \mapsto 12, s \mapsto 12 \mapsto Send \mapsto L \mapsto get\_statusLOp \mapsto A \mapsto s \mapsto 13 \mapsto 13, s \mapsto 13 \mapsto Receive \mapsto L \mapsto get\_statusLOp \mapsto A \mapsto s \mapsto 14 \mapsto 14, s \mapsto 14 \mapsto Send \mapsto A \mapsto statusOp \mapsto B \mapsto s \mapsto 15 \mapsto 15, s \mapsto 15 \mapsto Receive \mapsto A \mapsto statusOp \mapsto B \mapsto s \mapsto 16 \mapsto 16, s \mapsto 16 \mapsto Send \mapsto B \mapsto terminateOp \mapsto A \mapsto s \mapsto 17 \mapsto 17, s \mapsto 17 \mapsto Receive \mapsto B \mapsto terminateOp \mapsto A \mapsto s \mapsto 18 \mapsto 18, s \mapsto 18 \mapsto Send \mapsto A \mapsto terminateLOp \mapsto L \mapsto s \mapsto 19 \mapsto 19, s \mapsto 19 \mapsto Receive \mapsto A \mapsto terminateLOp \mapsto L \mapsto s \mapsto 20 \mapsto 20, s \mapsto 20 \mapsto Send \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 21 \mapsto 21, s \mapsto 21 \mapsto Receive \mapsto Pend \mapsto End \mapsto Pend \mapsto s \mapsto 22 \mapsto 22\}$

axm11: $S\_Next\_States = \{\{((s0\_B \mapsto (Send \mapsto OrderOp \mapsto A) \mapsto 1) \mapsto s1\_B)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (L \mapsto s0\_L)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\}, \{((s0\_A \mapsto (Receive \mapsto OrderOp \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto$

$s0\_L)\}, \{((s1\_A \mapsto (Send \mapsto deliverOp \mapsto L) \mapsto 2) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\}, \{((s0\_L \mapsto (Receive \mapsto deliverOp \mapsto A) \mapsto 2) \mapsto s1\_L)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s1\_L)\}, \{((s1\_L \mapsto (Send \mapsto deliver\_confOp \mapsto A) \mapsto 3) \mapsto s2\_L)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s1\_L)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\}, \{((s2\_A \mapsto (Receive \mapsto deliver\_confOp \mapsto L) \mapsto 3) \mapsto s3\_A)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s3\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\}, \{((s3\_A \mapsto (Send \mapsto deliveryOp \mapsto B) \mapsto 4) \mapsto s4\_A)\} \mapsto \{(A \mapsto s3\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\}, \{((s1\_B \mapsto (Receive \mapsto deliveryOp \mapsto A) \mapsto 4) \mapsto s2\_B)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\}, \{((s2\_B \mapsto (Send \mapsto terminateOp \mapsto A) \mapsto 5) \mapsto s4\_B)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\}, \{((s4\_A \mapsto (Receive \mapsto terminateOp \mapsto B) \mapsto 5) \mapsto s5\_A)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s5\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\}, \{((s2\_B \mapsto (Send \mapsto get\_statusOp \mapsto A) \mapsto 5) \mapsto s3\_B)\} \mapsto \{(A \mapsto s5\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s5\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s4\_A \mapsto (Receive \mapsto get\_statusOp \mapsto B) \mapsto 5) \mapsto s6\_A)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s6\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s5\_A \mapsto (Send \mapsto terminateLOp \mapsto L) \mapsto 6) \mapsto s7\_A)\} \mapsto \{(A \mapsto s5\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s7\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s2\_L \mapsto (Receive \mapsto terminateLOp \mapsto A) \mapsto 6) \mapsto s4\_L)\} \mapsto \{(A \mapsto s7\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s7\_A), (B \mapsto s3\_B), (L \mapsto s4\_L)\}, \{((s6\_A \mapsto (Send \mapsto get\_statusOp \mapsto L) \mapsto 7) \mapsto s8\_A)\} \mapsto \{(A \mapsto s6\_A), (B \mapsto s3\_B), (L \mapsto s4\_L)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s4\_L)\}, \{((s2\_L \mapsto (Receive \mapsto get\_statusOp \mapsto A) \mapsto 7) \mapsto s3\_L)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s3\_L)\}, \{((s3\_L \mapsto (Send \mapsto get\_statusLOp \mapsto A) \mapsto 8) \mapsto s2\_L)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s3\_L)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s8\_A \mapsto (Receive \mapsto get\_statusLOp \mapsto L) \mapsto 8) \mapsto s9\_A)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s9\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s9\_A \mapsto (Send \mapsto statusOp \mapsto B) \mapsto 9) \mapsto s4\_A)\} \mapsto \{(A \mapsto s9\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s3\_B \mapsto (Receive \mapsto statusOp \mapsto A) \mapsto 9) \mapsto s2\_B)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\}\}$

axm12: $A\_Next\_States = \{\{((s0\_B \mapsto (Send \mapsto OrderOp \mapsto A) \mapsto 1) \mapsto s1\_B)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s0\_B), (L \mapsto s0\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\}, \{((s0\_A \mapsto (Receive \mapsto OrderOp \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \{A \mapsto OrderOp \mapsto 0\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\}, \{((s0\_A \mapsto (Receive \mapsto OrderOp \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto \{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\}, \{((s1\_A \mapsto (Send \mapsto deliverOp \mapsto L) \mapsto 2) \mapsto s2\_A)\} \mapsto \{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\}, \{((s0\_L \mapsto (Receive \mapsto deliverOp \mapsto A) \mapsto 2) \mapsto s1\_L)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \{L \mapsto deliverOp \mapsto 1\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s1\_L)\}, \{((s0\_L \mapsto (Receive \mapsto deliverOp \mapsto A) \mapsto 2) \mapsto s1\_L)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s1\_L)\}, \{((s1\_L \mapsto (Send \mapsto deliver\_confOp \mapsto A) \mapsto 3) \mapsto s2\_L)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s1\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\}, \{((s2\_A \mapsto (Receive \mapsto deliver\_confOp \mapsto L) \mapsto 3) \mapsto s3\_A)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\} \mapsto \{A \mapsto deliver\_confOp \mapsto 2\} \mapsto \{(A \mapsto s3\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\}, \{((s2\_A \mapsto (Receive \mapsto deliver\_confOp \mapsto L) \mapsto 3) \mapsto s3\_A)\} \mapsto \{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s3\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\}, \{((s3\_A \mapsto (Send \mapsto deliveryOp \mapsto B) \mapsto 4) \mapsto s4\_A)\} \mapsto \{(A \mapsto s3\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s4\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\}, \{((s1\_B \mapsto (Receive \mapsto deliveryOp \mapsto A) \mapsto 4) \mapsto s2\_B)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\} \mapsto \{B \mapsto deliveryOp \mapsto 3\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\}, \{((s1\_B \mapsto (Receive \mapsto deliveryOp \mapsto A) \mapsto 4) \mapsto s2\_B)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s1\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s4\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\}, \{((s2\_B \mapsto (Send \mapsto terminateOp \mapsto A) \mapsto 5) \mapsto s4\_B)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s4\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\}, \{((s4\_A \mapsto (Receive \mapsto terminateOp \mapsto B) \mapsto 5) \mapsto s5\_A)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \{A \mapsto terminateOp \mapsto 4\} \mapsto \{(A \mapsto s5\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\}, \{((s4\_A \mapsto (Receive \mapsto terminateOp \mapsto B) \mapsto 5) \mapsto s5\_A)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s5\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\}, \{((s2\_B \mapsto (Send \mapsto get\_statusOp \mapsto A) \mapsto 5) \mapsto s3\_B)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s4\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s4\_A \mapsto (Receive \mapsto get\_statusOp \mapsto B) \mapsto 5) \mapsto s6\_A)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{A \mapsto get\_statusOp \mapsto 4\} \mapsto \{(A \mapsto s6\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s4\_A \mapsto (Receive \mapsto get\_statusOp \mapsto B) \mapsto 5) \mapsto s6\_A)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s6\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s5\_A \mapsto (Send \mapsto terminateLOp \mapsto L) \mapsto 6) \mapsto s7\_A)\} \mapsto \{(A \mapsto s5\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s7\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\}, \{((s2\_L \mapsto (Receive \mapsto terminateLOp \mapsto A) \mapsto$

$6) \mapsto s4\_L)\} \mapsto \{(A \mapsto s7\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \{L \mapsto terminateLOp \mapsto 5\} \mapsto \{(A \mapsto s7\_A), (B \mapsto s4\_B), (L \mapsto s4\_L)\}, \{((s2\_L \mapsto (Receive \mapsto terminateLOp \mapsto A) \mapsto 6) \mapsto s4\_L)\} \mapsto \{(A \mapsto s7\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s7\_A), (B \mapsto s4\_B), (L \mapsto s4\_L)\}, \{((s6\_A \mapsto (Send \mapsto get\_statusOp \mapsto L) \mapsto 7) \mapsto s8\_A)\} \mapsto \{(A \mapsto s6\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s2\_L \mapsto (Receive \mapsto get\_statusOp \mapsto A) \mapsto 7) \mapsto s3\_L)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{L \mapsto get\_statusOp \mapsto 5\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s3\_L)\}, \{((s2\_L \mapsto (Receive \mapsto get\_statusOp \mapsto A) \mapsto 7) \mapsto s3\_L)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s3\_L)\}, \{((s3\_L \mapsto (Send \mapsto get\_statusLOp \mapsto A) \mapsto 8) \mapsto s2\_L)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s3\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s8\_A \mapsto (Receive \mapsto get\_statusLOp \mapsto L) \mapsto 8) \mapsto s9\_A)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{A \mapsto get\_statusLOp \mapsto 6\} \mapsto \{(A \mapsto s9\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s8\_A \mapsto (Receive \mapsto get\_statusLOp \mapsto L) \mapsto 8) \mapsto s9\_A)\} \mapsto \{(A \mapsto s8\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s9\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s9\_A \mapsto (Send \mapsto statusOp \mapsto B) \mapsto 9) \mapsto s4\_A)\} \mapsto \{(A \mapsto s9\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s4\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\}, \{((s3\_B \mapsto (Receive \mapsto statusOp \mapsto A) \mapsto 9) \mapsto s2\_B)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \{B \mapsto statusOp \mapsto 7\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\}, \{((s3\_B \mapsto (Receive \mapsto statusOp \mapsto A) \mapsto 9) \mapsto s2\_B)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s3\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s4\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\}, \{((s2\_B \mapsto (Send \mapsto terminateOp \mapsto A) \mapsto 5) \mapsto s4\_B)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s2\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s4\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\}, \{((s4\_A \mapsto (Receive \mapsto terminateOp \mapsto B) \mapsto 5) \mapsto s5\_A)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \{A \mapsto terminateOp \mapsto 8\} \mapsto \{(A \mapsto s5\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\}, \{((s4\_A \mapsto (Receive \mapsto terminateOp \mapsto B) \mapsto 5) \mapsto s5\_A)\} \mapsto \{(A \mapsto s4\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s5\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\}, \{((s5\_A \mapsto (Send \mapsto terminateLOp \mapsto L) \mapsto 6) \mapsto s7\_A)\} \mapsto \{(A \mapsto s5\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s7\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\}, \{((s2\_L \mapsto (Receive \mapsto terminateLOp \mapsto A) \mapsto 6) \mapsto s4\_L)\} \mapsto \{(A \mapsto s7\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \{L \mapsto terminateLOp \mapsto 9\} \mapsto \{(A \mapsto s7\_A), (B \mapsto s4\_B), (L \mapsto s4\_L)\}, \{((s2\_L \mapsto (Receive \mapsto terminateLOp \mapsto A) \mapsto 6) \mapsto s4\_L)\} \mapsto \{(A \mapsto s7\_A), (B \mapsto s4\_B), (L \mapsto s2\_L)\} \mapsto \varnothing \mapsto \{(A \mapsto s7\_A), (B \mapsto s4\_B), (L \mapsto s4\_L)\}\}$

axm13: $Initial\_state\_value = s0$

axm14: $CP\_Final\_states\_value = \{s0\}$

axm15: $Prophecy\_value = 10$

Prophecy_value = 10

axm16: $End\_message = End$

axm17: $S\_GS\_value = \{A \mapsto s0\_A, B \mapsto s0\_B, L \mapsto s0\_L\}$

axm18: $A\_GS\_value = \{A \mapsto s0\_A, B \mapsto s0\_B, L \mapsto s0\_L\}$

axm19: $Last\_cp\_trans\_value = s0 \mapsto B \mapsto OrderOp \mapsto A \mapsto s1 \mapsto 1$

END