Université Abdelhamid Ibn Badis Mostaganem

Thèse de doctorat en informatique

Spécialité : Apprentissage Automatique et Web Intelligent

Intitulée:

Vers une évolution cohérente des systémes dynamiques á partir des méthodes formelles

Présentée par :

Sarah Benyagoub

le dimanche 02 février 2020

Devant le jury:

Président:

Amir Adbessamad, Professeur; Université de Mostaganem Abdelhamid Ibn Badis

Examinateurs:

Sehaba Karim, Maître de Conférences A; Université de Mostaganem Abdelhamid Ibn Badis

Dominique Mery, Professeur; Université de Nancy; LORIA; France

Aît ameur Yamine, Professeur; ENSEEIHT Institut National Polytechnique de Toulouse; France

Directeur de thése:

Medeghri Ahmed, Professeur; Université de Mostaganem Abdelhamid Ibn Badis

Co-directeur de thése:

Ouederni Meriem, Maître de Conférences; Institut National Polytechnique de Toulouse; France

Invités:

André-Luc Beylot, Professeur; ENSEEIHT Institut National Polytechnique de Toulouse; France

Ladjel Bellatreche, Professeur; Université LIAS/ISAE-ENSMA Poitiers; France

Henni Fouad, Maître de Conférences B; Université de Mostaganem Abdelhamid Ibn Badis

Atif Mashkoor, chargé de recherche; SCCH, Linz-Autriche

Résumé

Les systèmes contemporains complexes basés sur les interactions sont souvent construits en réutilisant des entités de communication distribuées existantes 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) communiquantes 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 échangés entre les entités en communication. En procédant ainsi, la construction des systèmes ne nécessite pas de construction des entités individuelles ni de leur composition car ils peuvent être obtenus par la projection de la chorégraphie. La rectitude de la conservation de tels messages échangés par chaque entité, obtenus par projection, est une problématique majeure, connue 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, etc. 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. 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.*

ملخص:

غالبًا ما يتم بناء النظم المعقدة الموزعة القائمة على التواصل من خلال إعادة إستخدام كيانات التواصل الموزعة الموجودة فعليا و التي يتعين عليها التنسيق فيما بينها لتلبية متطلبات العملاء ،النظام و بيئته. في هذه الأطروحة، نتناول إشكالية تصميم النظم الموزعة التي تتألف من كيانات التواصل (نظم انتقال الحالة) اللاتي تتواصل من خلال تبادل الرسائل فيما بينها. تعتبر الكوريغرافيا النموذج الرسمي الذي يسمح للمطور بوصف وتعيين التنسيق بين كيانات التواصل، كمجموعة من المحادثات. لا يتطلب هذا الأخير بناء كيانات جديدة ولا تركيبها، إذ يمكن الحصول عليها عن طريق الإسقاط. الحفاظ على الرسائل التي يتبادلها كل كيان ، والتي يتم الحصول عليها عن طريق الإسقاط ، تمثل صعوبة في حد ذاتها. هذه الأخيرة و المعروفة باسم مشكلة "**قابلية التحقيق**" و التي تعني وجوب التحقق من مدى إمكانية تصميم الكوريغرافيا لإنشاء التطبيقات و النظم دون أخطاء التنسيق ، على سبيل المثال ، غياب نقاط التوقف التام ، فقدان الرسائل أو الترتيب الخاطئ للرسائل الى آخره.

لهذا الغرض، اقترحنا مجموعه من طرق التركيب التي تسمح لمصممي النظم بتركيب كوريغرافيا قابله للتحقيق، موصوفة على شكل بروتوكولات محادثه. نعتمد في أعمالنا البحثية على الإثبات عن طريق التركيب لضمان **قابلية التحقيق**: فنحن نعتمد على طريقة التناسق باستعمال التركيب لإثبات أن كل عملية بناء باستخدام طرق التركيب يمكن تحقيقها. وينطبق نهجنا ويوسع نطاقه على مجموعة من حالات الاستخدام المستعارة من المؤلفات والمستخدمة من قبل الباحثين. كما يسمح النهج الذي نتبناه أيضا باكتشاف الأعطال واسترداد **قابلية التحقيق** في حالة عدم القدرة على الإحتفاظ بها.

Remerciement

Je tiens à remercier toutes les personnes qui ont rendu cette thèse possible par leurs aides et leurs contributions. J'adresse mes sincères remerciements à monsieur Medeghri Ahmed, professeur à l'université de Mostaganem Abdelhamid Ibn Badis, monsieur Yamine AIT-AMEUR, Professeur à ENSEEIHT Institut National Polytechnique de Toulouse et à madame Meriem OUEDERNI, maître de conférence à ENSEEIHT Institut National polytechnique de Toulouse pour leur encadrement et leur soutien tout au long des années de thèse.

Je remercie également tous les membres du jury :
— Monsieur Amir Adbessamad, Professeur ; Université de Mostaganem Abdelhamid Ibn Badis
— Sehaba Karim, Maitre de Conférences A ; Université de Mostaganem Abdelhamid Ibn Badis
— Monsieur Dominique Mery, Professeur ; Université de Nancy ; LORIA ; France
— Monsieur André-Luc Beylot, Professeur ; ENSEEIHT Institut National Polytechnique de Toulouse ; France
— Monsieur Ladjel Bellatreche, Professeur ; Université LIAS/ISAE-ENSMA Poitiers ; France
— Monsieur Henni Fouad, Maitre de Conférences B ; Université de Mostaganem Abdelhamid Ibn Badis
— Monsieur Atif Mashkoor, chargé de recherche ; SCCH, Linz-Autriche

Je tiens à remercier Neeraj Kumar Singh pour son soutien et ses conseils.

Je voudrais remercier Xavier Crégut, Katia Jaffrès-Runser, Neeraj Kumar Singh et Yamine AIT-AMEUR de m'avoir donné l'occasion d'enseigner.

Je tiens à remercier Sylvie Armengaud, Annabelle Sansus et Muriel De Guibert qui étaient toujours agréables, efficaces et aidantes dans toutes les démarches administratives.

Je remercie toutes les personnes travaillant à ENSEEIHT, les membres de l'équipe ACADIE.

Enfin, je garde une place toute particulière à mes parents qui m'ont toujours soutenu, encouragé et cru en moi.

Table des matières

Résumé ii
Abstract iii

Liste de Figures **xi**

Liste de Tables **xii**

1 Introduction **1**
1 Contexte de l'étude 1
2 Positionnement 2
3 Contributions 3
4 Organisation du manuscrit 5
5 Diffusion scientifique 7

I Contexte **9**

2 Modélisation de système à base d'état implicite : la méthode B-événementiel **11**
1 Modèles de systèmes 11
2 Modèles B-événementiel 12
3 Règles d'obligation de preuve 15
4 Sémantique 16
5 Raffinement 16
6 Outils 17
6.1 Animation et modèle checker 17
ProB 17
6.2 Prouveurs automatiques 18

3 Réalisabilité de systèmes de communication **19**
1 Protocoles de communication (CP) 20
2 Composition de système 22
2.1 Communication synchrone 22
2.2 Communication asynchrone 22
3 Réalisabilité de CP 23

3.1 Propriété d'équivalence . . . 23
3.2 Propriété de synchronisabilité . . . 24
3.3 Propriété de bonne définition (WF) . . . 24
3.4 Propriété de réalisabilité . . . 24
4 Réalisabilité : travaux connexes . . . 25
4.1 Approches correctes-par-construction . . . 26
4.2 Approches de vérification de modèles . . . 27
4.2.1 Vérification sur modèles de CP . . . 27
4.2.2 Vérification sur modèles spécifiques . . . 28
4.3 Compatibilité des chorégraphies . . . 30
4.4 Exploration partielle du graphe d'accessibilité . . . 32
5 Étude de l'évolution et de la reconfiguration de CP . . . 32
5.1 Cas de chorégraphies . . . 33
5.2 Cas de processus d'affaires . . . 34
5.3 Cas de Service Web . . . 35
6 Réparation de chorégraphies . . . 36
6.1 Renforcement de la réalisabilité . . . 37
6.2 Contrôle d'équivalence . . . 38
7 Discussion . . . 38

II Contributions 41

4 Construction incrémentale : algèbre de composition 43
1 Opérateurs de composition . . . 43
2 Réalisabilité-par-construction : définition des con-ditions suffisantes . 46
3 Réalisabilité-par-construction : théorèmes et preu-ves . . . 50
3.1 Schéma de preuve. . . . 50
3.2 Schéma de preuve. . . . 52
3.3 Théorèmes de réalisabilité pour la séquence, le choix et l'itération. . . . 52
3.4 Cas d'étude . . . 60
Exigences . . . 60
Un protocole de conversation possible . . . 61
Entités obtenues après projection . . . 62
Conception incrémentale du *CP* réalisables : un modèle formel 62

5 Formalisation : preuve formelle avec B-événementiel 69
1 Stratégie de raffinement avec B-événementiel . . . 70
2 Cadre mathématique pour la composition . . . 71
2.1 Variables et états de systèmes de composition . . . 71
2.2 Systèmes . . . 72
2.3 Initialisation et composition . . . 73
2.4 Conditions suffisantes . . . 73
2.5 Propriété de réalisabilité . . . 73
3 Cadre général de formalisation et preuve . . . 74

4 Un modèle B-événementiel pour la composition du système 75
4.1 Partie statique : définitions requises 76
4.2 Partie dynamique : modélisation du comportement des CP . . 78
4.3 Une approche évolutive de construction des CP réalisables . . 84
5 Instanciation du modèle B-événementiel par raffinement 84
5.1 Étape 1. Instanciation du contexte 84
5.2 Étape 2. Raffinement et témoins pour l'instanciation 85
6 Application à un cas d'étude 85
7 Animation avec ProB 88
8 Évaluation 89
8.1 Les preuves 89
8.2 Méthodes formelles correctes-par-construction 91

6 Réparation correcte-par-construction 93
1 Opérateurs de réparation : théorèmes et preuves 93
2 Étude de cas 96
3 Scénarios de réparation 98
3.1 Restauration de ISeqF 98
3.2 Restauration de PCF 99
4 Formalisation avec B-événementiel 101
4.1 Partie statique : définitions requises 101
4.2 Partie dynamique : modélisation de la réparation 102
5 Restauration de la réalisabilité et instanciation 103
6 Évaluation 105

7 Évaluation et mise en oeuvre 107
1 Cas d'étude de composition réalisable 107
1.1 CS1. Processus d'entreprise virtuelle 108
1.2 CS2. Protocole de transfert de fichiers 108
1.3 CS3. Accès à une application web 108
1.4 CS4. Accès à une base de données via une application web . . 109
1.5 CS5. Application d'achat en ligne 110
1.6 CS6. Protocole de visa de travail australien 111
1.7 CS7. Protocole d'un jeu fictif 112
2 Cas d'étude de réparation de CP non réalisables 113
2.1 CS8. Réparation de CS4 113
2.2 CS9. Réparation de CS5 114
2.3 CS10. Réparation de CS7 115
2.4 CS11. Réparation de CS2 116
3 Travaux connexes 116

III Conclusion 119

8 Conclusion et perspectives 121

IV Annexes **131**

A Modèle B-événementiel de composition **133**

B Modèle B-événementiel de réparation **160**

C Exemples d'instanciation **168**

Table des figures

1.1 Aperçu des contributions. 4

4.1 Transition ajoutée (hachurée) forme un cycle 45
4.2 Transition ajoutée (hachurée) forme un graphe dirigé 45
4.3 Choix non déterministe *CP* 46
4.4 Préservation de l'ordre des messages d'envoi-réception 47
4.5 Conservation de l'ordre d'envoi de messages 48
4.6 *CP* non réalisable violation de la condition ISeqF 48
4.7 *CP* réalisable satisfaisant PCF 49
4.8 *CP* non-réalisable PCF violé 49
4.9 Exemple illustratif d'un protocole de conversation avec séquence et choix satisfaisant *ISeqF* et *PCF*. 50
4.10 Protocole de conversation d'un processus d'approvisionnement simple au sein d'une entreprise virtuelle 61
4.11 Entités projetées d'un processus simple d'achat en ligne au sein d'une entreprise virtuelle 62

5.1 Modèle en B-événementiel. 75
5.2 Trace de modèle abstrait de conversation (CP). 88
5.3 Trace de modèle synchrone de conversation. 88
5.4 Trace de modèle asynchrone de conversation. 89
5.5 Les preuves associées aux modèle de composition B-événementiel. . . 92

6.1 Chorégraphie de protocole de transfert de fichiers 97
6.2 Entités projetées d'une chorégraphie de protocole de transfert de fichiers 97
6.3 Proposition 1 de la réparation de ISeqF. 98
6.4 Proposition 2 de la réparation de ISeqF. 99
6.5 Proposition 1 de la réparation du PCF. 100
6.6 Proposition 2 de la réparation du PCF. 100
6.7 *CP* réparé du protocole de transfert de fichiers 104
6.8 Entités projetées du *CP* réparé du protocole de transfert de fichiers . 104
6.9 Les preuves associées aux modèle de réparation B-événementiel 106

7.1 Protocole de conversation d'un processus d'approvisionnement simple au sein d'une entreprise virtuelle. 108
7.2 Chorégraphie de protocole de transfert de fichier simple. 108
7.3 Accès à une application web. 109
7.4 Accès à une base de données via une application Web. 110
7.5 Chorégraphie non réalisable d'achat en ligne. 110
7.6 Protocole de visa de travail australien. 111
7.7 Protocole d'un jeu fictif. 112
7.8 Accès à une application web, *CP* réparé. 114
7.9 Chorégraphie d'un achat en ligne. 114
7.10 Accès à une application web, *CP* réparé. 115
7.11 *CP* réparé du protocole de transfert de fichier. 116

Liste des tableaux

2.1 Structures de contextes et de machines B-événementiel 13
2.2 Exemples d'obligations de preuve pour un modèle B-événementiel . . 15

3.1 Les quatre classes d'approches existantes 39

4.1 Théorèmes liés au *CP* réalisables-par-construction 51

5.1 Trace synchrone de l'exemple. 89
5.2 Trace asynchrone de l'exemple. 90
5.3 Statistiques de preuves RODIN (modèle de composition). 91

6.1 Théorèmes liés aux *CP* réparés-par-construction 96
6.2 Statistiques des preuves RODIN (modèle de réparation) 105

7.1 Rapport technique des travaux connexes. 118

Chapitre 1

Introduction

Sommaire

1 **Contexte de l'étude** 1
2 **Positionnement** 2
3 **Contributions** 3
4 **Organisation du manuscrit** 5
5 **Diffusion scientifique** 7

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 consensuelle. Les composants n'ont pas d'accès direct à une connaissance partagée et doivent échanger des informations l'un entre l'autre pour atteindre leur but. Par exemple, les situations de la vie quotidienne nous rappellent souvent la difficulté de cette tâche. Des 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 de compatibilité d'une collection de composants est en effet loin d'être triviale. Avec l'avènement de l'informatique en nuage (cloud computing), de l'internet des objets (Internet of Things) et l'interconnexion des objets informatiques allant des bases de données volumineuses centralisées aux petits appareils de cuisine, les systèmes distribués acquièrent une influence et un effet encore plus fort sur nos vies. En conséquence, assurer leur fonctionnement approprié est crucial. Un simple défaut peut ainsi entraîner des complications physiques, environnementales, économiques, ou même mortelles. 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 des réponses à ces exigences de sécurité et de performance en instrumentant des structures et des preuves mathématiques pour décrire, modéliser et raisonner la conception de ces systèmes de manière à minimiser les risques de bugs et les coûts de dépannage. Elles proposent des garantis que les systèmes sont effectivement conformes à un ensemble de spécifications et de propriétés de comportement correctes. À cet égard, ce travail contribue à la formalisation d'un domaine d'interaction spécifique dans les systèmes distribués, appelé systèmes réalisables.

2 Positionnement

Dans une conception descendante des systèmes distribués, les interactions entre entités communicantes sont généralement définies à l'aide d'une spécification globale appelée chorégraphie ou protocole 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, quelques questions difficiles à aborder peuvent se poser. À savoir, existe-t-il un ensemble d'entités distribuées dont le comportement composé est identique à celui spécifié par le protocole de conversation ? Ceci est défini comme propriété de « Réalisabilité ».

Étant donné que les spécifications d'un système peuvent évoluer avec le temps, comment pouvons-nous assurer la cohérence de cette évolution ? Comment cela peut être vérifié et prouvé formellement ? De plus, à partir d'un protocole de conversation réalisable mis à jour pour certaines raisons, comment la réalisabilité peut être vérifiée (partiellement), après cette mise à jour du système ? Comment l'évolution d'entités communicantes peut être reflétée dans la spécification globale, c.à.d sous le protocole de conversation ?

Une problématique principale, déjà abordée par la communauté de la recherche, est : la vérification de la réalisabilité des CP. Elle se traduit par la vérification de l'existence d'un ensemble d'entités communicantes dont la composition génère les mêmes séquences de messages que celles spécifiées par le CP. Considérant un système asynchrone, le problème de réalisabilité est indécidable [25] avec un tel mode de communication, en raison du mécanisme de file d'attente illimitée (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 communicantes de manière asynchrone en utilisant des files d'attentes de taille illimitée. Ce travail résout le problème de la réalisabilité pour une sous-classe de systèmes asynchrones, à savoir, les systèmes synchronisables, c'est-à-dire un système composé d'entités communicantes se comportent de la même façon en passant d'une communication synchrone à une communication asynchrone. Ainsi, 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 unes 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) projection des entités communicantes à partir du *CP*, ii) vérification de la synchronisation i.e. l'équivalence entre les compositions synchrone et asynchrone des entités projetées, iii) vérification de l'équivalence entre le *CP* et les compositions synchrone et asynchrone des entités projetées, iv) et l'ensemble des files d'attente utilisées par les entités communicantes sont vide à la fin de la communication. Ce travail propose une vérification des systèmes asynchrones de taille raisonnable de point du vue du nombre : des états, des transitions et des entités communicantes.

3 Contributions

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

Trois contributions constituent les résultats de nos traveaux :

— **Langage de composition correcte-par-construction.** Considérant la communication asynchrone, plusieurs propriétés de systèmes sont indécidables dans le cadre général [26] en raison du mécanisme de file d'attente non bornée. La propriété principale traitée dans nos travaux est la réalisabilité des systèmes distribués asynchrones. Nous rappelons qu'intuitivement, un *CP* est réalisable s'il existe un ensemble d'entités communicantes mettant en œuvre ce *CP*. L'approche donnée dans [10] proposait une condition nécessaire et suffisante pour vérifier si un *CP* pouvait être implémenté par un ensemble d'entités communicantes en asynchrone avec des files d'attente FIFO non bornées. Nous proposons une résolution de problème de réalisabilité pour une sous-classe d'entités communicantes en asynchrone, à savoir les systèmes synchronisables. La synchronisabilité signifie que les compositions synchrones et asynchrones des entités de communication sont équivalentes. Ceci est une définition spécifique de la propriété d'élasticité (slack elasticity) qui a été prouvée comme décidable pour la communication en anneau orienté dans [41]. Dans [10], la réalisabilité du *CP* est vérifiée comme suit : i) calcul les LTSs des entités communicantes par projection à partir du *CP* ; ii) vérification de la synchronisabilité entre les systèmes synchrone et asynchrone ; et iii) vérification de l'équivalence entre les *CP* et la composition synchrone de leurs entités projetées. Les techniques de vérification de modèle s'appliquent à des systèmes distribués de taille raisonnable, c'est-à-dire nombre d'états, de transitions et d'entités communicantes. Notant qu'il s'agit d'un processus de vérification a posteriori. Par conséquent, si un *CP* est identifié comme non réalisable à l'aide de la vérification du modèle, il peut alors être appliqué en introduisant soit des messages de synchronisation supplémentaires, soit en

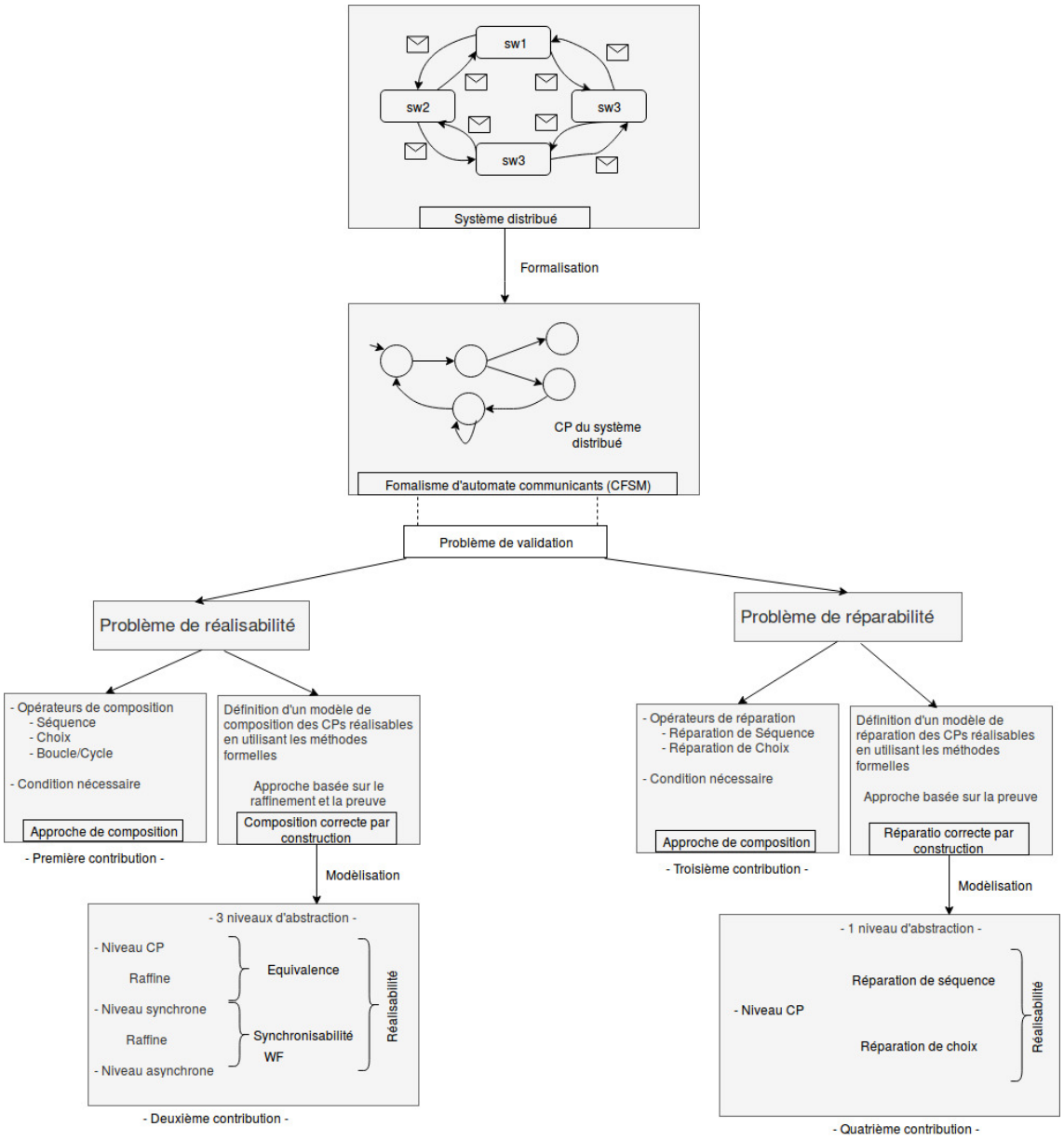


FIGURE *1.1 – Aperçu des contributions.*

calculant des moniteurs garantissant la réalisabilité [46].
En se basant sur la définition donnée dans [10], nous sommes les premiers à proposer une méthode a priori d'application de la réalisabilité. Nous spécifions formellement un *CP* utilisant le système de transitions étiquetées (LTS), modèle comportemental constitué d'un ensemble d'états liés à des transitions décrivant les interactions de système. Les entités communicantes sont également spécifiées avec des LTS où les transitions représentent les actions d'envoi ou de réception à déclencher et passent d'un état à un autre. Ils sont obtenus par projection à partir du *CP* et communiquent de manière asynchrone via des échanges de messages utilisant des files d'attente FIFO. Notre contribution garantit des *CP* réalisables par construction. Nous définissons

une algèbre d'opérateurs de composition pour la construction d'un *CP* réalisable. À partir d'un *CP* initial, nous composons progressivement un *CP* intermédiaire avec de nouveaux *CP* de base jusqu'à l'obtention du *CP* complet requis. Nous exigeons que chaque composition puisse être réalisée si et seulement si certaines propriétés associées sont satisfaites. Notre contribution réduit considérablement la complexité de la vérification car il n'est pas nécessaire de projeter et recomposer les entités communicantes distribuées. En conséquence, nous fournissons aux utilisateurs un nouveau langage de conception qui nous permet d'économiser des ressources (temps et mémoire) ainsi que les efforts nécessaires pour contrôler des systèmes complexes du monde réel.

— **Modélisation et preuve.** Nous présentons notre approche sous forme d'un développement à plusieurs niveaux de modélisation. Une preuve inductive, basée sur la préservation de l'invariant de réalisabilité, est mise en place avec Event-B [2] sur la plate-forme Rodin [1]. La stratégie de raffinement est utilisée pour décomposer cet invariant afin de faciliter les processus de preuve et de développement. Nous identifions, un premier niveau de raffinement, exprimant un modèle d'interaction synchrone. Ensuite, un deuxième niveau, obtenu également par raffinement produit un modèle asynchrone du protocole de conversation.
Le modèle générique que nous définissons est évolutif et ses paramètres ont des valeurs arbitraires (le nombre d'entités, la taille de la file d'attente, le nombre d'états et de transitions peuvent prendre n'importe quelle valeur dans leurs ensembles). De plus, ce modèle peut être instancié pour décrire tout *CP* par application incrémentale des opérateurs de composition que nous avons définis.

— **Validation, expériences et benchmarks.** Nous prouvons la validation de notre proposition en utilisant un point de repère de plusieurs études de cas réalisables et non réalisables empruntées de la littérature. 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.

— **Réparation.** Nous étudions la possibilité de réparation de chorégraphies non réalisables. L'objectif est d'identifier un ensemble de modifications à apporter à une chorégraphie non réalisable afin de restaurer la propriété de réalisabilité, tout en se basant sur l'ensemble des opérateurs définis pour la composition correcte-par-construction de *CP* réalisables. Nous présentons une technique de réparation automatique des chorégraphies non réalisables et fournissons des garanties formelles de correction.

4 Organisation du manuscrit

Le contenu de ce manuscrit de thèse est organisé comme suit.

1. La première partie est composée de deux chapitres.

   — Le premier chapitre présente la méthode B-événementiel, les obligations de preuve associées à la platforme Rodin et l'animateur ProB.
   — Le second chapitre résume les techniques mises en oeuvre dans nos travaux. Tel que,

      — tout d'abord, nous présentons la notion de protocole de conversation et de projection (le comportement de chacune des entités communicantes) ainsi que les deux modes de communication de systèmes : synchrone et asynchrone.

      — puis, une étude de l'état de l'art des travaux qui traitent des problématiques similaires, est présentée sous forme de trois classes.

         — La première est consacrée aux approches de vérification et de validation de la réalisabilité.
         — La deuxième classe concerne les approches traitant l'évolution et la reconfiguration des systèmes.
         — La dernière classe est dédié au problème de réparabilité des systèmes.
         — Nous concluons ce chapitre par la comparaison et la définition des limites de chaque catégorie d'approches présentées.

2. La deuxième partie présente nos contributions.

   — Le premier chapitre décrit notre approche correcte-par-construction avec la définition de l'ensemble des opérateurs de composition de *CP* réalisables et leurs propriétés. La correction de satisfaction des définitions est garantie par des théorèmes prouvés.

   — Nous détaillons ensuite notre deuxième contribution : la validation de l'approche de composition correcte-par-construction. Les démarches de modélisation adoptées ainsi que les différents raffinages sont détaillés. Puis nous présentons une validation de notre proposition sur un cas d'étude réalisable emprunté de la littérature.

   — Notre troisième contribution est une stratégie de réparation incrémentale. Un principe de réparation basé sur notre langage de composition ainsi que sa preuve formelle sont présentés et formalisés avec B-événementiel. Un cas d'étude identifié comme non-réalisable, emprunté de la littérature est mis en oeuvre.

   — En fin, nous présentons un ensemble de cas d'étude en guise de test et de validation.

3. La dernière partie est une conclusion générale qui présente une synthèse des travaux réalisés et identifie un ensemble de perspectives à nos travaux.

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. DOI :10.1002/smr.2209.* (Chapitre 8)

— Sarah Benyagoub, Meriem Ouederni and Yamine Aït Ameur. Incremental correct-by-construction of distributed Systems. *Journal of Computer Languages, 2020.* (Chapitre 5)

Conférence nationale avec comité de lecture

— Sarah Benyagoub : Construction incrémentale de chorégraphies réalisables, *AFADL 2018 : 62-63*(Chapitre 3)

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* (Chapitre 8)

— Sarah Benyagoub, Yamine Aït Ameur, Meriem Ouederni, Atif Mashkoor : Handling Reparation in Incremental Construction of Realizable Conversation Protocols. *MEDI Workshops 2018 : 159-166* (Chapitre 7)

— Sarah Benyagoub, Meriem Ouederni, Yamine Aït Ameur, Atif Mashkoor : Incremental Construction of Realizable Choreographies. *NFM 2018 : 1-19* (Chapitre 6)

— Sarah Benyagoub, Meriem Ouederni, Neeraj Kumar Singh, Yamine Aït Ameur : Correct-by-Construction Evolution of Realisable Conversation Protocols. *MEDI 2016 : 260-273* (Cha-pitre 5)

— Sarah Benyagoub, Meriem Ouederni, Yamine Aït Ameur : Towards correct Evolution of Conversation Protocols. *VECoS 2016 : 193-201*(Chapitre 5)

Projets et financements

— *Bourse ministérielle :* 8 mois de stage dans le cadre du Programme National Exceptionnel (P.N.E) 2016-2017

— *Projet IntegR - partenariat-Ministère autrichien des transports, COMET SCCH :* Composition et réparation formelle de systèmes critiques distribué (du 04-18 à 08-18)

— *Contrat ATER 2018-2019 :* Attaché temporaire d'enseignement et de recherche 192h de charge
— *Contrat ATER 2019-2020 :* Attaché temporaire d'enseignement et de recherche 192h de charge

Première partie

Contexte

Chapitre 2

Modélisation de système à base d'état implicite : la méthode B-événementiel

Sommaire

1 Modèles de systèmes 11
2 Modèles B-événementiel 12
3 Règles d'obligation de preuve 15
4 Sémantique 16
5 Raffinement 16
6 Outils 17
6.1 Animation et modèle checker 17
6.2 Prouveurs automatiques 18

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 modélisés à l'aide de changements d'état du système. Aussi, nous avons choisi de nous appuyer sur des systèmes de transition d'états comme modèle de base. Nous utilisons la méthode B-événementiel et la plate-forme Rodin pour la modélisation de ces systèmes et des preuves de satisfaction des exigences structurées à l'aide des raffinements. Nous présentons dans ce qui suit une brève définition de ces techniques formelles.

1 Modèles de systèmes

Les systèmes de transition ont été identifiés comme modèle générique approprié pour les systèmes distribués. 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 systèmes de transitions où st_i raffine st_{i-1}. Le raffinement introduit de plus en plus de détails passant d'un système abstrait à un système qui l'est concrétisé. De plus, nos travaux visent la définition de systèmes

corrects par constructions. Par conséquent, il est nécessaire de prouver la correction des modèles conçus en allant au-delà des tests ou de la vérification sur 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 des algèbres de processus CCS [68] ou LOTOS [36, 65]. Ces techniques sont descriptives et n'offrent pas d'opérations de raffinement. Nous ne les avons donc pas considérées dans notre travail.

La deuxième classe de méthodes formelles sont les méthodes basées sur l'expression d'état explicite. 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 (entre états) munies en général de pré-conditions et de post-conditions [51]. De plus, ces modèles formels peuvent être associés à une relation de raffinement permettant de définir une séquence de modèles liés par cette relation. Parmi ces méthodes, on peut citer Z [86, 89], VDM [23], B [3], TLA$^+$ [57], B-événementiel [6] et Statecharts [49]. 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 associés à ces techniques sont NuSMV [27], CADP [44], PROMELA/SPIN [52], ProB [61] 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 utilisées à un niveau «méta» : elles 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 [69], PVS [70] 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 formaliser nos modèles et prouver les propriétés associées.

La méthode B-événementiel [6] introduite à base d'état explicite est une évolution récente de la méthode B [3]. Cette méthode est basée sur les notions de pré-conditions et post-conditions de Hoare [51], la pré-condition la plus faible de Dijkstra [35] et le calcul de substitution généralisé [3]. C'est une méthode formelle basée sur des fondements mathématiques : logique du premier ordre et théorie des ensembles.

2 Modèles B-événementiel

Un modèle B-événementiel est composée de : machines et de contextes. Les machines représentent les parties dynamiques (états et transitions) d'un modèle, tandis que les contextes décrivent 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, un modèle peut *voir* un ou plusieurs contextes.

Un modèle B-événementiel est caractérisé par un ensemble de variables (représentant l'état d'un système), définies dans la clause $Variables$ et évoluant grâce aux événements déterminé dans la clause $Events$. Il définit un système de transitions d'état où les variables représentent l'état et les événements représentent les transitions entre états. Les événements sont un ensemble de traces caractérisant les comportements entrelacés du système (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 :* nom unique du composant dans un modèle.
— *Extends :* déclare le(s) contexte(s) étendu(s) par ce contexte.
— *Sets :* décrit des ensembles abstraits ou énumérés utilisés par les modèles.
— *Constants :* définit les constantes utilisées par un modèle.
— *Axioms :* décrit, à l'aide d'expressions logiques du premier ordre, les propriétés (définitions) des éléments déclarés dans les clauses *Constants* et *Sets*. Les types et les contraintes sont également décrits dans cette clause.
— *Theorems :* caractérise les propriétés 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 *2.1 – Structures de contextes et de machines B-événementiel*

De manière similaire aux contextes, les machines sont définies par un ensemble de clauses (Table 2.1).

— *Machine :* nom du composant unique.
— *Refines* déclare la machine raffinée par la machine décrite.
— *Sees* fait référence au contextes importés par cette machine.
— *Variables* définit l'ensemble des variables d'état du modèle. Les raffinements peuvent introduire de nouvelles variables afin d'enrichir le modèle décrit.

— *Invariants* décrit, à l'aide d'expressions logiques du 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 être préservées à tout moment. Cela signifie que les invariants doivent être garantis à l'initialisation et que les événements (plus précisément les actions) doivent les préserver. Cela garantit le maintien des invariants (induction). Cette classe permet d'exprimer également l'invariant de collage requis sous les raffinements.
— *Theorems* définissent un ensemble d'expressions logiques pouvant être déduites des invariants et/ou du (des) contexte(s). Il n'est pas nécessaire de les prouver 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 et d'atteignabilité.
— *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 2.1) :

— *Refines* déclare la liste des événements raffinés par l'événement décrit.
— *Any* liste les paramètres de l'événement.
— *Where* exprime la garde de l'événement. Un événement peut être déclenché lorsque sa garde (guard) est évaluée à vrai. Si plusieurs conditions sont évaluées à vrai, un évènement est déclenché avec un choix non-déterministe.
— *Then* Contient les actions de l'événement qui modifient 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 le variant,
— *convergent* : le variant doit décroître,
— *anticipated* : le variant ne doit pas croître. Il sera précisé lors du raffinement.

B-événementiel propose trois types d'actions (substitutions) :

— *affectation* (x := E) où l'expression E est affectée à la variable x. Cette action est déterministe.
Exemple : x := 4
— *choix* ($x :\in S$) où un élément d'un ensemble s, est affecté à une variable x de manière non déterministe.
Exemple : $x :\in N \setminus \{2\}$

où la variable x devient un nombre naturel quelconque autre que 2.

— Le prédicat *avant-après* (x :| BA(x, x')) est la forme d'action la plus générale. La variable après devient telle que le prédicat avant-après BA(x,x') est satisfait. C'est la substitution qui permet d'exprimer toutes les autres. Exemple : x,y :| x'>x ∧, x'+y' = 5
Cette substitution affirme que x et y deviennent telles que x' est supérieur à x et que la somme de x et y après est égale à 5.

3 Règles d'obligation de preuve

Des obligations de preuve (PO) sont associées à tout modèle B-événementiel afin d'exprimer des développements et des raffinements. Elles doivent être prouvées pour garantir la correction du modèle.

Les règles permettent de générer des obligations de preuve suivent le calcul des substitutions généralisées [3, 6], proche du calcul de plus faible pré-condition de Dijkstra [35]. Afin de définir les règles d'obligation de preuve, nous utilisons les notations définies dans la Table 2.2, où s indique les ensembles, c les constantes et v les variables de la machine. Les axiomes sont désignés par $A(s, c)$ et les théorèmes par $T_c(s, c)$, tandis que les invariants sont désignés par $I(s, c, v)$, les théorèmes 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 déclenchement de l'un des événements.

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

La Table 2.2 présente les principales obligations de preuve associées à un modèle B-événementiel.

— Les *obligations de preuve de théorème* (a) et (b) garantissent qu'un théorème de contexte proposé (a) ou un théorème de machine (b) est bien déduit des axiomes et de l'invariant.
— L*'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 *i.e.* peut être déclenché.

— L*'obligation de preuve du variant* garantit que chaque événement converge, un variant numérique proposé est un nombre naturel
— L*'obligation de preuve de variant* (f) indique que chaque événement convergent fait décroître 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 [6].

4 Sémantique

L'apport de B-événementiel [6], par rapport à la méthode B classique [3], est lié à la sémantique. En effet, les événements d'un modèle sont des événements atomiques d'un système de transitions. 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 [5] définie dans B-événementiel permet le développement de modèles de manière incrémentale. Un système de transitions 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 tout en restant cohérent avec le système abstrait. Tous les nouveaux événements raffinent l'événement de skip (c'est-à-dire l'événement qui ne fait rien et qui peut se produire à tout moment). Sous réserve de définir l'invariant de collage, le raffinement préserve les propriétés prouvées et il n'est donc pas nécessaire de les prouver à nouveau dans le système de transition raffiné. Cela aide à garder des preuves de tailles raisonnables en répartissant l'effort de preuve tout au long du raffinement.

Afin de prouver la correction d'un développement, il est nécessaire de prouver les différents raffinements qu'il contient. Les obligations de preuve suivantes sont les deux obligations de preuve clés dans le raffinement.

— 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, A désigne la conjonction des axiomes, I les invariants, G les conditions,

W les témoins (prédicats liant des variables concrètes et abstraites) et 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 affecte 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. Notons 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 Outils

L'outil support des développements basés sur B-événementiel est la plate-forme Rodin [1] [4]. Il s'agit d'un environnement de développement intégré doté d'éditeurs de contextes et de machines, d'un générateur d'obligations de preuve, de démonstrateurs automatiques et de capacités de vérification interactives. De plus, une large gamme de plug-ins est disponible, ce qui peut étendre les capacités de modélisation (avec des théories, par exemple) ou de vérification (comme le vérificateur de modèle ProB [62] ou l'utilisation de solveurs SMT).

6.1 Animation et modèle checker

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.

ProB

ProB est un animateur, un résolveur de contraintes et un vérificateur de modèles pour la méthode B. Il permet une animation entièrement automatique des spécifications B et peut être utilisé pour vérifier systématiquement une spécification pour un

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

large éventail d'erreurs. Les capacités de résolution de contraintes de ProB peuvent également être utilisées pour la recherche de modèle, la vérification des blocages et la génération de cas de test. Le langage B est basé sur la logique des prédicats, l'arithmétique et la théorie des ensembles et fournit un support pour les structures de données telles que les relations, les fonctions et les séquences (d'ordre supérieur). En plus du langage B, ProB prend également en charge B-événementiel [6], CSP-M, TLA+ [57] et Z [86, 89]. ProB peut être installé dans Rodin. ProB est utilisé par Siemens, Alstom, Thales et plusieurs autres sociétés pour la validation de données de propriétés complexes pour des systèmes critiques pour la sécurité. ProB est certifié T2 SIL4 conformément à la norme Cenelec EN 50128 pour une utilisation chez Thales.

6.2 Prouveurs automatiques

Parmi les prouveurs automatiques, on peut différencier les solveurs SMT (Satisfiability Modulo Theories) comme $Alt-Ergo$ [37], $CVC3$ [28] et $Z3$ [31] qui utilisent des solveurs SAT (pour les problèmes de satisfaisabilité booléenne), des prouveurs du premier ordre comme Zenon [74] et Vampire [75], 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.

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 3

Réalisabilité de systèmes de communication

Sommaire

1 Protocoles de communication (CP) 20
2 Composition de système 22
2.1 Communication synchrone 22
2.2 Communication asynchrone 22
3 Réalisabilité de CP 23
3.1 Propriété d'équivalence 23
3.2 Propriété de synchronisabilité 24
3.3 Propriètè de bonne définition (WF) 24
3.4 Propriété de réalisabilité 24
4 Réalisabilité : travaux connexes 25
4.1 Approches correctes-par-construction 26
4.2 Approches de vérification de modèles 27
4.3 Compatibilité des chorégraphies 30
4.4 Exploration partielle du graphe d'accessibilité 32
5 Étude de l'évolution et de la reconfiguration de CP . . . 32
5.1 Cas de chorégraphies 33
5.2 Cas de processus d'affaires 34
5.3 Cas de Service Web 35
6 Réparation de chorégraphies 36
6.1 Renforcement de la réalisabilité 37
6.2 Contrôle d'équivalence 38
7 Discussion 38

La modélisation du comportement dynamique de systèmes doit offrir des outils et mécanismes de vérification des propriétés attendues de cette opération. Le modèle défini doit être validé pour garantir l'absence d'anomalies. Ces dernières peuvent être, en particulier, des erreurs de conversation ou de réalisabilité.

Dans ce chapitre, nous présentons le formalisme que nous utilisons pour décrire notre contribution de thèse pour la modélisation et la vérification de la composition de systèmes. Nous explorons aussi les travaux similaires et connexes. Nous mettons l'accent, d'une part, sur les différentes techniques de validation de conversation basées sur un modèle de transitions étiquetés et, d'autre part, sur les différentes approches qui traitent l'aspect réalisabilité de protocoles de conversation.

1 Protocoles de communication (CP)

Cette section présente la notion de protocole de communication qui définit les échanges de messages entre un ensemble d'entités communicantes ($\mathcal{P}$). Cette communication génère un ensemble de traces décrivant les échanges de messages entre les entités. Nous considérons des protocoles de conversation déterministes et introduisons la notion de protocole de conversation de base formé de deux états et d'une seule transition. Le comportement des entités d'un *CP*, en appliquant l'opération de *Projection* est également définie. La définition des deux modes de communication des entités communicantes (synchrone et asynchrone) est également présentée.

Les systèmes de transitions étiquetés (LTS) sont utilisés comme modèle de base pour les protocoles de conversation (*CP*) ainsi que pour les entités communicantes ($\mathcal{P}$) incluses dans cette spécification. Ce modèle de comportement définit l'ordre des messages échangés pour un *CP*. Au niveau des entités communicantes, le LTS peut être obtenu par projection à partir d'un *CP*. Il 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* $\mathcal{P} =< S, s^0, \Sigma, T >$ *où, S est un ensemble fini d'états,* $s^0 \in S$ *est l'état initial du système,* $\Sigma = \Sigma^! \cup \Sigma^? \cup \{\tau\}$ *est l'ensemble fini (alphabet) défini comme l'union de l'ensemble des messages envoyés* $\{m!, m'!, \ldots\}$*, l'ensemble des messages reçus* $\{m?, m'?, \ldots\}$ *et du singleton comprenant l'action interne* τ *(transition vide), et* $T \subseteq S \times \Sigma \times S$ *définit la relation de transition.*

Nous notons $t = s \xrightarrow{l} s'$, où $l \in \Sigma$ *et* $t \in T$, comme une transition d'envoi si $l = m!$ ($m! \in \Sigma^!$) et comme une transition de réception si $l = m?$ ($m? \in \Sigma^?$). Un état s^f *dans* S^f est final si l'entité correspondante peut arrêter de envoyer/recevoir des messages dans cet état.

Définition 2 (Protocole de conversation (CP)) *Un protocole de conversation CP pour un ensemble d'entités communicantes* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *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,* s^0_{CP} *est l'état initial global,* L_{CP} *est l'ensemble des étiquettes (une étiquette est dénotée par* $m^{\mathcal{P}_i,\mathcal{P}_j}$ *où* $\mathcal{P}_i$ *et* $\mathcal{P}_j$ *sont, respectivement, les entités émettrice et réceptrice du message m avec* $\mathcal{P}_i \neq \mathcal{P}_j$*) et* $T_{CP} \subseteq S_{CP} \times L_{CP} \times S_{CP}$ *est la relation de transition.*

De plus, chaque échange de message possède 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.

— $t = s \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s'$ *décrit 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 la transition,* P_i *et* P_j *sont les entités émettrice et réceptrice et*
— $S^f_{CP} \neq \varnothing$ *et* $S^f_{CP} \subseteq S_{CP}$ *dénotes l'ensemble non-vide d'états finaux, i.e. , un état dans lequel un CP peut cesser de fonctionner.*

Définition 3 (Traces) *Les traces sont définies aux niveaux des CP et des* $\mathcal{P}$*, 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* $m^{\mathcal{P}_i \to \mathcal{P}_j}$.
Tr_{CP} désigne l'ensemble de ces traces. Nous notons une trace $tr_c\{s_0 \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_1, \ldots, s_n \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s_{n+1}\} \subseteq Tr_{CP}$ *où* s_i *est le* i^{eme} *état dans la trace* tr_c*. Lorsque* s_{n+1} *est un état final, la trace* tr_c *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 réception (de la forme* $m?$*) correspondant aux libellés des transitions dans* $\mathcal{P}$*.*

Tr_p désigne l'ensemble de ces traces, notées $tr_p = \{s^k_0 \xrightarrow{l} s^k_1, \ldots, s^k_n \xrightarrow{l'} s^k_{n+1}\} \subseteq Tr_p$*.* s^i_n *est le* i^{eme} *état d'une trace de séquence dans* Tr_k *avec* $l \in \Sigma$*,* $l = m!$ *ou* $l = m?$*.*
Lorsque s_{n+1} *est un état final, la trace est dite complète.*

Définition 4 (Choix Déterministe (DC)) *Seuls les protocoles de conversation déterministes sont étudiés, i.e. des CP qui réagissent toujours de la même façon à un événement, quoi qu'il se soit passé auparavant. Formellement, nous définissons un CP déterministe comme suit.*

Soit DC l'ensemble des CP déterministes.
Pour tout protocole de conversation 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 de base) *Un CP de base, noté* CP_b*, est un CP avec une seule transition* $CP_b =< S_{CP_b}, s^0_{CP_b}, L_{CP_b}, T_{CP_b} >$ *où* $T_{CP_b} = \{s^0_{CP_b} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s^f_{CP_b}\}$ *tel que* $s^0_{CP_b} \neq s^f_{CP_b}$*.*

Définition 6 (Projection) *La projection* $\downarrow CP = \{P_1, \ldots, P_n\}$ *d'un protocole de conversation CP produit un ensemble d'entités LTSs* $\mathcal{P}_i =< S_i, s^0_i, \Sigma_i, T_i >$ *où chaque entité* $\mathcal{P}_i, i \in [1..n], n \in \mathbb{N}$*, est un LTS obtenu en remplaçant dans* $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *chaque étiquette* $(\mathcal{P}_j, m, \mathcal{P}_k)$ *par,* $m!$ *si* $j = i$ *avec* $m?$ *si* $k = i$ *et avec* τ *(action interne) sinon ; et enfin, supprimer les transitions* τ *en appliquant les algorithmes de minimisation [54].*

2 Composition de système

Notre travail repose sur deux modèles de communication. Le modèle synchrone [94], et le modèle asynchrone [26] de protocole d'interactions d'entités communicantes. Le modèle synchrone nous permet de simplifier la spécification de protocoles avec l'absence de canaux de message (files d'attente). Dans le cas d'une communication entre deux entités, 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. Cette exigence facilite le raisonnement sur les LTSs des entités communicantes en évitant les problèmes liés à l'asynchronisme (blocage, perte d'information, désordre des messages échangés, etc). Cependant, bien que le modèle de communication asynchrone conduise à plusieurs limites de spécification et de vérification, cette sémantique correspond aux cas réels.

Dans ce qui suit, nous définissons d'abord le modèle de communication synchrone pour les LTSs, utilisés par nos travaux, puis nous présentons le cas des systèmes en sémantique asynchrone.

2.1 Communication synchrone

Une communication synchrone se base sur le principe des échanges en temps réel ou instantané (rendez-vous). Il s'agit d'une communication directe entre deux interlocuteurs ou plus (cas du multicast) qui participent de manière effective à un échange d'information durant la même session de communication. Nous définissons formellement un protocole de communication synchrone comme suit.

Définition 7 (CP synchrone) *Considérons des systèmes reposant sur des communications par rendez-vous. Aucun canal de messages n'est requis.*

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

- $S_{sync} \subseteq S_1 \times \ldots \times S_n$
- $s_{sync}^0 \in S$, $s^0 = (s_1^0, \ldots, s_n^0)$
- $\Sigma_{sync} = \cup_{i=1}^{n} \Sigma_i$
- $T_{sync} \subseteq S \times \Sigma \times S$, *et pour* $s_{sync} = (s_1, \ldots, s_n) \in S$ *et* $s'_{sync} = (s'_1, \ldots, s'_n) \in S$ *où définit la transition*

(interact) $s \xrightarrow{m} s' \in T_{sync}$ $\exists i, j \in \{1, \ldots, n\}, i \neq j$, $m \in \Sigma$ *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* $\forall k \in \{1, \ldots, n\}, k \neq i \wedge k \neq j \Rightarrow s'_k = s_k$

2.2 Communication asynchrone

En sémantique asynchrone, une entité communicante peut envoyer un message m lorsqu'elle se trouve dans un état permettant cet envoi, même si l'entité réceptrice n'est pas dans un état lui permettant de recevoir ce message m. Dans ce modèle, chaque entité dispose d'un canal (ou plusieurs, par exemple un pour les messages d'entrée et un pour les messages de sortie) qui conserve 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'états du système global peut être infini même si les entités communicantes impliquées sont initialement décrites à l'aide d'un espace d'états fini. Par conséquent, dans ce contexte, la vérification des propriétés du système, par exemple, l'atteignabilité ou l'absence du blocage, est considérée indécidable [26]. Nous définissons formellement le protocole de communication asynchrone comme suit.

Définition 8 (CP asynchrone) *Considérons des systèmes interagissant de manière asynchrone en envoyant et en recevant des messages sur des canaux de communication (non)bornés.*

Pour un ensemble d'entités $\downarrow CP = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *avec* $\mathcal{P}_i =< S_i, s_i^0, \Sigma_i, T_i >$, *où* Q_i *est la file d'attente associé à chaque entité communicante, la composition asynchrone* $Sys_{async}(\downarrow CP)$ *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* $s' = (s'_1, Q'_1, \ldots s'_n, Q'_n) \in S_a$. *Trois types de transitions sont possibles :*

(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)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

3 Réalisabilité de CP

Une problématique majeure, déjà abordée dans plusieurs travaux, concerne la propriété de *réalisabilité* d'un *CP*. Il s'agit de garantir l'existence d'un ensemble d'entités, communicant par envois et réceptions de messages, dont la composition génère les mêmes séquences d'échanges de messages que celles spécifiées par le *CP* décrivant ces échanges. La propriété de réalisabilité se décompose de trois conditions détaillées dans ce qui suit.

3.1 Propriété d'équivalence

Nous définissons formellement la relation d'équivalence entre le protocole de conversation et sa recomposition synchrone à partir des entités projetées comme suit :

Définition 9 (Equivalence) *Pour un ensemble d'entités communicantes, un CP est equivalent à sa composition* $Sys_{sync}(\downarrow CP)$ *, notée formellement,* $CP \equiv Sys_{sync}(\downarrow CP)$*, si et seulement si, les ensemble de séquences de messages échangés sont égaux, i.e. les traces sont des traces équivalentes.*

3.2 Propriété de synchronisabilité

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. Nous définissons formellement la propriété de synchronisabilité comme suit.

Définition 10 (Synchronisabilité) *Soit l'ensemble d'entités communicantes* $\downarrow CP = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$*, le CP synchrone* $Sys_{sync}(\downarrow CP) = (S_s, s_s^0, L_s, T_s)$*, et le CP asynchrone* $Sys_{async}(\downarrow CP) = (S_a, s_a^0, L_a, T_a)$*, et deux états* $r \in S_s$ *et* $s \in S_a$ *sont synchronisables 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* $Sys_{async} \in Sync \Leftrightarrow Sync\_st(s_s^0, s_a^0)$*.*

3.3 Propriété de bonne définition (WF)

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 (*i.e.* Q_i est non vide), le *CP* asynchrone atteint un état où Q_i est vide. Nous définissons formellement la propriété de bonne définition comme suit.

Définition 11 (WF) *Soit WF l'ensemble des systèmes bien formés sur un ensemble d'entités communicante* $\downarrow CP = \{P_1, \ldots, P_n\}$*.* $Sys_{async}(\downarrow CP) = (S_a, s_a^0, \Sigma_a, T_a)$ *est un CP asynchrone défini sur un ensemble d'entités communicantes* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *est bien formé, i.e.* $Sys_{async} \in WF$*, 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* $s_a^0 = (s_1^0, \epsilon, \ldots, s_n^0, \epsilon)$*, 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* $\forall Q'_i, \mid Q'_i \mid = 0$*.*

3.4 Propriété de réalisabilité

Un protocole de conversation *CP* est réalisable si et seulement si

- les entités obtenues par projection à partir de ce protocole sont synchronisables,

— le *CP* asynchrone résultant de la composition des entités est bien formé (WF est vérifiée), et
— la version synchrone du système distribué (*CP* synchrone) $\{P1, \ldots, Pn\}$ est équivalente au *CP* du départ. Cette définition est emprunter de l'approche présentée dans [38].

Nous définissons formellement la propriété de réalisabilité comme suit.

Définition 12 (Réalisabilité) *Soit R l'ensemble des CP réalisables. R est défini comme un sous-ensemble de CP déterministes tels que :*

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

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 [10] permet de décomposer le processus de vérification de la réalisabilité d'un CP selon les étapes suivantes :

— *Calcul de l'ensemble des entités impliquées* $\downarrow CP = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$ *par projection à partir du CP.*
— *Calcul des compositions synchrones et asynchrones.*
— *Vérification de la synchronisabilité entre le modèle de communication synchrone et asynchrone.*
— *Vérification de l'équivalence entre CP et la composition des entités communicantes.*
— *Vérification de la WF du CP.*

4 Réalisabilité : travaux connexes

Dans cette section, nous nous concentrons sur les approches de vérification et de validation des propriétés du système distribué décrits par des chorégraphies. La vérification et la validation de tels systèmes [10, 42, 64] 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 communicantes à partir d'un système global décrivant les échanges entre ces dernières. La formalisation et la vérification du modèle ont été étudiées par : les algèbres de processus [73,81,82,87], les systèmes à base de transitions d'états [46, 48, 64, 85], les réseaux de pétri [15, 29, 32] etc. Des

techniques de vérification sur le modèle obtenu sont mises en oeuvre et chaque approche propose un contexte de vérification différent.

Le but de l'étude des approches présentées dans cette section est de comprendre la notion du bon fonctionnement de système à base de *CP* et de pouvoir distinguer par la suite les différentes conditions de validation de tels systèmes, afin de pouvoir identifier des contributions pertinentes.

4.1 Approches correctes-par-construction

Cette section est dédiée aux approches de vérification correctes-par-construction basées sur le raffinement, à partir d'une spécification abstraite. La cohérence du raffinement est préservée à tous ses niveaux grâce à la conservation des propriétés liées au comportement et exprimées sur les différents modèles. Ces approches permettent une vérification a priori des propriétés exprimées.

Farah et al. [39]

Les auteurs utilisent les systèmes d'états étiquetés comme modèle de base pour la description des chorégraphies de systèmes distribués. En se basant sur la méthode B-événementiel, [39] proposent un modèle correct-par-construction qui permet de composer des systèmes distribués asynchrones qui réalisent a priori les chorégraphies décrites. Le travail établi par les auteurs 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. Ce système est ensuite raffiné dans sa version asynchrone avec des files d'attente de capacité illimitée. Grâce au invariants exprimant qu'une séquence de messages échangés est préservée à chaque étape de raffinement, cette approche fournit une preuve formelle de la réalisabilité des chorégraphies déterministes. Ces travaux utilisent une file d'attente globale indexée qui permet de préserver une topologie de communication en anneau (Finkel et al [41]).

Gadouche et al. [43]

Dans le travail de [43], 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 correct-par-construction. Le modèle est composé de niveaux d'abstraction obtenus par 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.

4.2 Approches de vérification de modèles

Cette partie est dédiée aux approches qui offrent une vérification formelle a posteriori de la propriété de réalisabilité des systèmes états-transitions.

4.2.1 Vérification sur modèles de CP

Dans cette section, nous résumons les approches de vérification de la propriétés de réalisabilité des systèmes modélisés sous forme de protocole de conversation CP

Basu et al. [10]

Dans nos travaux, nous nous appuyons sur la définition de réalisabilité des *CP* présentée dans [10]. Le principal but des travaux 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 ?* Pour répondre à cette question, les auteurs fournissent des conditions nécessaires et suffisantes pour garantir la réalisabilité des chorégraphies, à savoir : l'équivalence, la synchronisabilité et la bonne forme de système (Well-Formedness WF). L'efficacité de l'approche proposée a été démontrée en particulier sur (1) des chorégraphies de services Web, (2) des diagrammes de collaboration UML. La vérification est conduite par des techniques de model checking qui est limité par la taille des *CP* vérifiés.

Fu et al. [42]

Ici, les auteurs s'intéressent à la réalisabilité de systèmes asynchrones avec files d'attentes de capacité infinie. La proposition consiste à vérifier un protocole de conversation modélisé par un automate de büchi. Ce protocole est dit réalisable si et seulement si les trois conditions (1) le protocole de communication est spécifié par un automate de büchi, (2) la propriété de réalisabilité du *CP* est vérifiée par l'automate de büchi, (3) le *CP* est formé à partir des entités communicantes via la projection sont satisfaites par cet automate.

Kazhamiakin et al. [56], Peltz et al. [71]

Les approches proposées par [56, 71] sont basées sur la modélisation d'un *CP* à l'aide de systèmes à base de transitions d'états. Les auteurs considèrent deux définitions de réalisabilité de systèmes : la réalisabilté forte définie par la préservation de l'ordre des actions internes et externes par la chorégraphie, et la réalisabilité faible, où seul l'ordre des échanges de messages entre une paire d'entités communicantes (émettrice/réceptrice) est préservé. Cette classification de la réalisabilité permet de traiter efficacement ce problème en définissant des degrés de réalisabilité du protocole de conversation.
Dans ce cas, la réalisabilté est étudiée au niveau global (au niveau des interactions du système) et au niveau local (au niveau du 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 asynchrone avec des files d'attentes de

capacité infinie, en utilisant WS-CDL (Web Services Choreography Description Language).

Finkel et Lozes. [41]

Les auteurs [41] ont récemment prouvé que la synchronisabilité n'est pas décidable dans le cas général. Finkel et ses collaborateurs, se concentrent sur le mode de communication peer-to-peer. Des contre-exemples liées aux propriétés de modèles de communication, à la fois pour la sémantique peer-to-peer et pour la sémantique de boite aux lettres (Mailbox), relatifs aux résultats de [13, 14] sont produits. Ces travaux indiquent que la synchronisabilité des *CP* décrivent des échanges suivant une topologie d'anneau (oriented-ring) est décidable, à la fois en mode boite aux lettres (Mailbox) ou peer-to-peer.

4.2.2 Vérification sur modèles spécifiques

Cette catégorie de techniques est utilisable pour les protocoles dont les canaux de communication sont de capacité limitée. Leur principe est le parcours de tous les états possibles du protocole en vérifiant pour chaque état, les propriétés requises. Bien que cette catégorie de techniques permet le parcours exhaustif, elle demeure coûteuse en terme de temps d'exécution et d'espace mémoire.

— **Approches basées sur les sessions types.** Divers travaux permettent de modéliser et vérifier des protocoles multi-parties (impliquant plusieurs parties), 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 [26]. La sécurité des interactions (généralement indécidable) est vérifiée selon deux approches principales : (a) supposer la décidabilité d'une propriété de synchronisabilité [13, 14], puis vérifier des 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, 34, 53].

Scalas et al. [83]

Dans ce travail, les auteurs proposent une évolution de la théorie généralisée des types de sessions multi-parties (MPST). Ils préconisent une révision des fondements théoriques de la MPST. les MPSTs classiques ont une propriété de cohérence limitée, avec des restrictions inhérentes qui peuvent être affaiblies lors de cette évolution. [83] propose une nouvelle théorie MPST 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 comportementales générales. De plus, [83] établit des propriétés de la sécurité de protocoles et de processus. Ils produisent une analyse détaillée des propriétés, montrant comment, la nouvelle théorie permet d'assurer la décidabilité du contrôle de propriétés et garantit de manière statique que les

processus bénéficient, par exemple, l'absence de blocage et de propriétés de vivacité.

Lange et al. [59]

Les auteurs proposent dans ce travail une procédure de vérification des propriétés des automates de session communicants (csa), c'est-à-dire des automates correspondant à des types de session à multi-parties. [59] introduit une nouvelle propriété de compatibilité pour des csa asynchrone, appelée compatibilité k-multi-partie (k-mc), qui est un sur-ensemble strict de la compatibilité multi-partie synchrone proposée dans la littérature. La propriété est décomposée en deux sous-propriétés : (i) la k-sécurité qui garantit que, tous les messages envoyés peuvent être reçus et que chaque automate peut effectuer un changement d'état ; et (ii) la k-exhaustivité qui garantit que toutes les actions k-accessible d'envoi peuvent être déclenchées. [59] montrent aussi que les systèmes k-exhaustifs caractérisent correctement et complètement les systèmes csa, i.e. chaque automate se comporte de manière uniforme pour toute borne supérieure ou égale à k. Cette proposition s'applique efficacement sur de grands systèmes en utilisant des techniques de réduction d'ordre partiel. [59] montrent plusieurs exemples de la littérature qui sont k-mc.

Deniélou et al. [33]

Des langages de type session ont été introduits pour les systèmes de machines à états finis communicantes [33]. Plusieurs notions similaires à celle de la synchronisabilité ont également été étudiées dans différents contextes. L'élasticité relâchée (Slack elasticity) est le nom donné aux systèmes distribués asynchrones qui se comportent de la même manière pour toutes les communications. Cette propriété a été étudiée dans la conception matérielle [66], dans le but de garantir que certaines transformations de code conservent la sémantique, dans le calcul haute performance. Il s'agit de garantir l'absence de blocage et autres bugs dans les programmes d'interface de passage de message MPI [84, 88].

— **Approches basées sur les algorithmes ad-hoc de vérification.** Dans cette section, nous présentons les approches basées sur l'utilisation et la comparaison des algorithmes de vérifications existants.

Heussner et al. [50]

[50] a développé une plate-forme McScM (Model Checker for Systems of Communicating Machines) permettant de mettre en oeuvre et de comparer des algorithmes de vérification de systèmes de transitions à états finis échangeant des messages en utilisant des files FIFO non bornées. L'outil de vérification proposé implémente plusieurs techniques de vérification de modèle : CEGAR avec différentes méthodes : raffinement, interprétation abstraite et

vérification abstraite de modèle. McScM fournit l'infrastructure de base pour la mise en œuvre d'algorithmes de vérification. Elle permet aussi l'implémentation de nouvelles spécifications à un niveau d'abstraction élevé.

Genest et al. [45]

Dans ce travail, [45] prouve l'équivalence entre plusieurs formalismes de spécification d'ensembles bornés de MSCs (Message sequence charts), à savoir des machines communicantes à états finis, la logique monadique du second ordre et des graphes de séquence de messages. De nouveaux résultats ont été obtenus par une adaptation de la technique de Kuske qui permet de transférer les résultats sur les traces de Mazurkiewicz vers le domaine des MSCs. La construction de CFM (Communicating finite-state machine), qui accepte tous les MSCs B-bornés, repose sur une nouvelle caractérisation de cette classe de MSC. Étant donné que les différents formalismes peuvent être transformés efficacement l'un en l'autre, les auteurs obtiennent la décidabilité de plusieurs problèmes (passage à l'échelle, explosion combinatoire, etc) de vérification de modèle impliquant des CFM, des graphes de séquence de messages et la logique monadique du second ordre.

4.3 Compatibilité des chorégraphies

L'objectif principal des approches étudiées dans cette section 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 spécifiées 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 entités 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, nous considérons que le moyen de commu-

nication 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 en recevoir.

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 choisi, et en évaluant les propriétés sur cet ensemble (blocage, perte ou désordre des messages, etc). La spécification des : entités communicantes, modèles de communication, système global dans lequel les entités interagissent et propriétés de compatibilité, constituent un cadre pour la vérification de la compatibilité des entités communicantes.

La suite de la section décrit et formalise ce cadre et démontre sa validité en termes d'exactitude et de complétude.

Wu et al. [93]

Les auteurs proposent une notion de compatibilité pour vérifier deux services décrits en utilisant le π-calcul. Selon leur proposition, deux services sont compatibles, s'il existe au moins une séquence de transitions permettent d'atteindre les états finaux. Leur travail ne calcule pas la compatibilité détaillée des différents états de protocole et ne propose pas de détection d'incohérence.

Lohmann et al. [64]

Les auteurs [64] ont lié le problème de réalisabilité des chorégraphies au problème de contrôlabilité de l'orchestration. La relation étroite entre ces problèmes offre un moyen uniforme d'analyser et de modéliser des services en interaction arbitraire, en transformant une spécification de chorégraphie en une orchestration de services. Les algorithmes de contrôlabilité sont appliqués pour vérifier la réalisabilité sur les orchestrations. En outre, ils suggèrent des définitions alternatives pour la réalisabilité et discutent plusieurs propositions pour définir la réalisabilité qui diffèrent par le degré de couverture de l'interaction spécifiée. Ces techniques ont abouti à un cadre formel permettant de spécifier et d'analyser des chorégraphies avec une communication synchrone et asynchrone.

4.4 Exploration partielle du graphe d'accessibilité

Nous présentons dans cette section quelques techniques de validation de protocoles qui se basent sur des approches qui effectuent une exploration partielle du graphe d'accessibilité [91] tout en essayant de détecter le maximum d'erreurs logiques.

Liu et al. [63]

Dans ce travail, les auteurs étendent la technique d'analyse d'atteignabilité équitable généralisée aux protocoles cycliques avec transitions non déterministes et internes. Ils montrent qu'à l'exception de quelques modifications mineures, tous les résultats établis pour les protocoles cycliques utilisés dans les références connexes peuvent être reportés dans des protocoles cycliques avec transitions non déterministes et internes. Ils identifient aussi un nouveau type d'erreur logique appelée "indefiniteness". Ce dernier signifie qu'un protocole peut atteindre un état à partir duquel, l'un des processus peut exécuter indéfiniment des transitions internes, sans communiquer avec les autres processus du protocole. La décidabilité de la détection de cette erreur logique est également été démontrer.

West et al. [90]

Les auteurs de [90] propose une technique d'exploration aléatoire des états de protocole (Random Walk State Exploration (RWSE)). Cette technique peut être vue comme étant une analyse d'accessibilité réduite, au lieu de parcourir toutes les branches du graphe de protocole, on se contente de parcourir seulement quelques branches choisies d'une manière aléatoire. Les auteurs de cette technique, suppose qu'une analyse partielle du fonctionnement de protocole suffit pour détecter un nombre important d'erreur, cela permet de réduire la complexité de calcul tout en assurant un bon fonctionnement du protocole. Bien que cette technique s'avère très simple, elle ne spécifie pas des conditions de terminaisons. Dans l'analyse d'accessibilité exhaustive, le processus de vérification s'arrête une fois tout le graphe est parcouru. Dans l'analyse RWSE, puisque on fait un parcourt aléatoire, il est impossible de décider que l'analyse va se terminer après un parcourt de telle ou telle branche ou bien de dire que toutes les erreurs sont détectées. Donc, cette technique permet une détection des erreurs de conception mais ne garantit pas que le protocole n'aille pas présenter d'autres erreurs lors de son exécution.

5 Étude de l'évolution et de la reconfiguration de CP

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

5.1 Cas de chorégraphies

La Chorégraphie, est l'approche permettant de construire une composition. Plusieurs langages et plates-formes sont définis au tour de cette approche . Bien que certains de ces langages soient considérés comme des standards, l'absence de sémantique formelle et la possibilité d'interprétations ambiguës des documents décrivant ces standards, constituent un frein à leur utilisation et à leur généralisation, et donc compromettent le principe d'interopérabilité. Par conséquence, les services à composer doivent respecter un modèle théorique pour arriver à une spécification de la façon dont sont coordonnés ces services afin de répondre à un besoin, c'est-à-dire l'ordre et les conditions dans lesquelles ils s'enchaînent. Les approches de cette section utilisent la notion de chorégraphie pour modéliser les systèmes à évoluer.

Roohi et Salaün [79]

Les auteurs proposent une méthode de vérification de la reconfigurabilité des CP. L'approche considère deux CP réalisables, CP et CP', et deux ensembles d'entités communicantes PS et PS' obtenues par projection des deux CP, 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.

Wombacher et Andreas. [92]

Les auteurs dans [92] utilisent un modèle formel basé sur les automates d'états finis (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 sa chorégraphie correspondante. Les changements proposés sont soit des ajouts ou des éliminations des séquences de messages. Ces changements sont propagés aux entités distribuées tout en gardant la cohérence de la chorégraphie. L'approche proposée par les auteurs, fournit un moyen de propager les modifications du niveau de chorégraphie aux modifications du niveau d'orchestration. La principale contribution de ce travail consiste à utiliser un modèle formel pour représenter les propriétés syntaxiques des orchestrations et à appliquer une notion d'invariants sémantiques pour aligner avec la chorégraphie, le modèle d'orchestration d'une manière syntaxique et sémantique.

Mila et al. [72]

Les auteurs de ce travail définissent un nouveau langage de haut niveau nommé DIOC (langage de chorégraphie) destiné à la programmation cohérente des applications distribuées. La sémantique du langage DIOC repose sur les systèmes de transitions étiquetés. L'approche proposée 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. Ils définissent aussi un langage de bas niveau, appelé DPOC (langage des entités communicante), basé sur les primitives d'envoi et de réception standard. Ils proposent une fonction de projection préservant le comportement et compilant DIOC en DPOC. Une preuve formelle de correction de la projection est donnée. L'évolution est garantie même dans un scénario où le nouveau code utilisé pour les mises à jour change de façon dynamique à tout moment et sans préavis.

5.2 Cas de processus d'affaires

Nous nous concentrons sur les solutions proposées afin d'assurer la réalisabilité après chaque mise à jour du système, suite à une reconfiguration, adaptation ou toute autre évolution.

Les auteurs de [76] ont constaté l'utilisation 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. Ces derniers permettent la définition, l'exécution et la surveillance des processus opérationnels d'une entreprise.

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 chorégraphies 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.

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.
Dans ce qui suit, nous étudions les différents propositions des approches traitant l'évolution des systèmes.

5.3 Cas de Service Web

Les approches de cette section utilisent les services web pour l'analyse de l'évolution. Les Services Web, sont l'exemple le plus connu de la mise en œuvre d'une SOA (Services Oriented Architecture). La notion de Service Web désigne un composant logiciel mis à disposition par un fournisseur, et invocable sur Internet par des clients (des utilisateurs ou d'autres services), les interactions entre services Web se font via des opérations d'échange de messages. L'ambition apportée par les services Web est de permettre une plus grande interopérabilité entre applications sur Internet. On envisage ainsi des services Web capables de négocier entre eux, ou de se composer en des services plus complexes. Le but de cette vision consiste à créer des applications constituées uniquement de services Web interagissant entre eux. Dans une telle perspective, peu importe où est déployé le service. ce qui importe est alors que le service remplisse un rôle bien précis. L'une des sources de complexité provient de la capacité des services à se composer entre eux, et donner ainsi naissance, de façon transparente pour l'extérieur, à de nouveaux services.

Leite et al. [60]

Les auteurs examinent 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 : [55, 76, 78, 79, 92]. L'étude de l'évolution des systèmes a été abordée dans [67]. 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 logicielles, les transformations graphiques et autres, mais, aucun travail n'étudie les systèmes dont les interactions sont décrites sous forme de *CP*.

Rinderle et al. [76, 78]

Les auteurs proposent une méthode d'évolution contrôlée des systèmes où la propagation des changements apportés à une entité nécessitent la vérification de son effet sur les autres entités formant la chorégraphie.

Les approches concernent les changements structurels de processus. En particulier, pour les modifications des processus privés, elles permettent le recalcul automatiquement des vues publiques associées et, le cas échéant, ils propagent les modifications qui en résultent dans les processus partenaires. Les aspects très importants de ces travaux sont leurs pertinences pratique et leurs fondement formel. Dans [76], un modèle formel et des critères précis, permettant de décider automatiquement des adaptations nécessaires en raison de modifications des processus des partenaires privés sont fournis.

Fdhila et al. [40]

Dans ce travail, les auteurs se concentrent sur le domaine de la gestion des processus métier (BPML) et l'architecture orientée services (SOA). Ils décrivent des

chorégraphies de 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. L'approche proposée permet la propagation du changement dans les chorégraphies avec plusieurs processus partenaires en interaction. Le principe s'articule autour de six étapes : (i) vérifier si un changement doit être propagé ou isolé (c'est-à-dire, le changement est local), (ii) calculer les effets privé-public ; c'est-à-dire propager les changements du modèle privé de l'initiateur du changement vers son modèle public, (iii) calculer les effets public à public ; c'est-à-dire propager les modifications aux partenaires impliqués, (iv) négocier les modifications avec les partenaires concernés, (v) calculer les effets public-privé (si les négociations ont abouti). En d'autres termes, chaque partenaire calcule en interne les effets des modifications publiques sur son modèle privé et enfin (vi) vérifie la compatibilité et la cohérence de la chorégraphie et met en œuvre les modifications.

Clarke et al. [30]

Les auteurs décrivent une méthode pour extraire des fonctions d'abstraction du texte du programme. Dans cette méthode, les prédicats de flux de contrôle du programme à vérifier (code source) sont utilisés pour factoriser l'espace d'états en classes d'équivalence agissant en tant qu'états abstraits. Pour des raisons de performance, les fonctions d'abstraction dans la littérature sont souvent définies uniquement pour des variables individuelles, et la fonction d'abstraction est obtenue en tant que composition de fonctions d'abstraction individuelles. Contrairement à cela, les auteurs utilisent des fonctions d'abstraction qui sont définies sur des n-uplets finis de variables appelées grappes de variables extraites du code du programme. Cette approche permet d'exprimer des conditions impliquant plusieurs variables (telles que $x < y$) dans la fonction d'abstraction, ce qui permet de limiter la taille de l'espace abstrait. Ils introduisent des algorithmes pour affirmer si les contre-exemples abstraits sont parasites, ou ont des contre-exemples concrets correspondants. Si un contre-exemple est parasite, Ils identifient le préfixe (partie correspondante a une trace réelle) le plus court du contre-exemple abstrait qui ne correspond pas à une trace réelle dans le modèle concret. Le dernier état abstrait de ce préfixe («l'état d'échec») doit être scindé en états moins abstraits en affinant les classes d'équivalence de manière à éliminer le contre-exemple parasite. Ainsi, une fonction d'abstraction plus raffinée est obtenue.

6 Réparation de chorégraphies

Lorsqu'une chorégraphie est non réalisable, est-il possible de la réparer automatiquement de tel sorte que sa réalisabilité soit restaurée ? 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 abordant le même problème.

6.1 Renforcement de la réalisabilité

Dans certaines situations, la propriété de réalisabilité en cas de violation peut être rétablie ou réparée. Cette section passe en revue différentes approches relatives à la réparation.

Güdemann et al. [46]

Ici [46], la réalisabilté est renforcée par l'introduction de modules appelés moniteurs de synchronisation. Un moniteur est créé pour chaque entité du *CP*. Son rôle est de synchroniser les comportements des entités communicantes afin de satisfaire le protocole de conversation globale souhaité. Des tests de réalisabilité sont effectués afin de détecter les comportements incohérents que les auteurs souhaitent éviter, puis l'adaptation des comportements des entités via leurs moniteurs est réalisée. L'approche proposée est basée sur le test des contre exemples.

Basu et al. [11]

Dans ce travail, les auteurs étudient la possibilité de réparation des chorégraphies identifiées comme non réalisables, l'objectif étant d'identifier un ensemble de modifications à apporter sur une chorégraphie non réalisable donnée pour restaurer la propriété de réalisabilité. Ces travaux proposent une technique permettant de réparer automatiquement les chorégraphies non réalisables, basée sur deux stratégies : 1) la relaxation, où de nouveaux comportements sont ajoutés à la chorégraphie d'origine dans le cadre de la réparation et 2) la restriction, où des comportements non désirés (non spécifiés par la chorégraphie) dans le système obtenus en projetant la chorégraphie sont supprimés. Des garanties formelles de correction et de terminaison sont fournies. Cette technique a été appliquée avec succès sur plusieurs chorégraphies identifiées comme non réalisables dans le domaine du contrat Singulary OS et des services Web.

Autili et al. [8]

Les auteurs formalisent une méthode qui force la réalisabilité d'une 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 forment la chorégraphie. Le registre contient des services publiés par des fournisseurs qui ont identifié des opportunités commerciales dans le domaine d'intérêt. Le registre contient également l'enregistrement des utilisateurs intéressés par l'exploitation de 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

incontrôlées. De plus, ils décrivent un algorithme de coordination distribué implémenté par les CD pour réaliser la logique de coordination.

Autili et al. [7]

Dans leur travail, les auteurs proposent une approche formelle qui enforce 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 communication les entités synthétisées appliquent la collaboration prescrite par la spécification de chorégraphie. ils fournissent une formalisation complète de la méthode de synthèse automatisée pour la mise en application distribuée de la réalisabilité de la chorégraphie mise en œuvre par la plate forme CHOReVOLUTIONStudio. Elle prend en entrée la spécification de la chéographie sous la forme d'une machine à états et génère automatiquement un ensemble de délégués de coordination (CD).

Ils affirment aussi la correction de leur méthode de synthèse et illustrent sa formalisation sur un exemple explicatif.

6.2 Contrôle d'équivalence

Dans [47], les auteurs présentent une approche de vérification de chorégraphies à l'aide des techniques de contrôle des modèles et de l'équivalence. Ils ont développé une connexion à l'outil de vérification CADP [44] via la traduction de un des langages de spécification d'entrée CADP en algèbre de processus LNT. Cela permet la vérification de certaines tâches d'analyse, comme 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 *CP* à l'aide du contrôle d'équivalence, les contrôleurs distribués sont générés via un processus itératif pris principalement de [46]. Les contre-exemples générés par CADP 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.

7 Discussion

Les approches de modélisation et de vérification de systèmes distribués étudiées dans les sections précédentes répondent à trois besoins principaux :

— *Vérification de la validité des systèmes distribués existants.* Il s'agit de prouver que les systèmes existants répondent à leurs spécifications. La vérification est souvent statique et repose sur des assistants de preuve dotés d'un degré variable d'automatisation.

— *Conception de systèmes distribués corrects-par-construction.* En d'autres termes, dériver des systèmes de leurs spécifications. Cela repose généralement

sur une approche descendante avec un raffinement mécanique incrémental, éventuellement complétée par une traduction certifiée conforme des spécifications de systèmes.

— *Détecter et éviter les erreurs d'exécution des implémentations de systèmes distribués.* En pratique, il est possible que les implémentations des systèmes distribués étudiés n'aient pas été construites correctement ou prouver qu'elles soient sans erreur ou que des erreurs puissent survenir après de longues exécutions.

Le formalisme et le contexte d'analyse et de vérification des protocoles de conversation des travaux connexes sont définis de différentes façons. La Table 3.1 résume les quatre classes d'approches identifiées dans notre étude de l'état de l'art. Indépendamment du formalisme adopté, les approches étudiées peuvent être classées comme suit :

Approches de vérification et validation de la réalisabilité	Basu et al [10], Fu et al [42], Lohmann et al [64], Brand et al [26], Basu et al [13], Basu et al [14], Bocchi et al [24], Denielou et al [34], Honda et al [53].
Approches d'évolution des systèmes	Rinderle et al [76], Leite et al [60], Roohi et al [79], Wombacher et al [92], Jureta et al [55], Rinderle et al [78], Rinderle et al [76], Medvidovic et al [67], Fdhila et al [40], Preda et al [72], Clarke et al [30].
Approches correctes-par-construction	Farah et al [39], Gadouche et al [43].
Approches de réparation	Güdemann et al [46], Cámara et al [11], Autili et al [8], Autili et al [7], Güdemann et al [47].

TABLE *3.1 – Les quatre classes d'approches existantes*

1. La première famille d'approches englobe les processus d'analyse basés sur des outils de vérification de modèle ou modèle checking (CADP, ...). Les auteurs définissent un ensemble de contraintes à vérifier par les protocoles de conversation modélisant des systèmes répartis afin de valider le bon fonctionnement de ces derniers.

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

3. La troisième classe concerne les approches de vérification formelle correcte-par-construction des propriétés des modèles de systèmes *CP*.

4. 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 :

— *le problème de passage à l'échelle en raison de l'explosion combinatoire d'espace d'états lors de la validation des modèles complexes*, et

— *la difficulté de décharger les obligations de preuve et la complexité des étapes suivies afin de modéliser, vérifier formellement et valider la réalisabilité des modèles de système.*

L'approche proposée dans [39] 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). Ces derniers s'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 des problèmes de :

— explosion combinatoire d'espace d'état lors de la vérification de réalisabilité.

— La réparation a priori des systèmes non réalisables décrient sous forme de *CP*.

sont les objectifs fondamentaux de cette thèse.

Deuxième partie

Contributions

Chapitre 4

Construction incrémentale : algèbre de composition

Sommaire

1 Opérateurs de composition 43
2 Réalisabilité-par-construction : définition des con-ditions suffisantes 46
3 Réalisabilité-par-construction : théorèmes et preu-ves . 50
3.1 Schéma de preuve. 50
3.2 Schéma de preuve. 52
3.3 Théorèmes de réalisabilité pour la séquence, le choix et l'itération. 52
3.4 Cas d'étude 60

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 en un ensemble d'opérateurs et de propriétés de composition des *CP*s avec des théorèmes et preuves mathématiques 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 première contribution concerne 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 comprend : un opérateur de séquence noté par $\otimes_{(\gg, s^f_{CP})}$, un opérateur conditionnel notée $\otimes_{(+, s^f_{CP})}$ et un opérateur d'itération noté par $\otimes_{(\circlearrowleft, s^f_{CP})}$. Chacun de ces opérateurs est caractérisé par deux paramètres, le premier est un *CP réalisable*

possédant un état final $s^f_{CP} \in S^f_{CP}$ et le second est un (un ensemble de) protocole de conversation de base CP_b formé d'une unique transition. Chaque expression de la forme $\otimes_{(op,s^f_{CP})}(CP, CP_b)$ ou $op \in \{\gg, +, \circlearrowleft\}$ 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, noté $\varnothing$, avec tout CP_b (resp. $\{CP_{bi}\}$) produit un CP_b (resp. $\{CP_{bi}\}$) *i.e.* $\otimes_{(op,s^f_{CP})}(\varnothing, CP_b) = CP_b$ (resp. $\otimes_{(op,s^f_{CP})}(\varnothing, \{CP_{bi}\}) = \{CP_{bi}\}$. $\varnothing$ définit l'élément neutre pour ces opérateurs.

Dans ce qui suit, nous définissons formellement chaque opérateur du point de vue structurel. Les conditions sous lesquelles la réalisabilité est préservée sont présentées dans la section 2.

Définition 13 *Opérateur de séquence* $\otimes_{(\gg,s^f_{CP})}$.
Soient

- $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *un protocole de conversation avec un état final* $s^f_{CP} \in S_{CP}$ *et*
- 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* $\{s_{CP_b}, s'_{CP_b}\} \cap S_{CP} = \varnothing$.

La composition séquentielle de CP avec CP_b *à l'état* s^f_{CP} *noté par* $CP_{\gg} = \otimes_{(\gg,s^f_{CP})}(CP, CP_b)$ *est le* $CP_{\gg} =< S_{CP_{\gg}}, s^0_{CP_{\gg}}, L_{CP_{\gg}}, T_{CP_{\gg}} >$ *avec :*

- $S_{CP_{\gg}} = S_{CP} \cup \{s'_{CP_b} |\ s_{CP_b} \xrightarrow{l_{CP_b}} s'_{CP_b} \in T_{CP_b}\}$
- $s^0_{CP_{\gg}} = s^0_{CP}$
- $L_{CP_{\gg}} = L_{CP} \cup \{l_{CP_b}\}$
- $T_{CP_{\gg}} = T_{CP} \cup \{s^f_{CP} \xrightarrow{l_{CP_b}} s'_{CP_b}\}$
- $S^f_{CP_{\gg}} = (S^f_{CP} \setminus \{s^f_{CP}\}) \cup \{s'_{CP_b}\}$

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

Définition 14 *Opérateur conditionnelle* $\otimes_{(+,s^f_{CP})}$.
Soient

- $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *un protocole de conversation avec un état final* $s^f_{CP} \in S_{CP}$ *et*
- $\{CP_{bi} \mid 1 \leq i \leq n, n \geq 2\}$ *un ensemble fini de protocoles de conversation basique avec* $T_{CP_{bi}} = \{s_{CP_{bi}} \xrightarrow{l_{CP_{bi}}} s'_{CP_{bi}}\}$, $\{s_{CP_{bi}}, s'_{CP_{bi}}\} \cap S_{CP} = \varnothing$.

La composition conditionnelle 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 le* $CP =< S_{CP_+}, s^0_{CP_+}, L_{CP_+}, T_{CP_+} >$ *tel que*

- $S_{CP_+} = S_{CP} \cup \{s'_{CP_{b1}}, \ldots, s'_{CP_{bn}} |\ s_{CP_{bi}} \xrightarrow{l_{CP_{bi}}} s'_{CP_{bi}} \in T_{CP_{bi}}\}$

— $s^0_{CP_+} = s^0_{CP}$
— $L_{CP_+} = L_{CP} \cup \{l_{CP_{b1}}, \ldots, l_{CP_{bn}}\}$
— $T_{CP_+} = T_{CP} \cup \{s^f_{CP} \xrightarrow{l_{CP_{b1}}} s'_{CP_{b1}}, \ldots, s^f_{CP} \xrightarrow{l_{CP_{bn}}} s'_{CP_{bn}}\}$
— $S^f_{CP_+} = (S^f_{CP} \setminus \{s^f_{CP}\}) \cup \{s'_{CP_{b1}}, \ldots, s'_{CP_{bn}}\}$

Informellement, la composition conditionnel $CP_+ = \otimes_{(+,s^f_{CP})}(CP, \{CP_{bi}\})$ ajoute un ensemble de protocoles de conversation de base à l'état final s^f_{CP} ce qui forme un branchement (un choix) et met 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 $S^f_{CP_+}$.

Définition 15 *Composition en cycle* $\otimes_{(\circlearrowleft,s^f_{CP})}$.
Soient
— $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *un protocole de conversation avec un état final* $s^f_{CP} \in S_{CP}$ *et*
— CP_b *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 en cycle de CP avec CP_b *à l'état* s^f_{CP} *dénoté par*
$CP_\circlearrowleft = \otimes_{(\circlearrowleft,s_{CP})}(CP, CP_b)$ *est un* $CP =< S_{CP_\circlearrowleft}, s^0_{CP_\circlearrowleft}, L_{CP_\circlearrowleft}, T_{CP_\circlearrowleft} >$ *tel que*
— $S_{CP_\circlearrowleft} = S_{CP}$
— $s^0_{CP_\circlearrowleft} = s^0_{CP}$
— $L_{CP_\circlearrowleft} = L_{CP} \cup \{l_{CP_b}\}$
— $T_{CP_\circlearrowleft} = T_{CP} \cup \{s_{CP} \xrightarrow{l_{CP_b}} s'_{CP_b}\}$
— $S^f_{CP_\circlearrowleft} = S^f_{CP}$

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, i.e. , itération dans le CP (voir Figure 4.1).*
— CP_b *introduit un graphe dirigé, i.e. , la transition ajoutée ne définit pas de cycle dans* CP_b *(voir Figure 4.2).*

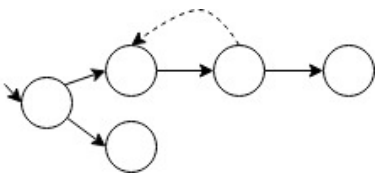


FIGURE *4.1 – Transition ajoutée (hachurée) forme un cycle*

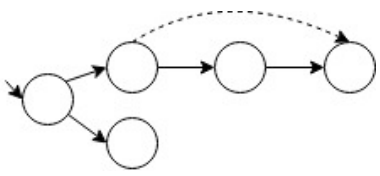


FIGURE *4.2 – Transition ajoutée (hachurée) forme un graphe dirigé*

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, conformément à la Définition 12. Les définitions de ces opérateurs sont syntaxiques et structurelles. Elles ne donnent aucune information sémantique pour garantir ou préserver la réalisabilité des *CP* 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 les conditions suffisantes pour la réalisabilité que nous avons identifiées. Ensuite, chaque condition est illustrée par un exemple ainsi qu'un contre exemple pour mieux la comprendre. Notons que ces conditions sont définies, sur les protocoles de conversation, au niveau structurel (syntaxique). Par conséquent, leur vérification sera effectuée à un niveau syntaxique sans nécessité une preuve au niveau sémantique avec projection et composition du système en mode de communication synchrone et asynchrone.

Condition 1 (Choix déterministe DC) *Tout protocole de conversation CP paramètre d'un opérateur de composition séquence, choix ou cycle doit appartenir à l'ensemble DC (voir Définition 4).*

Example 1 *La Figure 4.3 montre un exemple de CP non déterministe (Figure 4.3 (a)) et les entités obtenues par projection à partir de ce CP (Figure 4.3 (b)). Notons que, lors d'une communication asynchrone, les entités peuvent déclencher leurs transitions en pointillés. Avec un tel scénario de communication, l'entité* P1 *aboutit à un blocage directement après l'envoi du message* a! *étant donné que, la réception du message de* b? *n'est pas effectuée, i.e. , 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 ce type de blocage.*

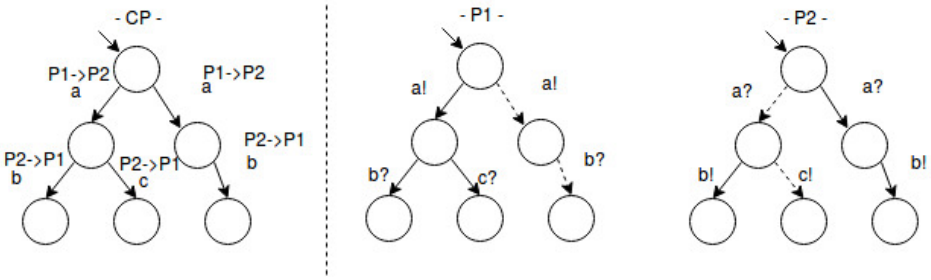


FIGURE *4.3 – Choix non déterministe CP*

Condition 2 (Absence de séquences indépendantes (ISeqF)) *La condition liée à deux échanges de messages consécutifs au niveau d'un CP est décomposée en deux conditions correspondantes à deux ordres différents 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 [41]. La seconde condition conserve l'ordre d'envoi des messages et aucune contrainte n'est imposée sur l'ordre de réception des messages.*

Cas1- Préservation de l'ordre d'envoi-réception des messages. *Soit ISeqF l'ensemble CP sans séquence indépendante.*
CP ∈ ISeqF si est seulement si

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

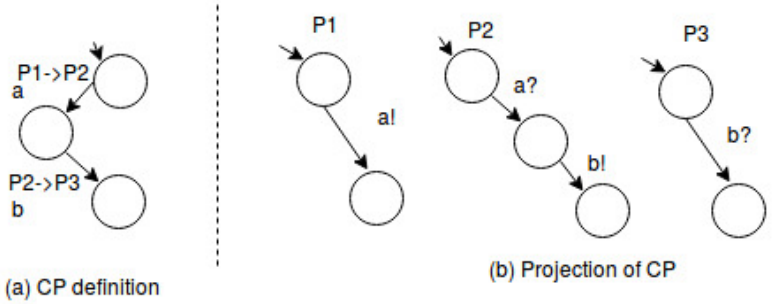


FIGURE *4.4 – Préservation de l'ordre des messages d'envoi-réception*

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

Cas2- Conservation de l'ordre d'envoi de messages. *Soit ISeqF l'ensemble de CP sans séquence indépendante.*

CP ∈ ISeqF si est seulement si

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

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,

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

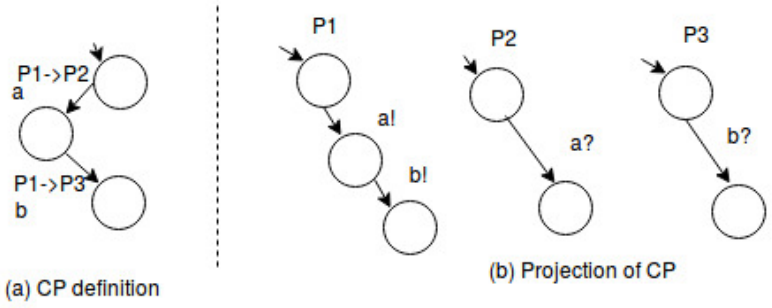


FIGURE *4.5 – Conservation de l'ordre d'envoi de messages*

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 *Les Figures 4.4 et 4.5 montrent des CP satisfaisant de la condition ISeqF par le CP réalisable. La communication des entités projetées indiquées (partie droite des Figures 4.4 et 4.5) suit la même séquence que celle requise par le CP (partie gauche des Figures 4.4 et 4.5). En d'autres termes, la vérification du CP suivant les étapes présentées en Section 2 confirme que la réalisabilité est assurée pour cet exemple. Ainsi, ici, la réalisabilité implique la Condition 2.*

Example 3 (Contre-Exemple) *La Figure 4.6 montre un CP non réalisable qui ne satisfait pas la condition ISeqF. L'ensemble des entités (montrées sur la Figure 3(a)) obtenues par projection du CP (partie gauche de la Figure 4.4) ne réalise pas le CP (Figure 4.4 (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. En effet, le CP viole la propriété ISeqF et cela ne permet pas la vérification de la réalisabilité.*

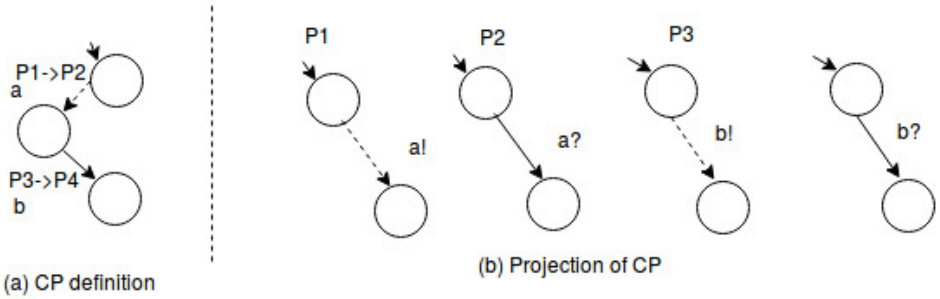


FIGURE *4.6 – CP non réalisable violation de la condition ISeqF*

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

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

PCF renforce la propriété DC des protocoles de conversation. Un *CP* peut être composé d'un ensemble de CP_b de base en utilisant l'opérateur de choix, *i.e.* $\otimes_{(+,s^f_{CP})}(CP, \{CP_{bi}\})$, si et seulement si toutes les entités d'envois sont identiques sur les transitions sortantes de l'état de branchement s^f_{CP}.

Example 4 *La Figure 4.7 illustre un CP réalisable où PCF est respecté. On peut appliquer les étapes de vérification de la Section 2 et conclure que ce CP est réalisable. D'autre part, le même CP peut être construit en utilisant notre opérateur de choix en maintenant PCF.*

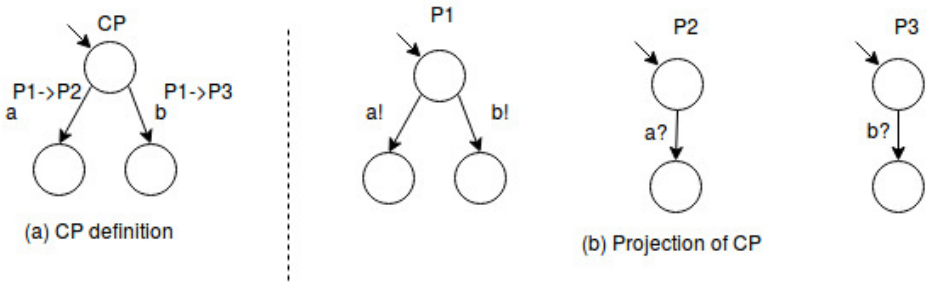


FIGURE *4.7 – CP réalisable satisfaisant PCF*

Example 5 (Contre-exemple) *La Figure 4.8 montre un contre exemple où les transitions de branchement ne respectent pas la propriété PCF dans l'état initial, i.e. deux entités différentes peuvent envoyer des messages à un même état du CP. Les entités montrées sur la partie droite de la Figure 4.8 sont obtenues par projection à partir du CP sur la partie gauche de la Figure 4.8. Cependant, ces entités ne réalisent pas leurs CP car elles peuvent toutes entrelacer les messages* a ! *et* b ! *lors d'une communication asynchrone. Il est donc possible de déclencher soit* a ! *en premier et ensuite* b !*, soit* b ! *en premier suivi de* a !*. CP n'autorise pas ces comportements entrelacés, de tel sorte que la réalisabilité soit assurée. Nous remarquons ici que la violation de PCF implique la non réalisabilité.*

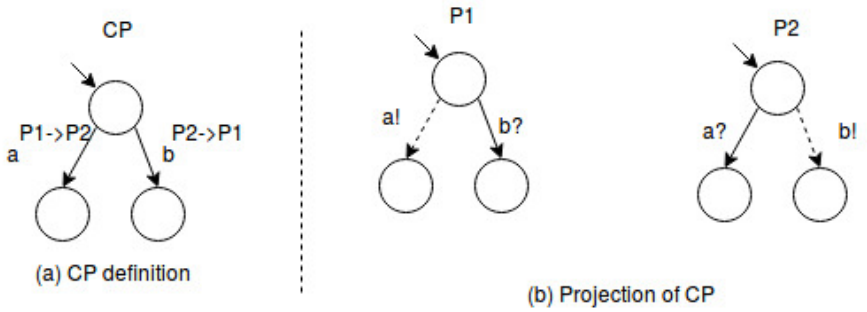


FIGURE *4.8 – CP non-réalisable PCF violé*

Les exemples 1, 3 et 5 donnés ci-dessus illustrent les problèmes d'interaction à éviter pour préserver la réalisation du système. Ces problèmes peuvent être évités 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 des messages d'envoi-réception appliqués par la séquence est toujours valable. La Figure 4.9 illustre cette situation.

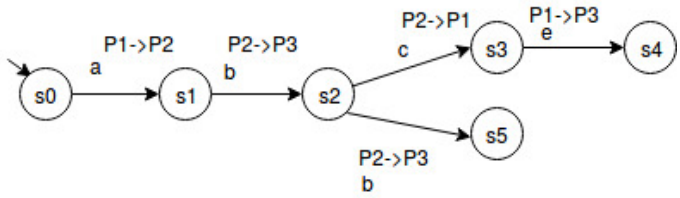


FIGURE *4.9 – Exemple illustratif d'un protocole de conversation avec séquence et choix satisfaisant ISeqF et PCF.*

3 Réalisabilité-par-construction : théorèmes et preuves

Dans cette section, nous prouvons mathématiquement la correction de notre proposition relative à la composition incrémentale des *CP* réalisables-par-construction.

La Table 4.1 présente les théorèmes garantissant la réalisabilité par la construction incrémentale de *CP*. Elle utilise les opérateurs de composition que nous avons introduits. 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.

Dans la Section 3.1, nous donnons le schéma de preuve. Le modèle générique de preuve est détaillé dans la Section 3.2. Il est instancié dans la Section 3.3 pour chaque théorème, de : composition séquentielle, en branche et en cycle. La Section 3.4 montre un cas d'étude.

3.1 Schéma de preuve.

Afin de démontrer les théorèmes de la table 4.1 , nous avons défini notre stratégie de preuve comme 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 cette condition, il est nécessaire de prouver l'équivalence (Définition 9), la synchronisabilité (Définition 10) et la WF (Définition 11). Les définitions de l'équivalence, de la synchronisabilité et de la WF,

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 *4.1 – Théorèmes liés au CP réalisables-par-construction*

nécessitent le calcul de la projection $\downarrow CP_{op}$ (Définition 6) du CP. Cette décomposition conduit à une preuve par raffinement. Ce processus est reproduit pour la preuve des différents théorèmes de la Table 4.1.

— *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 [57] pour prédire l'ensemble arbitraire de messages échangés. Cette variable définit un variant (cardinal de cet ensemble) qui se décroît 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 de base 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 $s_{CP} \in S_{CP}$, respectivement, un CP de base réalisable, un CP réalisable et un état de collage. Nous devons prouver que :

$$CP_{op} = \otimes_{(op, s_{CP})}(CP, CP_b) \in R \text{ est vrai}$$

lorsque la condition suffisante définie Op_{cond} correspondant aux Conditions 4, 3 et 2 de la Section 2 et associées à chacun des opérateurs Op est satisfaite.

3.2 Schéma de preuve.

À partir du schéma de la stratégie de preuve présenté précédemment, nous définissons le schéma de preuve de l'équation 1.

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

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

Ce schéma sera mis en oeuvre pour chaque opérateur de composition

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 (associés à CP_b basique, séquence, choix et itération) sont définis et prouvés.

Théorème 1 (Basic CP).
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.

Théorème 2 (Séquence).
Soit $CP = <S_{CP}, s^0_{CP}, L_{CP}, T_{CP}>$, un protocole de conversation CP_b et $s^f_{CP} \in S^f_{CP}$ un état final de CP.

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 schéma de preuve de l'équation 1 introduit en Section 3.2 et décrit en 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 d'après 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, (*i.e.* , à la fois l'état final de CP et l'état initial de CP_b).

Suivant 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

— $CP \equiv \ Sys_{sync}(\downarrow CP)$

— $CP_b \equiv Sys_{sync}(\downarrow CP_b)$.

Supposons que la condition suffisante pour la séquence ISeqF est vérifiée, *i.e.* $CP_{\gg} \in$ ISeqF. Nous devons prouver que $CP_{\gg} \equiv Sys_{sync}(\downarrow CP_{\gg})$ (l'équation (2.a)).

Considérons :

— une 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 protocole 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.

À partir de 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 se présentent.

1. **Soit** $\mathcal{P}_k = \mathcal{P}_t$, alors on obtient les traces suivantes pour les entités $\mathcal{P}_k = \mathcal{P}_t$, $\mathcal{P}_q$ et $\mathcal{P}_z$

— $\{\ldots, s_n^k \xrightarrow{m'!} s_{n+1}^k, s_{n+1}^k \xrightarrow{m''!} s_{n+2}^k\} \subseteq Tr_k$ ou s_{n+1}^k est l'état de collage et s_{n+2}^k est un nouvel état final de $\mathcal{P}_k$
— $\{\ldots, s_n^q \xrightarrow{m'?} s_{n+1}^q\} \subseteq Tr_q$
— $\{\ldots, s_{n+1}^z \xrightarrow{m''?} s_{n+2}^z\} \subseteq Tr_z$ ou s_{n+1}^z est l'état de collage, s_{n+2}^z est un nouvel état final du $\mathcal{P}_z$,

2. **ou** $\mathcal{P}_q = \mathcal{P}_t$, on obtient les traces suivantes pour les entités $\mathcal{P}_q = \mathcal{P}_t$, $\mathcal{P}_k$ et $\mathcal{P}_z$
— $\{\ldots, s_n^k \xrightarrow{m'!} s_{n+1}^k\} \subseteq Tr_k$
— $\{\ldots, s_n^q \xrightarrow{m'?} s_{n+1}^q, s_{n+1}^q \xrightarrow{m''!} s_{n+2}^q\} \subseteq Tr_q$ ou s_{n+1}^q est l'état de collage et s_{n+2}^q est le nouvel état final $\mathcal{P}_q$
— $\{\ldots, s_{n+1}^z \xrightarrow{m''?} s_{n+2}^z\} \subseteq Tr_z$ ou s_{n+1}^z est l'état de collage s_{n+2}^z 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 de CP_b est utilisée 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 complètes. La transition envoi/réception de CP_b devient alors 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.
Considérons :

— une 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

À partir des 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 $Sys_{async}(\downarrow CP_{\gg})$. 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. À partir des hypothèses de récurrence, nous avons
— $Sys_{async}(\downarrow CP) \in WF$ et
— $Sys_{async}(\downarrow CP_b) \in WF$.
Nous devons prouver que $Sys_{async}(\downarrow CP_{\gg}) \in WF$ (équation 2.c). Cela signifie que, par hypothèse, les files d'attente à l'état final de $Sys_{async}(\downarrow CP)$ sont vides puisqu'il est réalisable donc bien formé (WF). Nous devons montrer que la file d'attente est toujours vide après l'échange du message effectué par CP_b. Considérons :

— une 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 $T_{CP_b} = \{s_{b0} \xrightarrow{m''^{\mathcal{P}_t \to \mathcal{P}_z}} s_{b1}\}$ dans le CP_b réalisable

Puisque la condition ISeqF est vérifiée *i.e.* $\otimes_{(\gg, s^f_{CP})}(CP, CP_b) \in$ ISeqF, deux cas se présentent.

1. **Soit** $\mathcal{P}_k = \mathcal{P}_t$, alors on obtient les traces suivantes 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 complètes. A l'état s^k_{n+1}, l'entité $\mathcal{P}_k$ envoie le message m" et ses traces sont complètes. La file d'attente de l'entité $\mathcal{P}_z$ contient le message m" (transition de $\mathcal{P}_k$ à l'état s^k_{n+2}). À 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 file d'attente. Donc, toutes les files sont vides à l'état $n+2$ et toutes les traces sont complètes.
2. **ou** $\mathcal{P}_q = \mathcal{P}_t$, on obtient les traces suivantes 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 que précédemment 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 complètes. 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 file de l'entité $\mathcal{P}_z$ contient le message m" (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 file. Donc, toutes les files sont vides à l'état $n+2$ et toutes les traces seront complétes.

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

Théorème 3 (Branchement).
Soient $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$, $\{CP_{bi} \mid 2 \leq i \leq n\}$ *un ensemble de* n $(n \geq 2)$ *protocoles de conversation de base et* $s^f_{CP} \in S^f_{CP}$ *un état final dans* CP. *La propriété suivante est vraie.*

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

Preuve 3.
La preuve suit le schéma de preuve de l'équation 1 présenté en Section 3.2 et décrit en section 3.1.

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

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

D'après le 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. Soient $CP = \varnothing$ et un ensemble de protocoles de conversation de base $\{CP_{bi}\},\ i \geq 2, i \in \mathbb{N}$ 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 (condition DC).

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 seule entité émettrice suivant la condition PCF.

Considérons la trace $\{s_{b0} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s_{bi}\} \in Tr_{CP_+}$ de CP_+.
Les traces d'échanges 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. Soient $CP \in R$ est un protocole de conversation,
$\{CP_{bi} \in R,\ i \geq 2, i \in \mathbb{N}\}$ un ensemble de CP de base et $s_{CP} \in S^f_{CP}$ l'état de collage, (*i.e.* , à la fois l'état final de CP et l'état initial de chacun des $CP_b \in \{CP_{bi},\ i \geq 2, i \in \mathbb{N}\}$
D'après 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 avons :
— $CP \equiv Sys_{sync}(\downarrow CP)$ et
— $\{CP_{bi}\} \equiv Sys_{sync}(\downarrow \{CP_{bi}\})$.
Supposons que les conditions suffisantes $CP_+ \in$ ISeqF et $CP_+ \in$ PCF sont satisfaites. Nous devons prouver que $CP_+ \equiv Sys_{sync}(\downarrow CP_+)$ (équation 3.a).
Considérons :
— une 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 une 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. **Soit** $\mathcal{P}_k = \mathcal{P}_t$, alors on obtient les traces suivantes pour les entités $\mathcal{P}_k = \mathcal{P}_t$, $\mathcal{P}_q$ et $\mathcal{P}_z$
   — $\{\ldots, s^k_n \xrightarrow{m'!} s^k_{n+1}, s^k_{n+1} \xrightarrow{m''!} s^k_{n+2}\} \subseteq T_k$
   — $\{\ldots, s^q_n \xrightarrow{m'?} s^q_{n+1}\} \subseteq T_q$
   — $\{\ldots, s^z_n \xrightarrow{m''?} s^z_{n+1}\} \subseteq T_z$.

2. **ou** $\mathcal{P}_q = \mathcal{P}_t$, alors on obtient les traces suivantes pour les entités $\mathcal{P}_q = \mathcal{P}_t$, $\mathcal{P}_k$ et $\mathcal{P}_z$
   — $\{\ldots, s^k_n \xrightarrow{m'!} s^k_{n+1}\} \subseteq T_k$
   — $\{\ldots, s^q_n \xrightarrow{m'?} s^q_{n+1}, s^q_{n+1} \xrightarrow{m''!} s^q_{n+2}\} \subseteq T_q$
   — $\{\ldots, s^z_n \xrightarrow{m''?} s^z_{n+1}\} \subseteq T_z$

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 utilisée par la transition finale du CP réalisable. De plus, elle est toujours franchie une fois que les transitions envoi-réception du $Sys_{sync}(\downarrow CP_+)$ du CP réalisable sont franchies. 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.

Considérons :

— une 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 une 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 dans l'ensemble $\{CP_{bi}\}$, $i >= 2$, $i \in \mathbb{N}$

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 $Sys_{async}(\downarrow CP_+)$

— à partir de la condition *PCF*, pour chaque trace, les entités émettrices des dernières transitions du $\downarrow CP_+$ sont identiques. La dernière entité émettrice est $\mathcal{P}_z$

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

Ainsi, la dernière transition des traces de $\downarrow CP_+$ est l'une des branches correspondante à une transition de l'un des protocoles de base introduit par l'opérateur de choix, à l'état $s_{b0} = s^f_{CP}$. Cette dernière transition 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 $Sys_{async}(\downarrow CP_+)$, où, la file d'attente $\mathcal{P}_z$ contient le message m'' et est 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.

À partir des hypothèses de récurrence, nous prouvons $Sys_{async}(\downarrow CP) \in WF$, $Sys_{async}(\downarrow \{CP_{bi}\}) \in WF$. Nous devons prouver que $Sys_{async}(\downarrow CP_+) \in WF$ (équation (3.c)). Ainsi, par hypothèse, les files d'attente sont vides dans l'état final de $Sys_{async}(\downarrow CP)$ puisque CP 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 :

— une 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 réali-

sable
— et une 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 $\{CP_{bi}\}$.

Par hypothèse, les conditions *ISeqF* et *PCF* sont satisfaites
i.e. $\otimes_{(+,s^f_{CP})}(CP, \{CP_{bi}\}) \in$ *ISeqF* et $\otimes_{(+,s^f_{CP})}(CP, \{CP_{bi}\}) \in$ *PCF*.
Par les propriétés de 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 $\{CP_{bi}\}$.
Par 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 les traces T_{CP} et T_{CP_b} données 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 de choix défini préserve la réalisabilité. □

Théorème 4 *(***Itération***)*
Soit $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *,* CP_b *un protocole de conversation de base 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 (cycle dans le *CP*) 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 traités.

Dans la suite, nous considérons le cas où l'opérateur de cycle introduit une boucle.
La preuve est inductive et suit le schéma de preuve de l'équation 1. Ce modèle est réécrit (instancié) pour l'opérateur de cycle dans l'équation 4 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 de base. Soit $CP = \varnothing$ et CP_b un CP de base. $CP_{\circlearrowleft} = \otimes_{(\circlearrowleft, s^0_{CP})}(\varnothing, CP_b)$ 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 [41] 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 $s^f_{CP} \in S^f_{CP}$ l'état de collage.
Par définition de l'opérateur de cycle, nous obtenons
— s^f_{CP} est à la fois l'état final de CP et l'état initial de CP_b, alors $s^f_{CP} = s_{b0}$, et
— une boucle (un cycle dans le CP) donc $s_{b1} \in S_{CP}$
Considérons :
— la propriété *ISeqF* est vraie pour $CP_{\circlearrowleft}$, c'est-à-dire que l'ordre des messages d'envoi-réception est garanti,
— ce CP obtenu défini un anneau orienté, et d'après [41], le protocole de conversation associé est réalisable.

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

3.4 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 entracte de [76] où le problème de réalisabilité des systèmes a été également étudié.

Exigences

L'étude de cas concerne un processus d'achat en ligne. Dans [76], les exigences suivantes sont spécifiées.

Un simple processus d'achat dans un environnement virtuel est considéré. Il comprend un client (B), un service de comptabilité (A) et un département logistique (L). Le service de comptabilité (A) approuve une commande (message Order) envoyée par le client (B) et la transmet au service logistique (L) (message Deliver) qui 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 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 distinguées : 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 par la Figure 7.1.

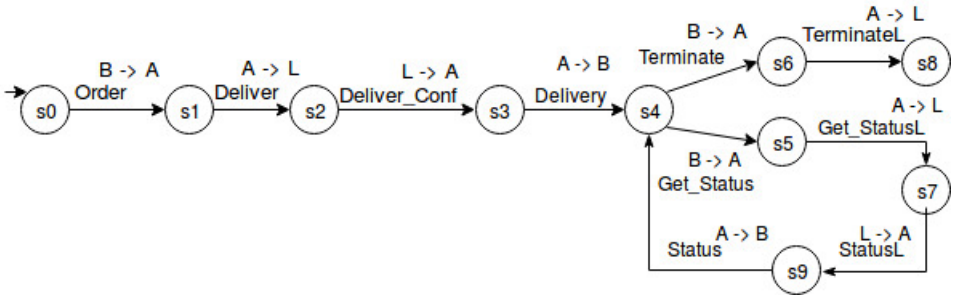


FIGURE *4.10 – Protocole de conversation d'un processus d'approvisionnement simple au sein d'une entreprise virtuelle*

Selon [76], 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), puis, l'entité service logistique répond par un message Deliver_ Conf (à l'état s_3). Lorsque l'entité service comptabilité (A) reçoit ce message, elle transmet au client via le message Delivery (état s_4). A ce niveau (état s_4), une alternative proposée. En effet, le service de comptabilité (A) peut recevoir un message Get_ status envoyé par le client (B) (état s_5), suivi 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. 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 entités atteignent leurs états finaux. Le protocole de conversation obtenu CP est formalisé avec 10 états, 7 messages échangés entre trois entités et 10 transitions. 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

Suivant la Définition 6, les entités communicantes sont obtenues à partir de *CP* par projection. La Figure 4.11 illustre le service de comptabilité (A), le client (B) et le service de logistique (L).

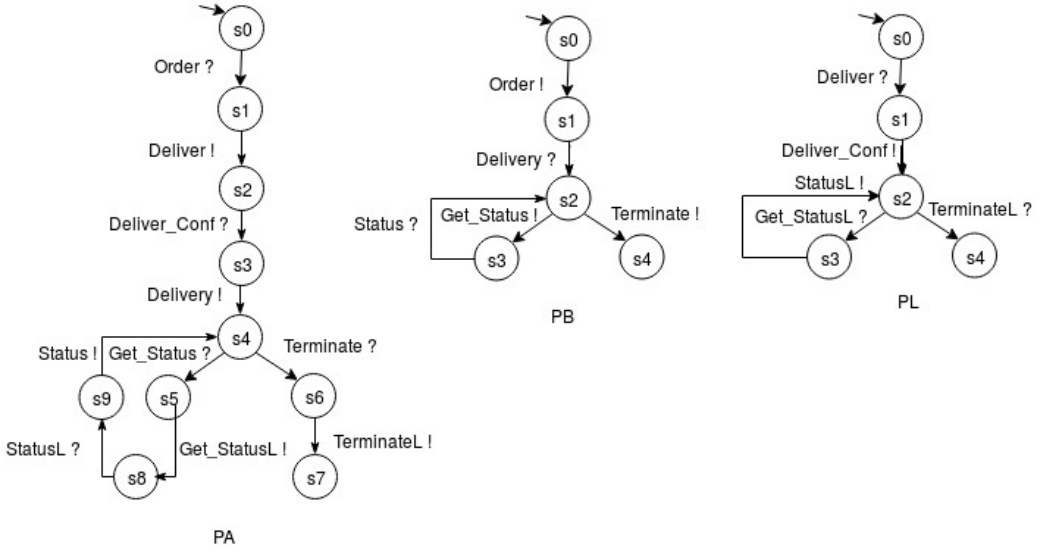


FIGURE *4.11 – Entités projetées d'un processus simple d'achat en ligne au sein d'une entreprise virtuelle*

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 4.10. En d'autres termes, le CP de la figure 4.10 est réalisable.

Dans ce qui suit, nous montrons, en utilisant l'approche que nous avons proposée, que le protocole de conversation de la figure 4.10 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 de base en vérifiant que les conditions suffisantes associées à chaque opérateur de composition sont satisfaites à chaque fois qu'un opérateur est appliqué. Une séquence d'étapes est configurée pour construire le protocole de conversation de la figure 4.10 comme suit.

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 protocoles de conversation de base CP_b

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

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

D'après le théorème 1, tous les protocoles de conversation de base précédents sont réalisables.

Étape 2. Initialisation du CP

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

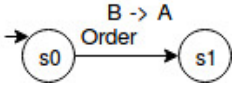
Étape 3. Application des opérateurs de composition

A ce niveau, les opérateurs de composition sont appliqués. Chaque fois qu'un opérateur est appliqué, les conditions suffisantes sont satisfaites. Un nouveau CP est obtenu à chaque application d'opérateur. La séquence d'applications d'opérateurs commence à partir du CP vide défini 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 par l'opérateur de séquence 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 obtenu CP_1 est définie comme suit.
   - $CP_1 = <$
     - $S_{CP_1} = \{s_0, s_1\}$,
     - $s^0_{CP_1} = \{s_0\}$,
     - $L_{CP_1} = \{Order\}$,
     - $T_{CP_1} = \{s_0 \xrightarrow{Order^{B\to A}} s_1\}$

     $>$

   CP_1 correspond au CP 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, i.e. l'entité réceptrice (A) de la dernière transition CP_1 est la même entité émettrice dans CP_{b1} avec le nouvel état final $s^f_{CP} = \{s_2\}$. Toujours d'après le théorème 2, le protocole de conversation réalisable CP_2 obtenu est défini comme suit.

— $CP_2 = <$
  — $S_{CP_2} = \{s_0, s_1, s_2\}$,
  — $s^0_{CP_2} = \{s_0\}$,
  — $L_{CP_2} = \{Order, Deliver\}$,
  — $T_{CP_2} = \{s_0 \xrightarrow{Order^{B\to A}} s_1, s_1 \xrightarrow{Deliver^{A\to L}} s_2\}$
  $>$

CP_2 correspond au CP 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 satisfaite pour cette composition, i.e. l'entité récéptrice (L) de la dernière transition du CP_2 est la même entité émettrice dans CP_{b2} avec le nouvel état final $s^f_{CP} = \{s_3\}$.

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

   CP_3 correspond au CP décrit ci-dessous.

4. $CP_4 = \otimes_{(\gg, s^3_{CP_3})}(CP_3, CP_{b3})$. CP_3 est composé de nouveau avec le protocole de conversation de base 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 satisfaite pour cette composition, i.e. l'entité réceptrice (A) de la dernière transition du CP_3 est la même entité émettrice du CP_{b3} avec $s^f_{CP} = \{s_4\}$.

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

>

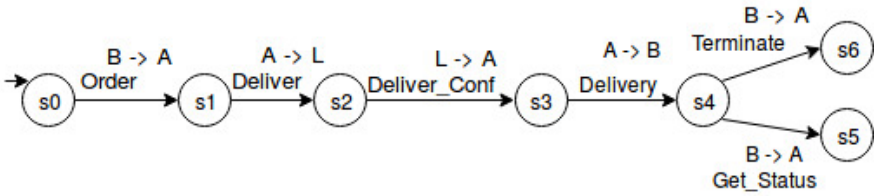
CP_4 correspond au CP décrit ci-dessous.

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. D'après 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}, i.e. l'entité réceptrice (B) de la dernière transition CP_4 est la même entité émettrice de CP_{b4} et de CP_{b5}. Puis, la condition PCF est satisfaite pour chaque transition en branche, i.e. 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.

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

     >

   CP_5 correspond au CP décrit ci-dessous.



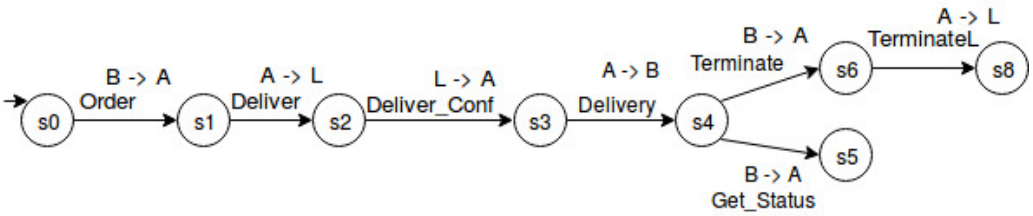
6. $CP_6 = \otimes_{(\gg,s^5_{CP_5})}(CP_5, CP_{b6})$. Nous suivons l'une des branches du CP obtenu CP_5. Le protocole de conversation CP_5 est composé avec le CP_{b6} de base en utilisant l'opérateur de séquence $\otimes_{(\gg,s^f_{CP})}$ à l'état final s_5 de CP_5. La condition suffisante ISeqF est satisfaite pour cette composition, i.e. que l'entité réceptrice (A) de la dernière transition CP_5 est identique à l'entité émettrice

du CP_{b6} avec les nouveaux états finaux $s^f_{CP} = \{s_6, s_7\}$.

D'après le théorème 2, le protocole de conversation CP_6 réalisable obtenu est défini comme suit.

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

CP_6 correspond au CP décrit ci-dessous.

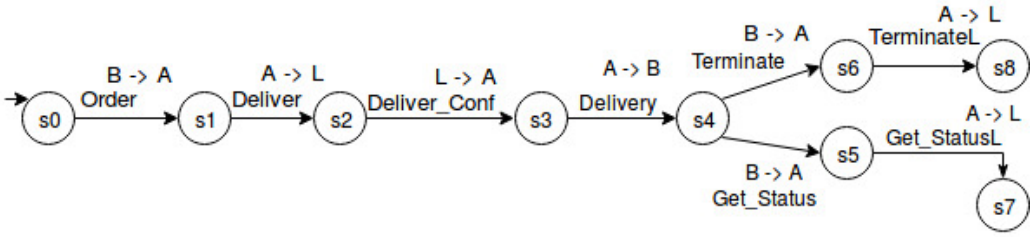


7. $CP_7 = \otimes_{(\gg, s^6_{CP_6})}(CP_6, CP_{b7})$. Ici, nous suivons et complétons l'autre branche de l'alternative. Le protocole de conversation CP_6 est composé avec le protocole de conversation de base CP_{b7} à l'aide de l'opérateur de séquence $\otimes_{(\gg, s^f_{CP})}$ à l'état final s_6 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 $s^f_{CP} = \{s_7, s_8\}$.

D'après le théorème 2, le CP_7 réalisable obtenu est défini comme suit.

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

CP_7 correspond au CP est décrit ci-dessous.

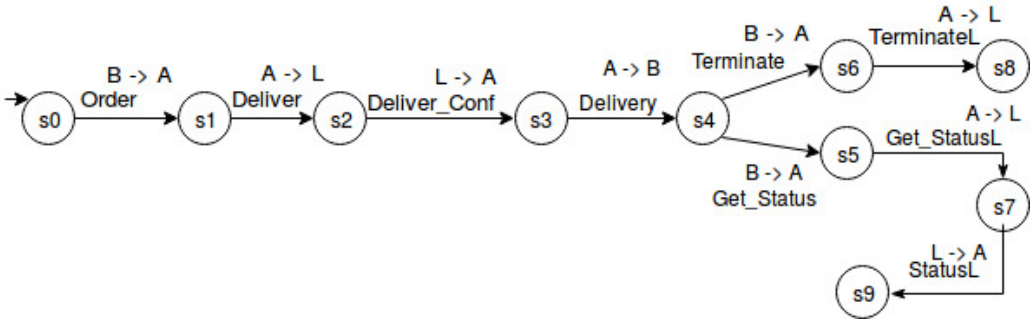


8. $CP_8 = \otimes_{(\gg, s^7_{CP_7})}(CP_7, CP_{b8})$. De retour sur l'autre branche, CP_7 est composé avec CP_{b8} à l'aide de l'opérateur de séquence $\otimes_{(\gg, s^f_{CP})}$ à l'état final s_7 de CP_6. La condition suffisante ISeqF est satisfaite pour cette composition, i.e. que l'entité réceptrice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

   D'après le théorème 2, le CP réalisable obtenu est défini comme suit.

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

   CP_8 correspond au CP décrit ci-dessous.

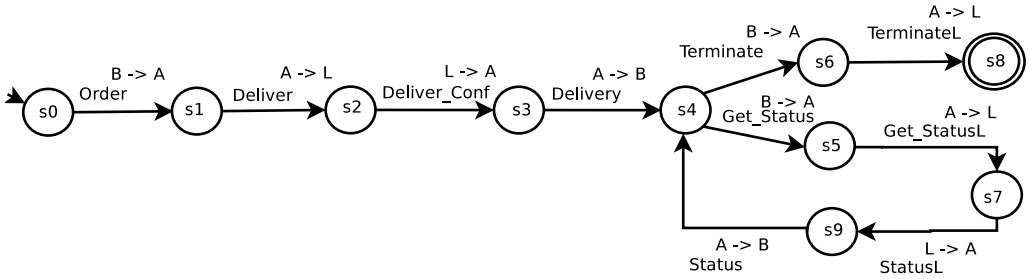


9. $CP_9 = \otimes_{(\circlearrowleft, s^9_{CP_8})}(CP_8, CP_{b9})$. Enfin, nous introduisons la boucle ou l'itération en ajoutant un cycle dans CP_8 obtenu précédemment. CP_8 est composé à l'aide de l'opérateur d'itération $\otimes_{(\circlearrowleft, s^f_{CP})}$ avec CP_{b9} à l'état final s_9 de CP_8.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

   D'après 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 4.10.



Chapitre 5

Formalisation : preuve formelle avec B-événementiel

Sommaire

1 Stratégie de raffinement avec B-événementiel 70
2 Cadre mathématique pour la composition 71
2.1 Variables et états de systèmes de composition 71
2.2 Systèmes 72
2.3 Initialisation et composition 73
2.4 Conditions suffisantes 73
2.5 Propriété de réalisabilité 73
3 Cadre général de formalisation et preuve 74
4 Un modèle B-événementiel pour la composition du système 75
4.1 Partie statique : définitions requises 76
4.2 Partie dynamique : modélisation du comportement des CP 78
4.3 Une approche évolutive de construction des CP réalisables 84
5 Instanciation du modèle B-événementiel par raffinement 84
5.1 Étape 1. Instanciation du contexte 84
5.2 Étape 2. Raffinement et témoins pour l'instanciation . . . 85
6 Application à un cas d'étude 85
7 Animation avec ProB 88
8 Évaluation 89
8.1 Les preuves 89
8.2 Méthodes formelles correctes-par-construction 91

Ce chapitre propose un modèle formel de composition incrémentale des systèmes réalisables développés prouvés et validés à l'aide de raffinement. B-événementiel est utilisé pour formaliser l'ensemble du développement formel des différents opérateurs de composition de *CP* réalisables. 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, i.e. entités communicantes, ni du type de composition.

1 Stratégie de raffinement avec B-événementiel

Cette section présente une description des modèles formels avec B-événementiel des opérateurs définis dans la Section 1. Un événement B-événementiel étant associé à chaque opérateur, permet de créer des *CP*s réalisables-par-construction. Chaque événement (*initialisation*, *Sequence*, *Choix* et *Cycle*) est gardé par les conditions suffisantes identifiées dans le Chapitre 4 Section2.
L'opérateur de raffinement offert par la méthode B-événementiel s'est avéré efficace pour gérer les preuves complexes associées à chaque opérateur. Le raffinement a permis de prouver 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 un *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 *CP* construits satisfont l'invariant du DC (Condition 1). Ce modèle déclare également la variable de prophétie comme variable d'état. Cette dernière définit un nombre arbitraire de messages échangés et est utilisée pour définir un variant 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. Un invariant de collage liant le *CP* à la projection synchrone est introduit. La propriété d'équivalence est prouvée à ce niveau. Elle est définie 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 définir une communication asynchrone. Les projections synchrones et asynchrones sont liées par un autre invariant de collage. Des actions d'envoi et de réception, ainsi que des actions de traitement des files d'attente et un variant décroissant de la variable de prophétie sont introduites. Elles sont nécessaires pour prouver la synchronisabilité et la WF exprimées par des invariants. Le raffinement par un modèle synchrone puis un modèle asynchrone facilite le processus de preuve.

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

2 Cadre mathématique pour la composition

La formalisation mathématique permettant de gérer la composition de *CP* est donnée 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.

2.1 Variables et états de systèmes de composition

Les trois variables d'état $Built\_CP$, $Built\_Synchrone$ et $Built\_Asynchrone$ représentent les états de définition des trois mode de communication d'un *CP* en cours de construction : *CP*, Sys_{sync} et Sys_{async}, respectivement. Initialisation, les trois protocoles de conversation sont vides comme suit :

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

La définition de ces derniers est construite à partir de l'ensemble des CP_b de base pour construire le *CP* réalisable par construction incrémentale, dénotée par CPs_B. L'ensemble des CP_bs de base synchrones pour construire le Sys_{sync} est dénoté par CPs_SYNC_B. Enfin, l'ensemble des CP_bs de base asynchrones pour construire le Sys_{async} est dénoté par CPs_ASYNC_B.

Les transitions de *CP* de base sont définies par les ensembles suivants :

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

où
— CP_STATES représente 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 dans
— $MESSAGES$ entre deux entités émettrice et réceptrice, dans
— $PEERS$ et d'un index dans l'ensemble des entiers naturels $\mathbb{N}$.

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

Les transitions des *CP* de base formant le *CP* synchrone Sys_{async} diffèrent de celles du *CP*, avec une étiquette d'échange de message découpée en deux parties : une première partie décrivant le message envoyé avec l'ensemble $ACTIONS$ instancié par l'action d'envoi ! et une seconde partie liée au message reçu instancié par l'action de réception ?. L'ensemble $ACTIONS$ définit aussi les transitions internes des entités communicantes, interprété par une transition vide au niveau *CP* dénotée par τ.

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

L'ensemble CPs_ASYNC_B comporte les transitions de base 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 entier naturel jouant le rôle d'index.

2.2 Systèmes

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

$$Système = \langle variables, variants, invariants, initialisation, operator \rangle$$

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 une file d'attente FIFO dénotée par $queue$. A_Traces représente les traces asynchrone de Sys_{async} comme suit :

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

— *invariants* 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 propriété 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 de système bien formé WF.

— *variant* définit un entier naturel définissant un variant à partir de valeurs variables :

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

$Prophecy\_of\_Sent\_Messages$ est une variable initialisée au nombre anticipé (prophécie) de messages echangés entre les entités communicantes, et $Number\_of\_send$ est le nombre de 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.

2.3 Initialisation et composition

L'initialisation du système global consiste à associer l'ensemble vide aux variables $Built\_CP$, $Built\_Synchrone$ et $Buit\_Asynchrone$. Les événements de composition construisent les CP, Sys_{sync} et Sys_{async} incrémentalement, de sorte que les trois systèmes satisfassent les invariants.

2.4 Conditions suffisantes

L'ensemble des conditions suffisantes identifiées en section 2 chapitre 4 et préservant la propriété de réalisabilité est formalisé comme suit :

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

tel que : 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 définie par la fonction totale PCF.

2.5 Propriété de réalisabilité

La propriété de réalisabilité à garantir à chaque étape de composition de systèmes est formalisée comme suit :

$$Réalisabilité \equiv Equivalence \wedge Synchronisability \wedge WF$$

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

La fonction de bijection "*Equivalence*" (one-to-one and onto relations) caractérise la relation d'équivalence entre un protocole de conversation abstrait *CP* identifié dans l'ensemble "CPs_B" et le protocole de conversation synchrone Sys_{sync} définie par "CPs_SYNC_B".

Le même raisonnement s'applique pour la caractérisation de la synchronisabilité entre la composition synchrone du protocole de conversation Sys_{sync} est définie par l'ensemble "CPs_SYNC_B" et sa composition asynchrone Sys_{async} représenté par l'ensemble "CPs_ASYNC_B".

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

3 Cadre général de formalisation et preuve

Notre proposition doit être appuyée par des théorèmes et preuves mathématiques pour chaque type de composition.

La Figure **??** illustre la stratégie de raffinement que nous avons mise en place afin de formaliser et de vérifier en B-événementiel notre modèle de composition de *CP* réalisables. Le modèle est composé de trois niveaux : 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 par des contextes B-événementiel. Afin de réduire la complexité des preuves, nous avons choisi de décomposer cette partie statique en trois sous parties, une dédiée à la définition des éléments relatifs au niveau *CP*, une deuxième consacrée à la définition des éléments du niveau synchrone et une dernière dans laquelle nous présentons les éléments nécessaires à la définition du niveau asynchrone. Les trois contextes sont liés par la relation "extends" d'extension. Nous exprimons les interactions du système (partie comportementale) au niveau des machines B-événementiel.

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

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

Le raffinement est également utilisé afin d'introduire de nouveaux comportements au niveau des modèles raffinés ce qui permet d'enrichir la description du modèle raffiné.

L'intérêt du raffinement est la décomposition du modèle en plusieurs niveaux est de permettre la bonne maîtrise du développement et de la complexité des preuves. Les propriétés de chaque niveau sont préservées dans la hiérarchie de raffinement grâce au maintien des invariants de collage.

Notre modèle B-événementiel se compose de trois contextes

— "*Lts_Context*",
— "*Lts_Sync_Context*" et

— "*Lts_Async_Context*"

aperçus respectivement par trois machines

— "*Lts_Model*",
— "*Lts_Synchronous_Model*" et
— "*Lts_Asynchronous_M-odel*".

La première machine "*Lts_Model*" modélise le niveau abstrait du système. La machine est raffinée ensuite par la deuxième machine "*Lts_Synchronous_Model*" qui modélise le système en mode de communication synchrone. Ensuite la machine "*Lts_Synchronous_Model*" est à son tour raffinée par une troisième machine "*Lts_Asynchronous_Model*" qui modélise le mode de communication asynchrone du système.

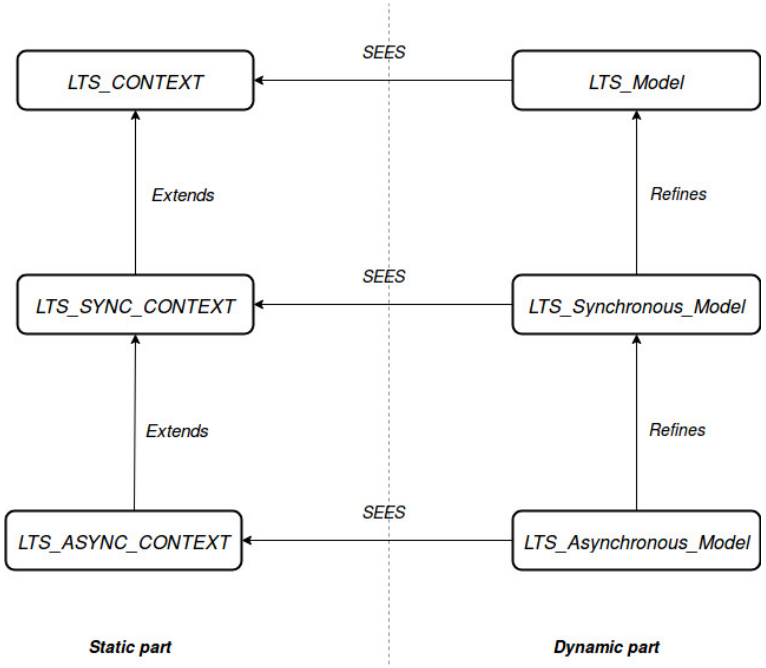


FIGURE *5.1 – Modèle en B-événementiel.*

4 Un modèle B-événementiel pour la composition du système

Le paramètre mathématique décrit en Section 2 a été complètement formalisé en B-événementiel. Le développement complet du modèle B est disponible en 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 protocole de conversation.

Le protocole de conversation 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 extension *Lts_Sync_Context* et la seconde extension *Lts_Async_Context*. Et de trois machines *Lts_Model*, son extension *Lts_Synchronous_Model* et la troisième extension *Lts_Asynchronous_Model*.

4.1 Partie statique : définitions requises

Le contexte racine. Il décrit la notion de *CP* et introduit la définition de chaque opérateur au niveau du *CP*.

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 5.1 – Un extrait du LTS_ CONTEXT.

Propriétés requises pour les CP (Listing 5.1). Le listing 5.1 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 *CP* de base, etc.

Un ensemble d'axiomes est utilisé pour définir les propriétés pertinentes de ces définitions. L'axiome $axm1$ définit un *CP* 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 complémentation dans l'axiome $axm5\_Cond1$. La définition de ISeqF est donnée par les axiomes $axm6\_Cond2$ et $axm7\_Cond2$.

L'entité source $P\_SRC(cp\_b)$ est comparée à 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, nous comparons dans les axiomes $axm8\_Cond3$ et $axm9\_Cond3$, l'entité expéditrice $P\_SRC(Trans)$ des transitions formant un branchement. De sorte que, les entités émettrices de toutes les branches doivent être identique.

La première extension : 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, les définitions de propriétés relatives aux échanges synchrones sont introduites, notamment l'équivalence ($\equiv$), notée $EQUIV$.

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

Listing 5.2 – Un extrait du LTS_SYNC_CONTEXT.

Propriétés requises pour la projection synchrone (Listing 5.2). La définition du système de transitions correspondant à la projection synchrone est donnée par l'ensemble CP_SYNC_B défini par l'axiome $axm1$ du Listing 5.2. Les actions (send ! et receive ?) sont introduites. Ensuite, deux autres axiomes, $axm1.a$ et $axm1.a1$ définissent la relation d'équivalence entre un CP et sa projection synchrone Sys_{sync}. La relation $EQUIV$ est ensuite introduite. Elle caractérise l'ensemble des CP équivalents à leur projection synchrone. L'axiom $axm1.a1$ formalise la définition 9 de la section 4.

Seconde extension : modèle asynchrone. Le second raffinement introduit la projection asynchrone avec les actions des entités émettrices et réceptrices. Les propriétés de système bien formé WF et de synchronisabilité restent à prouver pour compléter la preuve de la réalisabilité.

La projection asynchrone (Listing 5.3). La propriété de synchronisabilité ($Sync(Sys_{sync}, Sys_{async})$), de la Définition 10 est exprimée sous forme d'axiomes $axm\_1.b$ et $axm\_1.b1$, et la propriété de WF (WF (Sys_{async})) de la Définition 11 est exprimée sous forme d'axiomes $axm\_1.c$ et $axm\_1.c1$. Ces axiomes sont introduits

dans le contexte du Listing 5.3. Ces deux propriétés permettent d'effectuer la preuve de réalisabilité.

Context *Lts_Async_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 5.3 – Un extrait du LTS_ASYNC_CONTEXT.

4.2 Partie dynamique : modélisation du comportement des CP

4.2.1 La machine racine. Le modèle associé à cette machine correspond à la définition du *CP* par composition des opérateurs définis. Chaque opérateur correspond à un événement et contribue à la construction d'un *CP* donné représenté par la variable d'état $BUILT\_CP$ qui définit uniquement un *CP* déterministe (voir invariant $inv1$ du Listing 5.4).

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

Listing 5.4 – 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 gardé par les conditions suffisantes définies. Ces dernières garantissent qu'un opérateur n'est appliqué que si ces conditions sont satisfaites.

— **Initialisation.** Cet événement définit un $BUILT\_CP$ vide (dans l'action act1 du Listing 5.5) comme *CP* initial.

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

Listing 5.5 – Un extrait de l'événement d'initialisation.

— **Opérateur Add_Sequence.** L'événement *Add_Sequence* du Listing 5.6 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 cours de construction (en utilisant l'opération union dans l'action *act*1). Il configure également le nouvel état final par l'action *act*2. Cet événement est déclenché uniquement si les conditions pertinentes sont respectées. En d'autre termes, il est clairement indiqué, dans la condition *grd*3, que la propriété de séquence indépendante ISeqF doit être satisfaite 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 5.6 – Un extrait de l'opérateur de composition de séquence.

— **Opérateur Add_Choice.** L'événement *Add_Choice* du Listing 5.7 modélise l'opérateur de choix (Définition 14). Il complète le *CP* en cours avec 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* doit avoir le même état source et la même entité émettrice et 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.

— **Opérateur Add_Loop.** L'événement *Add_Loop* du Listing 5.8 formalise l'opérateur de boucle (Définition 15) en introduisant dans le *CP* en cours de construction une transition de boucle (boucle auto ou boucle de cycle) donnée en paramètre, à savoir *Some_cp_b* dans l'action *act*1. De plus, la propriété ISeqF dans *grd*2 et *grd*3 sont ajoutées pour assurer ces conditions associées à l'itération. Notons que *grd*4 fait la différence entre l'événement *Add_Loop* et l'événement *Add_Sequence* en imposant que les états source et destination de la transition à ajouter *Some_cp_b* sont deux états du *CP* déjà construit.

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 5.7 – Un extrait de l'opérateur de composition de choix.

Event $Add\_Loop \triangleq$
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 5.8 – Un extrait de l'opérateur de composition de boucle.

- Variant et événement Add_End. Un nouvel événement de terminaison *Add_End* est introduit au niveau abstrait. *Add_End* ajoute une nouvelle transition échangeant le message *End* entre deux entités communicantes dans l'action *act*1 du Listing 5.10. L'évènement est déclenché une seule fois à la fin de la composition incrémentale, quand le variant exprimé dans le Listing 5.9 devient nul, où la variable $Prophecy\_of\_Sent\_Messages$ est la variable de prophétie exprimant le nombre d'envoi souhaités et la variable $Number\_of\_send$ représente le nombre de messages envoyés. Il garantit que l'ensemble des messages a été traité.

Variant

$Prophecy\_of\_Sent\_Messages - Number\_of\_send$

End

Listing 5.9 – Variant du modéle LTS_ model.

Notons qu'un tel événement est introduit afin de déterminer la fin de la composition, le moment où la propriété WF doit être vérifiée. La dernière tran-

sition ajoutée est conservée dans une variable *Last_cp_trans* dans l'action *act*2 utilisée pour écrire l'invariant 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 5.10 – Un extrait de l'événement End.

4.2.2 La projection synchrone (cf. Listing 5.11). Le premier raffinement introduit la projection synchrone de $BUILT\_CP$ définie par la variable $BUILT\_SYNC$ du Listing 5.11. L'événement *Add_Sequence* ou l'opérateur de séquence (Listing 5.11) raffine le même événement du modèle racine. Il introduit l'ensemble $BUILT\_SYNC$ correspondant à la projection synchrone, comme indiqué dans Définition 7. De plus, *Add_Sequence* ne s'applique que si les conditions de réalisation de la séquence sont vraies. 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 ∈ cps_sync_b
    grd3 : S_SOURCE_STATE(Some_cp_sync_b) ∈ CP_Final_states grd4 : S_Some_cp_b ∈ ISeqF
    grd8 : MESSAGE(S_Some_cp_b) ≠ 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 ∪ {S_Some_cp_b}
    act2 : BUILT_SYNC := BUILT_SYNC ∪ {Some_cp_sync_b}
    ...
End
```

Listing 5.11 – Un extrait de l'opérateur de séquence du modéle LTS_Synchronous_model.

Le même principe s'applique sur les autres opérateurs de composition utilisés lors de la composition synchrone du protocole de conversation. Le raffinement synchrone des opérateurs *Add_Choice*, *Add_Loop* et *Add_End* est détaillé dans l'annexe A (partie *'Premier raffinement'*).

Invariant d'équivalence. Le Listing 5.12 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 5.12 décrit la propriété d'équivalence à l'aide de la relation $EQUIV$ définie dans le contexte synchrone (voir Listing 5.2). Ainsi, une partie de la propriété de réalisabilité (c'est-à-dire $CP \equiv Sys_{sync}$) de la Définition 9 est

prouvée à ce niveau de raffinement.

4.2.3 La projection asynchrone (cf. Listing 5.14). Le second raffinement introduit la projection asynchrone avec les actions (d'envoi et de réception) entre les entités émettrices et réceptrices. Les propriétés WF et synchronisabilité restent à prouver pour compléter la preuve de réalisabilité.

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

Listing 5.12 – Un extrait des invariants du modéle LTS_ Synchronous_ model.

Invariants de synchronisabilité et de WF. Les invariants associés à ce modèle sont présentés dans le Listing 6.4. En particulier, les propriétés de synchronisabilité, exprimées par l'invariant $axm\_1.b$ utilisée dans la Définition 10 ($Sync(Sys_{sync}, Sys_{async})$) , et de système bien formé WF, exprimée par l'invariant $axm\_1.c$ utilisé dans la Définition 11 ($WF(Sys_{async})$) sont introduites dans les invariants de ce niveau de raffinement. Ces deux propriétés complètent la preuve de réalisabilité.

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

Listing 5.13 – Un extrait des invariants du modéle LTS_ Asynchronous_ model.

- Les événements associés aux opérateurs de composition. 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 5.14 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 est complété, l'événement $Add\_Sequence\_send\_receive$ enregistre la fin de l'émission-réception. Cet événement incrémente le nombre de messages reçus (action $act5$) pour permettre de prouver le variant.

Les traces de $BUILT\_CP$, $BUILT\_SYNC$ et $BUILT\_ASYNC$ sont mises à jour en conséquence par les événements, elles servent à prouver les invariants en vérifiant à chaque étape les conditions d'équivalence, de synchronisabilité et de WF.

Le même principe s'applique sur les autres opérateurs de composition utilisés lors de la composition synchrone du protocole de conversation. Le raffinement synchrone des opérateurs *Add_Choice*, *Add_Loop* et *Add_End* est détaillé dans l'annexe A (partie *'Deuxième raffinement'*).

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

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

Then
$act1 : A\_TRACE := A\_TRACE \cup \{Reduces\_Trace\_states \mapsto St\_Num \mapsto Send \mapsto lts\_s \mapsto msg \mapsto lts\_d \mapsto Reduces\_Trace\_states \mapsto (St\_Num + 1) \mapsto A\_Trace\_index\}$
$act2 : queue, back := queue \cup \{lts\_d \mapsto msg \mapsto back\}, back + 1$
$act3 : A\_GS := A\_Next\_States(\{send\} \mapsto A\_GS \mapsto queue)$
$\ldots$
End

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

Where
$grd1 : queue \neq \varnothing \wedge lts\_d \mapsto msg \mapsto front \in queue$
$grd2 : \exists receive\_st\_src, receive\_st\_dest \cdot (((lts\_d \mapsto receive\_st\_src) \in A\_GS) \wedge ((receive\_st\_src \mapsto (Receive \mapsto msg \mapsto lts\_s) \mapsto index) \mapsto receive\_st\_dest) \in CPs\_ASYNC\_B \wedge \ldots$
$\ldots$

Then
$act1 : A\_TRACE := A\_TRACE \cup \{Reduces\_Trace\_states \mapsto St\_Num \mapsto Receive \mapsto lts\_s \mapsto msg \mapsto lts\_d \mapsto Reduces\_Trace\_states \mapsto (St\_Num + 1) \mapsto A\_Trace\_index\}$
$act2 : queue := queue \setminus \{lts\_d \mapsto msg \mapsto front\}$
$\ldots$
End

Event $Add\_Sequence\_Send - Receive$ **Refines** $Add\_Sequence \triangleq$
Any
$A\_Some\_cp\_b, A\_Some\_cp\_sync\_b, Send\_cp\_async\_b, Receive\_cp\_async\_b, R\_trace\_b$

Where
$grd1 : A\_MSG(Send\_cp\_async\_b) = A\_MSG(Receive\_cp\_async\_b)$
$grd2 : ACTION(Receive\_cp\_async\_b) = Receive \wedge ACTION(Send\_cp\_async\_b) = Send$
$grd3 : A\_Some\_cp\_b \in ISeqF$
$grd4 : MSG(A\_Some\_cp\_b) = A\_MSG(Send\_cp\_async\_b)$
$\ldots$

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
$act1 : BUILT\_CP := BUILT\_CP \cup \{A\_Some\_cp\_b\}$
$act2 : BUILT\_SYNC := BUILT\_SYNC \cup \{A\_Some\_cp\_sync\_b\}$
$act3 : BUILT\_ASYNC := BUILT\_ASYNC \cup \{Send\_cp\_async\_b\} \cup \{Receive\_cp\_async\_b\}$
$act4 : REDUCED\_TRACE := REDUCED\_TRACE \cup \{R\_trace\_b\}$
$\ldots$
End
End

Listing 5.14 – Un extrait de LTS_ Asynchronous_ model.

4.3 Une approche évolutive de construction des CP réalisables

Les raffinements des événements correspondant aux opérateurs de composition permettent de garantir les propriétés d'équivalence, de synchronisabilité et de WF, qui à leur tour permettent de prouver la réalisabilité. Ici, seules les constructions permettent de définir des *CP* réalisables au niveau générique sont données. Ces constructions permettent de définir un modèle générique de *CP* réalisables de manière symbolique. Une fois les preuves sont effectuées, il suffit d'instancier le modèle pour construire des *CP* réalisables particuliers. En procédant de cette manière, nous évitons de rejouer la preuve et on obtient une approche évolutive et incrémentale pour la construction de *CP* réalisables. De plus, il n'est pas nécessaire de construire explicitement les compositions synchrones et asynchrones des entités projetées. Les théorèmes fournis de la section 2 définissent des conditions qui ne sont vérifiées qu'une seule fois sur les *CP* construits. De plus, ces propriétés sont de nature syntaxique.

5 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 5.1. Ce modèle est divisé en deux parties : une partie concerne la modélisation du système, états, variables, variants et invariants ; et une seconde partie modélise le comportement des systèmes et du mécanisme de composition réalisable. L'instanciation consiste à configurer le modèle générique obtenu pour des systèmes spécifiques. Le modèle instancié est obtenu en deux étapes, décrites ci-dessous, ces étapes correspondent à l'instanciation de chaque partie de modélisation.

5.1 Étape 1. Instanciation du contexte

Initialement, les valeurs spécifiques des ensembles abstraits définis dans le contexte *Lts_Contexte* présenté à la section 4.1 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) ainsi que pour les constantes formant ces mêmes ensembles.

Un deuxième contexte d'instanciation *Lts_Sync_Instance* raffinant l'instance abstraite est donné, où des valeurs concrètes de l'ensemble CPs_SYNC_B du contexte synchrone *Lts_Sync_Contexte* qui raffine le contexte abstrait *Lts_Contexte* sont données.

Un dernier contexte d'instanciation *Lts_Async_Instance* raffinant l'instance concrète synchrone est donné. Les valeurs concrètes des ensembles (A_STATES,

A_TRACES, S_Next_States, A_Next_States) relatives à la communication asynchrone(*Lts-_Async_Contexte*) sont définies.

5.2 Étape 2. 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.

6 Application à un cas d'étude

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 7.1. 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 5.15, 5.16 et 5.17 montrent l'instanciation du modèle générique pour le *CP* de la Figure 7.1. Ils montrent également que les axiomes définis dans le modèle sont habités. Le vérificateur de modèle ProB [62] associé à B-événementiel sur la plate-forme Rodin a été utilisé pour la validation automatique.

Context $LTS\_CONTEXT\_Instance$ **Extends** $LTS\_CONTEXT$
Constants $A, B, L, Order, Deliver, Deliver\_Conf,$
$Delivery, Terminate, TerminateL, Get\_Status,$
$Get\_StatusL, StatusL, Status, s0, s1, s2, s3, s4, s5, s6, s7, \ldots, s11$

Axioms
$axm1 : partition(PEERS, \{A\}, \{B\}, \{L\}, \{Pend\})$
$axm2 : partition(MESSAGES, \{Order\}, \{Deliver\}, \{Deliver\_Conf\}, \{Delivery\}, \{Terminate\},$
$\{TerminateL\}, \{Get\_Status\}, \{Get\_StatusL\}, \{StatusL\}, \{Status\}, \{End\})$
$axm3 : partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\}, \{s6\}, \{s7\}, \{s8\}, \{s9\}, \{s10\}, \{s11\})$
$axm4 : CPs\_B = \{s0 \mapsto B \mapsto Order \mapsto A \mapsto s1 \mapsto 1,$
$s1 \mapsto A \mapsto Deliver \mapsto L \mapsto s2 \mapsto 2,$
$s2 \mapsto L \mapsto Deliver\_Conf \mapsto A \mapsto s3 \mapsto 3,$
$s3 \mapsto A \mapsto Delivery \mapsto B \mapsto s4 \mapsto 4,$
$s4 \mapsto B \mapsto Terminate \mapsto A \mapsto s6 \mapsto 5,$
$s6 \mapsto A \mapsto TerminateL \mapsto L \mapsto s8 \mapsto 6,$
$s4 \mapsto B \mapsto Get\_Status \mapsto A \mapsto s5 \mapsto 7,$
$s5 \mapsto A \mapsto Get\_StatusL \mapsto L \mapsto s7 \mapsto 8,$
$s7 \mapsto L \mapsto StatusL \mapsto A \mapsto s9 \mapsto 9,$
$s9 \mapsto A \mapsto Status \mapsto B \mapsto s4 \mapsto 10,$
$s8 \mapsto Pend \mapsto End \mapsto Pend \mapsto s10 \mapsto 11$
$\}$
End

Listing 5.15 – Un extrait de LTS_ Instance.

D'autres cas d'études prisent du monde de la recherche sur la réalisabilité ont été utilisés 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 5.15 fournit des valeurs concrètes pour les ensembles abstraits du contexte $Lts\_Contexte$. Ainsi, on value les ensembles suivants :

— $PEERS$ représentent l'ensemble des entités communicantes formant le CP,
— $MESSAGES$ l'ensemble des messages échangés entre entités,
— CP_STATES énumèrent des états globaux du CP,
— CPs_B l'ensemble des transitions de base, l'élément de base d'un CP.

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

$axm1 : CPs\_SYNC\_B = \{s0 \mapsto Send \mapsto Order \mapsto A \mapsto B \mapsto Receive \mapsto Order \mapsto s1 \mapsto 1,$
$s1 \mapsto Send \mapsto Deliver \mapsto L \mapsto A \mapsto Receive \mapsto Deliver \mapsto s2 \mapsto 2,$
$s2 \mapsto Send \mapsto Deliver\_Conf \mapsto A \mapsto L \mapsto Receive \mapsto Deliver\_Conf \mapsto s3 \mapsto 3,$
$s3 \mapsto Send \mapsto Delivery \mapsto B \mapsto A \mapsto Receive \mapsto Delivery \mapsto s4 \mapsto 4,$
$s4 \mapsto Send \mapsto Terminate \mapsto A \mapsto B \mapsto Receive \mapsto Terminate \mapsto s6 \mapsto 5,$
$s6 \mapsto Send \mapsto TerminateL \mapsto L \mapsto A \mapsto Receive \mapsto TerminateL \mapsto s8 \mapsto 6,$
$s4 \mapsto Send \mapsto Get\_Status \mapsto A \mapsto B \mapsto Receive \mapsto Get\_Status \mapsto s5 \mapsto 7,$
$s5 \mapsto Send \mapsto Get\_StatusL \mapsto L \mapsto A \mapsto Receive \mapsto Get\_StatusL \mapsto s7 \mapsto 8,$
$s7 \mapsto Send \mapsto StatusL \mapsto A \mapsto L \mapsto Receive \mapsto StatusL \mapsto s9 \mapsto 9,$
$s9 \mapsto Send \mapsto Status \mapsto B \mapsto A \mapsto Receive \mapsto Status \mapsto s4 \mapsto 10,$
$s8 \mapsto Send \mapsto End \mapsto Pend \mapsto Pend \mapsto Receive \mapsto End \mapsto s10 \mapsto 11$
$\}$

End

Listing 5.16 – Un extrait de Lts_Sync_Instance.

Le deuxième contexte d'instanciation $Lts\_Sync\_Instance$ du Listing 5.16 fournit des valeurs concrètes pour l'ensemble des transitions synchrones de base relatif au contexte $Lts\_Sync\_Contexte$:

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

Le dernier contexte d'instanciation $Lts\_Async\_Instance$ du Listing 5.17 fournit des valeurs concrètes pour les ensembles différés du contexte $Lts\_Async\_Contexte$. Tous les ensembles correspondant à la caractérisation statique du système asynchrone, sont valués par un ensemble d'instances. Ces ensembles caractérisent des systèmes de communication asynchrone spécifiques avec une file d'attente globale : Les ensembles suivants sont valués :

— A_STATES ensemble des états formant les entités projetées à partir du CP.
— CPs_ASYNC_B est l'ensemble des transitions asynchrone de base.
— R_TRACE_B ensemble des traces asynchrones réduites de Sys_{async}.
— A_TRACES ensemble des traces asynchrones complètes de Sys_{async}.
— S_Next_States états de système synchrone Sys_{sync}.
— A_Next_States états de système asynchrone 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
$axm1 : partition(A\_STATES, \{s0\_A\}, \{s1\_A\}, \{s2\_A\}, \{s3\_A\}, \{s4\_A\}, \{s5\_A\}, \{s6\_A\}, \{s7\_A\}, \{s8\_A\},$
$\{s0\_B\}, \{s1\_B\}, \{s2\_B\}, \{s3\_B\}, \{s4\_B\},$
$\{s0\_L\}, \{s1\_L\}, \{s2\_L\}, \{s3\_L\}, \{s4\_L\}, \{s5\_L\}$

$axm2 : CPs\_ASYNC\_B = \{((s0\_B \mapsto (Send \mapsto Order \mapsto A) \mapsto 1) \mapsto s1\_B),$
$((s0\_A \mapsto (Receive \mapsto Order \mapsto B) \mapsto 1) \mapsto s1\_A),$
$((s1\_A \mapsto (Send \mapsto Deliver \mapsto L) \mapsto 2) \mapsto s2\_A),$
$((s0\_L \mapsto (Receive \mapsto Deliver \mapsto A) \mapsto 2) \mapsto s1\_L),$
$\ldots$
$\}$
$axm3 : R\_TRACE\_B = \{s0 \mapsto B \mapsto Order \mapsto A \mapsto s1 \mapsto 1,$
$s1 \mapsto A \mapsto Deliver \mapsto L \mapsto s2 \mapsto 2,$
$s2 \mapsto L \mapsto Deliver\_Conf \mapsto A \mapsto s3 \mapsto 3,$
$s3 \mapsto A \mapsto Delivery \mapsto B \mapsto s4 \mapsto 4,$
$s4 \mapsto B \mapsto Terminate \mapsto A \mapsto s6 \mapsto 5,$
$s6 \mapsto A \mapsto TerminateL \mapsto L \mapsto s8 \mapsto 6,$
$s4 \mapsto B \mapsto Get\_Status \mapsto A \mapsto s5 \mapsto 7,$
$\ldots\}$
$axm4 : A\_TRACES = \{s \mapsto 0 \mapsto Send \mapsto B \mapsto Order \mapsto A \mapsto s \mapsto 1 \mapsto 1,$
$s \mapsto 1 \mapsto Receive \mapsto B \mapsto Order \mapsto A \mapsto s \mapsto 2 \mapsto 2,$
$s \mapsto 2 \mapsto Send \mapsto A \mapsto Deliver \mapsto L \mapsto s \mapsto 3 \mapsto 3,$
$s \mapsto 3 \mapsto Receive \mapsto A \mapsto Deliver \mapsto L \mapsto s \mapsto 4 \mapsto 4,$
$s \mapsto 4 \mapsto Send \mapsto L \mapsto Deliver\_Conf \mapsto A \mapsto s \mapsto 5 \mapsto 5,$
$s \mapsto 5 \mapsto Receive \mapsto L \mapsto Deliver\_Conf \mapsto A \mapsto s \mapsto 6 \mapsto 6,$
$s \mapsto 6 \mapsto Send \mapsto A \mapsto Delivery \mapsto B \mapsto s \mapsto 7 \mapsto 7,$
$s \mapsto 7 \mapsto Receive \mapsto A \mapsto Delivery \mapsto B \mapsto s \mapsto 8 \mapsto 8,$
$s \mapsto 8 \mapsto Send \mapsto B \mapsto Terminate \mapsto A \mapsto s \mapsto 9 \mapsto 9,$
$s \mapsto 9 \mapsto Receive \mapsto B \mapsto Terminate \mapsto A \mapsto s \mapsto 10 \mapsto 10,$
$s \mapsto 10 \mapsto Send \mapsto A \mapsto TerminateL \mapsto L \mapsto s \mapsto 11 \mapsto 11,$
$\ldots\}$
$axm6 : S\_Next\_States = \{$
$\{((s0\_B \mapsto (Send \mapsto Order \mapsto A) \mapsto 1) \mapsto s1\_B)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s0\_B), (L \mapsto s0\_L)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s0\_A \mapsto (Receive \mapsto Order \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto$
$\{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s1\_A \mapsto (Send \mapsto Deliver \mapsto L) \mapsto 2) \mapsto s2\_A)\} \mapsto$
$\{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto$
$\{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s0\_L \mapsto (Receive \mapsto Deliver \mapsto A) \mapsto 2) \mapsto s1\_L)\} \mapsto$
$\{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto$
$\{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s1\_L)\},$
$\ldots\}$
$axm7 : A\_Next\_States = \{$
$\{((s0\_B \mapsto (Send \mapsto Order \mapsto A) \mapsto 1) \mapsto s1\_B)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s0\_B), (L \mapsto s0\_L)\} \mapsto \varnothing \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s0\_A \mapsto (Receive \mapsto Order \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \{A \mapsto Order \mapsto 0\} \mapsto$
$\{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s0\_A \mapsto (Receive \mapsto Order \mapsto B) \mapsto 1) \mapsto s1\_A)\} \mapsto$
$\{(A \mapsto s0\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \varnothing \mapsto$
$\{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s1\_A \mapsto (Send \mapsto Deliver \mapsto L) \mapsto 2) \mapsto s2\_A)\} \mapsto$
$\{(A \mapsto s1\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \varnothing \mapsto$
$\{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\},$
$\{((s0\_L \mapsto (Receive \mapsto Deliver \mapsto A) \mapsto 2) \mapsto s1\_L)\} \mapsto$
$\{(A \mapsto s2\_A), (B \mapsto s1\_B), (L \mapsto s0\_L)\} \mapsto \{L \mapsto Deliver \mapsto 1\} \mapsto$
$\ldots\}$
End

Listing 5.17 – Un extrait de Lts_ Async_ Instance.

7 Animation avec ProB

La première étape consiste à générer la trace du modèle abstrait de la conversation. Cela nous donne la conversation de la Figure 5.2.

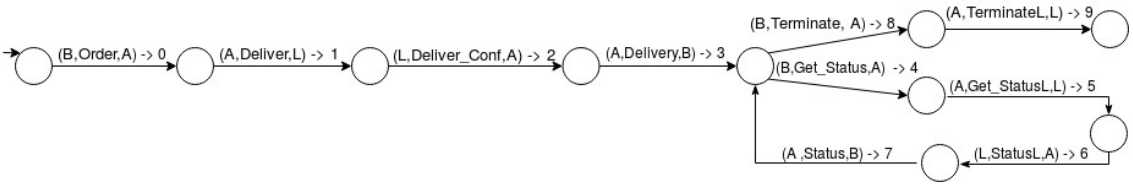


FIGURE *5.2 – Trace de modèle abstrait de conversation (CP).*

Le modèle abstrait donne une vue globale de la conversation et l'enchaînement souhaité sans détailler les opérations. L'abstraction donnée par la Figure 5.2 indique une interaction dans l'ordre $Order$, $Deliver$, $Deliver_Conf$, $Delivery$, Get_Status, $Get_StatusL$, $StatusL$, $Status$, $Terminate$ et enfin $TerminateL$.

Nous passons après au modèle synchrone, ici on introduit les opérations sous forme d'envoi/réception, l'origine et la destination des messages sont aussi spécifiés. Nous remarquons bien dans la Figure 5.3 que, l'ordre spécifié en mode abstrait est toujours maintenu.

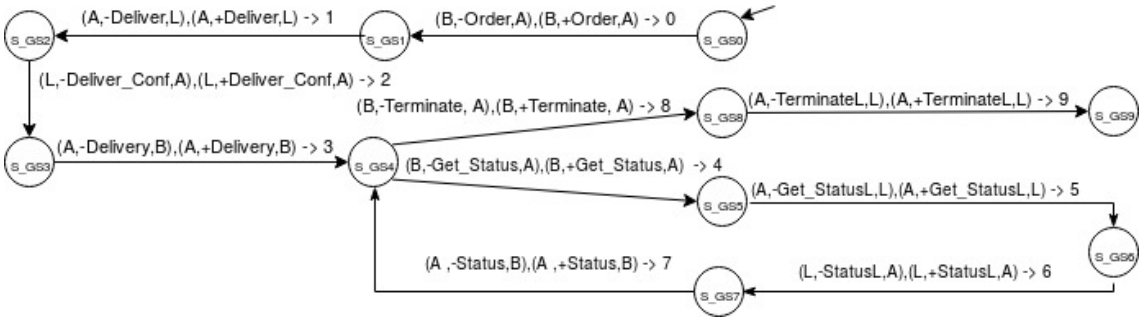


FIGURE *5.3 – Trace de modèle synchrone de conversation.*

Nous donnons dans la Table 5.1 les valeurs des états globaux du modèle synchrone obtenu par l'animation en ProB.

Le dernier niveau spécifie le comportement de chaque entité à part, tout en gardant l'ordre spécifié dans les modèles d'avant. On remarque ici qu'on a plus d'état global qui expriment le séquencement de la conversation. La trace de la conversation asynchrone est illustrée par la Figure 5.4.

La Table 5.2 détaille les valeurs des états globaux du modèle Asynchrone.

État global synchrone	**Valeurs des états locaux**
S_GS0	(A_S0, B_S0, L_S0)
S_GS1	(A_S1, B_S1, L_S0)
S_GS2	(A_S2, B_S1, L_S1)
S_GS3	(A_S3, B_S1, L_S2)
S_GS4	(A_S4, B_S2, L_S2)
S_GS5	(A_S5, B_S3, L_S2)
S_GS6	(A_S8, B_S3, L_S3)
S_GS7	(A_S9, B_S3, L_S2)
S_GS4	(A_S4, B_S2, L_S2)
S_GS8	(A_S6, B_S4, L_S2)
S_GS9	(A_S7, B_S4, L_S4)

TABLE *5.1 – Trace synchrone de l'exemple.*

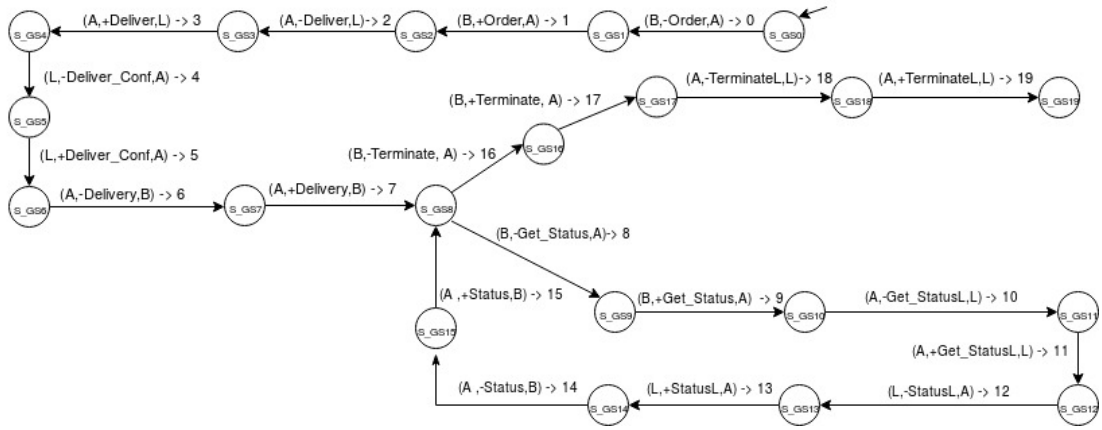


FIGURE *5.4 – Trace de modèle asynchrone de conversation.*

8 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, des preuves complexes ont été réalisées pour garantir la correction des différentes machines et des raffinements.

8.1 Les preuves

La Table 5.3 présente les statistiques de preuve pour l'ensemble des développements B-événementiel. Des efforts importants ont été nécessaires à la preuve interactive des différents modèles de composition. Toutes les obligations de preuve associées au développement formel B-événementiel présenté dans ce chapitre ont été prouvées, soit avec les prouveurs automatiques associés à la plate-forme Rodin, soit à l'aide du prouveur interactif sur la plate-forme Rodin.

État global asynchrone	**Contenu de queue / Valeurs des états locaux**
A_GS0	$\varnothing$/(A_S0, B_S0, L_S0)
A_GS1	$\{Order \mapsto 0\}$/(A_S0, B_S1, L_S0)
A_GS2	$\varnothing$/(A_S1, B_S1, L_S0)
A_GS3	$\{Deliver \mapsto 1\}$/(A_S2, B_S1, L_S0)
A_GS4	$\varnothing$/(A_S2, B_S1, L_S1)
A_GS5	$\{Deliver\_Conf \mapsto 2\}$(A_S2, B_S1, L_S2)
A_GS6	$\varnothing$/(A_S3, B_S1, L_S2)
A_GS7	$\{Delivery \mapsto 3\}$/(A_S4, B_S1, L_S2)
A_GS8	$\varnothing$/(A_S4, B_S2, L_S2)
A_GS9	$\{Get\_Status \mapsto 4\}$/(A_S4, B_S3, L_S2)
A_GS10	$\varnothing$/(A_S5, B_S3, L_S2)
A_GS11	$\{Get\_StatusL \mapsto 5\}$/(A_S8, B_S3, L_S2)
A_GS12	$\varnothing$/(A_S8, B_S3, L_S3)
A_GS13	$\{StatusL \mapsto 6\}$/(A_S8, B_S3, L_S2)
A_GS14	$\varnothing$/(A_S9, B_S3, L_S2)
A_GS15	$\{Status \mapsto 7\}$/(A_S4, B_S3, L_S2)
A_GS8	$\varnothing$/(A_S4, B_S2, L_S2)
A_GS16	$\{Terminate \mapsto 8\}$/(A_S4, B_S4, L_S2)
A_GS17	$\varnothing$/(A_S6, B_S4, L_S2)
A_GS18	$\{TerminateL \mapsto 9\}$/(A_S7, B_S4, L_S2)
A_GS19	$\varnothing$/(A_S7, B_S4, L_S4)

TABLE *5.2 – Trace asynchrone de l'exemple.*

Un point clé concerne l'instanciation pour des systèmes spécifiques. En effet, le développement présenté ci-dessus est un développement générique, défini à un niveau abstrait et générique, où les obligations de preuve en paramètre obtenues agissent comme des méta-théorèmes.

L'utilisation des événements paramétrés (constructions Any) montre que le développement prend en paramètre tout système de transitions décrit par un modèle de *CP* ainsi que les invariants associés exprimés dans les modèles B-événementiel correspondants. Cela semble très intéressant car le modèle définissant le mécanisme de composition des *CP* n'est prouvé qu'une fois. Cependant, la preuve est plus difficile que pour système concret spécifique seul. Par conséquent, la possibilité de réutiliser un modèle de composition générique pour plusieurs *CP* particulier réduit l'effort de preuve.

Notons que des techniques de vérification de modèle peuvent être appliquées pour vérifier automatiquement la correction de l'instanciation. L'exploration de tous les états devient possible car les ensembles sont finis dans les contextes d'instanciation.

Cependant, ces techniques peuvent être confrontées au problème de l'explosion d'états. Par exemple, la difficulté des preuves dans notre approche n'est pas affectée par la taille des systèmes ni par la complexité des échanges, alors qu'une méthode

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 *5.3 – Statistiques de preuves RODIN (modèle de composition).*

qui énumère explicitement toutes les valeurs possibles des composants du système serait limitée par le nombre considérable de possibilités offertes par la combinatoire. Une évaluation quantitative des différentes épreuves pour les différentes machines et contextes sont disponibles à la Figure 5.5.

8.2 Méthodes formelles correctes-par-construction

L'approche proposée est générique. Les contextes de niveau *CP*, 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, des niveaux *CP* synchrone et asynchrone afin de formaliser le modèle de comportement décrit avec les événements, initialisation, compositions séquentielle, en branche et en cycle. Notons que, le calcul des entités projetées à partir du *CP* ainsi que les traces des recompositions synchrone et asynchrone du *CP* 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

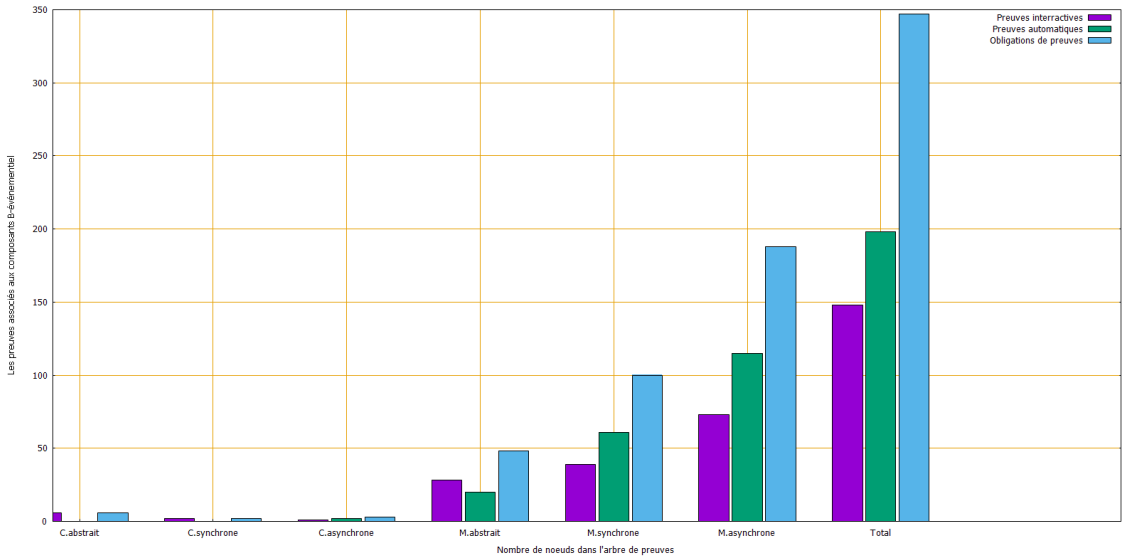


FIGURE *5.5 – Les preuves associées aux modèle de composition B-événementiel.*

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'invariant par les événements. Cela nous permet de scinder la preuve globale en preuves plus simples et donc plus faciles à gérer. Les preuves de la préservation des invariants et de la décroissance du variant sont obtenues au niveau abstrait de la machine *modèle abstrait.* Elles sont conservées par les autres machines qui la raffinent.

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

Notons que dans d'autres techniques correctes-par-construction traditionnelles comme Coq [9] ou Isabelle [69], des schémas classiques de preuve inductive sont proposés.

On a :

— décrire la structure inductive associée aux systèmes formalisés,
— donner un schéma de preuve inductif spécifique à cette structure inductive,
— prouver la correction de l'instanciation.

Dans ces techniques, le processus inductif associé à la composition réalisable des systèmes et la capacité de raffinement n'est pas intégré comme dans B-événementiel où cette notion est disponible via les variables d'état et les événements. Tel que, c'est au développeur de formaliser la preuve inductive du raffinement et d'instanciation, car aucun événement de ce dernier n'est disponible.

Par rapport à la méthode B-événementiel, une autre spécification de niveau méta et du processus de preuve est nécessaire.

Chapitre 6

Réparation correcte-par-construction

Sommaire

1 Opérateurs de réparation : théorèmes et preuves 93
2 Étude de cas 96
3 Scénarios de réparation 98
3.1 Restauration de ISeqF 98
3.2 Restauration de PCF 99
4 Formalisation avec B-événementiel 101
4.1 Partie statique : définitions requises 101
4.2 Partie dynamique : modélisation de la réparation 102
5 Restauration de la réalisabilité et instanciation 103
6 Évaluation 105

Dans ce chapitre, nous étudions la possibilité de réparation de protocoles de conversation non réalisables, l'objectif étant d'identifier un ensemble de modifications d'un protocole de conversation non réalisable donné pour le rendre réalisable. Nous présentons une stratégie permettant de réparer automatiquement les protocoles de conversation non réalisables en se basant sur les propriétés nécessaires présentées précédemment et fournissons des garanties formelles de correction 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* 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* identifié comme non réalisable, par composition incrémentale de plusieurs CP_b et $CP_{b_{Synch}}$ [1]. Ces opérateurs

1. $CP_{b_{Synch}}$ est une transition basique de synchronisation, formant l'ensemble des scénario de réparation possibles

désignent : une réparation séquentielle dénotée par $\otimes_{(\text{ISeqF}_{Restore}, s^f_{CP})}$ et une réparation du choix parallèle dénotée par $\otimes_{(\text{PCF}_{Restore}, s^f_{CP})}$. Chacun des opérateurs est caractérisé par deux paramètres, le premier est un *CP réalisable* avec un état final $s^f_{CP} \in S^f_{CP}$ et le second est un ensemble de protocoles de conversation de base formé de CP_b et $CP_{b_{Synch}}$ (ou d'un ensemble de CP_b et $CP_{b_{Synch}}$). 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 *restauration de ISeqF* $\otimes_{(ISeqF_{Restore}, s^f_{CP})}$.

Soient

- $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *un protocole de conversation avec un état final* $s^f_{CP} \in S_{CP}$ *et*
- $CP_{b_{Synch}}$ *un protocole de conversation de base de synchronisation 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 *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 définie comme suit :*

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

La Figure 6.2 illustre un CP non réalisable où la propriété ISeqF n'est pas respectée. Les Figures 6.3 et 6.4 montrent les proposition de restauration de ISeqF.

Ici, la restauration de la propriété ISeqF autrement dit, la 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 met à jour l'ensemble d'états $S_{CP_{\text{ISeqF}_{Restore}}}$, l'ensemble de messages échangés $L_{CP_{\text{ISeqF}_{Restore}}}$, l'ensemble de transitions $T_{CP_{\text{ISeqF}_{Restore}}}$ et l'ensemble d'états finaux $S^f_{CP_{\text{ISeqF}_{Restore}}}$.

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

Soient

- $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ *un protocole de conversation avec un état final* $s^f_{CP} \in S_{CP}$ *et*
- $\{CP_{bi} \mid 1 \leq i \leq n, n \geq 2\}$ *un ensemble fini de protocoles de conversation de base 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}, $1 \leq i \leq n, n \geq 2$.

— $\{CP_{b_{Synch\,i}} \mid 1 \leq i \leq n, n \geq 2\}$ *un ensemble de protocoles de conversation de base 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* $s'_{CP_{b_{Synch}}} = s_{CP_b}$.

Alors, la restauration de la propriété PCF

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

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

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

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

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

La Figure 6.2 illustre un *CP* non réalisable où la propriété PCF n'est pas respectée. Les Figures 6.5 et 6.6 montrent les proposition de restauration de PCF.

La restauration de la propriété PCF, autrement dit, la réparation de choix de protocole de conversation en choix $CP_{CP_{\mathrm{PCF}_{Restore}}} = \otimes_{(\mathrm{PCF}_{Restore}, s^f_{CP})}(CP, \{CP_{b_{Synch}\,bi}, CP_{bi}\})$ ajoute un ensemble de protocoles de conversation de base de synchronisation entre *CP* et CP_{bi} à l'état final s^f_{CP}. Elle forme un branchement (un choix) et met à jour l'ensemble des états $S_{CP_{\mathrm{PCF}_{Restore}}}$, l'ensemble des messages échangés $L_{CP_{\mathrm{PCF}_{Restore}}}$, l'ensemble des transitions $T_{CP_{\mathrm{PCF}_{Restore}}}$ et l'ensemble des états finaux $S^f_{CP_{\mathrm{PCF}_{Restore}}}$.

La Table 6.1 introduit les théorèmes assurant la réalisabilité après réparation incrémentale d'un *CP* identifié comme non réalisable car, il ne satisfait pas une des conditions suffisantes. Un théorème garantissant la réparation est associé à chacun des opérateurs de réparation. Ces théorèmes reposent sur les conditions définies dans le chapitre 4. La preuve de chaque théorème est également donnée.

Théorème 5 (Restauration de ISeqF).
Soit $CP = <S_{CP}, s^0_{CP}, L_{CP}, T_{CP}>$ *,un protocole de conversation de base de synchronisation*

$CP_{b_{Synch}}, CP_b$ *et* $s^f_{CP} \in S^f_{CP}$ *est un état final dans CP, alors la propriété suivante*

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

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

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 *6.1 – Théorèmes liés aux CP réparés-par-construction*

est vraie.

Preuve 5. Voir Preuve 2, section 3, chapitre 4

Théorème 6 (Réparation du branchement).
Soient $CP = <S_{CP}, s^0_{CP}, L_{CP}, T_{CP}>$ *un protocole de conversation avec l'état final* $s^f_{CP} \in S^f_{CP}$, $\{CP_{b_{Synch\,i}} \mid 2 \leq i \leq n\}$ *un ensemble de n* $(n \geq 2)$ *protocoles de conversation de base de synchronisation et* $\{CP_{bi} \mid 2 \leq i \leq n\}$ *un ensemble des protocoles de synchronisation* $\{CP_{b_{Synch\,i}} \mid 2 \leq i \leq n\}$ *tels que, les propriétés suivantes*

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*
$CP_{PCFRepair} = \otimes_{(PCF_{Repair}, s^f_{CP})}(CP, \{CP_{b_{Synch\,i}}\}, \{CP_{bi}\}) \in R.$

sont vraies.

Preuve 6. Voir Preuve 3, section 3, chapitre 4

2 Étude de cas

L'étude de cas suivante, Basu et al. [12] décrit une chorégraphie illustrée sur la Figure 6.1. Il s'agit d'un protocole de transfert de fichiers où P_1 est un client demandant le transfert de fichier, P_2 un serveur de fichiers et P_3 initialise la communication entre le client et le serveur. 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 terminée" (mc), dans ce cas le serveur répond par le message "Transfert terminé" (mf), qui, à nouveau , termine le protocole.

Lors de la projection, le *CP* de la Figure 6.1 produit les trois entités communicantes de la Figure 6.2.

La spécification de la chorégraphie décrit dans la Figure 6.1 n'est pas réalisable. Nous illustrons ceci en définissant une séquence d'applications d'opérateurs de composition. Nous obtenons les ensembles suivantes.

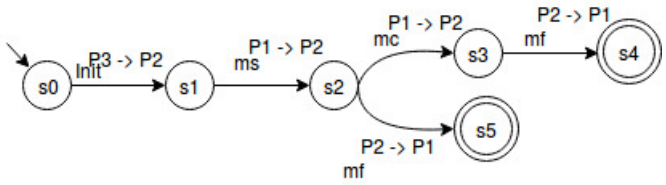


FIGURE *6.1 – Chorégraphie de protocole de transfert de fichiers*

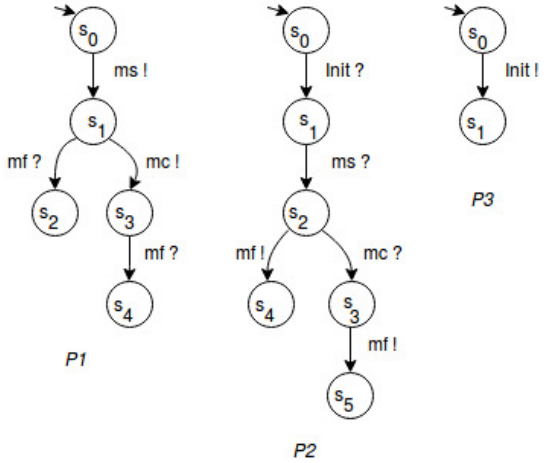


FIGURE *6.2 – Entités projetées d'une chorégraphie de protocole de transfert de fichiers*

— $CP = \varnothing$

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 des opérateurs de séquence et de choix, ne satisfont pas les conditions suffisantes correspondantes.

Le Listing 6.1 montre le contexte B-événementiel défini pour instancier le modèle générique de cette étude de cas.

Context
Constants
Axioms
$axm1$:$partition(PEERS, \{P1\}, \{P2\}, \{P3\}, \{Pend\})$
$axm2$:$partition(MESSAGES, \{Init\}, \{ms\}, \{mc\}, \{mf\}, \{End\})$
$axm3$:$partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\})$
$axm4$: $partition(A\_STATES, \{s0\_P1\}, \{s1\_P1\}, \dots)$
$axm5$: $CPs\_B = \{s0 \mapsto P3 \mapsto Init \mapsto P2 \mapsto s1 \mapsto 1,$
$s1 \mapsto P1 \mapsto ms \mapsto P2 \mapsto s2 \mapsto 2,$
$s2 \mapsto P1 \mapsto mc \mapsto P2 \mapsto s3 \mapsto 3,$
$s1 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s3 \mapsto 3,$
$s2 \mapsto P2 \mapsto mf \mapsto P1 \mapsto s4 \mapsto 4,$
$s4 \mapsto Pend \mapsto End \mapsto Pend \mapsto s5 \mapsto 5$
$\dots\}$
End

Listing 6.1 – Chorégraphie 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 6.1 pour les opérateurs de séquence et de choix. L'instanciation du modèle générique pour un tel *CP* confirme la violation de conditions suffisantes deux fois, lors de la composition de séquence et de la composition pour le 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 correct échanges 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 précédemment non validées. Deux cas de réparation peuvent être distingués pour les opérateurs de séquence et de branche comme suit.

3.1 Restauration de ISeqF

Réparation de la propriété de séquence ISeqF. Deux réparations possibles sont identifiées sur les Figures 6.3 et 6.4. Suivant la définition de ISeqF, nous introduisons une transition de synchronisation avec le message $Sync0$ (en gras pointillé dans les Figures 6.3 et 6.4) établissant la condition ISeqF.

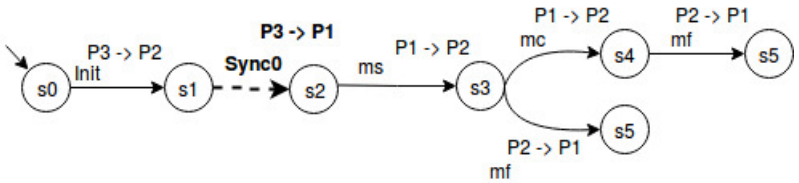


FIGURE *6.3 – Proposition 1 de la réparation de ISeqF.*

La spécification de la chorégraphie réparée de la Figure 6.3 est non réalisable. Nous illustrons ceci en définissant une séquence d'applications d'opérateurs. Nous obtenons les étapes de composition suivantes.

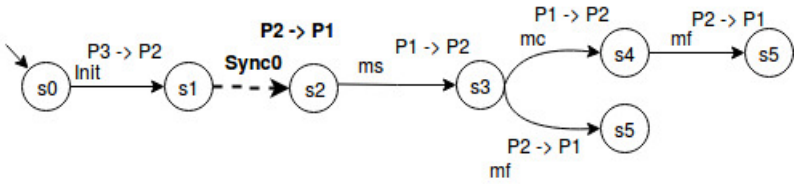


FIGURE *6.4 – Proposition 2 de la réparation de ISeqF.*

— $CP = \varnothing$

Transitions de base.

$CP_{b0} = s_0 \xrightarrow{Init^{P3\to P2}} s_1,$
$CP_{b1} = s_1 \xrightarrow{Sync0^{P3\to P1}} s_2,$
$CP_{b2} = s_2 \xrightarrow{ms^{P1\to P2}} s_3,$
$CP_{b3} = s_3 \xrightarrow{mc^{P1\to P2}} s_4,$
$CP_{b4} = s_3 \xrightarrow{mf^{P2\to P1}} s_5,$
$CP_{b5} = s_4 \xrightarrow{mf^{P2\to P1}} s_5.$

La prochaine étape consiste à appliquer une séquence d'opérateurs de composition prédéfinie, conduisant à la construction du CP de la Figure 6.3. Ces opérateurs ne sont appliqués que si les conditions suffisantes associées à chaque opérateur de composition sont satisfaites.

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 (la propriété PCF dans notre cas).

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

Réparation de la propriété de branchement PCF. De manière similaire à la réparation de la séquence, deux scénarios de réparation sont possibles selon la définition de PCF . Ils sont illustrés sur les Figures 6.5 et 6.6. La réparation de la transition de choix nécessite l'introduction d'un échange de messages de synchronisation $Sync1$ (en gras pointillé sur les Figures 6.5 et 6.6) précèdent les transitions en branche qui violent la condition PCF. Ces transitions échangent des messages de synchronisation entre la même entité émettrice que les autres branches et une entité réceptrice quelconque.

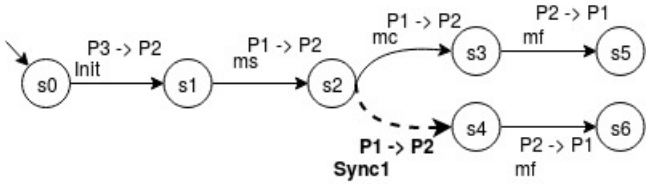


FIGURE *6.5 – Proposition 1 de la réparation du PCF.*

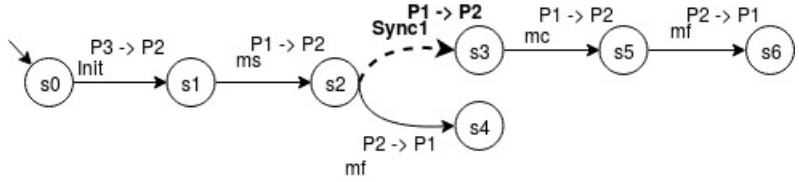


FIGURE *6.6 – Proposition 2 de la réparation du PCF.*

Le CP réparé de la Figure 6.6 est non réalisable. Nous illustrons ceci en définissant une séquence d'applications d'opérateurs. Nous obtenons les étapes de composition suivantes.

— $CP = \varnothing$

Transitions basiques.

$CP_{b0} = s_0 \xrightarrow{Init^{P3\to P2}} s_1,$
$CP_{b1} = s_1 \xrightarrow{ms^{P1\to P2}} s_2,$
$CP_{b2} = s_2 \xrightarrow{Sync1^{P1\to P2}} s_3,$
$CP_{b3} = s_2 \xrightarrow{mf^{P2\to P1}} s_4,$
$CP_{b4} = s_3 \xrightarrow{mc^{P1\to P2}} s_5,$
$CP_{b5} = s_5 \xrightarrow{mf^{P2\to P1}} s_6.$

Composition. Les deux premières applications d'opérateurs de composition de séquence (CP_0 et CP_1) suffisent à détecter la violation de conditions suffisantes. La propriété ISeqF n'est pas satisfaite par CP_1 dans notre cas.

— $CP_0 = \otimes_{(\gg, s^0_{CP})}(CP, CP_{b0})\checkmark$ $CP_0 \in$ 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 *CP* non-réalisables. Chaque événement ($ISeqF\_Restore$ et $PCF\_Restore$) est conditionné par les conditions suffisantes identifiées.

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 du modèle de réparation.

4.1 Partie statique : définitions requises

Le listing 6.2 définit le protocole de conversation, les propositions de réparation et les entités projetées 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 déterministe (NDC) par les axiomes $axm3\_Cond1$ et $axm4\_Cond1$. Ensuite, les transitions déterministes DC sont obtenues par complémentation dans l'axiome $axm5\_Cond1$.
- La définition de ISeqF est donnée par les axiomes $axm6\_Cond2$ et $axm7\_Cond2$. Elle compare l'entité source $P\_SRC(cp\_b)$ avec l'entité émettrice $LAST\_SDR\_Ps$ ou avec l'entité réceptrice $LAST\_RCV\_Ps$ de la dernière transition du CP. Si l'entité émettrice de la transition à ajouter est identique a l'une des deux entités $LAST\_SDR\_Ps$ ou $LAST\_RCV\_Ps$, alors ISeqF est satisfaite.
- De même, afin de définir la propriété PCF , dans les axiomes $axm8\_Cond3$ et $axm9\_Cond3$, les entités émettrices $PEER\_SRC(Trans)$ des transitions formant le branchement sont comparées. Si toutes les branches possèdent la même entité émettrice alors PCF est satisfaite.

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 ⊆ BUILT_CP_SET
axm7_Cond2 :∀cp_b·(cp_b ∈ BUILT_CP_SET ∧ (P_SRC(cp_b) = LAST_SDR_Ps(SRC_STATE(cp_b))∨
                    P_SRC(cp_b) = LAST_RCV_Ps(SRC_STATE(cp_b))))
                    ⇒ {cp_b} ⊆ ISeqF

− − −Parallel Choice freeness PCF
axm8_Cond3 :PCF ∈ CP_STATES → ℙ(BUILT_CP_SET)
axm9_Cond3 :∀Trans1, Trans2.(Trans1 ∈ BUILT_CP_SET ∧ Trans2 ∈ BUILT_CP_SET∧
                    SRC_STATE(Trans1) = SRC_STATE(Trans2)∧
                    P_SRC(Trans1) = P_SRC(Trans2)∧
                    DST_STATE(Trans1) ≠ DST_STATE(Trans2))∧
                    ⇒ PCF(SRC_STATE(Trans1)) = {Trans1, Trans2}
axm10 :Sequence_OK ⊆ BUILT_CP_SET
axm11 :∀Trans.(Trans ∈ BUILT_CP_SET ∧ {Trans} ⊆ ISeqF)
          ⇒ Trans ∈ Sequence_OK
axm12 :Branch_OK ⊆ BUILT_CP_SET
axm13 :∀Trans1, Trans2.(Trans1 ∈ BUILT_CP_SET ∧ Trans2 ∈ BUILT_CP_SET∧
          {Trans1, Trans2} = PCF(SOURCE_STATE(Trans1)))
          ⇒ {Trans1, Trans2} ⊆ Branch_OK
axm14 :OK = Sequence_OK ∪ Branch_OK
...
End
```

Listing 6.2 – Définition des conditions suffisantes

4.2 Partie dynamique : modélisation de la réparation

Les définitions des événements formalisant les opérateurs de réparation (restauration de ISeqF (Définition 16) ainsi que la restauration de PCF (Définition 17) sont données dans cette section. Chaque événement est protégé par les conditions suffisantes définies où, le *CP* et les entités projetées sont composées si et seulement si les gardes des évènements sont satisfaites.

```
Event ISeqF_Restore ≙
Any Some_reparation, cp_b_Breaks_ISeqF
Where
 grd1 :Some_reparation ∈ Repare_propositions
 grd2 :cp_b_Breaks_ISeqF ∈ CPs_B
 grd3 :BUILT_CP ≠ ∅ ∧ Some_reparation ∈ ISeqF
 grd4 :∃Trans.(Trans ∈ CPs_B ∧ Trans ∉ ISeqF)
 grd5 :BUILT_CP ≠ ∅ ⇒ cp_b_Breaks_ISeqF ∈ ISeqF
 grd6 :Repair_src_state(Some_reparation) ∈ CP_Final_states
 ...
With
Then
 act1 : BUILT_CP := BUILT_CP ∪ {Some_reparation} ∪ {cp_b_Breaks_ISeqF}
 act2 : CP_Final_states := (CP_Final_states ∪ {DST_STATE(cp_b_Breaks_ISeqF)})∖
                    {Repair_src_state(Some_reparation)}
End
```

Listing 6.3 – Un extrait de l'opérateur de réparation ISeqF_Restore

L'opérateur ISeqF_Restore. L'événement *ISeqF_Repair* du Listing 6.3 définit la réparation de la propriété ISeqF (Définition 2 Chapitre 4) en cas de violation. Il ajoute dans l'action *act*1 un CP_b de base, à savoir *Some_reparation*, donné en 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* (*grd*1). Cet événement de réparation est déclenché uniquement si la

construction incrémentale est interrompue si la condition ISeqF n'est pas 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 6.4 – Un extrait des invariants du modéle LTS_model

PCF_Restore operator. L'événement PCF_Repair du Listing 6.5 correspond à la réparation de la condition PCF (Définition 3). Cet événement ajoute un ensemble de CP_b de base déterministes, définis comme paramètres de l'événement, à savoir PCF_Repair ($grd4$), et formé d'au moins deux CP_b de base, à savoir $Branches\_reparation$ ($grd2$) avec même état et même 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
$act1 : BUILT\_CP := BUILT\_CP \cup Branches\_reparation \cup \{cp\_b\_Breaks\_PCF\}$
$act2 : CP\_Final\_states := (CP\_Final\_states \cup BR\_CP\_FINAL\_STATES(SRC\_STATE(branch\_repar-$
$ation)) \cup \{DST\_STATE(cp\_b\_Breaks\_PCF)\}) \setminus$
$(\{SRC\_STATE(branch\_reparation) \cup \{SRC\_STATE(cp\_b\_Breaks\_PCF)\})$
End

Listing 6.5 – Un extrait de l'opérateur de composition PCF_Repair

5 Restauration de la réalisabilité et instanciation

Les deux réparations proposées dans les Sections 3.1 et 3.2 conduisent à quatre scénarios de réparation. Parmi ces scénarios, nous sélectionnons le CP de la Figure 6.7 pour illustrer la correction de la réparation. Concrètement, le CP défini sur la Figure 6.1 est transformé à celui de la Figure 6.7, en ajoutant les nouvelles transitions de synchronisation (transitions en gras pointillé). Le CP obtenu devient réalisable.

Les entités projetées du CP de la Figure 6.7 sont définies sur la Figure 6.8.

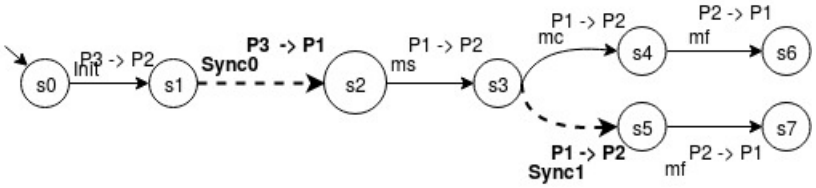


FIGURE *6.7 – CP réparé du protocole de transfert de fichiers*

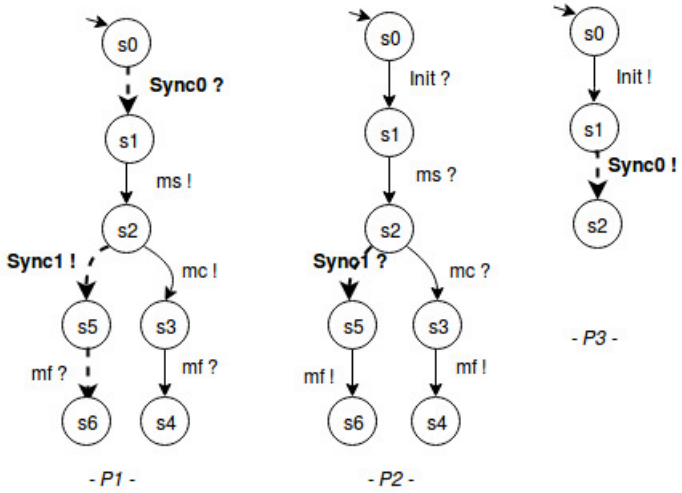


FIGURE *6.8 – Entités projetées du CP réparé du protocole de transfert de fichiers*

Ci-dessous, nous montrons la séquence des opérations de composition et de réparation permettant de construire le CP réparé de la Figure 6.7.

— $CP = \varnothing$

Transitions basiques.

$CP_{b0} = s_0 \xrightarrow{Init^{P3\to P2}} s_1,$
$CP_{b1} = s_1 \xrightarrow{Sync0^{P3\to P1}} s_2,$
$CP_{b2} = s_2 \xrightarrow{ms^{P1\to P2}} s_3,$
$CP_{b3} = s_3 \xrightarrow{mc^{P1\to P2}} s_4,$
$CP_{b4} = s_4 \xrightarrow{mf^{P2\to P1}} s_6,$
$CP_{b5} = s_3 \xrightarrow{Sync1^{P1\to P2}} s_5,$
$CP_{b6} = s_5 \xrightarrow{mf^{P2\to P1}} s_7.$

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 6.6 décrit le contexte B-événementiel définissant les instances du modèle générique de réparation décrivant le *CP* de la Figure 6.7.

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 6.6 – Instantiation de CP réparé d'un protocole de transfert de fichiers

6 Évaluation

La stratégie de réparation a conduit à un développement formel complètement prouvé disponible en annexe B. La Table 6.2 montre les résultats des expériences que nous avons menées à l'aide de la plate-forme Rodin et en B-événementiel. Le développement présenté a été entièrement formalisé et prouvé. L'absence de blocage a été prouvée. Les résultats montrent que peu d'obligations de preuve (POs) nécessitent des épreuves interactives (22 sur 84 POs générées). Les statistiques des différentes preuves pour la machine et le contexte sont présentées sur la Figure 6.9.

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 *6.2 – Statistiques des preuves RODIN (modèle de réparation)*

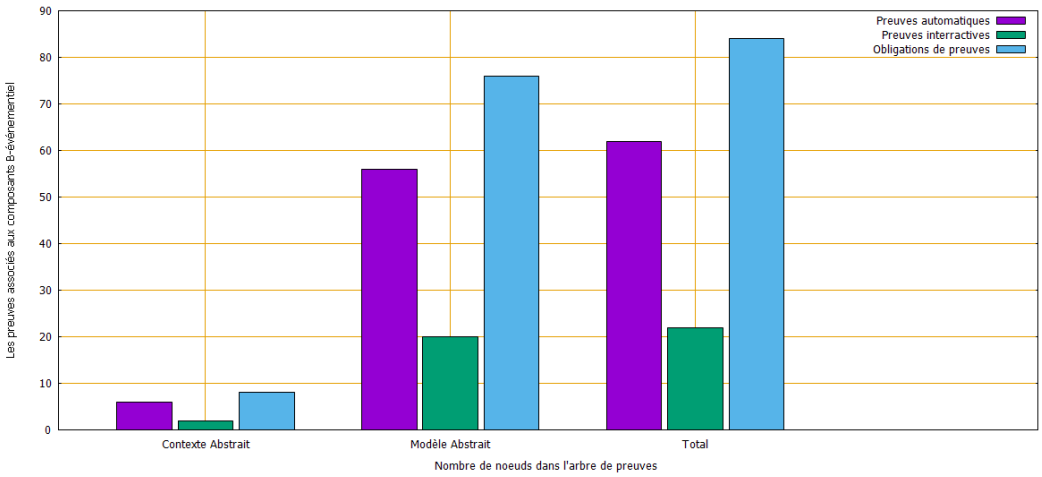


FIGURE *6.9 – Les preuves associées aux modèle de réparation B-événementiel*

Chapitre 7

Évaluation et mise en oeuvre

Sommaire

1 Cas d'étude de composition réalisable 107
1.1 CS1. Processus d'entreprise virtuelle 108
1.2 CS2. Protocole de transfert de fichiers 108
1.3 CS3. Accès à une application web 108
1.4 CS4. Accès à une base de données via une application web 109
1.5 CS5. Application d'achat en ligne 110
1.6 CS6. Protocole de visa de travail australien 111
1.7 CS7. Protocole d'un jeu fictif 112
2 Cas d'étude de réparation de CP non réalisables 113
2.1 CS8. Réparation de CS4 113
2.2 CS9. Réparation de CS5 114
2.3 CS10. Réparation de CS7 115
2.4 CS11. Réparation de CS2 116
3 Travaux connexes 116

Dans ce chapitre, nous présentons une validation de notre approche sur un ensemble d'études de cas issues de la littérature. Nous montrons également que les approches de composition ainsi que de la réparation permettent de détecter des cas de non réalisabilité pour lesquels nous introduisons la réparation.

1 Cas d'étude de composition réalisable

Dans les chapitres 4 et 5 précédents, nous avons formalisé une approche correcte par construction de *CP* réalisables. Notre objectif dans ce chapitre est de montrer que l'approche est non seulement formalisée et prouvée, mais également valide. Nous avons selectionné plusieurs études de cas issues de la litterature comme cas de validation. Toutes ces études de cas ont été soumises au modèle B-événementiel en tant qu'instances du modèle générique résumé dans le chapitre 5. Ces instances sont définies en 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 en Section 1.1 et d'une seconde étude de cas non réalisable en Section 1.2. Ensuite, dans la seconde partie, nous discutons des résultats obtenus pour d'autres études de cas composant nos indices de référence. Notons 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

Ici, la chorégraphie décrit un *CP* d'entreprise virtuelle décrit par la Figure 7.1 et étudié à la section 3.4 du chapitre 4. Il est emprunté à [77] et décrit un protocole de conversation réalisable.

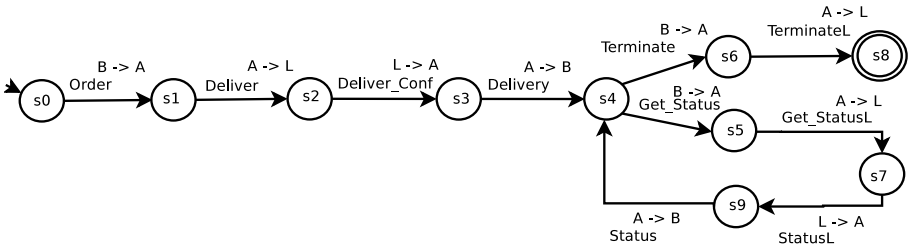


FIGURE *7.1 – Protocole de conversation d'un processus d'approvisionnement simple au sein d'une entreprise virtuelle.*

1.2 CS2. Protocole de transfert de fichiers

L'étude de cas présente une chorégraphie d'un protocole de transfert de fichiers non réalisable empruntée à [12]. Ce cas est décrit dans la section 1.2 du chapitre 6. Une réparation de cette chorégraphie est également proposée dans la section 2.

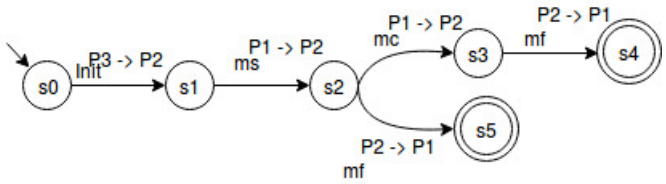


FIGURE *7.2 – Chorégraphie de protocole de transfert de fichier simple.*

1.3 CS3. Accès à une application web

Le protocole d'accès à une application web est un exemple concret décrit dans la Figure 7.3 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 déconnecter de l'interface (*logout*). Ce *CP* est réalisable.

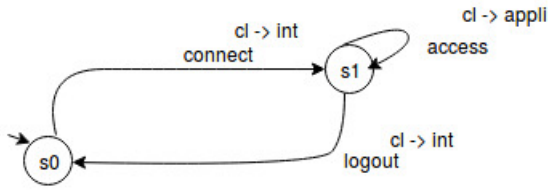


FIGURE *7.3 – Accès à une application web.*

Le Listing 7.1 résume le comportement du système décrit en Figure 7.3 par instanciation du modèle générique de la composition réalisable des *CP*.

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 7.1 – Accés á une application web

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

Le protocole d'accès à une base de données via une application web est une extension de CS1. Le *CP* est emprunté à [46]. Cela implique une entité 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 7.4 n'est pas réalisable, mais une réparation est donnée dans [46].

Le Listing 7.2 instantie le comportement du *CP* décrit dans la Figure 7.4.

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

$$
\begin{aligned}
axm5 : CPs\_B = \{&s0 \mapsto cl \mapsto connect \mapsto int \mapsto s1 \mapsto 1,\\
&s1 \mapsto int \mapsto setup \mapsto appli \mapsto s2 \mapsto 2,\\
&s2 \mapsto cl \mapsto access \mapsto appli \mapsto s2 \mapsto 3,\\
&s2 \mapsto cl \mapsto logout \mapsto int \mapsto s3 \mapsto 4,\\
&s3 \mapsto appli \mapsto log \mapsto db \mapsto s0 \mapsto 5,\\
&s0 \mapsto Pend \mapsto End \mapsto Pend \mapsto s4 \mapsto 6\\
&\}
\end{aligned}
$$

End

Listing 7.2 – Accés á une application web

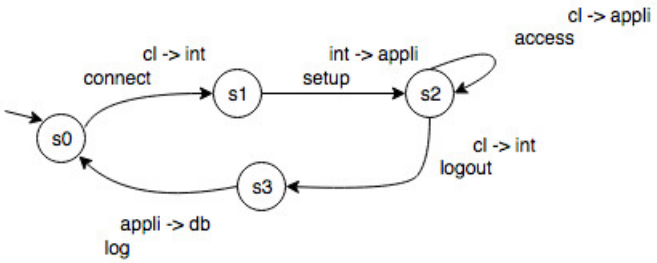


FIGURE *7.4 – Accès à une base de données via une application Web.*

1.5 CS5. Application d'achat en ligne

L'application d'achat en ligne décrite en Figure 7.5 est non réalisable. Elle est empruntée à [72], 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*. Si le paiement est terminé avec succès, l'application se termine dés que, la banque confirme le paiement au vendeur et au client, sinon ce paiement est annulé. La description formelle du *CP* de la Figure 7.5 est donné par le Listing 7.3.

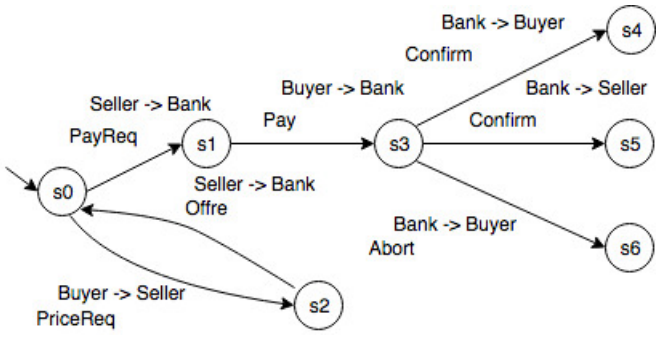


FIGURE *7.5 – Chorégraphie non réalisable d'achat en ligne.*

1.6 CS6. Protocole de visa de travail australien

La Figure 7.6 montre une représentation graphique d'un protocole pour un processus de demande de visa de travail australien emprunté à [80]. 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 (état s_8) en remplir 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 (même état) remplir la demande et fournir une lettre de recommandation de l'employeur.

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 7.3 – Accés á une application web

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 termine à l'état s_9 ou s_{10}.

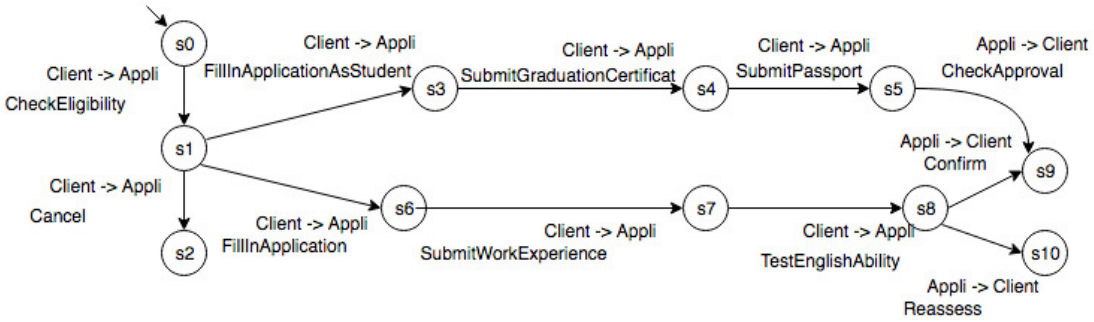


FIGURE *7.6 – Protocole de visa de travail australien.*

Le Listing 7.4 représente l'instance du comportement du *CP* de la Figure 7.4.

1.7 CS7. Protocole d'un jeu fictif

Le protocole d'un jeu fictif emprunté à [58], représenté sur la Figure 7.7, 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.

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 7.4 – Accés á une application web

2. Si B gagne (le message *bwin* se trouve en tête de la file d'attente AB et 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 informées 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.

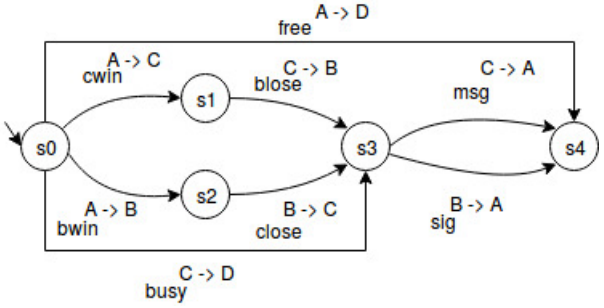


FIGURE 7.7 – *Protocole d'un jeu fictif.*

Le Listing 7.5 montre le contexte B-événementiel défini pour instancier le modèle générique de cette étude de cas.

2 Cas d'étude de réparation de CP non réalisables

Dans cette section, notre objectif est de montrer que la réparation proposée dans le Chapitre 6 est valide sur plusieurs études de cas de la littérature. Tous ces études de cas ont été instanciés dans modèle de réparation B-événementiel. Ces instances définissent les extensions d'ensembles arbitraires utilisés dans le modèle générique de réparation.

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

Listing 7.5 – Accés á une application web

2.1 CS8. Réparation de CS4

Le protocole de conversation décrit sur la Figure 7.8 représente une réparation possible du *CP* non-réalisable de la Figure 7.4.

Le *CP* non-réalisable est emprunté à [46]. Cela implique l'introduction de nouvelles transitions de synchronisation entre l'entité *appli* et le client *cl*, et une seconde synchronisation entre l'entité interface *int* et l'aplication *appli*. Le *CP* décrit restaure la réalisabilité. L'illustration du bon fonctionnement du *CP* après réparation est détaillé dans le Listing 7.6.

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

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

Listing 7.6 – Accés á une application web

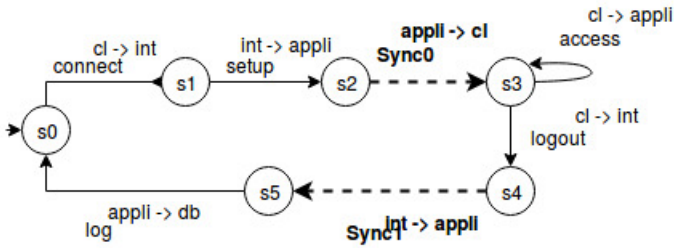


FIGURE *7.8 – Accès à une application web, CP réparé.*

2.2 CS9. Réparation de CS5

Le protocole de conversation d'achat en ligne de la Figure 7.5 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 7.9 avec les deux transitions $Sync0$ et $Sync1$.

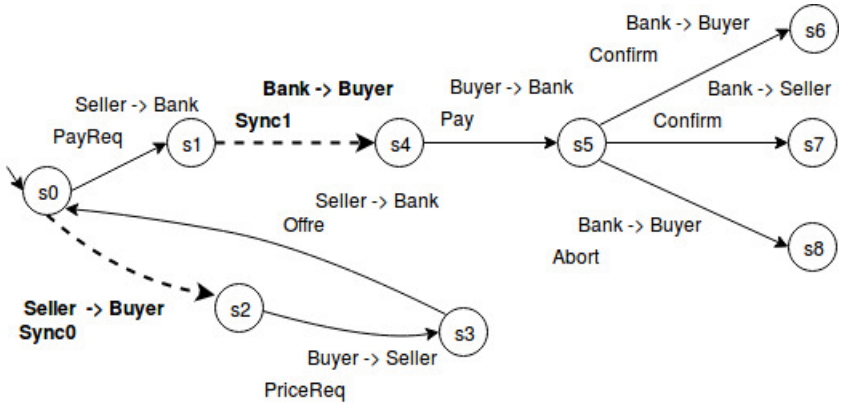


FIGURE *7.9 – Chorégraphie d'un achat en ligne.*

Le listing 7.7 formalise le *CP* réparé de la Figure 7.9

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

$axm5 : CPs\_B = \{s0 \mapsto Seller \mapsto PayReq \mapsto Bank \mapsto s1 \mapsto 2,$
$s2 \mapsto Buyer \mapsto PriceReq \mapsto Seller \mapsto s3 \mapsto 3,$
$s1 \mapsto Seller \mapsto Offre \mapsto Bank \mapsto s0 \mapsto 4,$
$s2 \mapsto Buyer \mapsto Pay \mapsto Bank \mapsto s3 \mapsto 6,$
$s3 \mapsto Bank \mapsto Abort \mapsto Buyer \mapsto s4 \mapsto 7,$
$s3 \mapsto Bank \mapsto Confirm \mapsto Seller \mapsto s5 \mapsto 8,$
$\ldots\}$
$axm6 : Repare\_propositions = \{s0 \mapsto Seller \mapsto Sync0 \mapsto Buyer \mapsto s1 \mapsto 1,$
$s0 \mapsto Buyer \mapsto Sync0 \mapsto Seller \mapsto s1 \mapsto 1,$
$s1 \mapsto Seller \mapsto Sync1 \mapsto Buyer \mapsto s3 \mapsto 3,$
$s1 \mapsto Bank \mapsto Sync1 \mapsto Buyer \mapsto s3 \mapsto 3,$
$\}$
End

Listing 7.7 – Accés á une application web

2.3 CS10. Réparation de CS7

Un scénario de réparation possible est représenté dans la Figure 7.10. Le *CP* avant réparation décrit dans la Figure 7.7 est caractérisé comme non-réalisable. Une transition de synchronisation entre les entité A et C est ajouté à l'état initial du *CP* afin de rétablir la réalisabilité du système ($Sync0$ sur Figure 7.10).

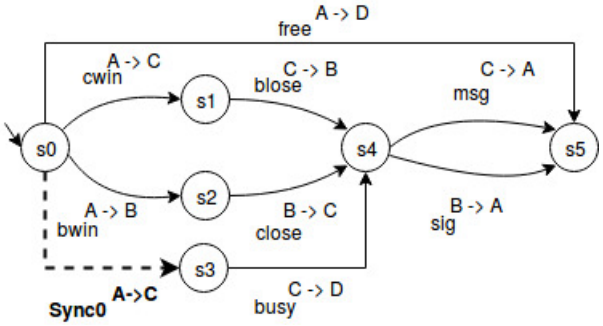


FIGURE *7.10 – Accès à une application web, CP réparé.*

Le listing 7.8 illustre les scénarios de réparation possibles du protocole de jeu fictif non-réalisable.

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

Listing 7.8 – Accés á une application web

2.4 CS11. Réparation de CS2

La Figure 7.11 décrit la réparation du *CP* non réalisable du protocole de transfert de fichier, décrit à la Figure 6.7 et étudié à la Section 1. Dans le *CP* réparé, les transitions de synchronisations $Sync0$ et $Sync1$ sont introduites.

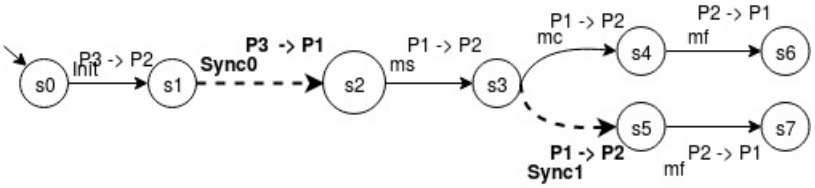


FIGURE *7.11 – CP réparé du protocole de transfert de fichier.*

3 Travaux connexes

La majorité des techniques de vérification et de validation de la réalisabilité dans la littérature est basée sur le processus de model-checking, qui présente des limites à savoir, le passage à l'échelle et le problème d'explosion combinatoire du nombre d'états du système à vérifier. Cela empêche le test de systèmes complexes i.e. des systèmes avec nombre d'entités élevé.

Nous avons bien pris en considération dans nos contributions les deux problèmes majeurs cités au auparavant. Notre approche est fondée sur la preuve mathématique avec 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 que le *CP* d'origine dans les différents niveaux d'abstraction, tout en évitant le passage par la construction des systèmes synchrone et asynchrone comme dans [10]. Cela 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 à l'échelle ainsi que des modèles complexes, et la démarche correcte-par-construction permet d'éviter l'explosion combinatoire.

Nous discutons ci-dessous 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 cas d'études issus de la littérature. La Table 7.1 résume les résultats des différentes études de cas présentées ci-dessus. Elle 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. Elle indique si le *CP* est réalisable ou non-réalisable et donne les détails des opérateurs de composition utilisés pour construire le *CP*. Elle 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 aborder le cas d'un opérateur parallèle en interprétant cet

composition comme un entrelacement utilisant les opérateurs de séquence et de choix, avec

— $\gg$: pour l'opérateur de séquence.
— $+$: pour l'opérateur de choix.
— $\circlearrowleft$: pour l'opérateur de boucle.
— $-$: pour signifier qu'il n'y a pas de propriétés violées. *i.e.* 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. 7.1. Comme indiqué dans le tableau 7.1, 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. Elles sont utilisés comme témoins lors de l'instanciation.

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

Notons que, notre proposition est basée sur des conditions syntaxiques et non sémantique que ça soit pour le cas de composition ou de réparation.

Références	États	Entités	Messages	Transitions	Réalisable/ Non-Réalisable	Opérateurs	Propriétés violé	Automatique vérification	Total des États	Total des Transitions
CS1. Processus d'entreprise virtuelle [77]	10	3	9	10	Réalisable	$\gg, \circlearrowleft$	-	190(63%)	298	322
CS2. Protocole de transfert de fichier [12] web application [46]	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. Accès à une base de données [46]	4	4	5	5	Non-réalisable	$\gg, +, \circlearrowleft$	ISeqF	7(28%)	22	21
CS5. Application d'achat en ligne [72]	7	3	6	7	Non-réalisable	$\gg, +, \circlearrowleft$	ISeqF, PCF	8(30%)	24	23
CS6. Protocole de visa Australien [80]	20	2	21	21	Réalisable	$\gg, +, \circlearrowleft$	-	90(52%)	29	32
CS7. Protocole d'un jeu fictif [58]	5	4	8	8	Non-réalizable	$\gg, +, \circlearrowleft$	PCF	6(62%)	9	8
CS8. Accès à un bdd réparé [46]	6	4	7	7	Réalisable	$\gg, +, \circlearrowleft$	-	110(79%)	108	39
CS9. Application réparé d'achat en ligne [72]	9	3	9	9	Réalisable	$\gg, +, \circlearrowleft$	-	18(48%)	38	29
CS10.Protocole réparé d'un jeu fictif [58]	6	4	9	9	Réalisable	$\gg, +, \circlearrowleft$	-	10(72%)	306	341
CS11. Protocole réparé de transfert [12]	6	4	7	7	Réalisable	$\gg, +, \circlearrowleft$	-	110(79%)	108	39

TABLE *7.1 – Rapport technique des travaux connexes.*

Troisième partie

Conclusion

Chapitre 8

Conclusion et perspectives

L'objectif central de cette thèse est la formalisation et la validation de la composition et de réparation 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 d'analyser le comportement des entités communicantes formant un système distribué.

L'état de l'art étudié nous a permis de classer les différentes méthodes de modélisation des systèmes distribués et d'analyse et de validation des propriétés assurant le bon fonctionnement de ces derniers.

Nous avons distingué trois catégories principales : les approches de vérification et validation de la propriété de réalisabilité, les approches d'évolution et de reconfiguration des *CP* et les approches de réparation des protocoles de conversation.

Ce classement a été effectué selon les limites des propositions face au problème d'explosion combinatoire compte tenu de la complexité élevée des systèmes actuels.

Afin d'éviter les problèmes des approches de validation de la réalisabilité des systèmes basées sur le modèle checking. nous avons proposé un ensemble de conditions nécessaires pour la préservation de la réalisation des *CP* tout en évitant le passage par les étapes de recomposition des *CP* 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 propositions des travaux connexes.

Nous avons identifié des conditions suffisantes qui transfèrent le problème de la vérification au niveau des *CP* via des propriétés suffisantes décrites à un niveau syntaxique.

Notre approche permet la construction des *CP* réalisables. La méthode de composition est inductive (incrémentale) et 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 prédéfinies à savoir, la propriété d'absence des séquences indépendantes et la propriété d'absence des choix parallèles, ce qui garantit la préservation de la réalisation des *CP* résultants.

L'approche de composition a été complètement formalisée en B-événementiel sur

la platforme Rodin. Chaque opérateur de composition prend en entrée un *CP* réalisable et produit en sortie un *CP* également réalisable i.e. chacun des opérateurs de composition garantit 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 qui introduit un ensemble d'invariants pour les différentes propriétés de la réalisabilité à savoir, l'équivalence, la synchronisabilité et la WF (invariants du modèle B-événementiel). Ces propriétés sont préservées par le raffinement.

Nous avons défini un langage permettant la construction incrémentale de *CP* réalisables simples ou complexes à partir d'un ensemble de transitions de base. De plus, nous avons montré que le développement proposé est générique et peut être instancié sur n'importe quel ensemble de *CP* composites et de base.

La solution donnée dans cette thèse est optimale et présente plusieurs avantages par rapport aux propositions du domaine. De plus, notre approche supporte le passage a l'échelle et traite des systèmes de taille arbitraire.

Le raffinement nous a permis aussi de disséminer les preuves sur les différents niveaux d'abstraction (le niveau *CP*, le niveau synchrone et le niveau asynchrone), ce qui facilite la formalisation et la preuve du modèle.

De plus, nous avons proposé une approche également formalisée en B-événementiel qui suggère des corrections pour les *CP* non réalisables en se basant sur les conditions nécessaires définies 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 satisfaction des conditions de réalisation du *CP*. Nous avons aussi formalisé et prouvé cette stratégie de réparation en B-événementiel. L'approche a été validée par plusieurs études de cas issues de la littérature.

Ce travail a ouvert plusieurs perspectives :

- Étendre le modèle avec l'introduction de l'opérateur de composition parallèle qui n'est pas supporté dans le modèle actuel.
- Étendre notre modèle avec de nouveaux opérateurs permettant de composer des *CP* entiers au lieu d'exiger une composition incrémentale en utilisant le protocole de conversation de base.
- Définir l'ensemble des modèles pour les *CP* 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 par ces opérateurs.
- Enfin, nous visons à fournir aux concepteurs un moteur pour l'instanciation automatique des *CP* réalisables.

À plus long terme, nous prévoyons :

- de gérer d'autres systèmes asynchrones car nous avons utilisé un modèle causal dans lequel l'envoi et la réception de messages sont dans le même ordre (anneau).

— d'améliorer l'approche proposée par la définition des conditions nécessaires et suffisantes pour la composition des *CP* réalisables.

Bibliographie

[1] Event B language, Technical report. *Available at http ://rodin.cs.ncl.ac.uk/deliverables/rodinD7.pdf*, (3):496–503, 2005.

[2] Jean-Raymond Abrial : *Modeling in Event-B : System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st édition, 2010.

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

[4] 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.

[5] 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.

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

[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 : 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.

[12] 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.

[13] Samik Basu et Tevfik Bultan : On deciding synchronizability for asynchronously communicating systems. *Theoretical Computer Science*, 656:60–75, 2016.

[14] 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.

[15] Maya Benabdelhafid, Béatrice Bérard et Mahmoud Boufaïda : Analyzing behavioral compatibility for web service choreography using colored petri nets and ask-ctl. 2014.

[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] D. Brand et P. Zafiropulo : On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, 1983.

[26] Daniel Brand et Pitro Zafiropulo : On communicating finite-state machines. *Journal of the ACM (JACM)*, 30(2):323–342, 1983.

[27] 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.

[28] C. Tinelli C. Barrett : *CVC3*. International Conference on Computer Aided Verification (CAV '07), Berlin, Germany, 19th édition, 2007.

[29] M Emilia Cambronero, Gregorio Di, Hermenegilda Macià *et al.* : A petri net approach for the design and analysis of web services choreographies. *The Journal of Logic and Algebraic Programming*, 78(5):359–380, 2009.

[30] E. Clarke, O. Grumberg, S. Jha, Y. Lu et H. Veith : Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50, 2003.

[31] L. M. de Moura et N. Bjorner : *Z3 : An Efficient SMT Solver*. Lecture Notes in Computer Science, 2008.

[32] G. Decker et M. Weske : Local Enforceability in Interaction Petri Nets. *In Proc. of BPM'07*, volume 4714 de *LNCS*. Springer, 2007.

[33] Pierre-Malo Deniélou et Nobuko Yoshida : Multiparty session types meet communicating automata. *In European Symposium on Programming*, pages 194–213. Springer, 2012.

[34] 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.

[35] 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.

[36] 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.

[37] E. Contejean F. Bobot, S. Conchon et S. Lescuyer : *Implementing Polymorphism in SMT solvers*. International Workshop on Satisfiability Modulo, 6th édition, 2008.

[38] 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.

[39] 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.

[40] 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.

[41] Alain Finkel et Etienne Lozes : Synchronizability of communicating finite state machines is not decidable. *arXiv preprint arXiv :1702.07213*, 2017.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] Nawal Guermouche : *Etude des interactions temporisées dans la composition de services web*. Thèse de doctorat, 2010.

[49] David Harel : Statecharts : A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[50] 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.

[51] Charles Antony Richard Hoare : An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[52] Gerard J Holzmann : *The SPIN model checker : Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

[53] 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.

[54] J. E. Hopcroft et J. D. Ullman : *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

[55] Ivan J Jureta, Stéphane Faulkner et Philippe Thiran : *Dynamic Requirements Specification for adaptable and open Service-oriented Systems*. Springer, 2007.

[56] Pistore M. Kazhamiakin, R. : Analysis of realizability conditions for web service choreographies. volume 4052 de *LNCS*, page 61–76. Springer, 2003.

[57] Leslie Lamport : *Specifying systems : the TLA+ language and tools for hardware and software engineers*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[58] Julien Lange, Emilio Tuosto et Nobuko Yoshida : From Communicating Machines to Graphical Choreographies. Proceedings of the 42nd Annual ACM POPL, pages 221–232, 2015.

[59] Julien Lange et Nobuko Yoshida : Verifying asynchronous interactions via communicating session automata. *arXiv preprint arXiv :1901.09606*, 2019.

[60] 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.

[61] Michael Leuschel et Michael Butler : Prob : A model checker for b. *In International Symposium of Formal Methods Europe*, pages 855–874. Springer, 2003.

[62] 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.

[63] Hong Liu et Raymond E Miller : Generalized fair reachability analysis for cyclic protocols with nondeterministic and internal transitions. *In Proceedings of International Conference on Network Protocols*, pages 6–13. IEEE, 1995.

[64] N. Lohmann et K. Wolf : Realizability Is Controllability. *In Proc. of WS-FM'09*, volume 6194 de *LNCS*. Springer, 2010.

[65] ISO Lotos : A formal description technique based on the temporal ordering of observational behaviour. *ISO8807, 1XS989*, 1989.

[66] Rajit Manohar et Alain J. Martin : Slack elasticity in concurrent computing. *In Mathematics of Program Construction, (MPC)*, pages 272–285, 1998.

[67] Nenad Medvidovic : ADLs and dynamic Architecture Changes. *In Proc. of SIGSOFT'96 workshops*, pages 24–27. ACM, 1996.

[68] Robin Milner : *A Calculus of Communicating Systems*, volume 92 de *Lecture Notes in Computer Science*. Springer, 1980.

[69] Tobias Nipkow, Lawrence C Paulson et Markus Wenzel : *Isabelle/HOL : a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[70] Sam Owre, John M Rushby et Natarajan Shankar : Pvs : A prototype verification system. *In International Conference on Automated Deduction*, pages 748–752. Springer, 1992.

[71] C. Peltz : Web services orchestration and choreography. volume 4052 de *LNCS*, page 46–52. Springer, 2003.

[72] 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.

[73] Z. Qiu, X. Zhao, C. Cai et H. Yang : Towards the Theoretical Foundation of Choreography. *In Proc. of WWW'07*. ACM Press, 2007.

[74] D. Delahaye R. Bonichon et D. Doligez. Zenon : *An Extensible Automated Theorem Prover Producing Checkable Proofs*. In Dershowitz and Voronkov.

[75] A. Riazanov et A. Voronkov : *The design and implementation of Vampire*. Journal of AI Communicationse, 2002.

[76] S. Rinderle, A. Wombacher et M. Reichert : On the Controlled Evolution of Process Choreographies. *In Proc. of ICDE'06*, pages 124–124. IEEE, 2006.

[77] S. Rinderle, A. Wombacher et M. Reichert : On the Controlled Evolution of Process Choreographies. Proc. of ICDE, pages 100–124, 2006.

[78] 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.

[79] N. Roohi et G. Salaün : Realizability and Dynamic Reconfiguration of Chor Specifications. *Informatica (Slovenia)*, 35(1):39–49, 2011.

[80] 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.

[81] G. Salaün et T. Bultan : Realizability of Choreographies using Process Algebra Encodings. *In Proc. of IFM'09*, volume 5423 de *LNCS*. Springer, 2009.

[82] Gwen Salaün et Nima Roohi : On realizability and dynamic reconfiguration of choreographies. *Proc. of WASELF*, 9, 2009.

[83] Alceste Scalas et Nobuko Yoshida : Less is more : multiparty session types revisited. *Proceedings of the ACM on Programming Languages*, 3(POPL):30, 2019.

[84] 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.

[85] Jocelyn Simmonds, Yuan Gan, Marsha Chechik, Shiva Nejati, Bill O'Farrell, Elena Litani et Julie Waterhouse : Runtime monitoring of web service conversations. *IEEE Transactions on Services Computing*, 2(3):223–244, 2009.

[86] J Michael Spivey et JR Abrial : *The Z notation*. Prentice Hall Hemel Hempstead, 1992.

[87] Jun Sun, Yang Liu, Jin Song Dong, Geguang Pu et Tian Huat Tan : Model-based methods for linking web service choreography and orchestration. *In 2010 Asia Pacific Software Engineering Conference*, pages 166–175. IEEE, 2010.

[88] 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.

[89] RESULT OF VOTING : Information technology–z formal specification notation–syntax, type system and semantics. 2002.

[90] CH West : Protocol validation by random state exploration. *In Protocol Specification, Testing, and Verification*, pages 233–242, 1987.

[91] John Whitbeck, Marcelo Dias de Amorim, Vania Conan et Jean-Loup Guillaume : Graphes d'accessibilité dynamiques. 2012.

[92] Andreas Wombacher : Alignment of Choreography Changes in BPEL Processes. *In Proc. of SCC'09*, pages 1–8. IEEE, 2009.

[93] Zhaohui Wu, Shuiguang Deng, Ying Li et Jian Wu : Computing compatibility in dynamic service composition. *Knowledge and information systems*, 19(1): 107–129, 2009.

[94] 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