Verification of cryptographic protocols with lists of unbounded lengths

Miriam Paiola

Université Paris Diderot (Paris 7) Sorbonne Paris Cité

Ecole Doctorale de Science mathématiques de
Paris Centre - ED 386

Verification of cryptographic protocols with lists of unbounded lengths

Vérification de protocoles cryptographiques avec listes de longueur non bornée

Thèse

dirigée par Bruno Blanchet

pour l'obtention du

Doctorat de l'Université Paris Diderot

spécialité : Informatique

présentée et soutenue publiquement *le 28 Mai 2014* par

Miriam Paiola

Laboratoire: INRIA Paris-Rocquencourt, Prosecco

Devant le jury composé de:

Directeur de thèse :	Bruno Blanchet	-	INRIA (Prosecco)
Rapporteurs :	Stéphanie Delaune	-	ENS Cachan
	Matteo Maffei	-	Universität des Saarlandes
Examinateurs :	Roberto Amadio	-	Université Paris Diderot
	Cédric Fournet	-	Microsoft Research Cambridge
	Ralf Küsters	-	Universität Trier
	Michaël Rusinowitch	-	INRIA, LORIA

Remerciements

Je souhaite remercier mon directeur de thèse, Bruno Blanchet, en premier lieu pour m'avoir accueillie à Paris pour mon mémoire de master et ensuite pour m'avoir guidée pendant ces années de thèse. J'ai toujours apprécié sa disponibilité, sa tranquillité et son aide. Merci pour le temps que vous m'avez consacré.

Ensuite, ma grande reconnaissance se porte vers mes rapporteurs, Stéphanie Delaune et Matteo Maffei, pour avoir accepté de lire ce manuscrit. Merci aussi aux autres membres du jury.

Je remercie chaleureusement Mark Ryan et Myrto Arapinis pour m'avoir hébergée pendant trois mois à Birmingham. Les discussions avec eux m'ont ouvert sur un sujet différent de celui de ma thèse.

J'ai commencé cette thèse au sein de l'équipe de Crypto de l'ENS et je l'ai terminé dans l'équipe Prosecco. Je remercie tous les membres des deux équipes pour leur convivialité. En particulier, je tiens à remercier Karthikeyan Bhargavan pour les suggestions techniques par rapport aux enveloppes SOAP. Grazie aux autres deux membres de la mafia italienne, Elizabeth et Angelo pour les bons moments passés ensemble. Thanks à Ben Smyth, non seulement pour les discussions techniques, mais aussi pour avoir toujours partagé avec moi les conseils appris quand il était thésard lui même (beer ?).

Une thèse n'est qu'une partie de la vie de quelqu'un...mais le reste est également important pour l'accomplir sans soucis ! Par conséquent, je remercie les amis italiens à Paris (parce que on est *a casa* avec eux), les étrangers (la recherche n'est pas lié que à l'académie, mais aussi aux découvertes sociales et culturelles) et finalement les amis en Italie parce qu'ils ne m'ont pas (encore ?) oubliée. Entre tous ces amis, j'en remercie en particulier deux pour le temps partagé ensemble : Caroline et Ilaria, grâce à vous la collocation n'a pas été seulement une cohabitation, mais aussi une belle amitié.

Mes derniers remerciements vont à mes parents qui, même d'Italie, ont toujours vécu ma vie quotidiennement. Je ne pourrai jamais les remercier suffisamment pour les soirées passées sur Skype avec moi, à me soutenir, écouter, et comprendre. (Et bien sûr, pour être ici aujourd'hui !) Un merci vaut plus que mille mots. Grazie.

Contents

Chapitre 1

Introduction

Contents

If you do not read French, you can find this chapter in English on page 23.

Les protocoles cryptographiques entourent notre vie : quand on retire de l'argent à un guichet, ou qu'on fait un achat sur internet, ou encore quand on utilise son téléphone, les échanges de messages sont réglés par des protocoles cryptographiques qui permettent de sécuriser les communications. Pour garantir la confidentialité de la communication, il faut que les messages soient chiffrés grâce à une clé. En faisant cela, on rend le message incompréhensible pour quiconque ne posséde pas la clé. On pourrait penser que la fiabilité de ces protocoles soit facile à assurer. Ce n'est pas le cas : de nouvelles attaques sont découvertes régulièrement et parfois sur des protocoles connus depuis des années. Par exemple le protocole de Needham-Schroeder à clé publique [41], développé en 1978, a été attaqué que seulement en 1995, par Gavin Lowe [34] : on a attendu 17 ans avand de découvrir que ce protocole avait une faille. Un autre exemple est le protocole Transport Layer Security (TLS) : il est le protocole le plus utilisé pour garantir la sécurité sur Internet. Malgré son usage répandu, il est sujet à de nombreuses attaques importantes. Pour augmenter la fiabilité de ces protocoles, on les vérifie formellement : on vérifie des propriétés de sécurité en considérant les primitives cryptographiques comme des boîtes noires.

1.1 Protocoles Cryptographiques

1.1.1 Primitives Cryptographiques

Le but principal de la cryptographie est de protéger ou cacher un message, si bien qu'il devient illisible à quiconque n'est pas autorisé à le lire. La conversion d'un un message, appelé le *message en clair*, en un texte illisible, le *texte chiffré*, est appelée *chiffrement*. L'opération inverse qui permet de décoder du texte illisible est le *déchiffrement*. Ces opérations utilisent une clé : il s'agit d'un secret (idéalement connu seulement par l'émetteur et les destinataires) nécessaire pour chiffrer ou déchiffrer le message. Pour un message initial m, on utilise la notation $\{\!|m|\!\}_k$ pour le chiffrement de m sous la clé k. On peut diviser le domaine de cryptographie en deux grands secteurs : le chiffrement *à clé symétrique* et le chiffrement *à clé publique*.

Le chiffrement à clé symétrique prévoit que l'émetteur et le destinataire du message partagent une clé secrète k : l'émetteur chiffre le message avec la clé secrète k et le destinataire déchiffre avec la même clé k. Idéalement, un troisième agent ne devrait pas réussir à déchiffrer le chiffré s'il ne connaît pas la clé secrète partagée entre l'émetteur et le destinataire. L'un des plus anciens systèmes de chiffrement à clé symétrique est le *chiffrement de César*, décrit pour la première fois par l'historien roman Gaius Suetonius Tranquillus dans *Vie des douze Césars, (De vita caesarum)* et utilisé par Jules César pour sa correspondance privée :

> *Exstant et ad Ciceronem, item ad familiares domesticis de rebus, in quibus, si qua occultius perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum verbum effici posset ; quae si qui investigare et persequi velit ; quartam elementorum litteram, id est D pro A et perinde reliquas commutet.*
>
> On possède enfin [ses lettres] à Cicéron, et aussi à ses proches sur des affaire domestiques, dans lesquelles, s'il voulait garder quelque chose secret, il écrivait en chiffré, c'est-à-dire, en changeant l'ordre des lettres de l'alphabet, de telle sorte qu'aucun mot ne pouvait être deviné. Si quelqu'un voulait les déchiffrer et les lire, il devait remplacer la quatrième lettre de l'alphabet, en d'autres termes un D par un A et ainsi de suite.

Comme Suetonius le dit, ce simple système de chiffrement consistait à décaler de façon circulaire chaque lettre de quelques crans . La longueur du décalage est la clé de chiffrement et aussi celle de déchiffrement. Par exemple, on peut chiffrer "Ave Caesar" en "Ezi Geiwev" avec clé 4, c'est-à-dire en remplaçant $A \rightarrow E$, $B \rightarrow F$,

..., $Z \rightarrow D$. Pour déchiffrer ce message, il faut inverser le décalage des lettres, $E \rightarrow A, F \rightarrow B, \ldots, D \rightarrow Z$. De toute évidence, on peut facilement casser cet algorithme en essayant toutes les clés possible, c'est-à-dire les 26 décalages possibles. Une autre façon de casser cet algorithme et aussi d'autres bien plus compliqués est d'effectuer une analyse statistique sur le chiffré. Par exemple, les analyses fréquentielles étudient la fréquence d'apparition des lettres ou des groupes de lettres dans un chiffré. Même en utilisant des systèmes de chiffrement plus compliqués, un intrus peut deviner la clé secrète : il peut essayer toutes les possibilités en effectuant une *attaque par force brute*. C'est le cas du Data Encryption Standard (DES), publié comme Federal Information Processing Standard par les États-Unis en 1977 : dans les années 1990, pendant que cet système était largement utilisé, il fut montré que cette méthode était vulnérable aux attaques par force brute à cause de la longueur trop courte des clés.

Le désavantage principal du chiffrement à clé symétrique est la *gestion des clés* : chaque paire d'agents doit partager et garder une clé secrète, par conséquent le nombre de clés augmente quadratiquement par rapport au nombre d'agents. En outre, quand deux agents veulent communiquer, il doivent avoir échangé la clé partagée avant : cet échange peut être fait seulement si ils ont déjà access à une chaîne sécurisée ou s'ils partagent déjà une clé.

Le chiffrement à clé publique (aussi dit *chiffrement asymétrique*) prévoit que la clé utilisée par le destinataire du message pour déchiffrer soit différente de celle utilisée par l'émetteur du message pour chiffrer. Cela signifie que chaque agent possède une paire de clés : la clé *privée* est connue seulement par le propriétaire et est utilisée pour déchiffrer et la clé *publique* est partagée avec quiconque veut chiffrer des messages. Bien que différentes, les deux parties de cette paire de clés sont liées mathématiquement : les systèmes de chiffrement à clé publique sont en effet basés sur des problèmes mathématiques qui n'admettent pas de solution efficace, ce qui veut dire que pour un agent il est facile de générer sa paire de clés et l'utiliser pour chiffrer et déchiffrer. Cependant, il est difficile pour un intrus de déterminer la clé privée d'un agent à partir de sa clé publique. Par conséquent, on peut publier la clé publique sans compromettre la sécurité, alors que la clé privée doit rester secrète. En 1976, W. Diffie et M. Hellman [25] définirent la notion de chiffrement à clé publique, sans pourtant donner aucun système réel. Ce fut seulement en 1978 que le premier système de chiffrement à clé publique fut presenté, quand R.L. Rivest, A. Shamir and L. Adleman [47] inventèrent RSA.

Contrairement aux systèmes de chiffrement à clé symétrique, les systèmes de chiffrement à clé publique résolvent le problème de la gestion des clés : le nombre de clés nécessaires est linéaire dans le nombre d'agents et l'échange de clés n'est plus nécessaire avant la communication.

Une autre primitive cryptographique, appelée *signature numérique*, sert à démontrer l'authenticité d'un message numérique ou d'un document. Une signature valide donne au destinataire raison pour croire que le message a été créé par un émetteur connu et qu'il n'a pas été modifié pendant la communication. De cette façon l'émetteur ne peut pas nier avoir envoyé le message. Les signatures numériques utilisent aussi deux clés : une clé publique de vérification et une clé privée de signature. Un agent signe un document avec sa clé privée de signature et n'importe qui peut vérifier le document avec la clé publique de vérification de l'agent. Contrairement au chiffrement à clé publique, les signatures numériques ne cachent pas le contenu du document. Pourtant il est possible de combiner les deux primitives et obtenir un document chiffré et signé.

1.1.2 Protocoles

Même en supposant que les primitives cryptographiques soient *parfaites*, c'est-à-dire, qu'il soit impossible de déchiffrer un message chiffré sans connaître la clé correspondante, on ne peut pas garantir l'absence d'attaques dans la communication. Considérons, par exemple, le protocole suivant, une version simplifiée du protocole de Needham-Schroeder à clé publique [41].

$$
\begin{aligned}
&(1)\ A \rightarrow B : \{\!|(N_A, A)|\!\}_{pk_B} \\
&(2)\ B \rightarrow A : \{\!|(N_A, N_B)|\!\}_{pk_A} \\
&(1)\ A \rightarrow B : \{\!|(N_B)|\!\}_{pk_B}
\end{aligned}
$$

Ce protocole décrit l'échange de messages entre deux *participants*, qu'on appellera Alice et Bob. Chaque participant possède une paire de clés publique et privée, (pk_A, sk_A) pour la paire d'Alice et (pk_B, sk_B) pour celle de Bob. D'abord Alice crée aléatoirement un nombre à usage unique, dit *nonce*, N_A, elle le couple avec sa propre identité A et chiffre la paire avec la clé publique de Bob, pk_B. Alice envoie ensuite le message 1 à Bob. Quand Bob reçoit ce message, il déchiffre avec sa clé privée sk_B et obtient N_A. Ensuite, il crée aléatoirement un nonce N_B et envoie à Alice le chiffrement avec pk_A de la paire de N_A et N_B. Finalement, Alice reçoit le message 2, le déchiffre et renvoie N_B chiffré avec pk_B. Comme Bob est le seul à connaitre sk_B, quand Alice reçoit le message 2, elle est convaincue qu'elle communique avec lui, parce que seul Bob peut déchiffrer le message 1 et obtenir N_A. De la même façon, à la fin du protocole Bob est convaincu de communiquer avec Alice. Cela signifie que les deux participants sont *authentifiés*.

Cependant, cela est vrai seulement si Alice parle avec des participants honnêtes. En effet, on peut découvrir l'attaque suivante, décrite en 1995 par Gavin Lowe [34] et illustré à la Figure 1.1. Cette attaque est dite *attaque de l'homme du milieu* et suppose la présence d'un troisième participant malhonnête, Charles, qui possède la

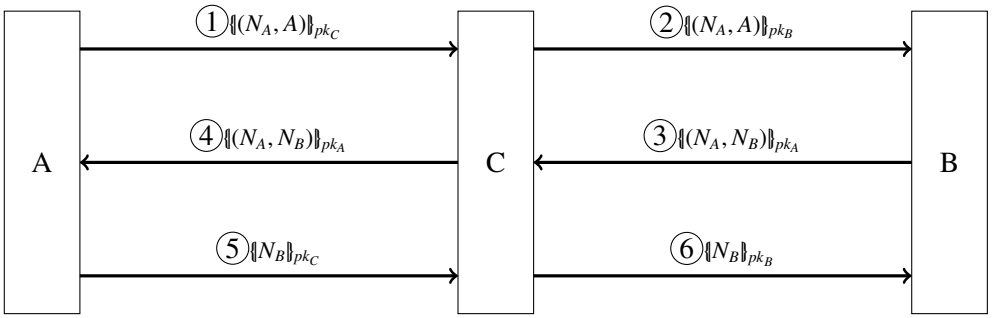

FIGURE 1.1 – Une attaque sur le protocole de Needham-Schroeder

paire de clés (pk_C, sk_C). Au début du protocole, Alice commence une session avec Charles et envoie le message ①. Charles rechiffre ce message avec la clé publique de Bob et envoie ensuite le message ② à Bob. Quand Bob reçoit le premier message, il pense que l'émetteur est Alice, donc il répond avec le message ③, que Charles transfère à Alice. Remarquez que Charles à ce moment ne peut pas obtenir N_B en déchiffrant le message ③, qui est destiné à Alice. Quand elle reçoit le message ④, Alice reconnaît son nonce N_A, donc elle pense que Charles a répondu correctement. Ensuite elle renvoie N_B chiffré avec la clé publique de Charles, qui finalement découvre N_B sans déchiffrer le message ②. Maintenant, Charles peut rechiffrer N_B avec la clé publique de Bob et terminer correctement la session avec Bob. Dans [34], Lowe donne aussi une version correcte du protocole, qui est maintenant connu comme le protocole de Needham-Schroeder-Lowe. La correction prévoit que le deuxième message du protocole soit modifié : Lowe ajoute l'identité de Bob à la paire chiffrée, c'est-à-dire qu'il remplace $\{|(N_A, N_B)|\}_{pk_A}$ par $\{|(N_A, N_B, B)|\}_{pk_A}$.

1.1.3 Propriétés de Sécurité

Comme expliqué dans la section précédente, le protocole de Needham-Schroeder ne garantit pas l'*authentification* des participants. L'authentification est une des propriétés de sécurité qu'un protocole devrait garantir. Les plus importantes sont le *secret* et l'*authentification* et seront traitées dans cette thèse.

Secret Le secret a comme but de prévenir la divulgation d'informations à des agents non autorisés. Un protocole préserve le secret d'une donnée s seulement si des agents non autorisés ne peuvent pas l'apprendre. Par exemple, dans le protocole de Needham-Schroeder, Charles apprend le nonce N_B sans être autorisé à déchiffrer le message $\{|(N_A, N_B)|\}_{pk_A}$. De ce fait, le secret de N_B n'est pas garanti par

le protocole. Dans cette thèse on utilisera la notion de secret suivante : un protocole P garantit le secret du message M si P ne publie pas M, ni rien qui permettrait de calculer M. Une notion plus forte de secret est liée au concept de indistinguabilité [2, 3] : le *secret fort* (*strong secrecy* en anglais) indique qu'un attaquant ne peut pas détecter quand la valeur du secret change. Cela signifie que un message s est secret si la session qui utilise s est indistinguable d'une session qui utilise un message s' à la place de s.

Authentification De manière informelle, l'authentification permet de vérifier l'identité d'un individu, c'est-à-dire, de déterminer si un individu est en effet celui qu'il prétend d'être. Le problème de définir formellement une notion d'authentification est que cette propriété dépend de l'utilisation du protocole. Par exemple, pour le protocole de Needham-Schroeder, on peut utiliser la propriété suivante : si Bob termine le protocole apparemment avec Alice, alors Alice a commencé le protocole apparemment avec Bob. Dans [35], G. Lowe donne plusieurs définitions formelles de authentification. Dans cette thèse, la propriété d'authentification qu'on considère est l'*accord non injectif* : si un participant B termine le protocole apparemment avec A avec certains paramètres, alors A a commencé le protocole apparemment avec B avec les mêmes paramètres.

1.2 Vérification des Protocoles

Comme on a montré dans la Section 1.1.2, même si les primitives cryptographiques sont correctes, des attaques sont encore possibles à cause des défauts dans la conception des protocoles. La fiabilité des protocoles de sécurité peut être augmentée grâce à une analyse formelle qui prouve les propriétés de sécurité souhaitées. Pour effectuer cette vérification, on doit représenter les protocoles mathématiquement. Il y a deux modèles pour décrire les protocoles.

Modèle calculatoire Dans le modèle calculatoire, les messages envoyés sur le réseau sont considérés comme des séquences binaires (*bitstrings*, en anglais). Les primitives cryptographiques sont représentées par des algorithmes polynomiaux sur ces séquences. Le comportement d'un adversaire est modélisé comme une machine de Turing probabiliste. Ces modèles sont proches de la réalité et offrent des garanties fortes sur la sécurité des protocoles cryptographiques. Cependant, les preuves sont d'habitude difficiles, sujettes aux erreurs et jusqu'à il y a peu de temps encore elles étaient manuelles. Plusieurs outils ont été développés pour obtenir des preuves de sécurité dans le modèle calcu-

latoire. CryptoVerif [14] produit des preuves par suites de jeux pour des protocoles écrits dans une algèbre de processus. CertiCrypt [11] et EasyCrypt [10] sont deux outils qui permettent de construire et vérifier des preuves cryptographiques par suites de jeux. Leurs technologies sont différentes : CertiCrypt repose sur l'assistant de preuve Coq, tandis que EasyCrypt utilise des solveurs SMT et des Démonstrateurs automatiques de théorèmes.

Modèle symbolique Dans le modèle symbolique, aussi appelé modèle de Dolev-Yao [26], les primitives cryptographiques sont représentées par des fonctions et sont considérées comme des boîtes noires, les messages sont des termes sur ces fonctions et l'adversaire peut utiliser seulement des primitives définies. Ce modèle suppose une hypothèse de chiffrement parfait : l'adversaire ne peut pas casser les primitives. Ce modèle, même s'il est moins réaliste, est plus facile à automatiser, en calculant tous les messages (termes) que l'adversaire peut déduire. Plusieurs techniques ont été conçues pour la vérification dans le modèle symbolique. ProVerif [1, 16] établit des propriétés de sécurité des protocoles pour un nombre de sessions non bornée ; pour cette raison il peut ne pas se terminer. Cependant, quand une propriété ne peut pas être prouvée, ProVerif peut construire une description de l'attaque. Scyther [23] garantit la terminaison, mais il peut analyser seulement un nombre borné de sessions quand un résultat ne peut pas être obtenu pour un nombre non borné. AVISPA [5] fournit une interface commune pour plusieurs outils d'analyse et aide à trouver des attaques sur les protocoles.

A cause des abstractions faites dans le modèle symbolique, les garanties offertes par la vérification d'une propriété de sécurité sont normalement plus faibles que les garanties fournies par la vérification de la même propriété dans le modèle calculatoire. D'habitude une attaque dans le modèle symbolique implique une attaque dans le modèle calculatoire, cependant la réciproque n'est pas vraie dans tous les cas. Récemment, plusieurs travaux ont essayé de relier les deux modèles et prouver que, sous des hypothèses supplémentaires, si un protocole est sûr dans le modèle symbolique, alors il est sûr aussi dans le modèle calculatoire. Un premier résultat qui traite de cette *correction calculatoire* (*computational soundess*, en anglais) a été réalisé par Abady and Rogaway [4]. Ils ont montré que, si deux expressions sont équivalentes dans le modèle symbolique, elles sont aussi indistinguable dans le modèle calculatoire, quand la seule primitive utilisée est le chiffrement symétrique et sous des restrictions supplémentaires (en particulier, ils considèrent seulement un intrus passif, c'est-à-dire, un intrus qui peut seulement écouter le réseau, mais qui ne peut pas modifier les messages). Ce premier résultat a suscité plusieurs extensions. Le résultat de Micciancio et Warinschi [40] montre la correspondance entre états et traces dans les deux modèles pour le chiffrement publique et en présence d'un adversaire actif. Dans [21], Comon et Cortier ont montré un résultat qui lie l'équivalence observationnelle dans le modèle symbolique à l'indistinguabilité calculatoire

en présence d'un adversaire actif, pour des protocoles qui utilisent le chiffrement symétrique. Pour plus de détails sur d'autres résultats de correction calculatoire, on renvoie le lecteur à [22].

1.3 Vérification des Protocoles avec Listes de Longueur Non Bornée

La vérification formelle des protocoles de sécurité avec des structures de données de taille fixée a été considérablement étudiée. Cependant, certains protocoles, comme par exemple les protocoles XML pour les services web, utilisent des structures de données plus compliquées, telles que des listes. La vérification des protocoles qui manipulent ces structures de donnés a été moins étudiée et présente des difficultés supplémentaires : ces structures complexes ajoutent une autre cause d'indécidabilité. Par ailleurs, certains autres protocoles, dites *protocoles de groupe*, décrivent un échange de messages entre un nombre non borné de participants, d'habitude pour établir une clé secrète pour communiquer entre eux sur un réseau peu sûr. La difficulté liée à la vérification des protocoles de groupe vient du fait qu'un protocole de groupe peut effectuer un nombre arbitraire d'étapes, la taille du groupe n'étant pas bornée. De plus, la forme des messages peut dépendre du nombre de participants.

1.3.1 Contexte

Protocoles de groupe La première approche considérée pour prouver des protocoles avec des structures de données récursives a été d' utiliser des démonstrateurs de théorèmes : Paulson [44] et Bryans et al [19] étudient un protocole à authentification récursive pour un nombre non borné de participants, en utilisant Isabell/HOL pour [44], et les fonctions de rang et PVS pour [19]. Cependant, cette approche nécessite un effort humain considérable. Meadows et al [38] ont utilisé l'analyseur de protocoles NRL (NPA), fondé sur une combinaison entre le model checking et les techniques de démonstration de théorèmes, pour vérifier la suite de protocoles Group Domain of Interpretation (GDOI). Comme NPA ne pouvait pas traiter les structures de données infinies requises pour représenter les protocoles de groupe, ils utilisent une seule clé à la place d'une hiérarchie de clés. Plusieurs problèmes, comme par exemple des attaques dues à des défauts de type, ont été découverts sur le protocole et corrigés dans les versions suivantes de GDOI. La première vérification du protocole A.GDH-2 en utilisant NPA [37] semble avoir ignoré des attaques [45], bien que l'outil traite l'élévation à une puissance de Diffie-Hellman [39].

Steel et Bundy [49] ont utilisé CORAL, un outil pour trouver des contre-exemples à des conjectures inductives fausses, pour représenter des protocoles pour l'accord et le management de clé en groupe, sans aucune restriction sur le scénario. Ils ont découvert de nouvelles attaques contre plusieurs protocoles de groupe, mais ils ne peuvent pas prouver qu'un protocole est correct.

Kremer, Mercier et Treinen [30] vérifient le secret pour des protocoles de groupe avec élévation à la puissance modulaire et XOR, pour un nombre de participants quelconque et un nombre de sessions non borné, mais seulement pour un adversaire passif.

Plusieurs travaux considèrent un nombre non borné de sessions. Pereira et Quisquater [45] ont découvert plusieurs attaques sur la suite de protocoles CLIQUES [50], qui étend la méthode pour la mise en accord de clé Diffie-Hellmann à des groupes dynamiques (A-GDH). Ils ont converti le problème de la vérification des propriétés de sécurité en la résolution de systèmes d'équations linéaires. Dans [46], ils ont prouvé un premier résultat d'insécurité pour des protocoles d'authentification : il est impossible de construire un protocole authentifié de mis en accord de clé de groupe correct en partant du protocole A-GDH. Truderung [51] a montré un résultat de décidabilité (en NEXPTIME) pour le secret dans des protocoles récursifs. Ce résultat a été étendu à une classe de protocoles récursifs avec XOR [32] en 3-NEXPTIME. Chridi et al [20] présentent une extension de l'approche basée sur les contraintes dans la vérification symbolique de protocoles pour traiter une classe de protocoles avec listes de longueur non bornée dans les messages. Ils ont prouvé que le problème de l'insécurité pour ces protocoles est décidable pour un nombre borné de sessions.

Protocoles XML Pour ce qui concerne la vérification de protocoles XML, Bhargavan et al [13] ont développé l'outil TulaFale pour vérifier automatiquement des propriétés d'authentification pour des protocoles SOAP. Ils ont proposé un nouveau langage de spécification pour représenter des protocoles de sécurité basés sur SOAP et leur propriétés, ainsi qu'une implémentation qui compiles TulaFale dans le pi calcul appliqué et utilise ensuite ProVerif. Kleiner et Roscoe [48, 29] ont proposé une méthode pour traduire des protocoles WS-Security dans des protocoles de cryptographie standards, en préservant défauts et attaques. En particulier ils ont donné une traduction dans la notation de Casper et ils ont analysé des protocoles avec FDR. Backes et al [8] ont analysé la sécurité du Secure WS-ReliableMessaging Scenario, un scénario pour services web qui combine des éléments de sécurité avec des traits d'un autre standard sur la qualité de service, la messagerie fiable. Dans ce travail, ils donnent deux types d'analyse : une analyse automatique basée sur des techniques de vérification symbolique et une autre basée sur des hypothèses cryptographiques

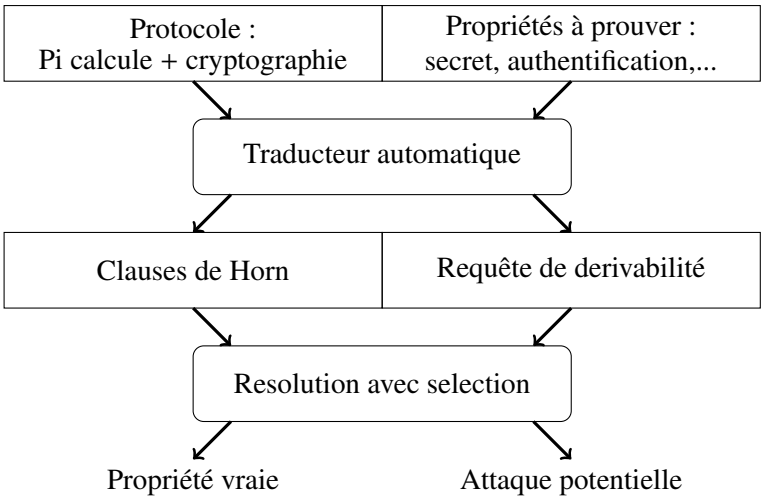

FIGURE 1.2 – ProVerif

explicites sur les algorithmes de cryptographie utilisés. Toutes ces approches ont peu ou pas du tout de support pour des listes de longueur non bornée. Par exemple, TulaFale traite l'appartenance à une liste pour des listes de longueur non bornée, mais rien d'autre.

1.3.2 Contributions de Cette Thèse

Dans cette thèse, nous proposons deux techniques différentes pour vérifier des protocoles de sécurité qui utilisent des listes de longueur non bornée. Les deux résultats sont basés sur le vérificateur automatique ProVerif [1, 16].

La Figure 1.2 montre la structure de ProVerif. ProVerif est un vérificateur de protocoles automatique qui prend en entrée un protocole écrit dans une variante du pi calcul appliqué [2], le traduit dans une représentation en clauses de Horn, et enfin utilise un algorithme de résolution pour déterminer si un fait est dérivable à partir des clauses. On peut en déduire des propriétés de sécurité du protocole. Par exemple, on peut utiliser un fait att(s) pour signifier que l'attaquant peut avoir le message M. Si att(M) n'est pas dérivable à partir de l'ensemble de clauses, alors s est secret. Le but principal de cette approche est de prouver les propriétés de sécurité des protocoles sans limiter le nombre de sessions du protocole. Comme d'autres vérificateurs de protocoles, ProVerif peut analyser des protocoles avec listes si on fixe les longueurs des listes à priori. Cependant, si le protocole est vérifié seulement pour certaines longueurs, on pourrait avoir des attaques pour des longueurs différentes. S'il y a une attaque contre un protocole, alors il existe une longueur telle

que cette attaque apparaît avec listes plus courtes que cette longueur, mais cette longueur dépend du protocole et n'est pas facile à calculer. C'est pour cette raison qu'on doit prouver les protocoles pour des listes de longueur quelconque.

Partie 1 : Listes homogènes Pour la première technique, présentée dans la Partie 1 de cette thèse, on étend le langage des clauses de Horn en introduisant un nouveau type de clauses, les clauses de Horn généralisées, pour pouvoir représenter des listes de longueur quelconque. Ce résultat s'applique à des protocoles qui traitent tous les éléments des listes de la même façon. En raison de cette uniformité, on pourrait penser intuitivement que le secret pour des listes de taille un implique le secret pour des listes de taille quelconque. On montre que cette intuition n'est pas exactement vraie : le secret pour des listes de taille un n'implique pas en général le secret pour de liste de taille quelconque, comme expliqué dans la Section 5.3. Cependant on montre que, pour une certaine classe de clauses de Horn, si le secret est prouvé par notre technique pour des listes de longueur un, alors le secret est vrai aussi pour des listes de longueur non bornée. Ce résultat dépend des abstraction correctes faites par la traduction en clauses de Horn. En outre, on donne un algorithme d'approximation qui peut transformer des clauses de Horn généralisées dans des clauses qui appartiennent à la classe qui valide notre résultat. Tout fait dérivable à partir des clauses initiales est aussi dérivable à partir des clauses créées par l'algorithme d'approximation, donc on peut prouver le secret sur ces dernières clauses, et conclure le secret pour les clauses initiales. Par conséquent, ce résultat donne une méthode facile pour obtenir une forte garantie de secret : on prouve avec ProVerif que att(s) n'est pas dérivable à partir des clauses pour des listes de longueur un, et on peut immédiatement conclure que le secret est valide aussi pour des listes de longueur non bornée, pour un nombre non borné de sessions.

Partie 2 : Listes hétérogènes Dans la deuxième partie de cette thèse, on étend à nouveau le langage des clauses de Horn généralisées, pour pouvoir représenter des listes de longueur quelconque dans des protocoles qui traitent des éléments différents de façons différentes. On adapte l'algorithme de résolution pour qu'il traite ces nouvelles clauses, et on prouve la correction du nouvel algorithme. L'algorithme obtenu peut prouver des propriétés de secret et d'authentification. (Plus précisément, la propriété d'authentification que l'on considère est l'accord non injectif [35] : si un participant B termine le protocole apparemment avec A avec certains paramètres, alors A a commencé le protocole apparemment avec B avec les mêmes paramètres.) Cet algorithme produit des approximations, donc il pourrait échouer à prouver une propriété de sécurité qui en fait est vraie, et ne termine pas toujours. Cette non terminaison est inévitable, parce que le problème de vérifier des protocoles pour un nombre non borné de sessions est indécidable. Pourtant, l'algorithme se com-

porte bien en pratique : on l'a implémenté et on l'a testé avec succès sur plusieurs exemples de protocoles (voir Section 8.3). Enfin, on donne un langage d'entrée pour protocoles plus commode : on étend le langage d'entrée de ProVerif pour modéliser les protocoles avec des listes de longueur non bornée. On donne une sémantique formelle au nouvel calcul de processus, et une traduction automatique en clauses de Horn généralisées. On prouve que cette traduction est correcte. On peut appliquer l'algorithme de résolution aux clauses de Horn généralisées obtenues à partir de cette traduction, pour prouver des propriétés de secret et d'authentification du protocole initial.

Applications des Résultats Les deux techniques présentées dans cette thèse peuvent être utilisées pour vérifier les protocoles suivants :

- certains protocoles de groupe qui utilisent des listes non bornées, avec un nombre non borné de participants. Dans la première partie de la thèse, on traite principalement des protocoles de groupe et on illustre le travail sur le protocole d'Asokan-Ginzboorg [6]. On prouve le secret de la clé de session échangée dans ce protocole en vérifiant avec ProVerif la version du protocole avec listes de longueur un et un groupe de taille un. Dans notre analyse, on considère un attaquant externe : cela suffit pour cet exemple. (Notre outil permet de modéliser des agents corrompus. On détaille ces points dans la Section 4.2.2. On ne considère pas la corruption dynamique dans ce travail). La vérification de protocoles de groupe est aussi une application de la deuxième technique présentée dans cette thèse.
- des protocoles XML pour les services web, puisqu'on peut modéliser les documents XML avec des listes imbriquées. Dans la deuxième partie de la thèse, on traite principalement ce type de protocoles. On illustre notre travail sur l'extension de SOAP [18] aux signatures XML [9]. Les signatures XML permettent de signer plusieurs parties d'un document XML, en ajoutant au document un élément SignedInfo qui peut être modélisé comme une liste de références aux parties signées du document. Une attaque bien connue [36] peut se produire quand le document contient plusieurs éléments Body, et l'élément Body signé n'est pas celui qui est utilisé ensuite. Notre outil trouve cette attaque, et prouve qu'une version corrigée est sûre.

1.3.3 Aperçu de la Thèse

Le prochain chapitre est un rappel de la technique utilisée par ProVerif. Dans le Chapitre 4, on définit formellement les clauses de Horn généralisées, et leur sé-

mantique en en donnant la traduction en clauses de Horn. De plus, on introduit un premier exemple pour motiver l'introduction de ce nouveau type de clauses. Dans le Chapitre 5, on montre notre théorème principal : pour une classe de clauses de Horn généralisées, si att(s) n'est pas dérivable pour des listes de longueur un, alors il n'est pas dérivable pour des listes de longueur quelconque. En outre, on donne un algorithme d'approximation pour transformer les clauses de Horn généralisées dans des clauses qui satisfont les hypothèses de notre théorème.

Le Chapitre 6 ouvre la deuxième partie de la thèse. On introduit un deuxième exemple et on motive l'extension des clauses de Horn généralisées, qu'on utilisera pour modéliser les protocoles qui traitent des éléments différents de façon différentes. On définit formellement les nouvelles clauses de Horn généralisées, et leur sémantique en en donnant la traduction en clauses de Horn. Dans le Chapitre 7, on adapte les définitions de subsumption, résolution et unification aux clauses de Horn généralisées et on prouve que les nouvelles notions sont correctes. Dans le Chapitre 8, on adapte l'algorithme de résolution et on prouve sa correction. Enfin, dans le Chapitre 9, on définit le nouveau calcul de processus et sa sémantique. On donne la traduction automatique d'un processus généralisé en clause de Horn généralisées et on prouve que cette traduction est correcte.

Les résultats présentés dans la Partie 1 correspondent aux publications [42, 43] et ceux des Chapitres 6, 7 et 8 à [17].

CHAPTER 2

Introduction

Contents

Our lives are surrounded by cryptographic protocols: when we withdraw some money, or when we buy something on the internet, or again when use our mobile, the exchange of messages follows a system of digital rules to secure the communication. In order to guarantee the confidentiality of the communication, messages need to be encrypted with a key. In this way the messages become incomprehensible to anyone that does not own the key. One may think that these protocols are always reliable. However this is not the case: new attacks are found regularly, even on protocols used for many years. For example, as we will see in Section 2.1.2, the Needham-Schroeder public-key protocol, introduced in 1978 [41], has been attacked only in 1995 by Gavin Lowe [34]: hence, people thought that the protocol was correct, but in fact it had a flaw. Another example is the Transport Layer Security (TLS) protocol: it is the most widely used security protocol on the Internet. Despite its widespread use, it is subjected to several significant attacks. To increase the confidence in these protocols we need to perform a formal analysis: we can verify security properties by considering the cryptographic primitives as black boxes.

2.1 Cryptographic Protocols

2.1.1 Cryptographic Primitives

The main goal of cryptography is to protect or hide a message, so that it becomes unreadable to anyone except the sender and the intended receivers. The process of converting an ordinary message, called the *plaintext*, into illegible text, the *ciphertext*, is called *encryption*. The reverse operation that allows to decode unintelligible text is called *decryption*. These operations rely on the presence of a *key*: this is a secret (ideally known only by the sender and receivers) needed to encrypt or decrypt the message. For an initial message m, we use the notation $\{\!|m|\!\}_k$ for the encryption of m using the key k. We can divide the field of cryptography into two big areas of study: *symmetric-key* encryption and *public-key* encryption.

In symmetric-key encryption the sender and the receiver of the message share a secret key k: the sender encrypts his message using the secret key k and the reader decrypts the cipher using the same key k. Ideally, a third party cannot decrypt the cipher if he does not know the secret key shared between the sender and the receiver. One of the oldest symmetric encryption scheme is the *Caesar cipher*, first described by the Roman historian Gaius Suetonius Tranquillus in *The Twelve Caesars, (De vita caesarum)* and used by Julius Caesar for his private letters:

> *Exstant et ad Ciceronem, item ad familiares domesticis de rebus, in quibus, si qua occultius perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum verbum effici posset; quae si qui investigare et persequi velit; quartam elementorum litteram, id est D pro A et perinde reliquas commutet.*
>
> There are also [his letters] to Cicero, as well as to his intimates on private affairs, in which, if he wanted to keep something secret, he wrote it in cipher, that is, by changing the order of the letters of the alphabet, so that no word could be made out. If one wished to seek out and read them, he should substitute the fourth letter of the alphabet, that is D for an A, and the same with the others.

As Suetonius tells us, this simple encryption scheme consisted of replacing each letter in the text with a letter further along in the alphabet. The length of the shift is the encryption and decryption key. For example, "Ave Caesar" would be encrypted into "Ezi Geiwev" with key 4, that is $A \to E$, $B \to F$, $\ldots$, $Z \to D$. To decrypt this message, it suffices to reverse the shift of letters, $E \to A, F \to B, \ldots, D \to Z$. This algorithm is clearly easily breakable by testing all possible keys, that is all the

26 possible shifts. Another way to break this and other more complicated schemes is to perform a statistical analysis on the ciphertext. For example, frequency analysis studies the frequency of letters or groups of letters in a ciphertext. Even with more complicated schemes, an intruder might guess the shared secret key by trying every possibility by performing a *brute-force attack*. This is the case of the Data Encryption Standard (DES), published as an official Federal Information Processing Standard for the United States in 1977: in the late 1990s, while this encryption scheme was still used, this method was shown to be vulnerable to brute force attack due to the relatively short length of the keys.

The main disadvantage of symmetric-key encryption is *key management*: each pair of users has to share and store a secret key, hence the number of keys increases quadratically with the number of users. Moreover, when two users want to communicate, they first have to exchange the shared key, which can only be done if they have access to a secure channel or already share a key.

In public-key encryption (or *asymmetric encryption*) the key that the receiver uses to decrypt the message is different from the one used by the sender to encrypt. This means that each user possesses a pair of keys: the *private* key is only known by the owner and used to decrypt ciphers and the *public* key is shared to anyone and used to encrypt messages. Although different, the two parts of this key pair are mathematically linked: public-key schemes are based on mathematical problems which currently admit no efficient solution. This means that for a user it is computationally easy to generate his pair of key and to use them for encryption and decryption. However, it is hard (computationally difficult) for an intruder to determine the private key of a user from its public key. Therefore, the public key can be published without compromising security, while the private key must be kept secret. In 1976, W. Diffie and M. Hellman [25] proposed guidelines for developing public-key cryptosystems, without giving any practical scheme. It was only in 1978 that the first public-key encryption scheme was presented, when R.L. Rivest, A. Shamir and L. Adleman [47] invented RSA.

Differently from symmetric encryption schemes, public encryption schemes solve the problem of key-management: the number of keys required is linear in the number of users and it is not required to securely exchange the keys before the communication.

Another cryptographic primitive, called *digital signature*, serves for demonstrating the authenticity of a digital message or document. A valid signature gives a recipient some reason to believe that the message was created by a known sender and was not altered during the communication. In this way the sender cannot deny having sent the message. Digital signatures also use two keys: a public verification key and

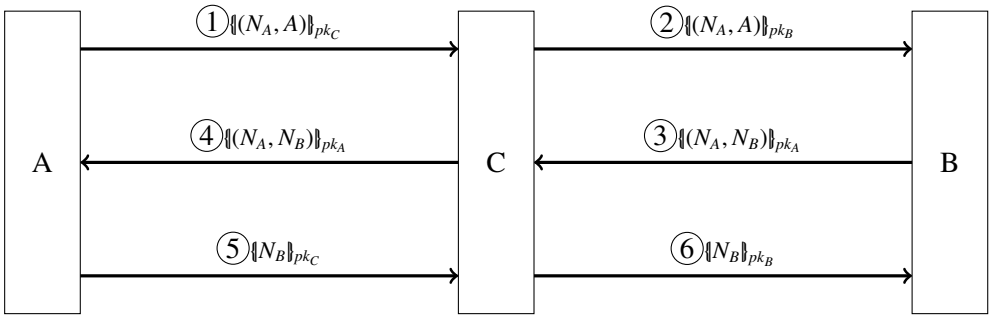

Figure 2.1: An attack on Needham-Schroeder

a private signature key. A participant signs a document using his secret signing key and anyone can verify the document using the public verification key of the participant. Differently from public-key encryption, digital signatures do not hide the content of the document. However it is possible to combine the two primitives and obtain an encrypted signed document.

2.1.2 Protocols

Even assuming that the cryptographic primitives are *perfect*, that is, that it is impossible to decrypt an encrypted message without the corresponding key, we cannot guarantee the absence of attacks on the communication. Consider, for example, the following protocol, a simplified version of the Needham-Schroeder public-key protocol [41].

$$(1)\ A \to B : \{\!|(N_A, A)|\!\}_{pk_B}$$
$$(2)\ B \to A : \{\!|(N_A, N_B)|\!\}_{pk_A}$$
$$(1)\ A \to B : \{\!|(N_B)|\!\}_{pk_B}$$

This protocol describes the exchange of messages between two *participants*, or *parties*, that we will call Alice and Bob. Both participants own a pair of public and private keys, (pk_A, sk_A) for Alice's pair and (pk_B, sk_B) for Bob's. At first, Alice randomly creates a fresh *nonce* N_A, pairs it with her identity A, and encrypts the pair under the public key of Bob, pk_B. Alice then sends message 1 to Bob. When Bob receives this message, he decrypts it using his secret key sk_B and obtains N_A. Then, he randomly creates a nonce N_B and sends to Alice the encryption under pk_A of the pair of N_A and N_B. Finally, Alice receives message 2, decrypts it and sends back N_B encrypted under pk_B. Since Bob is the only one to know sk_B, when Alice receives message 2, she is sure that she is talking to him, because only Bob can decrypt message 1 and obtain N_A. Similarly at the end of the protocol, Bob is sure that he is talking to Alice. This means that both the participants are *authenticated*.

However, this is true only if Alice talks only to honest participants. In fact, we can discover the following attack, described in 1995 by Gavin Lowe [34] and illustrated in Figure 2.1. This type of attack is called *man in the middle* attack and supposes the presence of a third dishonest participant, Charlie, whose pair of keys is (pk_C, sk_C). At the beginning, Alice starts a session with Charlie and sends message ①. Charlie re-encrypts message ① under the public key of Bob and then sends message ② to Bob. When Bob receives the first message he thinks the sender is Alice, so he replies with message ③, that Charlie forwards to Alice. Notice that Charlie at this point cannot obtain N_B by decrypting message ③, that is intended for Alice. When she receives message ④, Alice recognizes her nonce N_A, so she thinks that Charlie replied correctly. She then sends back N_B encrypted under the public key of Charlie, that finally discovers N_B without needing to decrypt message ②. Charlie can now re-encrypt N_B with the public key of Bob and end correctly the session with Bob. In [34] Lowe also provides a fixed version of the protocol, that is now known as the Needham-Schroeder-Lowe protocol. The fix involves the modification of the second message of the protocol: he adds the identity of Bob to the encrypted pair, that is, he replaces $\{\!|(N_A, N_B)|\!\}_{pk_A}$ with $\{\!|(N_A, N_B, B)|\!\}_{pk_A}$.

2.1.3 Security Properties

As illustrated in the previous section, the Needham-Schroeder protocol did not guarantee the *authentication* of the participants. Authentication is one of the security properties that we may want a protocol to achieve. The most important ones are *secrecy* and *authentication* and this thesis will focus on them.

Secrecy Secrecy refers to preventing the disclosure of information to unauthorized users. A protocol preserves the secrecy of some secret s only if unauthorized users cannot learn it. For example, in the Needham-Schroeder protocol, Charlie learns the nonce N_B without being authorized to decrypt the message $\{\!|(N_A, N_B)|\!\}_{pk_A}$. Therefore the secrecy of N_B is not guaranteed by the protocol. In this thesis we will use the following informal definition of secrecy: a protocol P preserves the secrecy of message M if P never publishes M, or anything that would permit the computation of M. A stronger notion of secrecy is related to the concept of indistinguishability [2, 3]: *strong secrecy* means that an attacker cannot differentiate between the different messages. This means that a message s is secret if the session that uses s is indistinguishable from any other session using a message s' instead of s.

Authentication Informally authentication allows to determine whether an individual is, in fact, who he claims to be. The problem in defining a formal notion of authentication is that this property depends upon the use to which the protocol is put. For example, for the Needham-Schroeder protocol, we can use the following property: if Bob ends the protocol apparently with Alice, then Alice started the protocol apparently with Bob. In [35], G. Lowe gives different formal definitions of authentication. In this thesis, the authentication property that we consider is *non-injective agreement*: if some participant B terminates the protocol apparently with A with some parameters, then A started the protocol apparently with B with the same parameters.

2.2 Verification of Protocols

As we have shown in Section 2.1.2, even though the cryptographic primitives are correct, attacks are still possible due to flaws in the design of protocols. The confidence in security protocols can be increased by a formal analysis that proves the desired security properties. In order to do this verification, we need to represent protocols mathematically. There are two models to describe protocols.

Computational model The computational model considers messages sent over the network as bitstrings. Cryptographic primitives are represented by polynomial algorithms on these bitstrings. The behavior of an adversary is modeled as a polynomial probabilistic Turing machine. These models are close to reality and offer strong guarantees on the security of cryptographic protocols. However the proofs are usually difficult, error prone and until recently were only manual. Several tools have been developed to obtain proofs of security in the computational model. CryptoVerif [14] generates proofs by sequences of games for protocols written in a process calculus. CertiCrypt [11] and EasyCrypt [10] are both fully machine-checked frameworks for constructing and verifying cryptographic proofs by sequences of games. They differ in their underlying technologies: CertiCrypt relies on the Coq proof assistant, whereas EasyCrypt uses SMT solvers and automated theorem provers.

Symbolic model In the symbolic model, known also as the Dolev-Yao model [26], the cryptographic primitives are represented by functions and considered as black boxes, the messages are terms on these functions and the adversary can only compute using the defined primitives. This model assumes perfect cryptography: an adversary cannot break the primitives itself. This model, even if less realistic, is more suitable to automation, by computing all the messages (terms) that the adversary can deduce. Several techniques have been designed

for the verification in the symbolic model. ProVerif [1, 16] establishes security properties of protocols for an unbounded number of sessions and thus may not terminate. However when a property cannot be proved a description of the attack may be constructed. Scyther [23] guarantees termination, but may only analyze a bounded number of sessions when a result cannot be obtained in the unbounded case. AVISPA [5] provides a common interface for a number of analysis tools and helps finding attacks on protocols.

Because of the abstractions made in the symbolic model, the guarantees provided by the verification of a security property are usually weaker than the guarantees provided by the verification of the same property in the computational model. Normally an attack in the symbolic model implies an attack in the computational model, however the contrary is not necessary true. Recently, several works have tried to link the two models and prove that, under some conditions, if a protocol is secure in the symbolic model, then it is also secure in the computational model. The first work that investigates this *computational soundness* is by Abadi and Rogaway [4]. They show that, if two expressions are equivalent in the symbolic model, they are also indistinguishable in the computational one, when the only primitive used is symmetric encryption and under some additional restrictions (in particular, they consider only a passive intruder, that is, an intruder that can only listen to the network, but cannot alter messages). This initial result has given rise to many extensions. The result of Micciancio and Warinschi [40] shows the correspondence between states and traces in the two models for public-key encryption and in the presence of an active adversary. In [21] Comon and Cortier showed a computational soundness result that links observational equivalence in the symbolic model to computational indistinguishability in presence of an active adversary, for protocols using symmetric encryption. For more details on further results on computational soundness, we refer the reader to [22].

2.3 Verification of Protocols with Lists of Unbounded Length

The formal verification of security protocols with fixed-size data structures has been extensively studied. However, some protocols, for instance XML protocols for web services, use more complex data structures, such as lists. The verification of protocols that manipulate such data structures has been less studied and presents additional difficulties: these complex data structures add another cause of undecidability. Moreover, some other protocols, called *group protocols*, describe an exchange of messages between an unbounded number of principals, usually establishing a secure key for communicating between themselves over an insecure network. The difficulty in verifying group protocols relates to the fact that a group protocol may

perform an arbitrary number of steps, as the size of the group is a priori unbounded. The form of messages may also depend on the number of participants.

2.3.1 Related Work

Group protocols The first approach considered for proving protocols with recursive data structures was interactive theorem proving: Paulson [44] and Bryans et al [19] study a recursive authentication protocol for an unbounded number of participants, using Isabelle/HOL for [44], and rank functions and PVS for [19]. However, this approach requires considerable human effort. Meadows et al [38] used the NRL protocol analyzer (NPA), based on a combination of model checking and theorem-proving techniques, to verify the Group Domain of Interpretation (GDOI) protocol suite. NPA could not handle the infinite data structures required for modeling general group protocols, so a single key was used instead of a key hierarchy. Several problems including type flaw attacks were found in the protocol and fixed in later versions of GDOI. The early verification of the A.GDH-2 protocol using NPA [37] seems to have missed attacks [45], although the tool supports the Diffie-Hellman exponentiation [39].

Steel and Bundy [49] have used CORAL, a tool for finding counterexamples to incorrect inductive conjectures, to model protocols for group key agreement and group key management, without any restrictions on the scenario. They have discovered new attacks against several group protocols, but cannot prove that protocols are correct.

Kremer, Mercier, and Treinen [30] verify secrecy for group protocols with modular exponentiation and XOR, for any number of participants and an unbounded number of sessions, but only for a passive adversary (eavesdropper).

Several works consider the case of a bounded number of sessions. Pereira and Quisquater [45] discovered several attacks on the CLIQUES protocol suite [50], which extends the Diffie-Hellman key agreement method to support dynamic group operations (A-GDH). They converted the problem of the verification of security properties to the resolution of linear equation systems. In [46], they proved a first generic insecurity result for authentication protocols showing that it is impossible to design a correct authenticated group key agreement protocol based on the A-GDH protocol. Truderung [51] showed a decidability result (in NEXPTIME) for secrecy in recursive protocols. This result was extended to a class of recursive protocols with XOR [32] in 3-NEXPTIME. Chridi et al [20] present an extension of the constraint-based approach in symbolic protocol verification to handle a class

of protocols (Well-Tagged protocols with Autonomous keys) with unbounded lists in messages. They prove that the insecurity problem for Well-Tagged protocols with Autonomous keys is decidable for a bounded number of sessions.

XML Protocols Concerning verification of XML protocols, Bhargavan et al [13] developed the TulaFale language to automatically verify authentication properties of SOAP protocols. They proposed a new specification language for representing SOAP-based security protocols and their properties and an implementation that compiles TulaFale into the applied pi calculus and then runs ProVerif. Kleiner and Roscoe [48, 29] proposed a method for translating WS-Security protocols to traditional cryptographic protocols, preserving flaws and attacks. In particular they gave a mapping to Casper notation ant then analyzed some protocols with FDR. Backes et al [8] analyzed the security of the Secure WS-ReliableMessaging Scenario, a web services scenario that combines security elements with features from another quality-of-service standard, reliable messaging. In their work they provided two types of analysis: an automated analysis based on symbolic verification techniques and an analysis based on explicit cryptographic assumptions underlying cryptographic algorithms used. All these approaches have little or no support for lists of unbounded length. For instance, TulaFale has support for list membership with unbounded lists, but does not go further.

2.3.2 Contributions of This Thesis

In this thesis, we propose two different techniques for verifying security protocols that manipulate lists of unbounded length. Both results are based on the automatic verifier ProVerif [1, 16].

Figure 2.2 shows the structure of ProVerif. ProVerif is an automatic protocol verifier that takes as input a protocol written in a variant of the applied pi calculus [2], translates it into a representation in Horn clauses, and uses a resolution algorithm to determine whether a fact is derivable from the clauses. One can then infer security properties of the protocol. For instance, we use a fact att(M) to mean that the attacker may have the message M. If att(s) is not derivable from the clauses, then s is secret. The main goal of this approach is to prove security properties of protocols without bounding the number of sessions of the protocol.

Like other protocol verifiers, ProVerif can analyze protocols with lists if we fix the lengths of the lists a priori. However, if the protocol is verified only for some lengths, attacks may exist for other lengths. If there is an attack against a protocol,

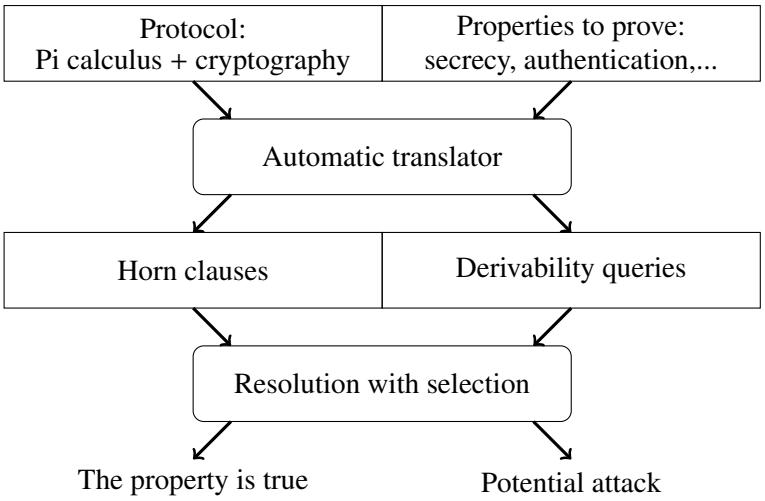

Figure 2.2: ProVerif

then there is a bound such that this attack appears with lists shorter than the bound, but this bound depends on the protocol and is not easy to compute. So our goal is to prove the protocols for lists of any length.

Part 1: homogeneous lists For the first technique, that we present in Part 1 of this thesis, we extend the language of Horn clauses, introducing a new kind of clauses, generalized Horn clauses, to be able to represent lists of any length. This result applies to protocols that manipulate list elements in a uniform way. Because of this uniformity, one might intuitively think that secrecy for lists of length one implies secrecy for lists of any length. We show that this intuition is not exactly true: in general, secrecy for lists of length one does not imply secrecy for lists of any length, as demonstrated in Section 5.3. However, we show that, for a certain class of Horn clauses, if secrecy is proved by our Horn clause technique for lists of length one, then secrecy also holds for lists of unbounded length. This result relies on the sound abstractions made by the translation into Horn clauses. Additionally, we provide an approximation algorithm that can transform generalized Horn clauses into clauses of the class on which our result holds. All facts derivable from the initial clauses are also derivable from the clauses generated by the approximation algorithm, so that we can prove secrecy on the latter clauses, and conclude secrecy for the initial clauses. This result therefore provides an easy way of obtaining a strong security guarantee: we prove using ProVerif that att(s) is not derivable from the clauses for lists of length one, and we can then immediately conclude that secrecy holds for lists of unbounded length, with an unbounded number of sessions.

Part 2: heterogeneous lists In the second part of the thesis, we further extend the language of generalized Horn clauses, to be able to represent lists of any length in protocols that manipulate different list elements in different ways. We adapt the resolution algorithm to deal with these new clauses, and prove the soundness of the new algorithm. The obtained algorithm can prove secrecy and authentication properties. (More precisely, the authentication property that we consider is non-injective agreement [35]: if some participant B terminates the protocol with some parameters, then A started the protocol with the same parameters.) This algorithm performs approximations, so it may fail to prove security properties that actually hold, and it does not always terminate. This is unavoidable because the problem of verifying protocols with an unbounded number of sessions is undecidable. However, the algorithm works well in practice: we have implemented it and successfully tested it on several protocol examples (see Section 8.3). Finally, we provide a more convenient input language for protocols: we extend the input language of ProVerif to model protocols with lists of unbounded length. We give a formal semantics to the new process calculus, and an automatic translation to generalized Horn clauses. We prove that this translation is sound. One can apply the resolution algorithm to the generalized Horn clauses obtained by our translation, to prove secrecy and authentication properties of the initial protocol.

Applications of these results The two techniques presented in this thesis can be used to verify the following protocols:

- some group protocols that manipulate unbounded lists, with an unbounded number of participants. In the first part of the thesis, we focus mainly on group protocols and illustrate our work on the Asokan-Ginzboorg protocol [6]. We prove secrecy of the session key exchanged in this protocol by verifying with ProVerif its version with lists of length one and the size of the group equal to one. In our analysis, we only considered an external attacker: this is sufficient for this example. (Our framework still allows one to model corrupted parties if desired. We detail these points in Section 4.2.2. We do not consider dynamic corruption in this work.) Verification of group protocols is also an application to the second technique presented in this thesis.
- XML protocols such as web services, XML documents being modeled using possibly nested lists. In the second part of the thesis, we focus mainly on these protocols. We illustrate our work on the SOAP extension [18] to XML signatures [9]. XML signatures allow one to sign several parts of an XML document, recorded in a SignedInfo element, which can be modeled as a list of references to the signed parts of the document. A known attack [36] may occur when the document contains several Body elements, and the signed

Body is not the one that is used for subsequent treatments. Our tool detects this attack, and proves that a corrected version is secure.

2.3.3 Outline

The next chapter recalls the technique used by ProVerif. In Chapter 4, we formally define generalized Horn clauses, and their semantics by giving their translation into Horn clauses. Additionally, we introduce a first running example and motivate the introduction of this new type of clauses. In Chapter 5, we show our main theorem: for a class of generalized Horn clauses, if att(s) is not derivable for lists of length one, then it is also not derivable for lists of any length. Moreover, we provide an approximation algorithm for transforming generalized Horn clauses into clauses that satisfy the hypothesis of our main theorem.

Chapter 6 opens the second part of the thesis. We introduce a second running example and motivate the extension of generalized Horn clauses, that we will use to model protocols that manipulate different list elements in different ways. We formally define the new generalized Horn clauses, and their semantics by giving their translation into Horn clauses. In Chapter 7, we adapt the definitions of subsumption, resolution and unification to generalized Horn clauses and prove the soundness of the new notions. In Chapter 8, we adapt the resolution algorithm and prove its soundness. Finally, in Chapter 9, we define the new process calculus and its semantics. We give the automatic translation of generalized processes into generalized Horn clauses and prove that this translation is sound.

The results presented in Part 1 correspond to the publications [42, 43] and those in Chapters 6, 7, and 8 to [17].

CHAPTER 3

Background: ProVerif

Contents

In this chapter we recall the technique used by the automatic verifier ProVerif. ProVerif takes as input a process written in a variant of the applied pi calculus [2]. It then translates the process into an abstract representation with Horn clauses. Finally it runs a resolution algorithm to determine whether a fact can be derived from the set of Horn clauses and prove security properties on the initial process.

3.1 Process Calculus

3.1.1 Syntax and Semantics

The syntax of the process calculus used by ProVerif is defined in Figure 3.1. This syntax assumes an infinite set of names $a, b, c, k, s,$ to be used for representing

$M, N ::=$	terms
x, y, z	variable
a, b, c, k, s	name
$f(M_1, \ldots, M_n)$	constructor application
$P, Q ::=$	processes
$\mathsf{out}(M, N).P$	output
$\mathsf{in}(M, x).P$	input
$\mathbf{0}$	nil
$P \mid Q$	parallel composition
$!P$	replication
$(\nu a)P$	restriction
$\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q$	destructor application
$\mathsf{let}\ pat = M\ \mathsf{in}\ P\ \mathsf{else}\ Q$	pattern-matching
$\mathsf{event}(e(M)).P$	event

Figure 3.1: Syntax of the process calculus

atomic data items, such as keys or nonces, and an infinite set of variables x, y, z. There is also a set of function symbols for constructors (f) and destructors (g), each with an arity. Constructors build new terms of the form $f(M_1, \ldots, M_n)$. Therefore, messages are represented by terms M, N, which can be a variable, a name, or a constructor application $f(M_1, \ldots, M_n)$. Destructors manipulate terms in processes; they are defined by rewrite rules as detailed below.

Protocols are represented by processes P, Q, of the following forms:

- The output process $\mathsf{out}(M, N).P$ outputs the message N on the channel M and then executes P.
- The input process $\mathsf{in}(M, x).P$ inputs a message on the channel M and then executes P with x bound to the input message.
- The nil process $\mathbf{0}$ does nothing.
- The process $P \mid Q$ is the parallel composition of P and Q.
- The replication $!P$ represents an infinite number of copies of P in parallel.
- The restriction $(\nu a)P$ creates a new name a and then executes P.
- The destructor application $\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q$ tries to evaluate $g(M_1, \ldots, M_n)$; if this succeeds, then x is bound to the result and P is executed, else Q is executed. More precisely, a destructor g is defined by a set $def(g)$ of rewrite rules of the form $g(M_1, \ldots, M_n) \to M$ where $M_1, \ldots, M_n, M$ are terms without free names, and the variables of M also occur in $M_1, \ldots, M_n$. Then

$g(M_1, \ldots, M_n)$ evaluates to M if and only if it reduces to M by a rewrite rule of $def(g)$. Using constructors and destructors, one can represent data structures and cryptographic operations as follows.

- The constructor $senc(M, N)$ builds the symmetric encryption of a message M under the symmetric key N. The corresponding destructor $sdec$ is defined by the following rule:

  $$sdec(senc(x, y), y) \to x$$

  $sdec(senc(M), N)$ decrypts a message $senc(M, N)$ with the symmetric key N and returns the cleartext M.
- The constructor pk builds a new public key $pk(M)$ from a secret key M. The constructor $penc(M, N)$ encrypts the message M with the public key N. The corresponding destructor is defined by

  $$pdec(penc(x, pk(y)), y) \to x$$

  $pdec(penc(M, pk(N)), N)$ decrypts a message $penc(M, pk(N))$ with the secret key N and returns the cleartext M.
- The constructor $sign$ is such that $sign(M, N)$ represents the signature of M under the key N. We assume that the signature hides the message. It has one corresponding destructor:

  $$checksign(sign(x, y), pk(y), x) \to x$$

  Hence, $checksign(sign(M, N), pk(N), M)$ checks if $sign(M, N)$ is a correct signature of message M under the secret key N; if yes, it returns the message M; otherwise, it fails.
- A data constructor is a constructor f of arity n that comes with n associated destructors f_i^{-1} $(1 \leq i \leq n)$, defined by rewrite rules $f_i^{-1}(f(x_1, \ldots, x_n)) \to x_i$, so that the arguments of f can be recovered. Data constructors are typically used to represent data structures.

- The pattern-matching $\mathsf{let}\, pat = M \,\mathsf{in}\, P \,\mathsf{else}\, Q$ matches M with the pattern pat, and executes P when the matching succeeds and Q when it fails. The pattern pat can be a variable x or a data constructor application $f(pat_1, \ldots, pat_n)$. Patterns pat are linear, that is, they never contain several occurrences of the same variable. Pattern-matching can be encoded using destructor application: $\mathsf{let}\, x = M \,\mathsf{in}\, P \,\mathsf{else}\, Q$ is an abbreviation for $\mathsf{let}\, x = id(M) \,\mathsf{in}\, P \,\mathsf{else}\, Q$, where the destructor id is defined by $id(x) \to x$ and $\mathsf{let}\, f(pat_1, \ldots, pat_n) = M \,\mathsf{in}\, P \,\mathsf{else}\, Q$ is an abbreviation for

  $$\begin{array}{l}\mathsf{let}\ x_1 = f_1^{-1}(M)\ \mathsf{in}\ \ldots\ \mathsf{let}\ x_n = f_n^{-1}(M)\ \mathsf{in}\\ \quad \mathsf{let}\ pat_1 = x_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ pat_n = x_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q\\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q\end{array}$$

where the variables $x_1, \ldots, x_n$ are fresh and the variables of $pat_1, \ldots, pat_n$ do not occur in Q.

- ProVerif models authentication as correspondence assertions, such as "if event $e(x)$ has been executed, then event $e'(x)$ has been executed". The process calculus provides an instruction for executing such events: the process $\mathsf{event}(e(M))\mathbf{.}P$ executes the event $e(M)$, then executes P.

The conditional $\mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q$ can be encoded as the destructor application $\mathsf{let}\ x = equal(M, N)\ \mathsf{in}\ P\ \mathsf{else}\ Q$ where x does not occur in P and the destructor $equal$, defined by $equal(x, x) \rightarrow x$, succeeds if and only if its two arguments are equal. We often omit a trailing $\mathbf{0}$.

The name a is bound in P in the process $(va)P$. The variable x is bound in P in the processes $\mathsf{in}(M, x)\mathbf{.}P$ and $\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q$. The variables of pat are bound in P in the process $\mathsf{let}\ pat = M\ \mathsf{in}\ P\ \mathsf{else}\ Q$. A process is closed if it has no free variables; it may have free names. We denote by $fn(P)$ the free names of P.

The formal semantics of this calculus can be defined either by a structural equivalence and a reduction relations, in the style of [2], or by a reduction relation on semantic configurations, as in [15].

3.1.2 Example

In this chapter we introduce a running example to illustrate how ProVerif determines secrecy and authentication properties. We use a simplified version of the Denning-Sacco key distribution protocol [24] that we adapt in order to model both secrecy and authentication:

$$(1)\ A \rightarrow B : \{\!|(k, \{\!|k|\!\}_{sk_A})|\!\}_{pk_B}$$
$$(2)\ B \rightarrow A : \{\!|s|\!\}_k$$

This protocols involves two principals A and B. A owns a secret key sk_A and a corresponding public key pk_A, and similarly sk_B and pk_B are the secret and the public key of B, respectively. The symmetric key k is created by A at the beginning of the protocol. Then, A builds the pair of this key together with its signature with her secret key sk_A and then encrypts the pair under the public key of B, pk_B. When B receives this message, he decrypts it and then sends a secret s encrypted under the symmetric key k. To model authentication we use two events b (begin) and e (end). The event $b(pk)$ means that A starts a session with the owner of the public key pk; the event $e(pk_B)$ means that the owner B of the public key pk_B ends the session with A. Our goal is to show that if the owner of pk ends the session, then A really started

the session with him and not with somebody else. In other words, we want to prove that if $e(x)$ has been executed, then $b(x)$ has been executed.

We can represent this protocol as follows:

$$
\begin{array}{ll}
P := & (\nu sk_A)(\nu sk_B)\mathsf{let}\ pk_A = pk(sk_A)\ \mathsf{in} \\
& \mathsf{let}\ pk_B = pk(sk_B)\ \mathsf{in}\ \mathsf{out}(c, pk_A).\mathsf{out}(c, pk_B).(P_A \mid P_B) \\
P_A := & (\nu k)\mathsf{event}(b(pk_B)).\mathsf{out}(c, penc((k, sign(k, sk_A)), pk_B)).\mathsf{in}(c, x). \\
& \mathsf{let}\ z = sdec(x, k)\ \mathsf{in}\ \mathbf{0} \\
P_B := & \mathsf{in}(c, x).\mathsf{let}\ (y, z) = pdec(x, sk_B)\ \mathsf{in} \\
& \mathsf{let}\ w = checksign(z, pk_A, y)\ \mathsf{in} \\
& \mathsf{out}(c, senc(s, y)).\mathsf{event}(e(pk_B))
\end{array}
$$

The protocols creates the secret keys sk_A and sk_B, computes the corresponding public keys and publishes the public keys on the channel c. The process P_A first creates a new name k, for the symmetric key, and then executes an event $b(pk_B)$, meaning that A starts the protocol with the owner of the public key pk_B, B. Then he builds his message and sends it on the public channel c.

The process P_B receives on channel c the message x and then decrypts it with his own secret key sk_B obtaining the pair (y, z). Then he checks whether z is the signature of y under the public key of A, pk_A. Next, he encrypts the secret s with y and sends it on the public channel c. Finally, he executes the event $e(pk_B)$, meaning that B has ended the session with A.

This model corresponds to a very basic version of the protocol, in which A talks only to B, and only one session runs. We can extend this processes to represent a more general protocol P' in which A and B run an unbounded number of sessions and A can talk to any participant and she receives his public key in x_{pk_B}, as illustrated in Figure 3.2.

$$
\begin{array}{ll}
P' := & (\nu sk_A)(\nu sk_B)\mathsf{let}\ pk_A = pk(sk_A)\ \mathsf{in} \\
& \mathsf{let}\ pk_B = pk(sk_B)\ \mathsf{in}\ \mathsf{out}(c, pk_A).\mathsf{out}(c, pk_B).(!P'_A \mid !P'_B) \\
P'_A := & \mathsf{in}(c, x_{pk_B}).(\nu k)\mathsf{event}(b(x_{pk_B})).\mathsf{out}(c, penc((k, sign(k, sk_A)), x_{pk_B})).\mathsf{in}(c, x). \\
& \mathsf{let}\ z = sdec(x, k)\ \mathsf{in}\ \mathbf{0} \\
P'_B := & \mathsf{in}(c, x).\mathsf{let}\ (y, z) = pdec(x, sk_B)\ \mathsf{in} \\
& \mathsf{let}\ w = checksign(z, pk_A, y)\ \mathsf{in} \\
& \mathsf{out}(c, senc(s, y)).\mathsf{event}(e(pk_B))
\end{array}
$$

Figure 3.2: Denning-Sacco representation with the process calculus

3.2 Horn Clauses

ProVerif translates a protocol written in the process calculus into a set of Horn clauses. The syntax of these clauses is defined in Figure 3.3.

$p ::=$	clause terms
$\quad x, y, z, v, w$	$\quad$ variable
$\quad a[p_1, \ldots, p_n]$	$\quad$ name
$\quad f(p_1, \ldots, p_n)$	$\quad$ constructor application
$F ::= pred(p_1, \ldots, p_l)$	facts
$R ::= F_1 \wedge \cdots \wedge F_n \Rightarrow F$	Horn clause

Figure 3.3: Syntax of Horn clauses

The terms, named *clause terms* to distinguish them from the terms that occur in processes, represent the messages that are exchanged between participants of the protocol. A term p can be a variable x, a name $a[p_1, \ldots, p_n]$, or a constructor application $f(p_1, \ldots, p_n)$. A variable can represent any term. In the process calulus, a new name is created at each execution of the restriction (νa). However, in the clauses, instead of creating a fresh name at each run of the protocol, the created names are considered as functions represented by the clause term $a[p_1, \ldots, p_n]$. These functions take as arguments the messages previously received by the principal that creates the name as well as session identifiers, which are variables that take a different value at each run of the protocol, to distinguish names created in different runs. When proving secrecy, we can omit the session identifiers (but they are necessary for proving correspondences, as authentication). As shown in, e.g., [15], this representation of names is a sound approximation. When a name has no arguments, we write a instead of $a[\,]$.

A fact $F = pred(p_1, \ldots, p_l)$ can be of the following forms: message(p, p') means that the message p' may appear on channel p; att(p) means that the attacker may have the message p; m-event(p) represents that the event p must have been executed; event(p) represents that the event p may have been executed.

A clause $F_1 \wedge \cdots \wedge F_n \Rightarrow F$ means that, if all facts F_i are true, then the conclusion F is also true. We use R for a clause, H for its hypothesis, and C for its conclusion. The hypothesis of a clause is considered as a multiset of facts. A clause with no hypothesis $\Rightarrow F$ is written simply F.

3.3 Translation from the Process Calculus to Horn Clauses

As explained in [15], ProVerif uses two sets of clauses: the clauses for the attacker and the clauses for the protocol.

3.3.1 Clauses for the Attacker

Initially the attacker has all the names in a set S, hence the clauses $\text{att}(a[\,])$ for each $a \in S$. Moreover, the abilities of the attacker are represented by the following clauses:

$$\text{att}(b[x]) \tag{Rn}$$

$$\text{for each constructor } f \text{ of arity } n, \quad \text{att}(x_1) \wedge \cdots \wedge \text{att}(x_n) \Rightarrow \text{att}(f(x_1, \ldots x_n)) \tag{Rf}$$

$$\text{for each destructor } g, \text{ for each rule } g(M_1, \ldots, M_n) \to M \text{ in } def(g), \quad \text{att}(M_1) \wedge \cdots \wedge \text{att}(M_n) \Rightarrow \text{att}(M) \tag{Rg}$$

$$\text{message}(x, y) \wedge \text{att}(x) \Rightarrow \text{att}(y) \tag{Rl}$$

$$\text{att}(x) \wedge \text{att}(y) \Rightarrow \text{message}(x, y) \tag{Rs}$$

Clause (Rn) represents the ability of the attacker to create fresh names: all fresh names that the attacker may create are represented by the names $b[x]$ for any x. Clauses (Rf) and (Rg) mean that if the attacker has some terms, then he can apply constructors and destructors to them. Clause (Rl) means that if the attacker has a channel x then he can listen on it and clause (Rs) means that the attacker can send messages in all the channels he has.

3.3.2 Clauses for the Protocol

The protocol is represented by a closed process P_0. To compute the clauses, we first rename the bound names of P_0 so that they are pairwise distinct and distinct from free names of P_0. This renaming is important because bound names are also used as function symbols in terms in the generated clauses. We say that the renamed process, denoted P'_0, is a *suitable renaming* of P_0.

Next, we instrument the process P'_0 by labeling each replication $!P$ with a distinct session identifier s, so that it becomes $!^s P$, and labeling each restriction (νa) with the clause term that corresponds to the fresh name a, $a[x_1, \ldots, x_n, s_1, \ldots, s_{n'}]$, where

$x_1, \ldots, x_n$ are the variables that store the messages received in inputs above (νa) in P'_0 and $s_1, \ldots, s_{n'}$ are the session identifiers that label replications above (νa) in the instrumentation of P'_0. The instrumentation of processes is formally defined by induction on the syntax of the processes, as follows.

Definition 1 (Instrumentation)**.** *Given a process P, a list of variables $Vars = x_1, \ldots, x_n$, and a list of session identifiers $SessId = s_1, \ldots, s_{n'}$, we define the instrumented process as follows:*

- $\mathrm{instr}(\mathsf{in}(M, x).P, Vars, SessId) = \mathsf{in}(M, x).\mathrm{instr}(P, (Vars, x), SessId)$*;*
- $\mathrm{instr}(!P, Vars, SessId) = !^s\mathrm{instr}(P, Vars, (SessId, s))$*;*
- $\mathrm{instr}((\nu a)P, Vars, SessId) = (\nu a : a[Vars, SessId])\mathrm{instr}(P, Vars, SessId)$*;*
- *In all other cases, the same instrumentation is applied recursively on the subprocesses. For instance,* $\mathrm{instr}(P \mid Q, Vars, SessId) = \mathrm{instr}(P, Vars, SessId) \mid \mathrm{instr}(Q, Vars, SessId)$.

We let $\mathrm{instr}(P) = \mathrm{instr}(P, \emptyset, \emptyset)$.

Then we compute the clauses as follows. Let ρ be a function that associates a clause term with each name and variable. We extend ρ as a substitution by $\rho(f(M_1, \ldots, M_n)) = f(\rho(M_1), \ldots, \rho(M_n))$ if f is a constructor.

The translation $[\![P]\!]\rho H$ of an instrumented process P is a set of clauses, where the environment ρ is a function defined as above and H is a sequence of facts $\mathrm{message}(\cdot, \cdot)$ and $\mathrm{m\text{-}event}(\cdot)$. The empty sequence is $\emptyset$ and the concatenation of a fact F to the sequence H is denoted by $H \wedge F$. The translation $[\![P]\!]\rho H$ is defined as follows, and explained below.

- $[\![\mathsf{out}(M, N).P]\!]\rho H = [\![P]\!]\rho H \cup \{H \Rightarrow \mathrm{message}(\rho(M), \rho(N))\}$.
- $[\![\mathsf{in}(M, x).P]\!]\rho H = [\![P]\!](\rho[x \mapsto x])(H \wedge \mathrm{message}(\rho(M), x))$.
- $[\![\mathbf{0}]\!]\rho H = \emptyset$.
- $[\![P \mid Q]\!]\rho H = [\![P]\!]\rho H \cup [\![Q]\!]\rho H$.
- $[\![!^s P]\!]\rho H = [\![P]\!](\rho[s \mapsto s])H$.
- $[\![(\nu a : a'[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P]\!]\rho H =$
  $[\![P]\!](\rho[a \mapsto a'[\rho(x_1), \ldots, \rho(x_n), \rho(s_1), \ldots, \rho(s_{n'})]])H$.

- $[\![\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H = \bigcup\{[\![P]\!]((\sigma\rho)[x \mapsto \sigma' p'])(\sigma H) \mid g(p'_1, \ldots, p'_n) \to p' \text{ is in } def(g) \text{ and } (\sigma, \sigma') \text{ is a most general pair of substitutions such that } \sigma\rho(M_1) = \sigma' p'_1, \ldots, \sigma\rho(M_n) = \sigma' p'_n\} \cup [\![Q]\!]\rho h.$
- $[\![\mathsf{event}(e(M)).P]\!]\rho H = [\![P]\!]\rho(H \wedge \text{m-event}(e(\rho(M)))) \cup \{H \Rightarrow \text{event}(e(\rho(M)))\}.$

The translation of an output $\mathsf{out}(M, N).P$ adds a clause, meaning that the reception of the messages in H can produce the output in question. The translation of an input $\mathsf{in}(M, x).P$ is the translation of P with the concatenation of the input to H. The translation of $\mathbf{0}$ is empty, as this process does nothing. The translation of the parallel composition $P \mid Q$ is the union of the translation of P and Q. The translation of the replication adds the session identifier to ρ; as the clauses can be applied many times, replication is otherwise ignored. The translation of a restriction $(\nu a)P$ is the translation of P in which a is replaced with the corresponding clause term that depends on previously received messages and on session identifiers of replications above the restriction. The translation of a destructor application is the union of the translation for the case where the destructor succeeds and that for the case where it fails, so the translation does not have to determine whether the destructor succeeds or not, but considers both possibilities. We consider that the else branch may always be executed, which overapproximates the possible behaviors of the process. The translation of an event adds the hypothesis m-event($e(\rho(M))$) to H, meaning that P can be executed only if the event has been executed first. Furthermore, it adds a clause, meaning that the event is triggered when all conditions in H are true.

Let $\rho_0 = \{a \mapsto a[\,] \mid a \in fn(P_0)\}$. The set of the clauses corresponding to the closed process P_0 is defined as:

$$\mathcal{R}_{P'_0,S} = [\![\text{instr}(P'_0)]\!]\rho_0\emptyset \cup \{\text{att}(a[\,]) \mid a \in S\} \cup \{(\text{Rn}), (\text{Rf}), (\text{Rg}), (\text{Rl}), (\text{Rs})\}$$

where P'_0 is a suitable renaming of P_0 and S is the set of names initially known by the attacker.

3.3.3 Example

We first instrument the process P' illustrated in Figure 3.2:

$$
\begin{array}{ll}
P'' := & (\nu sk_A : sk_A)(\nu sk_B : sk_B)\mathsf{let}\ pk_A = pk(sk_A)\ \mathsf{in} \\
& \mathsf{let}\ pk_B = pk(sk_B)\ \mathsf{in}\ \mathsf{out}(c, pk_A).\mathsf{out}(c, pk_B).(!^{s_A} P''_A \mid !^{s_B} P''_B) \\
P''_A := & \mathsf{in}(c, x_{pk_B}).(\nu k : k[x_{pk_B}, s_A])\mathsf{event}(b(x_{pk_B})). \\
& \mathsf{out}(c, penc((k, sign(k, sk_A)), x_{pk_B})).\mathsf{in}(c, x). \\
& \mathsf{let}\ z = sdec(x, k)\ \mathsf{in}\ \mathbf{0} \\
P''_B := & \mathsf{in}(c, x).\mathsf{let}\ (y, z) = pdec(x, sk_B)\ \mathsf{in} \\
& \mathsf{let}\ w = checksign(z, pk_A, y)\ \mathsf{in} \\
& \mathsf{out}(c, senc(s, y)).\mathsf{event}(e(pk_B))
\end{array}
$$

Then, we give in Figure 3.4 the translation into Horn clauses of the process P''.

Clauses for the attacker	
	$\mathrm{att}(x) \wedge \mathrm{att}(y) \Rightarrow \mathrm{message}(x, y)$
	$\mathrm{message}(x, y) \wedge \mathrm{att}(x) \Rightarrow \mathrm{att}(y)$
Destructors and constructors:	
	$\mathrm{att}(sk) \Rightarrow \mathrm{att}(pk(sk))$
	$\mathrm{att}(m) \wedge \mathrm{att}(pk) \Rightarrow \mathrm{att}(penc(m, pk))$
	$\mathrm{att}(penc(m, pk(sk))) \wedge \mathrm{att}(sk) \Rightarrow \mathrm{att}(m)$
	$\mathrm{att}(m) \wedge \mathrm{att}(k) \Rightarrow \mathrm{att}(senc(m, k))$
	$\mathrm{att}(senc(m, k)) \wedge \mathrm{att}(k) \Rightarrow \mathrm{att}(m)$
	$\mathrm{att}(m) \wedge \mathrm{att}(sk) \Rightarrow \mathrm{att}(sign(m, sk))$
	$\mathrm{att}(m) \wedge \mathrm{att}(sign(m, sk)) \wedge \mathrm{att}(pk(sk)) \Rightarrow \mathrm{att}(m)$
Names generation:	$\mathrm{att}(b[x])$
Clauses for the protocol	
Initial knowledge:	$\mathrm{att}(pk(sk_A)), \mathrm{att}(pk(sk_B)), \mathrm{att}(a)(a \in S)$
Event b:	$\mathrm{att}(x_{pk_B}) \Rightarrow \mathrm{event}(b(x_{pk_B}))$
First message:	$\mathrm{att}(x_{pk_B}) \wedge \text{m-event}(b(x_{pk_B}))$ $\Rightarrow \mathrm{att}(penc((k[x_{pk_B}, s_A], sign(k[x_{pk_B}, s_A], sk_A)), x_{pk_B}))$
Second message:	$\mathrm{att}(penc((y, sign(y, sk_A)), pk_B))$ $\Rightarrow \mathrm{att}(senc(s, y))$
Event e:	$\mathrm{att}(penc((y, sign(y, sk_A)), pk_B)) \Rightarrow \mathrm{event}(e(pk_B))$

Figure 3.4: Denning-Sacco representation with Horn clauses

Notice that in this set of clauses, all occurrences of $\mathrm{message}(c, M)$ where $c \in S$ are replaced by $\mathrm{att}(M)$. The two facts are equivalent thanks to clauses (Rl) and (Rs). The attacker initially has the public keys of the protocol participants. He also has all names a in the set S, therefore we add $\mathrm{att}(a)$ to the clauses. The clause for event b means that if the attacker has x_{pk_B} then the event $b(x_{pk_B})$ can be executed. The clause for the first message means that if the attacker has x_{pk_B} and the event $b(x_{pk_B})$ has been executed, then the attacker can get $penc((k[x_{pk_B}, s_A], sign(k[x_{pk_B}, s_A], sk_A)), x_{pk_B})$, because P''_A sends this message after receiving x_{pk_B} and executing the event. The clause

for the second message corresponds to the output in P''_B and the last one to the event e in P''_B.

From this set of Horn clauses, it is possible to derive the fact att(s). This means that the attacker can get the secret s. In fact the attacker can create a fresh name a and use it to compute $pk(a)$, with the clause for constructing pk. Then, he obtains att($penc((k[pk(a)], sign(k[pk(a)], sk_A)), pk(a))$ with the clause for the first message. Next, the attacker decrypts this message using the clause for the destructor $pdec$. In this way, he obtains the pair $(k[pk(a)], sign(k[pk(a)], sk_A))$ and re-encrypts this under pk_B: $penc((k[pk(a)], sign(k[pk(a)], sk_A)), pk_B))$. Using the clause for the second message, he obtains $senc(s, k[pk(a)])$. On the other hand, from the pair $(k[pk(a)], sign(k[pk(a)], sk_A))$, he gets $k[pk(a)]$; using the destructor $sdec$, he finally obtains the secret s.

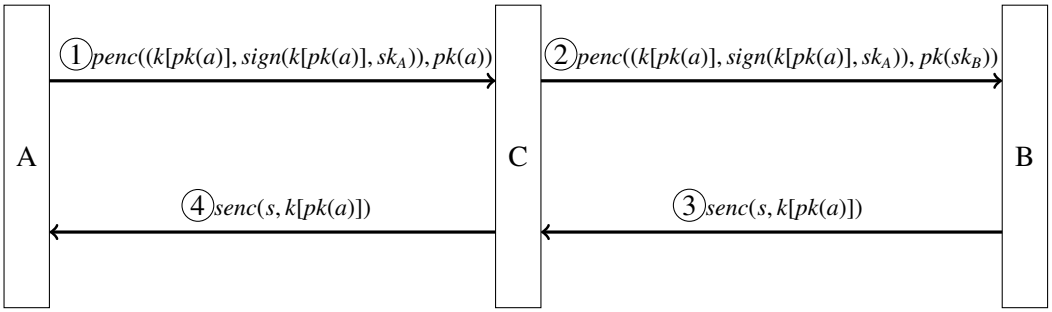

Figure 3.5: An attack on Denning-Sacco

This corresponds to the attack described in Figure 3.5. The attacker opens two sessions, one between A and a corrupted participant C of secret key a and one between C and B. The attacker gets the message ①, decrypts it, re-encrypts it with the public key of B and sends message ② to B. For B this message seems like the first message of a session between him and A, so he replies with message ③, which can be decrypted by the attacker since it obtains k from message ①. A possible fix for this protocol is to add the encryption of the public key of B in the first message. In this case, att(s) is not derivable from the clauses, preserving the secrecy of s. Notice that this protocol has an attack also on authentication: at the end of the first session B thinks to be talking to A, but is talking to C instead. In fact, from the set of Horn clauses, it is possible to derive a clause that conclude event($e(pk_B)$), but that does not have in the hypothesis m-event($b(pk_B)$).

3.3.4 Determining Security Properties

The representation of protocols by Horn clauses is approximate, mainly because Horn clauses that represent the protocol can be applied any number of times instead of only once per session. However, it is sound: if the attacker knows p after the events $p_1, \ldots, p_n$ have been executed, then att(p) is derivable from $\mathcal{R}_1$ and the facts m-event(p_1), ..., m-event(p_n). In particular, if att(p) is not derivable from $\mathcal{R}_1$ and any facts m-event(p'), then the protocol preserves the secrecy of p. If the event p is executed after the events $p_1, \ldots, p_n$, then event(p) is derivable from $\mathcal{R}_1$ and the facts m-event(p_1), ..., m-event(p_n). (This is proved by Theorem 1 in [15] when the clauses are generated from a pi calculus model of the protocol.)

Next, we formally define when a given fact is derivable from a given set of clauses. The hypothesis of a clause $F_1, \ldots, F_n$ are considered as a multiset: the order of the hypothesis is irrelevant, but the number of times an hypothesis occurs is important. We use R for clauses, H for hypothesis, and C for conclusion.

Definition 2 (Subsumption)**.** *We say that $R_1 = H_1 \Rightarrow C_1$ subsumes $R_2 = H_2 \Rightarrow C_2$, and we write $R_1 \sqsupseteq R_2$, if and only if there exists a substitution σ such that $\sigma C_1 = C_2$ and $\sigma H_1 \subseteq H_2$ (multiset inclusion).*

We say that R_1 subsumes R_2 when R_2 can be obtained by adding hypotheses to a particular instance of R_1. In this case, all facts that can be derived by R_2 can also be derived by R_1, so R_2 can be eliminated.

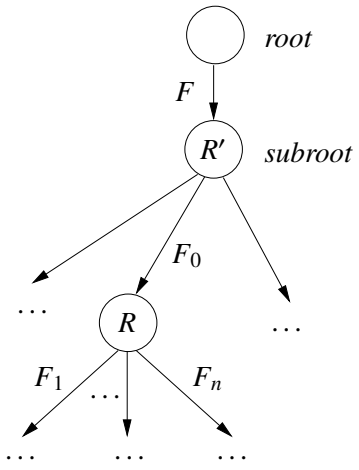

Figure 3.6: Derivation of F

Definition 3 (Derivability)**.** *Let F be a closed fact, that is, a fact without variable. Let $\mathcal{R}$ be a set of clauses. F is derivable from $\mathcal{R}$ if and only if there exists a derivation of F from $\mathcal{R}$, that is, a finite tree defined as follows:*

1. *Its nodes (except the root) are labeled by clauses $R \in \mathcal{R}$;*
2. *Its edges are labeled by closed facts;*
3. *If the tree contains a node labeled R with one incoming edge labeled by F_0 and n outgoing edges labeled by $F_1, \ldots, F_n$, then $R \sqsupseteq F_1 \wedge \cdots \wedge F_n \Rightarrow F_0$.*
4. *The root has one outgoing edge labeled by F. The unique son of the root is named the subroot.*

This definition is illustrated in Figure 3.6. In a derivation, if there is a node labeled by R with one incoming edge labeled by F_0 and n outgoing edges $F_1, \ldots, F_n$ then F_0 can be derived by $F_1, \ldots, F_n$ by the clause R. Therefore there exists a derivation of F from $\mathcal{R}$ if and only if F can be derived from clauses in $\mathcal{R}$ (in classical logic).

3.3.5 Soundness of the Clauses

By testing derivability of facts from the set of clauses, we can prove security properties of the process P_0, as shown by the following two results. Let $\mathcal{F}_{\text{me}}$ be any set of facts of the form m-event(p).

Theorem 1 (Secrecy)**.** *Let P'_0 be a suitable renaming of P_0. Let M be a term. Let p be the clause term obtained by replacing names a with $a[\,]$ in M. If* att(p) *is not derivable from $\mathcal{R}_{P'_0,S} \cup \mathcal{F}_{\text{me}}$ for any $\mathcal{F}_{\text{me}}$, then P_0 preserves the secrecy of M from adversaries with initial knowledge S.*

Theorem 2 (Authentication)**.** *Let P'_0 be a suitable renaming of P_0. Suppose that, for all $\mathcal{F}_{\text{me}}$, for all p, if* event($e(p)$) *is derivable from $\mathcal{R}_{P'_0,S} \cup \mathcal{F}_{\text{me}}$, then* m-event($e'(p)$) $\in \mathcal{F}_{\text{me}}$*. Then P_0 satisfies the correspondence "if $e(x)$ has been executed, then $e'(x)$ has been executed" against adversaries with initial knowledge S.*

3.4 Resolution Algorithm

ProVerif determines whether att(p) or event(p) is derivable from the clauses using resolution with free selection [7]: it combines pairs of clauses by resolution; the literals upon which we perform resolution are chosen by a selection function. We recall this resolution algorithm below.

When the conclusion of a clause R unifies with a hypothesis of a clause R', resolution creates a new clause that corresponds to applying R and R' one after the other.

SATURATE($\mathcal{R}_0$) =

1. $\mathcal{R} \leftarrow$ ELIM($\mathcal{R}_0$).
2. Repeat until a fixpoint is reached
   for each $R \in \mathcal{R}$ such that $sel(R) = \emptyset$,
   for each $R' \in \mathcal{R}$, for each $F_0 \in sel(R')$ such
   that $R \circ_{F_0} R'$ is defined,
   $\mathcal{R} \leftarrow$ ELIM($\{R \circ_{F_0} R'\} \cup \mathcal{R}$).
3. Return $\{R \in \mathcal{R} \mid sel(R) = \emptyset\}$.

Figure 3.7: ProVerif's Algorithm

Definition 4 (Resolution)**.** *Let R and R' be two clauses, $R = H \Rightarrow C$ and $R' = H' \Rightarrow C'$. Assume that there exists $F_0 \in H'$ such that C and F_0 are unifiable and σ is their most general unifier. In this case, we define $R \circ_{F_0} R' = \sigma(H \cup (H' \setminus \{F_0\})) \Rightarrow \sigma C'$. The clause $R \circ_{F_0} R'$ is the result of resolving R' with R upon F_0.*

The facts upon which we resolve are selected through a selection function:

Definition 5 (Selection function)**.** *A selection function is a function from clauses to sets of facts, such that $sel(H \Rightarrow C) \subseteq H$. If $F \in sel(R)$, we say that F is selected in R. If $sel(R) = \emptyset$, we say that no hypothesis is selected in R, or that the conclusion of R is selected.*

The algorithm is correct with any selection function that never selects facts of the form m-event(p), but the choice of the selection function can change the behavior of the algorithm. Facts att(x) where x is a variable can be unified with all facts att(M), so we should avoid selecting att(x) to reduce the number of possible resolution steps. Hence a good selection function satisfies the following formula:

$$sel(H \Rightarrow C) = \begin{cases} \{F_0\} \text{ where } F_0 \in H, \\ \qquad \text{for all variables } x, F_0 \neq \text{att}(x), \text{ and} \\ \qquad \text{for all clause terms } p, F_0 \neq \text{m-event}(p) \\ \emptyset \qquad \text{if there exists no such } F_0 \end{cases}$$

The resolution algorithm is shown in Figure 3.7. It transforms the initial set of clauses into a new one that derives the same facts. The algorithm SATURATE($\mathcal{R}_0$) has 3 steps:

- The first step inserts in $\mathcal{R}$ the clauses in $\mathcal{R}_0$ after elimination of subsumed

clauses by ELIM: when $R' \sqsupseteq R$ and both R and R' are in $\mathcal{R}$, R is removed by ELIM($\mathcal{R}$).

- The second step is a fixpoint iteration that adds the clauses created by resolution: if the conclusion of a clause R such that $sel(R) = \emptyset$ unifies with $F_0 \in sel(R')$, then the resolution $R \circ_{F_0} R'$ is added to $\mathcal{R}$. Then, subsumed clauses are eliminated by ELIM.
- Finally, the algorithm returns the clauses in $\mathcal{R}$ with no selected hypothesis.

3.4.1 Soundness of the Resolution Algorithm

Let $\mathcal{F}_{\text{me}}$ be any set of facts of the form m-event(p).

Theorem 3 (Lemma 2 in [15])**.** *Let F be a closed fact and $\mathcal{R}_0$ a set of clauses. F is derivable from $\mathcal{R}_0 \cup \mathcal{F}_{\text{me}}$ if and only if it is derivable from* SATURATE($\mathcal{R}_0$) $\cup \mathcal{F}_{\text{me}}$.

To prove that a closed fact att(p) is not derivable from $\mathcal{R}_0 \cup \mathcal{F}_{\text{me}}$, we use the following result, where att′ is a new predicate:

Corollary 4. *Let P'_0 be a suitable renaming of P_0. Let M be a term. Let p be the clause term obtained by replacing names a[ ] in M. Let $\mathcal{R}_1 =$* SATURATE($\mathcal{R}_{P'_0,S}$). *If* SATURATE($\mathcal{R}_1 \cup \{\text{att}(p) \Rightarrow \text{att}'(p)\}$) *contains no clause of the form $H \Rightarrow \text{att}'(p')$, then* att($p$) *is not derivable from $\mathcal{R}_1 \cup \mathcal{F}_{\text{me}}$ for any $\mathcal{F}_{\text{me}}$. So, by Theorem 1, P_0 preserves the secrecy of M from adversaries with initial knowledge S.*

Indeed, if SATURATE($\mathcal{R}_1 \cup \{\text{att}(p) \Rightarrow \text{att}'(p)\}$) contains no clause of the form $H \Rightarrow \text{att}'(p')$, then att′($p$) is not derivable from SATURATE($\mathcal{R}_1 \cup \{\text{att}(p) \Rightarrow \text{att}'(p)\}) \cup \mathcal{F}_{\text{me}}$, so by Theorem 3, att′($p$) is not derivable from $\mathcal{R}_1 \cup \{\text{att}(p) \Rightarrow \text{att}'(p)\} \cup \mathcal{F}_{\text{me}}$, so att($p$) is not derivable from $\mathcal{R}_1 \cup \mathcal{F}_{\text{me}}$. Similarly, we also have:

Corollary 5 (Corollary 2 in [15])**.** *Let P'_0 be a suitable renaming of P_0. Let M be a term. Let p be the clause term obtained by replacing names a[ ] in M. Let $\mathcal{R}_1 =$* SATURATE($\mathcal{R}_{P'_0,S}$). *Suppose that all clauses of* SATURATE($\mathcal{R}_1$) *that conclude* event($e(p)$) *for some p are of the form* m-event($e'(p')$) $\wedge H \Rightarrow$ event($e(p')$) *for some H and p'. Then, for all $\mathcal{F}_{\text{me}}$, for all p, if* event($e(p)$) *is derivable from $\mathcal{R}_1 \cup \mathcal{F}_{\text{me}}$, then* m-event($e'(p)$) $\in \mathcal{F}_{\text{me}}$. *So by Theorem 2, P_0 satisfies the correspondence "if $e(x)$ has been executed, then $e'(x)$ has been executed" against adversaries with initial knowledge S.*

Part I

Homogeneous Lists

CHAPTER 4

Abstract Representation of Protocols with Lists

Contents

This part of the thesis presents a first result, [42, 43], published at POST2012 and in the Journal of Computer Security. This technique proves secrecy properties for security protocols that manipulate lists of unbounded length and is based on the Horn clause approach used in ProVerif. Like other protocol verifiers, ProVerif can analyze protocols with lists if we fix the lengths of the lists a priori. However, if the protocol is verified only for some lengths, attacks may exist for other values. To reach this goal, we extend the language of Horn clauses, introducing a new kind of clauses, generalized Horn clauses, to be able to represent lists of any length. We consider a class of protocols that manipulate list elements in a uniform way.

In this chapter we define an abstract representation of protocols with lists by generalized Horn clauses. After introducing a running example and motivating the choices made, we give the syntax and semantics of generalized Horn clauses. We define a type system to guarantee that all variables use indices that vary in the appropriate interval. We finally define a correspondence between generalized Horn clauses and Horn Clauses, which formalizes the semantics of generalized Horn clauses.

4.1 Motivation

4.1.1 Running Example

As a running example, we use a version of the Asokan-Ginzboorg protocol [6] for key agreement. Let the set of players be $\{a_i, i = 1, \ldots, L\}$ for $L \geq 1$ and C be the chair (or the leader). The protocol describes the establishment of a session key between the chair and the other L participants.

$$
\begin{array}{lll}
(1)\ C \to ALL : & (C, \{\!|pk(d)|\!\}_{pw}) \\
(2)\ \ a_i \to C : & (a_i, \{\!|\{\!|(r_i, s_i)|\!\}_{pk(d)}|\!\}_{pw}) \\
(3)\ \ C \to a_i : & \{\!|(s_1, \ldots, s_L, s')|\!\}_{r_i} \\
(4)\ \ a_i \to C : & (a_i, \{\!|(s_i, h(s_1, \ldots, s_L, s'))|\!\}_K), \\
 & \text{for some } i, \text{ where } K = f(s_1, \ldots, s_L, s')
\end{array}
$$

At the beginning, the participants share the knowledge of a password pw and of two $L + 1$-input hash functions f and h. (We consider the password pw as a strong key and ignore dictionary attacks against it.) The chair owns a private key d, whose corresponding public key is $pk(d)$. First, the chair sends to all other participants his identity paired with the public key $pk(d)$ encrypted symmetrically with the password pw. Each participant a_i for $i \in \{1, \ldots, L\}$ decrypts $\{\!|pk(d)|\!\}_{pw}$ and then creates a fresh key r_i and a fresh nonce s_i which will be his contribution to the final session key. Next, he encrypts with $pk(d)$ and then with pw the pair (r_i, s_i) and sends the double encryption $\{\!|\{\!|(r_i, s_i)|\!\}_{pk(d)}|\!\}_{pw}$ paired with his identity. When C receives this message, he decrypts it and assumes that it has been created by a_i. After receiving all L messages, the chair creates his contribution s' to the final key and sends to each participant a_i for $i \in \{1, \ldots, L\}$ the list of all contributions encrypted with the key r_i that a_i previously sent. If step 3 is completed successfully, each participant can compute the session key $K = f(s_1, \ldots, s_L, s')$. In the end, for key confirmation, the chair randomly picks one of the other players and asks him for step 4: a_i computes the session key K as $f(s_1, \ldots, s_L, s')$ and uses it to encrypt a pair made up by his contribution s_i and the hash $h(s_1, \ldots, s_L, s')$. He then sends this encryption paired with his identity. The chair verifies this message by decrypting it.

4.1.2 Need for Generalizing Horn Clauses

We would like to model the example protocol of Section 4.1.1 by Horn clauses and use ProVerif to verify it. Since we consider a parametric group size, we encounter several problems. First, we have to deal with lists whose length is not fixed but is the size L of the group, such as $s_1, \ldots, s_L$ in message 3 of the example.

Next, we need conjunctions of L facts (and L is again unbounded) to represent that some agents receive one message from each group member. For example, when translating message 3 into Horn clauses, the chair expects messages 2 of the form $(a_i, \{\!|(v_i, w_i)|\!\}_{pk(d)}\}_{pw})$ from each a_i. (The incoming values of r_i, s_i are here variables v_i, w_i: the chair, in fact, cannot verify them). Then the chair replies with message 3 $\{\!|(w_1, \ldots, w_L, s')|\!\}_{v_i}$ where s' is a fresh name generated by C, modeled as a function of the previously received messages $s'[v_1, w_1, \ldots, v_L, w_L]$. The attacker can send the incoming messages and intercept L's reply, so we find the clause

$$\begin{aligned} &\mathrm{att}((a_1, \mathit{senc}(\mathit{penc}((v_1, w_1), \mathit{pk}(d)), \mathit{pw}))) \wedge \cdots \wedge \\ &\quad \mathrm{att}((a_L, \mathit{senc}(\mathit{penc}((v_L, w_L), \mathit{pk}(d)), \mathit{pw})) \Rightarrow \\ &\qquad \mathrm{att}(\mathit{senc}((w_1, \ldots, w_L, s'[v_1, w_1, \ldots, v_L, w_L]), v_i)). \end{aligned} \tag{4.1}$$

where *senc* and *penc* are the functions for symmetric encryption and public encryption, respectively. We solve those two problems by adding two new constructs to the syntax of Horn clauses: $\mathit{list}(i \leq L, p_i)$ for the list of elements p_i with index i in the set $\{1, \ldots, L\}$, that is, $\mathit{list}(i \leq L, p_i)$ stands for $\langle p_1, \ldots, p_L \rangle$ (inspired by the *mpair* construct of [20]) and $\bigwedge_{i_1 \leq L, \ldots, i_h \leq L} F$ for the conjunction of facts F with indices $i_1, \ldots, i_h$ in $\{1, \ldots, L\}$.

For instance, with these two new constructs, we can model clause (4.1) as:

$$\begin{aligned} &\bigwedge_{j \leq L} \mathrm{att}((a_j, \mathit{senc}(\mathit{penc}((v_j, w_j), \mathit{pk}(d)), \mathit{pw}))) \Rightarrow \\ &\qquad \mathrm{att}(\mathit{senc}((\mathit{list}(j \leq L, w_j), s'[\mathit{list}(j \leq L, (v_j, w_j))]), v_i)) \end{aligned} \tag{4.2}$$

In conclusions, we do not use the construct $\bigwedge_{i_1 \leq L, \ldots, i_h \leq L} F$ but write F directly, leaving indices in F free; this has the desired meaning since free indices are universally quantified, like other free variables of the clause.

4.2 Syntax and Semantics

This section formally defines the syntax and semantics of *generalized Horn clauses*.

The syntax of these new clauses is defined in Figure 4.1. The clause terms p^G that represent messages are enriched with several new constructs. The variables may have indices $x_{i_1, \ldots, i_h}$. The term for function application $f(p_1^G, \ldots, p_l^G)$ includes not only constructor application but also names $a[p_1^G, \ldots, p_l^G]$ where a is a name without index. The indexed name $a_i[p_1^G, \ldots, p_l^G]$ represents a fresh name a indexed by i in $[1, L]$. For instance, in the context of group protocols, it may represent a name created by the group member number i, inside a group of size L. We added

$p^G, s, t ::=$	clause terms
$x_{i_1,\ldots,i_h}$	variable ($h \geq 0$)
$f(p_1^G, \ldots, p_l^G)$	function application
$a_i[p_1^G, \ldots, p_l^G]$	indexed names
$list(i \leq L, p^G)$	list constructor
$F^G ::= \bigwedge_{i_1 \leq L_1,\ldots,i_h \leq L_h} \text{att}(p^G)$	facts
$R^G ::= F_1^G \wedge \cdots \wedge F_n^G \Rightarrow \text{att}(p^G)$	generalized Horn clause

Figure 4.1: Syntax of generalized Horn clauses

a particular constructor $list(i \leq L, p^G)$ to represent lists of length L, where L is an unknown bound.

In the Asokan-Ginzboorg protocol, we can write, for example, at message 3: $senc((list(j \leq L, s_j), s'), r_i)$ for $senc((s_1, \ldots, s_L, s'), r_i)$. The last element s' is not included in the list $list(j \leq L, s_j)$, to distinguish s' that has just been created by the chair from s_i with $i = 1, \ldots, L$ that has just been received by him: $s_1, \ldots, s_L$ are treated in a uniform way while s' is treated differently.

We extend facts to model the possibility of having a conjunction of facts depending on indices, so that the syntax for facts becomes $\bigwedge_{i_1 \leq L_1,\ldots,i_h \leq L_h} \text{att}(p^G)$. For example, intuitively, $\bigwedge_{i \leq L} \text{att}(p^G)$ represents $\text{att}(p^G\{i \mapsto 1\}) \wedge \cdots \wedge \text{att}(p^G\{i \mapsto L\})$, where $p^G\{i \mapsto i'\}$ denotes p^G in which i has been replaced with i'. The conjunction $\bigwedge_{i_1 \leq L_1,\ldots,i_h \leq L_h}$ with $h = 0$ is omitted: the fact is then simply $\text{att}(p^G)$.

The generalized Horn clause $F_1^G \wedge \cdots \wedge F_n^G \Rightarrow \text{att}(p^G)$ means that, if the facts F_1^G, $\ldots$, F_n^G hold, then the fact $\text{att}(p^G)$ also holds. The conclusion of a clause does not contain a conjunction $\bigwedge_{i_1 \leq L_1,\ldots,i_h \leq L_h}$: we can simply leave the indices of $\text{att}(p^G)$ free to mean that $\text{att}(p^G)$ can be concluded for any value of these indices.

4.2.1 Representation of the Protocol

The representation of the abilities of the attacker includes the clauses given in Section 3.3.1. For our running example, $\text{att}(a_i)$ and $\text{att}(C)$ represent that the attacker

initially knows a_i and C, and the clauses

$$\begin{aligned}
&\mathrm{att}(a)\\
&\mathrm{att}(x) \wedge \mathrm{att}(y) \Rightarrow \mathrm{att}(\mathit{senc}(x, y))\\
&\mathrm{att}(\mathit{senc}(x, y)) \wedge \mathrm{att}(y) \Rightarrow \mathrm{att}(x)\\
&\mathrm{att}(x) \Rightarrow \mathrm{att}(f(x))\\
&\mathrm{att}(x) \Rightarrow \mathrm{att}(h(x))\\
&\mathrm{att}(x) \wedge \mathrm{att}(y) \Rightarrow \mathrm{att}((x, y))\\
&\mathrm{att}((x, y)) \Rightarrow \mathrm{att}(x)\\
&\mathrm{att}((x, y)) \Rightarrow \mathrm{att}(y)
\end{aligned}$$

represent that the attacker can create fresh names, encrypt and decrypt messages, apply hash functions, compose and decompose pairs.

In addition, we have clauses for *list*, which generalize clauses for pairs:

$$\bigwedge_{i \le L} \mathrm{att}(x_i) \Rightarrow \mathrm{att}(\mathit{list}(j \le L, x_j)) \tag{4.3}$$

$$\mathrm{att}(\mathit{list}(j \le L, x_j)) \Rightarrow \mathrm{att}(x_i) \tag{4.4}$$

Let us now give the clauses that represent the protocol itself. We suppose that each principal always plays the same role in the protocol. The chair C sends the first message $(C, \{\!|pk(d)|\!\}_{pw})$ and the attacker intercepts it, so we have the fact:

$$\mathrm{att}((C, \mathit{senc}(pk(d), pw))).$$

Each agent a_i with $i = 1, \ldots, L$ sends message 2 if he has received a correct message 1. From the point of view of a_i, messages 1 and 2 are:

$$\begin{aligned}
&(1)\ C \to a_i : (C, \{\!|y|\!\}_{pw})\\
&(2)\ a_i \to C : (a_i, \{\!|\{\!|(r_i, s_i)|\!\}_y|\!\}pw)
\end{aligned}$$

Each agent a_i cannot verify the value of the public key $pk(d)$ chosen by the chair, so it becomes a variable y. After receiving a message of the form $(C, \{\!|y|\!\}_{pw})$, a_i creates two new names r_i and s_i, encoded as functions of the key y just received and then replies with message 2. If the attacker sends the first message $(C, \{\!|y|\!\}_{pw})$ to a_i, a_i replies with $(a_i, \{\!|\{\!|(r_i, s_i)|\!\}_y|\!\}_{pw})$, and the attacker can intercept this reply, so we obtain the clause:

$$\mathrm{att}((C, \mathit{senc}(y, pw))) \Rightarrow \mathrm{att}((a_i, \mathit{senc}(\mathit{penc}((r_i[y], s_i[y]), y), pw))) \tag{4.5}$$

The chair sends message 3 after receiving a correct message 2 from each agent a_i. The messages 2 and 3 of the protocol as seen from the chair's side are:

$$(2)\ a_i \to C : (a_i, \{\!|(v_i, w_i)|\!\}_{pk(d)}\!|\!\}_{pw})$$
$$(3)\ C \to a_i : \{\!|(w_1, \ldots, w_L, s')|\!\}_{v_i}$$

The chair expects from each a_i with $i = 1, \ldots, L$ a message of the form $(a_i, \{\!|(v_i, w_i)|\!\}_{pk(d)}\!|\!\}_{pw})$. He creates the list of all the contributions adding his own contribution s', modeled as a function of the previously received messages $s'[v_1, w_1, \ldots, v_L, w_L]$. Then he sends this list to each a_i encrypting it with v_i, so we obtain the clause (4.2) already given in Section 4.1.2:

$$\bigwedge_{j \le L} \mathrm{att}((a_j, senc(penc((v_j, w_j), pk(d)), pw))) \Rightarrow \\ \mathrm{att}(senc((list(j \le L, w_j), s'[list(j \le L, (v_j, w_j))]), v_i)) \tag{4.6}$$

Finally, one agent a_i sends message 4 if he received correct messages 1 and 3. From his point of view, messages 1, 3, and 4 of the protocol are:

$$(1)\ C \to a_i : (C, \{\!|y|\!\}_{pw})$$
$$(3)\ C \to a_i : \{\!|(z_1, \ldots, z_L, z)|\!\}_{r_i}$$
$$(4)\ a_i \to C : (a_i, \{\!|(s_i, h(z_1, \ldots, z_L, z))|\!\}_K),$$
$$\text{for some } i, \text{ where } K = f(z_1, \ldots, z_L, z)$$

If a_i has received a message 1 of the form $(L, \{\!|y|\!\}_{pw})$ and a message 3 of the form $\{\!|(z_1, \ldots, z_L, z')|\!\}_{r_i}$ encoded as $\{\!|(list(j \le L, z_j), z')|\!\}_{r_i[y]}$,[1] then a_i computes the session key $K = f((list(j \le L, z_j), z'))$ and one a_i sends to the chair message 4: $(a_i, \{\!|(s_i[y], h((list(j \le L, z_j), z')))|\!\}_K)$. Hence the final clause is:

$$\mathrm{att}((C, senc(y, pw))) \wedge \mathrm{att}(senc((list(j \le L, z_j), z'), r_i[y])) \Rightarrow \\ \mathrm{att}((a_i, senc((s_i[y], h((list(j \le L, z_j), z'))), K))) \\ \text{where } K = f((list(j \le L, z_j), z')) \tag{4.7}$$

We want to prove the secrecy of the session key K. However, this key depends on data received by protocol participants, so we cannot simply test the derivability of att(K). We can use the following trick: to test the secrecy of the key K that

[1] In the protocol, the participant a_i can check whether the component z_i of the list is his own contribution $s_i[y]$, but cannot check the other components. Our representation of lists does not allow us to model such a test: in fact, we cannot substitute a_i directly because, in the construct for lists $list(j \le L, z_j)$, all elements z_j need to have the same form. Moreover, we have built examples of protocols with such tests, for which our result does not hold: intuitively, the test breaks the uniform treatment of the elements of lists, so proving secrecy by the Horn clause technique for lists of length one does not imply secrecy for lists of unbounded length. We shall prove secrecy without the test on z_i; this implies a fortiori secrecy with this test, because the clause without test subsumes the one with the test. In general, removing these tests may obviously lead to false attacks.

participant a_i has, we consider that a_i sends the encryption $\{\!|s''_a|\!\}_K$ of a secret s''_a under K. If K is secret, the adversary will not be able to decrypt the message, so s''_a will remain secret. Therefore, we add the clause

$$\begin{aligned}&\mathrm{att}((C, senc(y, pw))) \wedge \mathrm{att}(senc((list(j \leq L, z_j), z'), r_i[y])) \Rightarrow \\ &\quad \mathrm{att}(senc(s''_a, f((list(j \leq L, z_j), z'))))\end{aligned} \tag{4.8}$$

to model the output of $\{\!|s''_a|\!\}_K$, and we test the derivability of $\mathrm{att}(s''_a)$. We have also used a similar clause to prove the secrecy of the key K that C has; in this case we consider that chair sends the encryption $\{\!|s''_C|\!\}_K$ of a secret s''_C under K, and the key K that C has is secret if and only if s''_C is secret:

$$\begin{aligned}&\bigwedge_{j \leq L} \mathrm{att}((a_j, senc(penc((v_j, w_j), pk(d)), pw))) \wedge \\ &\mathrm{att}((a_i, senc((w_i, h((list(j \leq L, w_j), s'[list(j \leq L, (v_j, w_j))])), \\ &\qquad f((list(j \leq L, w_j), s'[list(j \leq L, (v_j, w_j))])))) \Rightarrow \\ &\quad \mathrm{att}(senc(s''_C, f((list(j \leq L, w_j), s'[list(j \leq L, (v_j, w_j))]))))\end{aligned} \tag{4.9}$$

4.2.2 On Corruption and Participants Playing Multiple Roles

In the previous model, we have no explicit model for corrupted participants. In the particular case of the Asokan-Ginzboorg protocol, statically corrupted participants come for free. Indeed, we are going to show the secrecy of the session keys only for a session between honest participants, since for other sessions, the adversary obviously obtains the session keys via the corrupted participants. Other sessions, which involve a different group containing at least one corrupted participant, will use a different password pw'. We can assume that the adversary has this password. Hence, the adversary has all secrets needed to perform all actions of these sessions (for both the honest and the corrupted participants), so we do not need to add Horn clauses for them.

In the general case, however, a more complex model may be needed. For example, a protocol may use a long-term secret key for each participant, which is used both in sessions with only honest participants and in sessions with some corrupted participants. In this case, we need to model sessions that mix honest and corrupted participants. This can be done by modeling each honest participant so that it accepts to talk both to honest and dishonest participants, for instance by receiving the identities of its interlocutors from the adversary as the first message of the session. (This is a technique often used in ProVerif models of protocols.)

Furthermore, we can also model that a certain role is played by several participants (possibly including corrupted participants), for instance as follows. We use a pred-

$$\frac{i : [1, L] \in \Gamma}{\Gamma \vdash i : [1, L]}(\text{EnvIndex}) \qquad \frac{x\_ : [1, L_1] \times \cdots \times [1, L_h] \in \Gamma}{\Gamma \vdash x\_ : [1, L_1] \times \cdots \times [1, L_h]}(\text{EnvVar})$$

$$\frac{\Gamma \vdash x\_ : [1, L_1] \times \cdots \times [1, L_h] \qquad \Gamma \vdash i_1 : [1, L_1] \ldots \Gamma \vdash i_h : [1, L_h]}{\Gamma \vdash x_{i_1,\ldots,i_h}}(\text{Var})$$

$$\frac{\Gamma \vdash p_1^G \ldots \Gamma \vdash p_h^G}{\Gamma \vdash f(p_1^G, \ldots, p_h^G)}(\text{Fun})$$

$$\frac{\Gamma \vdash p_1^G \ldots \Gamma \vdash p_h^G \qquad \Gamma \vdash i : [1, L]}{\Gamma \vdash a_i[p_1^G, \ldots, p_h^G]}(\text{Name}) \qquad \frac{\Gamma, i : [1, L] \vdash p^G}{\Gamma \vdash \mathit{list}(i \leq L, p^G)}(\text{List})$$

$$\frac{\Gamma, i_1 : [1, L_1], \ldots, i_h : [1, L_1] \vdash p^G}{\Gamma \vdash \bigwedge_{i_1 \leq L_1, \ldots, i_h \leq L_h} \text{att}(p^G)}(\text{Fact})$$

$$\frac{\Gamma \vdash F_1^G \ldots \Gamma \vdash F_n^G \qquad \Gamma \vdash F^G}{\Gamma \vdash F_1^G \wedge \cdots \wedge F_n^G \Rightarrow F^G}(\text{Clause})$$

Figure 4.2: Type system for generalized Horn clauses

icate $\text{id}(\mathit{idname}, \mathit{key}_1, \ldots, \mathit{key}_n)$ to represent a participant of identity *idname*, that uses long-term keys $\mathit{key}_1, \ldots, \mathit{key}_n$ (these may include secret and public keys). We define as many facts as needed for this predicate, which may include honest and dishonest participants. For dishonest participants, we give the keys to the adversary by adding the appropriate att facts. We can define infinite sets of participants by including a variable in the terms that define the identities and keys. We define the Horn clauses for the various roles by adding hypotheses $\text{id}(x_{id}, k_1, \ldots, k_n)$ to the standard Horn clauses for these roles and using the variables $x_{id}, k_1, \ldots, k_n$ in place of the respective keys of the considered role. This yields an economical model of a role that can be played by any participant with an identity among those defined by the predicate id. To fit our framework formally, since we use att as only predicate, we can encode the facts $\text{id}(p_0, \ldots, p_n)$ as $\text{att}(f_{\text{id}}(p_0, \ldots, p_n))$ where f_{id} is a *private* function, that is, the adversary does not have clauses to compute f_{id} nor to decompose it.

4.2.3 Type System for the New Clauses

In this section, we define a simple type system for the generalized Horn clauses. The goal of this type system is to guarantee that all variables use indices that vary in the appropriate interval.

Definition 6. *An index i is bound if:*

- *it appears as an index of a conjunction defining a fact, so, for instance, in the*

fact $\bigwedge_{i_1 \leq L_1, \ldots, i_h \leq L_h} \mathrm{att}(p^G)$*,* $i_1, \ldots, i_h$ *are bound in* $\mathrm{att}(p^G)$.

- *it appears as an index for a list constructor, that is, in the clause term list*$(i \leq L, p^G)$*,* i *is bound in* p^G.

Indices that are not bound are free.

For simplicity, we suppose that the bound indices of clauses have pairwise distinct names, and names distinct from the names of free indices. This can easily be guaranteed by renaming the bound indices if needed. This renaming is not performed in the examples given here, because it would use too many letters.

In the type system defined in Figure 4.2, the type environment Γ is a list of type declarations:

- $i : [1, L]$ means that i is of type $[1, L]$, that is, intuitively, the value of index i can vary between 1 and the value of the bound L.
- $x_{\_} : [1, L_1] \times \cdots \times [1, L_h]$ means that the variable x expects indices of types $[1, L_1], \ldots, [1, L_h]$.

The type system defines the judgments:

- $\Gamma \vdash i : [1, L]$, which means that i has type $[1, L]$ in environment Γ, by rule (EnvIndex);
- $\Gamma \vdash x_{\_} : [1, L_1] \times \cdots \times [1, L_h]$, which means that x expects indices of types $[1, L_1], \ldots, [1, L_h]$ according to environment Γ, by rule (EnvVar);
- $\Gamma \vdash p^G$, $\Gamma \vdash F^G$, $\Gamma \vdash R^G$, which mean that p^G, F^G, R^G, respectively, are well typed in environment Γ.

Most type rules are straightforward. For instance, the rule (Var) means that $x_{i_1, \ldots, i_h}$ is well typed when the types expected by x for its indices match the types of $i_1, \ldots, i_h$. The rule (Name) deserves an additional explanation: we have no information in Γ to set the type of the index of name a, and hence the index of a can have any type. A priori, it is obviously expected that the index of a certain name a always has the same type. However, the additional freedom given by the type rule will be useful: the transformations of Section 5.4 can create clauses in which the same name a has indices of different types. The formal meaning of such clauses can be defined by assuming that the name a exists for indices up to the value of the largest bound.

It is easy to verify that the clauses of Section 4.2.1 are well typed in our type system. Clause (4.3) is well typed in the environment $x\_ : [1, L]$, (4.4) in the environment $x\_ : [1, L], i : [1, L]$, and the other clauses in the environment in which all free indices have type $[1, L]$ and the variables expect indices of type $[1, L]$.

4.2.4 Translation from Generalized Horn Clauses to Horn Clauses

A generalized Horn clause represents several Horn clauses: for each value of the bounds L and of the free indices i that occur in a generalized Horn clause R^G, R^G corresponds to a certain Horn clause. This section formally defines this correspondence.

$\overline{p} ::=$	clause terms
$\quad x_{\bar{\imath}_1,\ldots,\bar{\imath}_h}$	$\quad$ variable
$\quad a_{\bar{\imath}}[\overline{p}_1, \ldots, \overline{p}_h]$	$\quad$ name
$\quad f(\overline{p}_1, \ldots, \overline{p}_h)$	$\quad$ constructor application
$\quad \langle \overline{p}_1, \ldots, \overline{p}_h \rangle$	$\quad$ list
$\overline{F} ::= \text{att}(\overline{p})$	facts
$\overline{R} ::= \overline{F}_1 \wedge \ldots \wedge \overline{F}_n \Rightarrow \overline{F}$	Horn clauses

Figure 4.3: Syntax of Horn clauses

The syntax of Horn clauses obtained by translation of generalized Horn clauses is given in Figure 4.3. This syntax is similar to that of initial Horn clauses (Figure 3.3) except that variables and names can now have indices $\bar{\imath}$, which are integer values, and that we include a term $\langle \overline{p}_1, \ldots, \overline{p}_h \rangle$ for representing lists (which will be generated by translation of *list*).

Definition 7. *Given a generalized Horn clause R^G well typed in Γ, an environment T for R^G is a function that associates to each bound L a fixed positive integer L^T and to each free index i that appears in R^G, an index $i^T \in \{1, \ldots, L^T\}$, if $\Gamma \vdash i : [1, L]$.*

Given an environment T and values $\bar{\imath}_1, \ldots, \bar{\imath}_h$, we write $T[i_1 \mapsto \bar{\imath}_1, \ldots, i_h \mapsto \bar{\imath}_h]$ for the environment that associates to indices $i_1, \ldots, i_h$ the values $\bar{\imath}_1, \ldots, \bar{\imath}_h$ respectively and that maps all other indices as in T.

Given an environment T, a generalized Horn clause R^G is translated into the standard Horn clause R^{GT} defined next. We denote respectively $p^{GT}, F^{GT}, \ldots$ the translation of $p^G, F^G, \ldots$ using the environment T.

The translation of a clause term p^G is defined as follows:

- $(x_{i_1,\ldots,i_h})^T = x_{i_1^T,\ldots,i_h^T}$.
- $f(p_1^G,\ldots,p_l^G)^T = f(p_1^{GT},\ldots,p_l^{GT})$.
- $a_i[p_1^G,\ldots,p_l^G]^T = a_{i^T}[p_1^{GT},\ldots,p_l^{GT}]$.
- $list(i \leq L, p^G)^T = \langle p^{GT[i\mapsto 1]},\ldots,p^{GT[i\mapsto L^T]}\rangle$.

The translation of *list* is a list; we stress that this translation uses a list symbol $\langle\ldots\rangle$ different from the tuple symbol $(\ldots)$: *list* is the only construct that can introduce the list symbol $\langle\ldots\rangle$. This is important to make sure that confusions between tuples that may occur in the protocol and *list* do not occur for particular list lengths. In the implementation of the protocol, one must also make sure to use distinct encodings for *list* and for tuples.

The translation of a fact $F^G = \bigwedge_{i_1\leq L_1,\ldots,i_h\leq L_h} \mathrm{att}(p^G)$ is

$$F^{GT} = \mathrm{att}(p_1) \wedge \ldots \wedge \mathrm{att}(p_k)$$

where $\{p_1,\ldots,p_k\} = \{p^{GT'} \mid T' = T[i_1 \mapsto \bar{\imath}_1,\ldots,i_h \mapsto \bar{\imath}_h]$ where $\bar{\imath}_j \in \{1,\ldots,L_j^T\}$ for all j in $\{1,\ldots,h\}\}$, and $(F_1^G \wedge \cdots \wedge F_n^G)^T = F_1^{GT} \wedge \cdots \wedge F_n^{GT}$.

Finally, we define the translation of the generalized Horn clause $R^G = H^G \Rightarrow \mathrm{att}(p^G)$ as $R^{GT} = H^{GT} \Rightarrow \mathrm{att}(p^{GT})$.

For instance, the translation of the clause (4.6) in the environment $T = \{L \mapsto 1, i \mapsto 1\}$ is

$$\mathrm{att}((a_1, senc(penc((v_1,w_1),pk(d)),pw))) \Rightarrow \mathrm{att}(senc((\langle w_1\rangle, s'[\langle(v_1,w_1)\rangle]),v_1)).$$

In the environment $T = \{L \mapsto 2, i \mapsto 1\}$, it is

$$\mathrm{att}((a_1, senc(penc((v_1,w_1),pk(d)),pw))) \wedge \mathrm{att}((a_2, senc(penc((v_2,w_2),pk(d)),pw))) \Rightarrow \mathrm{att}(senc((\langle w_1,w_2\rangle, s'[\langle(v_1,w_1),(v_2,w_2)\rangle]),v_1)).$$

When $\mathcal{R}^G$ is a set of generalized Horn clauses, we define $\mathcal{R}^{G\mathcal{T}} = \{R^{GT} \mid R^G \in \mathcal{R}^G,\ T$ is an environment for $R^G\}$. In terms of abstract interpretation, the sets of generalized Horn clauses ordered by inclusion constitute the abstract domain, the sets of Horn clauses ordered by inclusion the concrete domain, and $\mathcal{R}^{G\mathcal{T}}$ is the concretization of $\mathcal{R}^G$.

CHAPTER 5

From Any Length to Length One

Contents

In this chapter, we define a mapping from lists of any length to lists of length one, and show that derivability for lists of any length implies derivability for lists of length one, for a particular class of Horn clauses. So, for protocols represented by such clauses, if ProVerif proves secrecy for lists of length one, then the secrecy is preseved for lists of any length. Additionally, we provide an approximation algorithm for transforming generalized Horn clauses into clauses that falls in the class above.

5.1 Main Result

As we explained in the previous chapter, given a generalized Horn clause R^G we can obtain several Horn clauses by giving a value to the bounds and the free indices that occur in R^G. In particular, there exists only one environment T for R^G such that all bounds are equal to 1. Hence by now we use R^{G1} for the only possible translation of R^G when all bounds are 1. We define $\mathcal{R}^{G1} = \{R^{G1} \mid R^G \in \mathcal{R}^G\}$.

Next, we define a translation from clauses in which bounds can have any value,

following the syntax of Figure 4.3, to clauses in which the bounds are fixed to 1:

$$\mathbb{I}(\overline{p}) = \begin{cases} \{x_{\underbrace{1,\ldots,1}_{h}}\} & \text{if } \overline{p} = x_{\tilde{\imath}_1,\ldots,\tilde{\imath}_h} \\ \{f(p'_1,\ldots,p'_h) \mid p'_1 \in \mathbb{I}(\overline{p}_1),\ldots,p'_h \in \mathbb{I}(\overline{p}_h)\} & \text{if } \overline{p} = f(\overline{p}_1,\ldots,\overline{p}_h) \\ \{a_1[p'_1,\ldots,p'_h] \mid p'_1 \in \mathbb{I}(\overline{p}_1),\ldots,p'_h \in \mathbb{I}(\overline{p}_h)\} & \text{if } \overline{p} = a_{\tilde{\imath}}[\overline{p}_1,\ldots,\overline{p}_h] \\ \{\langle p\rangle \mid p \in \mathbb{I}(\overline{p}_1) \cup \cdots \cup \mathbb{I}(\overline{p}_h)\} & \text{if } \overline{p} = \langle \overline{p}_1,\ldots,\overline{p}_h\rangle \end{cases}$$

This translation maps all indices of variables and names to 1. The translation of a list is a list with one element, containing the translation of any element of the initial list. Several choices are possible for the translation of a list; $\mathbb{I}(\overline{p})$ returns the set of all possible clause terms.

Given a fact $\overline{F} = \text{att}(\overline{p})$, its translation when the bounds are fixed to 1 is $\mathbb{I}(\text{att}(\overline{p})) = \{\text{att}(p) \mid p \in \mathbb{I}(\overline{p})\}$. Given a conjunction of facts $\overline{F}_1 \wedge \cdots \wedge \overline{F}_h$, its translation when the bounds are fixed to 1 is $\mathbb{I}(\overline{F}_1 \wedge \cdots \wedge \overline{F}_h) = \mathbb{I}(\overline{F}_1) \cup \cdots \cup \mathbb{I}(\overline{F}_h)$.

We say that a term or fact is *linear* when it contains at most one occurrence of each variable $x_\_$ (with any indices, so it cannot contain x_i and x_j for instance). Finally, we can state the main result of this first part of the thesis.

Theorem 6. *Let $\mathcal{R}^G$ be a set of generalized Horn clauses such that, for each clause $R^G \in \mathcal{R}^G$, R^G is well typed, that is, there exists Γ such that $\Gamma \vdash R^G$, with the following conditions:*

1. *the free indices of R^G have pairwise distinct types in Γ;*
2. *the conclusion of R^G is linear and the bound indices in the conclusion of R^G have pairwise distinct bounds, and bounds different from the bounds of free indices of R^G in Γ.*

For all facts $\overline{F}$, if $\overline{F}$ is derivable from $\mathcal{R}^{G\mathcal{T}}$, then for all $F \in \mathbb{I}(\overline{F})$, F is derivable from $\mathcal{R}^{G1}$.

If we show that, for some $F \in \mathbb{I}(\overline{F})$, F is not derivable from $\mathcal{R}^{G1}$, then using this theorem, $\overline{F}$ is not derivable from $\mathcal{R}^{G\mathcal{T}}$.

Suppose that we want to show that s is secret in a protocol represented by the clauses $\mathcal{R}^G$. It is sufficient to prove using ProVerif that the fact $\text{att}(s)$ is not derivable from $\mathcal{R}^{G1}$, that is, we prove secrecy when the bounds are all fixed to 1. In fact, by Theorem 6, we can conclude that $\text{att}(s)$ is not derivable from $\mathcal{R}^{G\mathcal{T}}$, so we obtain secrecy for any bounds.

Unfortunately, this theorem does not apply to all Horn clauses: Hypotheses 1 and 2 have to be satisfied. In terms of protocols, the condition that bound indices of the conclusion must have different types means that each sent message does not contain two lists of the same length. The free indices are typically indices inside a group of protocol participants, and we generally have a single free index ranging in this group; this index should have a bound different from the length of lists in sent messages, that is, the number of members of the group should be different from the length of lists in sent messages.

The clauses of our running example do not satisfy these hypotheses, as we consider several lists of the same length as the number of group members. We shall see in Section 5.4 how to transform the clauses so that they satisfy the required hypotheses.

5.2 Proof of Theorem 6

The proof of Theorem 6 proceeds by building a derivation of F from $\mathcal{R}^{G1}$, from a derivation of $\overline{F}$ from $\mathcal{R}^{G\mathcal{T}}$, by induction on this derivation. Informally, the derivation of F from $\mathcal{R}^{G1}$ is obtained by applying $\mathbb{I}$ to the derivation of $\overline{F}$ from $\mathcal{R}^{G\mathcal{T}}$. If $\overline{F}$ is derived by $R^{GT} = H^{GT} \Rightarrow C^{GT}$, $\overline{F}$ is an instance of C^{GT} by a substitution σ: $\overline{F} = \sigma C^{GT}$; we show that any $F \in \mathbb{I}(\overline{F})$ is an instance of C^{G1} by a substitution σ' obtained from σ (Lemma 7 and Corollary 8): $F = \sigma' C^{G1}$. Hence, in order to derive F using $R^{G1} = H^{G1} \Rightarrow C^{G1}$, we need to derive $\sigma' H^{G1}$ from $\mathcal{R}^{G1}$, knowing a derivation of σH^{GT} from $\mathcal{R}^{G\mathcal{T}}$. Informally, to show that this is possible, we prove that $\sigma' H^{G1} \subseteq \mathbb{I}(\sigma H^{GT})$ (Lemma 9 and Corollary 10) and conclude by induction. Next, we formally detail this proof. The proofs of the lemmas are detailed in Appendix A.1.

First, we define how the substitution σ' (for bounds fixed to 1) is computed from σ (for any bounds): $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$, as defined next.

Definition 8. *Given a clause R^G (resp. a fact F^G, a clause term p^G) well typed in Γ, we define the set of types $Types(\Gamma \vdash R^G)$: $[1, L] \in Types(\Gamma \vdash R^G)$ if and only if $i : [1, L] \in \Gamma$ for some index i free in R^G, or L appears as bound in list$(i \leq L, p^G)$ or in $\bigwedge_{\ldots, i \leq L, \ldots} \mathrm{att}(p^G)$ in R^G.*

We define a function τ, which associates to each type an index value. Given an environment T, we say that τ is consistent with T for $\Gamma \vdash R^G$ when τ is defined on $Types(\Gamma \vdash R^G)$ and, for each type $[1, L] \in Types(\Gamma \vdash R^G)$, we have:

- $1 \leq \tau([1, M]) \leq M^T$

- $i^T = \tau([1, L])$ *if* i *is a free index of* R^G *and* $\Gamma \vdash i : [1, L]$.

We consider closed substitutions, that is, substitutions that map the variables in their domain to closed clause terms, and are not defined on other variables. We denote the domain of the substitution σ *by* $dom(\sigma)$. *We designate by* $fv(R^G)$, $fv(p^G)$ *the free variables of a clause, resp. clause term.*

Given a closed substitution σ, *we define the domain of the translated substitutions* $dom\mathbb{I}(\sigma) = \{x_{\underbrace{1,\ldots,1}_{h}} \mid x_{\underbrace{-,\ldots,-}_{h}} \in dom(\sigma)\}$.

Given a closed substitution σ *such that* $dom(\sigma) = fv(R^{GT})$ *and a function* τ *consistent with* T *for* $\Gamma \vdash R^G$, *the translation* $\mathbb{I}_{\Gamma,\tau}(\sigma)$ *of the substitution* σ *is the set of closed substitutions* σ' *such that:*

- $dom(\sigma') = dom\mathbb{I}(\sigma)$,
- $\forall x_{\underbrace{1,\ldots,1}_{h}} \in dom\mathbb{I}(\sigma)$, $\sigma' x_{\underbrace{1,\ldots,1}_{h}} \in \mathbb{I}(\sigma x_{\bar{\imath}_1,\ldots,\bar{\imath}_h})$ *where* $\Gamma \vdash x_{\_} : [1, L_1]\times\cdots\times[1, L_h]$ *and* $\bar{\imath}_j = \tau([1, L_j])$ *for* $j = 1, \ldots, h$.

Remark 1. *In the definition above, we have* $x_{\bar{\imath}_1,\ldots,\bar{\imath}_h} \in dom(\sigma) = fv(R^{GT})$, *so that* σ' *is well defined. Indeed, since* $x_{\underbrace{1,\ldots,1}_{h}} \in dom\mathbb{I}(\sigma)$, $x_{\underbrace{-,\ldots,-}_{h}} \in dom(\sigma)$, *so* $x_{i_1,\ldots,i_h} \in fv(R^G)$ *for some* $i_1, \ldots, i_h$. *If* i_j *is free in* R^G, *then* $\Gamma \vdash i_j : [1, L_j]$, *so* $i_j^T = \tau([1, L_j]) = \bar{\imath}_j$, *since* τ *is consistent with* T *for* $\Gamma \vdash R^G$. *If* i_j *is bound in* R^G, *then* i_j *is bound by* $list(i_j \leq L_j, p^G)$ *or by* $\bigwedge_{\ldots,i_j \leq L_j,\ldots} \mathrm{att}(p^G)$, *so in* R^{GT}, i_j *takes all values in* $1, \ldots, L_j^T$, *hence* i_j *takes in particular the value* $\tau([1, L_j]) = \bar{\imath}_j$. *Therefore,* $x_{\bar{\imath}_1,\ldots,\bar{\imath}_h} \in fv(R^{GT})$.

Remark 2. *For a given* T, *there does not always exist a* τ *consistent with* T *for* $\Gamma \vdash R^G$. *Indeed, if* T *maps two free indices of* R^G *of the same type to distinct values, there exists no* τ *consistent with* T *for* $\Gamma \vdash R^G$.

Lemma 7. *Let* p^G *be a linear clause term, well typed in* Γ, *such that its free indices have pairwise distinct types, its bound indices have pairwise distinct bounds, and bounds distinct from the bounds of free indices. Let* T *be an environment for* p^G. *Let* σ *be a closed substitution such that* $dom(\sigma) = fv(p^{GT})$. *Then, for all* $p \in \mathbb{I}(\sigma p^{GT})$, *there exist* τ, *consistent with* T *for* $\Gamma \vdash p^G$, *and* $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$ *such that* $\sigma' p^{G1} = p$.

Corollary 8. *Let* $F^G = \mathrm{att}(p^G)$ *be a linear fact, well typed in* Γ, *such that its free indices have pairwise distinct types, its bound indices have pairwise distinct bounds, and bounds distinct from the bounds of free indices. Let* T *be an environment for* F^G. *Let* σ *be a closed substitution such that* $dom(\sigma) = fv(F^{GT})$. *Then, for all* $F \in \mathbb{I}(\sigma F^{GT})$, *there exist* τ, *consistent with* T *for* $\Gamma \vdash F^G$, *and* $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$ *such that* $\sigma' F^{G1} = F$.

Proof. Obvious by applying Lemma 7 to p^G. □

Lemma 9. *Let p^G be a clause term, well typed in Γ. Let T be an environment for p^G and σ be a closed substitution such that $dom(\sigma) = fv(p^{GT})$. Then, for all τ consistent with T for $\Gamma \vdash p^G$, for all $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$, we have $\sigma' p^{G1} \in \mathbb{I}(\sigma p^{GT})$.*

Corollary 10. *Let $F^G = \bigwedge_{i_1 \leq L_1, \ldots, i_h \leq L_h} \mathrm{att}(p^G)$ be a fact, well typed in Γ. Let T be an environment for F^G and σ be a closed substitution such that $dom(\sigma) = fv(F^{GT})$. Then, for all τ consistent with T for $\Gamma \vdash F^G$, for all $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$, we have $\sigma' F^{G1} \in \mathbb{I}(\sigma F^{GT})$.*

Proof. The proof proceeds similarly to the case *list* of Lemma 9, by applying Lemma 9 to p^G. □

Lemma 11. *Let $p^G = f(p_1^G, \ldots, p_h^G)$ be a clause term well typed in Γ. Let σ be a closed substitution such that $dom(\sigma) = fv(p^{GT})$. For all τ consistent with T for $\Gamma \vdash p^G$, for all $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$, for each $j = 1, \ldots, h$, the following holds:*

$$\sigma'_{|fv(p_j^{G1})} \in \mathbb{I}_{\Gamma,\tau_j}(\sigma_{|fv(p_j^{GT})}),$$

where τ_j is the restriction of τ to $Types(\Gamma \vdash p_j^G)$.

Proof of Theorem 6. Suppose that $\overline{F}$ is derivable from $\mathcal{R}^{G\mathcal{T}}$, and consider a derivation of $\overline{F}$ from $\mathcal{R}^{G\mathcal{T}}$. Let $F \in \mathbb{I}(\overline{F})$. We prove, by induction on the derivation of $\overline{F}$, that F is derivable from $\mathcal{R}^{G1}$.

Figure 5.1: Base case of the proof

Base step: Let the derivation of $\overline{F}$ be as in Figure 5.1(a). By definition of a derivation, we have that $R^{GT} = H^{GT} \Rightarrow C^{GT} \sqsupseteq \overline{F}$, which means that there exists a substitution σ such that:

- $\sigma H^{GT} \subseteq \emptyset$: this means that $H^{GT} = \emptyset$; then $R^G = C^G$.

- $\sigma C^{GT} = \overline{F}$. By hypothesis, there exists Γ such that $\Gamma \vdash R^G$, which means that $\Gamma \vdash C^G$ too. Hence, by Corollary 8, for all $F \in \mathbb{I}(\overline{F})$, there exist τ consistent with T for $\Gamma \vdash C^G$ and $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$ such that $\sigma' C^{G1} = F$.

Hence F can be derived from C^{G1}, so F is derivable from $\mathcal{R}^{G1}$, by the derivation of Figure 5.1(b).

Inductive step: Let the derivation of $\overline{F}$ be as in Figure 5.2.

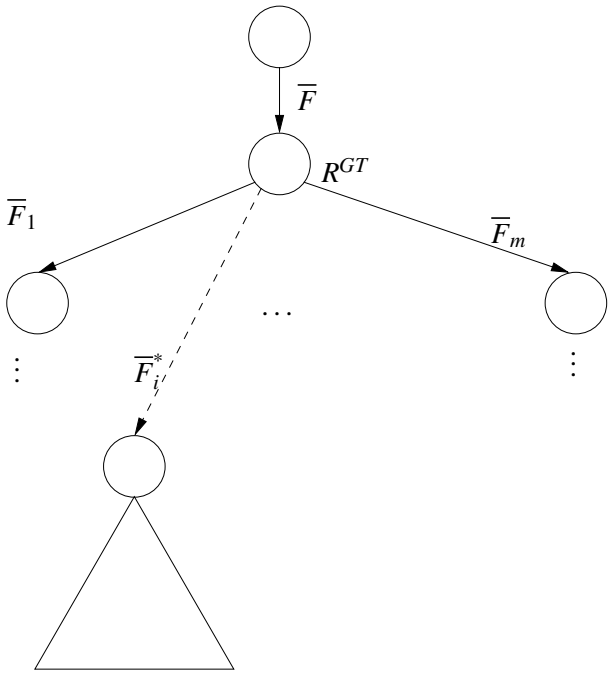

Figure 5.2: Derivation of $\overline{F}$ from $\mathcal{R}^{G\mathcal{T}}$

By definition of a derivation, we have that $R^{GT} \sqsupseteq \overline{F}_1 \wedge \cdots \wedge \overline{F}_m \Rightarrow \overline{F}$, which means that there exists a substitution σ such that $\sigma C^{GT} = \overline{F}$ and $\sigma H^{GT} \subseteq \overline{F}_1 \wedge \cdots \wedge \overline{F}_m$, and $\overline{F}_1, \ldots, \overline{F}_m$ are derivable from $\mathcal{R}^{G\mathcal{T}}$ by subtrees of the derivation of $\overline{F}$.

By hypothesis, we have that $\Gamma \vdash R^G$, so $\Gamma \vdash C^G$. Let $\sigma_C = \sigma_{|fv(C^{GT})}$: we have that $\sigma_C C^{GT} = \overline{F}$, $\Gamma \vdash C^G$ and hypotheses 1 and 2, so by Corollary 8, for each $F \in \mathbb{I}(\overline{F})$, there exist τ_C consistent with T for $\Gamma \vdash C^G$ and $\sigma'_C \in \mathbb{I}_{\Gamma,\tau_C}(\sigma_C)$ such that $\sigma'_C C^{G1} = F$.

Now, we build a τ that extends τ_C to $Types(\Gamma \vdash R^G)$ and a $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$ that extends σ'_C. For each type $[1, L] \in Types(\Gamma \vdash R^G)$:

- if $[1, L] \in Types(\Gamma \vdash C^G)$, then we define $\tau([1, L]) = \tau_C[1, L]$

- if $[1, L] \notin \mathit{Types}(\Gamma \vdash C^G)$ and there exists an index i free in R^G such that $\Gamma \vdash i : [1, L]$, then we define $\tau([1, L]) = i^T$. (This is possible since the free indices of R^G have pairwise distinct types by hypothesis 1.)
- otherwise, we choose any value such that $1 \leq \tau([1, L]) \leq L^T$.

Clearly, as τ_C is consistent with T for $\Gamma \vdash C^G$, also τ is consistent with T for $\Gamma \vdash R^G$. For each $x_{\underbrace{1,\ldots,1}_{h}} \in \mathit{dom}\mathbb{I}(\sigma)$:

- if $x_{\underbrace{1,\ldots,1}_{h}} \in \mathit{dom}(\sigma'_C)$, then we define $\sigma' x_{\underbrace{1,\ldots,1}_{h}} = \sigma'_C x_{\underbrace{1,\ldots,1}_{h}}$.
  In this case, since $\sigma'_C \in \mathbb{I}_{\Gamma,\tau_C}(\sigma_C)$, we obtain that $\sigma' x_{\underbrace{1,\ldots,1}_{h}} = \sigma'_C x_{\underbrace{1,\ldots,1}_{h}} \in \mathbb{I}(\sigma_C x_{\bar{\imath}_1,\ldots,\bar{\imath}_h}) = \mathbb{I}(\sigma x_{\bar{\imath}_1,\ldots,\bar{\imath}_h})$, where $\Gamma \vdash x_{\_} : [1, L_1] \times \cdots \times [1, L_h]$ and $\bar{\imath}_j = \tau_C([1, L_j]) = \tau([1, L_j])$ for all $j = 1, \ldots, h$.
- otherwise, we define $\sigma' x_{\underbrace{1,\ldots,1}_{h}} \in \mathbb{I}(\sigma x_{\bar{\imath}_1,\ldots,\bar{\imath}_h})$ where $\Gamma \vdash x_{\_} : [1, L_1] \times \cdots \times [1, L_h]$ and $\bar{\imath}_j = \tau([1, L_j])$ for each $j = 1, \ldots, h$.

Clearly, $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$.

Now let $H^G = F_1^G \wedge \cdots \wedge F_k^G$. For each $i = 1, \ldots, k$, let $\tau_i = \tau_{|\mathit{Types}(\Gamma \vdash F_i^G)}$ and $\sigma'_i = \sigma'_{|fv(F_i^{G1})} \in \mathbb{I}_{\Gamma,\tau_i}(\sigma_{fv(F_i^{GT})})$ by a reasoning similar to Lemma 11. By Corollary 10, we have that $\sigma'_i F_i^{G1} \in \mathbb{I}(\sigma F_i^{GT})$. Let $F_i = \sigma'_i F_i^{G1} = \sigma' F_i^{G1}$. Therefore, we have $\sigma' C^{G1} = F$ and $\sigma' H^{G1} \subseteq F_1 \wedge \cdots \wedge F_k$, so $R^{G1} \sqsupseteq F_1 \wedge \cdots \wedge F_k \Rightarrow F$.

We have $F_i \in \mathbb{I}(\sigma F_i^{GT})$. Since $\sigma F_i^{GT} \subseteq \sigma H^{GT} \subseteq \overline{F}_1 \wedge \cdots \wedge \overline{F}_m$, we have $F_i \in \mathbb{I}(\overline{F}_i^*)$ for some $\overline{F}_i^* \in \{\overline{F}_1, \ldots, \overline{F}_m\}$. So for each $i = 1, \ldots, k$, $\overline{F}_i^*$ is derivable from $\mathcal{R}^{G\mathcal{T}}$ by subtrees of the initial derivation of $\overline{F}$, hence by induction hypothesis, F_i is derivable from $\mathcal{R}^{G1}$. Therefore, F is derivable from $\mathcal{R}^{G1}$ by the derivation of Figure 5.3. □

5.3 Examples

In this section we illustrate with some examples why the hypotheses of Theorem 6 are necessary. In the first two examples, the theorem does not hold because some

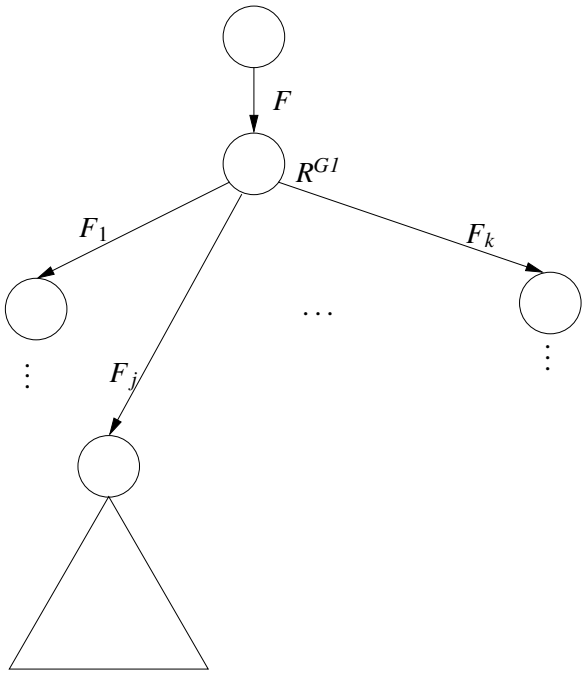

Figure 5.3: Derivation of F from $\mathcal{R}^{G1}$

hypotheses are not satisfied. Consider the following protocol:

(1) $A \to B : \{\!|(a, a)|\!\}_k$
(2) $B \to A : (\{\!|(b, b)|\!\}_k, \{\!|s|\!\}_{f(a,b)})$
(3) $A \to C : \langle \{\!|(a_1, a'_1)|\!\}_k, \ldots, \{\!|(a_L, a'_L)|\!\}_k \rangle$
(4) $C \to A : \langle \langle f(a_1, a'_1), \ldots, f(a_1, a'_L) \rangle, \ldots, \langle f(a_L, a'_1), \ldots, f(a_L, a'_L) \rangle \rangle$

At the beginning, the participants A, B, C share a key k. A first sends to B a fresh nonce a paired with itself and encrypted under k. When B receives it, he creates a fresh nonce b, computes the hash $f(a, b)$ and sends the pair $(\{\!|(b, b)|\!\}_k, \{\!|s|\!\}_{f(a,b)})$, where s is some secret. A can then decrypt $\{\!|(b, b)|\!\}_k$, obtain b, compute $f(a, b)$, decrypt $\{\!|s|\!\}_{f(a,b)}$, and obtain s, but an adversary should be unable to compute s. In the second part of the protocol (Messages 3 and 4), A sends to C a list of L fresh pairs (a_i, a'_i) encrypted with k and C replies with the matrix of the hashes $f(a_i, a'_j)$.

Now, if an attacker sends $\langle \{\!|(a, a)|\!\}_k, \{\!|(b, b)|\!\}_k \rangle$ to C as Message 3, he obtains $f(a, b)$ by decomposition of the list $\langle \langle f(a, a), f(a, b) \rangle, \langle f(b, a), f(b, b) \rangle \rangle$ and can now decrypt $\{\!|s|\!\}_{f(a,b)}$ and obtain the secret s.

However, if we consider only lists of one element, there is no attack: the last message consists of $\langle \langle f(a, a') \rangle \rangle$ if Message 3 was $\{\!|(a, a')|\!\}_k$, so the adversary would need to have $\{\!|(a, b)|\!\}_k$ in order to obtain $f(a, b)$.

The generalized Horn clause for Message 4 is:

$$\text{att}(\mathit{list}(i' \leq L, \mathit{senc}((x_{i'}, y_{i'}), k))) \Rightarrow \text{att}(\mathit{list}(i \leq L, \mathit{list}(j \leq L, f(x_j, y_i))))$$

In this clause, the Hypothesis 2 of Theorem 6 is not satisfied, because the bound indices i and j have the same bound L. If we translate this clause for lists of one element, we obtain

$$\text{att}(\langle \mathit{senc}((x_1, y_1), k)\rangle) \Rightarrow \text{att}(\langle\langle f(x_1, y_1)\rangle\rangle)$$

and with this clause (and other clauses representing this protocol), att(s) is not derivable because att($f(a, b)$) is not derivable, while with lists of length two, as we previously showed, there is an attack: att(s) is derivable. This example confirms that bound indices in the conclusion must have pairwise distinct bounds.

Similarly, we can define a group protocol between a participant B, a chair C, and L group members A_i:

$$\begin{array}{l}(1)\ C \to B : \{\!|(a, a)|\!\}_p \\ (2)\ B \to C : (\{\!|(b, b)|\!\}_p, \{\!|s|\!\}_{f(a,b)}) \\ (3)\ C \to A_i : \langle \{\!|(a_1, a'_1)|\!\}_p, \ldots, \{\!|(a_L, a'_L)|\!\}_p\rangle \\ (4)\ A_i \to C : \langle f(a_1, a'_i), \ldots, f(a_L, a'_i)\rangle\end{array}$$

In this case, the generalized Horn clause for Message 4 is:

$$\text{att}(\mathit{list}(i' \leq L, \mathit{senc}((x_{i'}, y_{i'}), p))) \Rightarrow \text{att}(\mathit{list}(j \leq L, f(x_j, y_i)))$$

where again the Hypothesis 2 of Theorem 6 is not satisfied: the bound index j has the same bound L as the free index i, because they index the same variable $x\_$. As above, att(s) is derivable from the clauses for lists of length 2 but not for lists of length one. There are similar examples regarding Hypothesis 1, for instance with the clause

$$\text{att}(\mathit{list}(i' \leq L, \mathit{senc}((x_{i'}, y_{i'}), p))) \Rightarrow \text{att}(f(x_j, y_i))$$

in which the free indices i and j have the same type $[1, L]$, but it is more difficult to find a concrete protocol that would generate such a clause.

Next, we consider a different kind of example: for the following protocol, the set of Horn clauses satisfies the hypothesis of Theorem 6, so we can apply the theorem. However, the protocol preserves secrecy for lists of length one but not for lists of unbounded length. This illustrates that the approximations made in the translation to Horn clauses are key for our theorem to hold: att(s) is derivable from the clauses, even for lists of length one. Let A and B be the two participants of the protocol that share a key k. Let h be a hash function.

$$\begin{array}{l}(1)\ A \to B : \{\!|e|\!\}_k, (b_1, b_2), \{\!|s|\!\}_{h(h(b_1,e),h(b_2,e))} \\ (2)\ B \to A : \langle x_1, \ldots, x_L\rangle \\ (3)\ A \to B : \langle h(x_1, e), \ldots, h(x_L, e)\rangle\end{array}$$

A chooses a fresh key e and two random nonces b_1, b_2, and sends to B the message $\{\!|e|\!\}_k, (b_1, b_2), \{\!|s|\!\}_{h(h(b_1,e),h(b_2,e))}$ where s is a secret. B obtains e by decryption, computes the key $h(h(b_1, e), h(b_2, e))$, and obtains s by decrypting with this key. Later, B sends a list $\langle x_1, \ldots, x_L \rangle$ and A returns the list containing for each component its hash with e. Clearly, if we consider this protocol for lists of length $L \geq 2$, there is an attack: the attacker can send to A the list $\langle b_1, b_2, \ldots \rangle$ and he obtains at Message 3 the list $\langle h(b_1, e), h(b_2, e), h(\ldots, e) \rangle$. He can then compute the hash $h(h(b_1, e), h(b_2, e))$ and decrypt $\{\!|s|\!\}_{h(h(b_1,e),h(b_2,e))}$ to obtain the secret s. However, if we translate this protocol to lists of length one, we do not find the attack: the attacker can only ask for $\langle h(b_1, e) \rangle$ or $\langle h(b_2, e) \rangle$, but cannot obtain both. For this point to hold, it is important that the participants do not repeat the Messages 2-3 more than once for each session.

ProVerif finds an attack against this protocol (which is a false attack for lists of length one): the abstraction done with the representation by Horn clauses in fact allows the participants to repeat their messages more than once. The translation of the protocol into clauses for lists of length one contains:

A sends the first message:

$$\text{att}((senc(e, k), (b_1, b_2), senc(s, h(h(b_1, e), h(b_2, e))))) \tag{5.1}$$

A receives Message 2 and sends Message 3:

$$\text{att}(\langle x \rangle) \Rightarrow \text{att}(\langle h(x, e) \rangle) \tag{5.2}$$

plus clauses for tuples, encryption, and the hash function h, where $\langle \cdot \rangle$ is a unary function such that $\text{att}(\langle x \rangle) \Rightarrow \text{att}(x)$ and $\text{att}(x) \Rightarrow \text{att}(\langle x \rangle)$. Now, if we query for the secrecy of s, ProVerif will find the attack: $\text{att}(s)$ is derivable from these clauses. Indeed, we get b_1 and b_2 from (5.1), then obtain $h(b_1, e)$ and $h(b_2, e)$ by two applications of (5.2) (note that we apply this clause twice for the same e, while the corresponding action can in fact be applied only once in the protocol itself), then compute $h(h(b_1, e), h(b_2, e))$, and finally obtain s by decrypting $senc(s, h(h(b_1, e), h(b_2, e)))$.

5.4 An Approximation Algorithm

In Section 4.2.1, we gave the representation of the Asokan-Ginzboorg protocol with generalized Horn clauses. However, some of them do not satisfy the hypotheses of Theorem 6. For example, the clause (4.7) does not have a linear conclusion and the same bound appears twice in the conclusion.

5.4.1 Approximation Algorithm

Here we give an algorithm for transforming generalized Horn clauses into clauses that satisfy the hypothesis of Theorem 6. We suppose that the initial set of clauses $\mathcal{R}^G$ satisfies:

Hypothesis 1. *For each clause $R^G \in \mathcal{R}^G$, R^G is well-typed, that is, there exists Γ such that $\Gamma \vdash R^G$, and each variable has indices of pairwise distinct types, that is, if $\Gamma \vdash x\_ : [1, L_1] \times \ldots, \times [1, L_h]$, then $L_1, \ldots, L_h$ are pairwise distinct.*

This hypothesis on the initial clauses is often satisfied in practice. In particular, it is satisfied by our running example, and it should generally be satisfied by group protocols. Indeed, the variables typically have only one index (the number of the group member).

Given a clause R^G well typed in Γ, the approximation algorithm performs the following three steps, until it reaches a fixpoint:

1. Suppose $R^G = H^G \Rightarrow \text{att}(p^G)$, where H^G contains a free index i such that $\Gamma \vdash i : [1, L]$ and p^G contains a bound index j with bound L, or R^G contains two free indices i, j such that $\Gamma \vdash i : [1, L]$ and $\Gamma \vdash j : [1, L]$.

   The algorithm chooses a fresh variable $y\_ = \rho x\_$ for each variable $x\_$ that occurs in R^G with index i, and replaces all occurrences of variables $x\_$ that have index i with $\rho x\_$ (the indices remain the same).

   The obtained clause can then be typed in an environment Γ' equal to Γ except that $\Gamma' \vdash i : [1, L']$ for some fresh bound L' and that $\Gamma' \vdash y\_ : [1, L'_1] \times \cdots \times [1, L'_h]$ if $y\_ = \rho x\_$, $\Gamma \vdash x\_ : [1, L_1] \times \cdots \times [1, L_h]$, and for each $k = 1, \ldots, h$, $L'_k = L_k$ if $L_k \neq L$ and $L'_k = L'$ if $L_k = L$. The indices i and j then have different types in the obtained clause.

2. Suppose $R^G = H_1^G \wedge H_2^G \Rightarrow \text{att}(p^G)$, where p^G contains a clause term $list(i \leq L, p_1^G)$ as well as a clause term $list(j \leq L, p_2^G)$ or a free index j such that $\Gamma \vdash j : [1, L]$, H_1^G contains all hypotheses of R^G in which the bound L appears or a free index of type $[1, L]$ appears, and H_2^G contains the other hypotheses of R^G.

   The algorithm chooses a fresh bound L' and replaces R^G with

   $$H_1^G \wedge H_1'^G \wedge H_2^G \Rightarrow \text{att}(p'^G)$$

   where:

- ρ is a substitution that replaces each variable $x\_$ of H_1^G and p_1^G such that $\Gamma \vdash x\_ : [1, L_1] \times \cdots \times [1, L_h]$ and $L_k = L$ for some $k \in \{1, \ldots, h\}$ with a fresh variable $y\_$ (the indices remain the same); the obtained clause will be typed in an environment Γ' obtained from Γ by adding $\Gamma' \vdash y\_ : [1, L'_1] \times \cdots \times [1, L'_h]$ where, for each $k = 1, \ldots, h$, $L'_k = L_k$ if $L_k \neq L$ and $L'_k = L'$ if $L_k = L$;
- $H_1'^G$ is obtained from ρH_1^G by replacing the bound L with L';
- p'^G is obtained from p^G by replacing $list(i \leq L, p_1^G)$ with $list(i \leq L', p_1'^G)$, where $p_1'^G$ is p_1^G in which all occurrences of variables $x\_$ that have index i have been replaced with $\rho x\_$.

3. Suppose $R^G = H_1^G \wedge H_2^G \Rightarrow \text{att}(p^G)$ where p^G contains at least two occurrences of a variable $x\_$, H_1^G contains all hypotheses of R^G in which $x\_$ appears, and H_2^G contains the other hypotheses of R^G.

   The algorithm chooses a fresh variable $y\_$ and replaces R^G with

   $$H_1^G \wedge H_1'^G \wedge H_2^G \Rightarrow \text{att}(p'^G)$$

   where $H_1'^G$ is obtained from H_1^G by replacing each occurrence of $x\_$ with $y\_$ (the indices remain the same), and p'^G is obtained from p^G by replacing one occurrence of $x\_$ with $y\_$.

Step 1 is applied first, until it cannot be applied. Then step 2 is applied, until there are no list constructors that match the condition. Step 2 may already rename some variables that occur more than once in the conclusion of the clause. Then, when a fixpoint is reached with step 2, we start applying step 3, until no variable occurs more than once in the conclusion. Step 1 ensures that free indices have pairwise distinct types and that free indices of the hypothesis have types distinct from those of bound indices in the conclusion. Step 2 ensures that the bound indices in the conclusion have pairwise distinct bounds and bounds distinct from the bounds of free indices in the conclusion. Step 3 ensures that the conclusion is linear.

This algorithm is similar to the algorithm that transforms any Horn clauses into Horn clauses of the class $\mathcal{H}_1$ [28]. Both algorithms ensure the linearity of the conclusion in the same way (step 3). Step 2 uses an idea similar to step 3 to guarantee that the types of the indices are distinct.

We illustrate this algorithm our running example in Section 5.4.2. The next theorem shows its correctness.

Theorem 12. *Let $\mathcal{R}^G$ be a set of clauses that satisfies Hypothesis 1. The approximation algorithm terminates on $\mathcal{R}^G$ and the final set of clauses $\mathcal{R}'^G$ satisfies the hypothesis of Theorem 6. Moreover, for any fact F, if F is derivable from $\mathcal{R}^{G\mathcal{T}}$, then F is also derivable from $\mathcal{R}'^{G\mathcal{T}}$.*

Proof. First, if a generalized Horn clause R^G is well typed in Γ, then there exists Γ' such that the clause R'^G obtained by the approximation algorithm is well typed in Γ'. Let us construct such a Γ'.

- If R^G is transformed by step 1, then Γ' is equal to Γ except that the type of i is replaced with $i : [1, L']$, and for each variable $x\_$ that occurs in R^G with index i such that $\Gamma \vdash x\_ : [1, L_1] \times \cdots \times [1, L_h]$ and $y\_ = \rho x\_$, we add $y\_ : [1, L'_1] \times \cdots \times [1, L'_h]$ to Γ', where for each $k = 1, \ldots, h$, $L'_k = L_k$ if $L_k \neq L$ and $L'_k = L'$ if $L_k = L$.

  We show that $\Gamma' \vdash R'^G$ by induction on the derivation of $\Gamma \vdash R^G$. In particular, we show that, if p'^G is obtained from p^G by replacing occurrences of variables $x\_$ that have index i with $\rho x\_$, $\Gamma_1 \vdash p^G$, and Γ'_1 is constructed from Γ_1 as Γ' from Γ above, then $\Gamma'_1 \vdash p'^G$. The only interesting case is the one of variables with index i: $\Gamma_1 \vdash x_{i_1,\ldots,i_h}$ has been obtained by type rule (Var) so $\Gamma_1 \vdash x\_ : [1, L_1] \times \cdots \times [1, L_h]$, $\Gamma_1 \vdash i_k : [1, L_k]$ for all $k \in \{1, \ldots, h\}$, $i_l = i$, and $L_l = L$ for some $l \in \{1, \ldots, h\}$. We have $L_k \neq L$ for all $k \neq l$ by Hypothesis 1. Hence, $\Gamma'_1 \vdash y\_ : [1, L'_1] \times \cdots \times [1, L'_h]$ where for each $k = 1, \ldots, h$, $L'_k = L_k$ if $L_k \neq L$ and $L'_k = L'$ if $L_k = L$, $\Gamma'_1 \vdash i_k : [1, L'_k]$ for all $k \in \{1, \ldots, h\}$, so $\Gamma'_1 \vdash y_{i_1,\ldots,i_h}$ by type rule (Var).

- If R^G is transformed by step 2, then $\Gamma' = \Gamma, \Gamma''$, where Γ'' is defined as follows. For each variable $x\_ \in fv(H^G_1) \cup fv(p^G_1)$ such that $\Gamma \vdash x\_ : [1, L_1] \times \cdots \times [1, L_h]$ with $L_k = L$ for some $k \in \{1, \ldots, h\}$ and $\rho x\_ = y\_$, we add $y\_ : [1, L'_1] \times \cdots \times [1, L'_h]$ to Γ'', where for each $k = 1, \ldots, h$, $L'_k = L_k$ if $L_k \neq L$ and $L'_k = L$ if $L_k = L$.

  Again, we show that $\Gamma' \vdash R'^G$ by induction on the derivation of $\Gamma \vdash R^G$.

- If R^G is transformed by step 3, then $\Gamma' = \Gamma, y\_ : [1, L_1] \times \cdots \times [1, L_h]$, where $\Gamma \vdash x\_ : [1, L_1] \times \cdots \times [1, L_h]$.

  Again, we show that $\Gamma' \vdash R'^G$ by induction on the derivation of $\Gamma \vdash R^G$.

Let us show that the algorithm terminates. With step 1, we strictly decrease the number of pairs of indices (i, j) such that i is free in the hypothesis, $\Gamma \vdash i : [1, L]$, and j is bound in the conclusion with bound N, or i and j are both free in R^G with $\Gamma \vdash i : [1, L]$ and $\Gamma \vdash j : [1, L]$, so step 1 terminates. Next, with step 2, we strictly decrease the number of pairs of indices (i, j) such that the conclusion of the clause contains both a clause term $list(i \leq L, p^G_1)$ and a clause term $list(j \leq L, p^G_2)$ or a free index j such that $\Gamma \vdash j : [1, L]$ with bound L, so step 2 terminates. Finally, with step 3, we strictly decrease the number of occurrences of variables that occur more than once in the conclusion, so step 3 terminates. Moreover, since steps 2 and 3 do not create new indices of an already existing type, they do not create new opportunities of applying step 1. Since step 3 does not modify the indices of

the variables, it does not create any list constructor that satisfies the condition for applying step 2. Hence, when step 3 terminates, the algorithm terminates.

The final set of clauses $\mathcal{R}'^G$ satisfies the hypothesis of Theorem 6: for each clause $R'^G \in \mathcal{R}'^G$, R'^G is well typed in some Γ' by the first point shown above, the free indices of R'^G have pairwise distinct types in Γ' (otherwise, we could apply step 1), the conclusion of each clause R'^G is linear (otherwise, we could apply step 3) and the bound indices in the conclusion have pairwise distinct bounds, and bounds different from the bounds of free indices in the judgment $\Gamma' \vdash R'^G$ (otherwise, we could apply step 1 or step 2).

Finally, let us show that, if a fact F is derivable from $\mathcal{R}^{G\mathcal{T}}$, then F is also derivable from $\mathcal{R}'^{G\mathcal{T}}$. We show that, for each transformation step of R^G into R'^G, if a fact F is derivable from $F_1, \ldots, F_l$ using $R^G = H^G \Rightarrow C^G$, that is, there exist T and σ such that $\sigma H^{GT} \subseteq \{F_1, \ldots, F_l\}$ (set inclusion) and $\sigma C^{GT} = F$, then F is also derivable from $F_1, \ldots, F_l$ using R'^G.

- If R^G is transformed into R'^G by step 1, we let $T' = T[L' \mapsto L^T]$ and σ' be the extension of σ with $\sigma' y_{\bar{\imath}_1,\ldots,\bar{\imath}_h} = \sigma x_{\bar{\imath}_1,\ldots,\bar{\imath}_h}$ for all $\bar{\imath}_1, \ldots, \bar{\imath}_h$ when $\rho x_{\_} = y_{\_}$.
  Then $\sigma' R'^{GT'} = \sigma R^{GT}$, so F is also derivable from $F_1, \ldots, F_l$ using R'^G.
- If $R^G = H_1^G \wedge H_2^G \Rightarrow \mathrm{att}(p^G)$ is transformed into $R'^G = H_1^G \wedge H_1'^G \wedge H_2^G \Rightarrow \mathrm{att}(p'^G)$ by step 2, we let $T' = T[L' \mapsto L^T]$ and σ' be the extension of σ with $\sigma' y_{\bar{\imath}_1,\ldots,\bar{\imath}_h} = \sigma x_{\bar{\imath}_1,\ldots,\bar{\imath}_h}$ for all $\bar{\imath}_1, \ldots, \bar{\imath}_h$ when $\rho x_{\_} = y_{\_}$. Then $\sigma' R'^{GT'} = \sigma R^{GT}$ up to copies of hypotheses, since $\sigma' H_1'^{GT'} = \sigma H_1^{GT}$ and $\sigma' p'^{GT'} = \sigma p^{GT}$. Therefore, F is also derivable from $F_1, \ldots, F_l$ using R'^G.
- If R^G is transformed into R'^G by step 3, we let $T' = T$ and σ' be the extension of σ with $\sigma' y_{\bar{\imath}_1,\ldots,\bar{\imath}_h} = \sigma x_{\bar{\imath}_1,\ldots,\bar{\imath}_h}$ for all $\bar{\imath}_1, \ldots, \bar{\imath}_h$. Then $\sigma' R'^{GT'} = \sigma R^{GT}$ up to copies of hypotheses, so F is also derivable from $F_1, \ldots, F_l$ using R'^G. □

5.4.2 Running Example

We apply the approximation algorithm to our running example.

For instance, let us transform the clause (4.7):

$$\begin{aligned}&\mathrm{att}((C, \mathit{senc}(y, pw))) \wedge \mathrm{att}(\mathit{senc}((\mathit{list}(j \le L, z_j), z'), r_i[y]))\\ &\Rightarrow \mathrm{att}((a_i, \mathit{senc}((s_i[y], h((\mathit{list}(j \le L, z_j), z'))), f((\mathit{list}(j \le L, z_j), z')))))\end{aligned}$$

First, as there are two list constructors with the same bound L in the conclusion, we apply step 2 of the algorithm: we rename the bound and variables of one of the two

occurrences of $list(j \leq L, z_j)$ in the conclusion, so we obtain:

$$\begin{aligned}
&\text{att}((C, senc(y, pw))) \wedge \\
&\text{att}(senc((list(j \leq L, z_j), z'), r_i[y])) \wedge \text{att}(senc((list(j \leq L', x_j), z'), r_i[y])) \\
&\Rightarrow \text{att}((a_i, senc((s_i[y], h((list(j \leq L, z_j), z'))), f((list(j \leq L', x_j), z')))))
\end{aligned}$$

Next, as variable z' appears twice in the conclusion, we apply step 3 and obtain:

$$\begin{aligned}
&\text{att}((C, senc(y, pw))) \wedge \\
&\text{att}(senc((list(j \leq L, z_j), z'), r_i[y])) \wedge \text{att}(senc((list(j \leq L', x_j), z'), r_i[y])) \wedge \\
&\text{att}(senc((list(j \leq L, z_j), x'), r_i[y])) \wedge \text{att}(senc((list(j \leq L', x_j), x'), r_i[y])) \\
&\Rightarrow \text{att}((a_i, senc((s_i[y], h((list(j \leq L, z_j), z'))), f((list(j \leq L', x_j), x')))))
\end{aligned} \tag{5.3}$$

Finally, this clause satisfies the hypothesis of Theorem 6. All clauses $\mathcal{R}^G$ listed in Section 4.2.1, which represent our running example, are transformed in a similar way, yielding clauses $\mathcal{R}'^G$. The clause (4.5) is transformed by giving distinct names to the three occurrences of y in the conclusion by step 3, yielding the clause:

$$\begin{aligned}
&\text{att}((C, senc(y, pw))) \wedge \text{att}((C, senc(x, pw))) \wedge \text{att}((C, senc(z, pw))) \\
&\Rightarrow \text{att}((a_i, senc(penc((r_i[y], s_i[x]), z), pw)))
\end{aligned}$$

The clause (4.6) is transformed by applying step 2 twice so that the two lists and the free index i of the conclusion have different bounds:

$$\begin{aligned}
&\bigwedge_{j \leq L} \text{att}((a_j, senc(penc((v_j, w_j), pk(d)), pw))) \wedge \\
&\bigwedge_{j \leq L'} \text{att}((a_j, senc(penc((x_j, y_j), pk(d)), pw))) \wedge \\
&\bigwedge_{j \leq L''} \text{att}((a_j, senc(penc((z_j, t_j), pk(d)), pw))) \\
&\Rightarrow \text{att}(senc((list(j \leq L'', t_j), s'[list(j \leq L', (x_j, y_j))]), v_i))
\end{aligned}$$

As we want to prove the secrecy of the session key K, we need to transform also the clause (4.9), by applying step 2 so that the two lists of the conclusion have different bounds:

$$\begin{aligned}
&\bigwedge_{j \leq L} \text{att}((a_j, senc(penc((v_j, w_j), pk(d)), pw))) \wedge \\
&\text{att}((a_i, senc((w_i, h((list(j \leq L, w_j), s'[list(j \leq L, (v_j, w_j))])), \\
&\qquad f((list(j \leq L, w_j), s'[list(j \leq L, (v_j, w_j))])))) \wedge \\
&\bigwedge_{j \leq L'} \text{att}((a_j, senc(penc((x_j, y_j), pk(d)), pw))) \wedge \\
&\text{att}((a_i, senc((y_i, h((list(j \leq L', y_j), s'[list(j \leq L', (x_j, y_j))])), \\
&\qquad f((list(j \leq L', y_j), s'[list(j \leq L', (x_j, y_j))])))) \\
&\Rightarrow \text{att}(senc(s''_C, f((list(j \leq L, w_j), s'[list(j \leq L', (x_j, y_j))]))))
\end{aligned}$$

The other clauses are left unchanged since they already satisfy the hypothesis of Theorem 6.

We can now translate these clauses into Horn clauses with lists of length one, obtaining the set $\mathcal{R}'^{G1}$. For example, clause (5.3) can be translated into:

$$\begin{aligned}&\text{att}((C, \mathit{senc}(y, pw))) \wedge \\ &\text{att}(\mathit{senc}((\langle z_1\rangle, z'), r_1[y])) \wedge \text{att}(\mathit{senc}((\langle x_1\rangle, z'), r_1[y])) \wedge \\ &\text{att}(\mathit{senc}((\langle z_1\rangle), x'), r_1[y])) \wedge \text{att}(\mathit{senc}((\langle x_1\rangle, x'), r_1[y])) \\ &\Rightarrow \text{att}((a_1, \mathit{senc}((s_1[y], h((\langle z_1\rangle, z'))), f((\langle x_1\rangle, x')))))\end{aligned}$$

The other clauses are translated in a similar way. We have then shown that $\text{att}(s''_a)$ and $\text{att}(s''_C)$ are not derivable from $\mathcal{R}'^{G1}$, using the automatic verifier ProVerif with the input file given at the page `http://prosecco.gforge.inria.fr/personal/mpaiola/publications/files/Thesis/AsokanGinzboorg`. By Theorem 6, we conclude that $\text{att}(s''_a)$ (resp. $\text{att}(s''_C)$) is not derivable from $\mathcal{R}'^{G\mathcal{T}}$, so by Theorem 12, $\text{att}(s''_a)$ (resp. $\text{att}(s''_C)$) is not derivable from $\mathcal{R}^{G\mathcal{T}}$. Therefore, the Asokan-Ginzboorg protocol preserves the secrecy of s''_a (resp. s''_C), that is, it preserves the secrecy of the key K that a_i has (resp. that the chair C has).

Part II

Heterogeneous Lists

CHAPTER 6

Generalized Horn Clauses for Heterogeneous Lists

Contents

The result presented in Part 1, is limited to secrecy and to a restricted class of protocols. In fact with that result we can only handle protocols that treat all elements of lists uniformly. When the reduction we propose does not apply, we need a different approach. In this second part of the thesis, we provide a practical algorithm that can prove both secrecy and authentication properties of protocols that manipulate different list elements in different ways. To reach this goal, we extend the automatic verifier ProVerif to handle protocols with lists of unbounded lengths. In this chapter, we first introduce a new running example, a version of the SOAP extension to XML signatures and then motivate this work. We illustrate the need of a further extension of generalized Horn clauses, give the new syntax and semantics and define a type system for these clauses. Finally we define the correspondence between this new version of generalized Horn clauses and Horn Causes.

6.1 Motivation

6.1.1 Running Example

As a running example, we consider a simple protocol based on the SOAP extension to XML signatures [18]. SOAP envelopes are XML documents with a mandatory Body together with an optional Header. The Body may contain a request, response or a fault message. The Header contains information about the message: in particular, the SOAP header can carry a digital signature, as follows:

```
<Envelope>
  <Header>
    <Signature>
      <SignedInfo>
        <Reference URI="#theBody">
          <DigestValue> hash of the body </DigestValue>
        </Reference>
        <Reference URI="#x1">
          <DigestValue> hash of the content of x1
          </DigestValue>
        </Reference>
        ...
      </SignedInfo>
      <SignatureValue>
        signature of SignedInfo with key skC
      </SignatureValue>
    </Signature>
  </Header>
  <Body Id="#theBody"> request </Body>
</Envelope>
```

The Signature header contains two components. The first component is a SignedInfo element: it contains a list of references to the elements of the message that are signed. Each reference is designated by its identifier and carries a DigestValue, a hash of the corresponding content. This hash may be computed with the hash function SHA-1. The second component of the Signature header is the signature of the SignedInfo element with a secret key sk_C.

We consider a simple protocol in which a client C sends such a document to a server S. The server processes the document and checks the signature before authorizing the request given in the Body: if the SignedInfo contains a Reference to an element with tag Body, then he will authorize the request. This protocol should guarantee that the server authorizes only requests signed by legitimate clients.

However, as shown in [36], this protocol is subject to a wrapping attack: an attacker can intercept an envelope and create a new document wrapping the Body in the Header and adding a new body with a different id and a new request. The server will verify the signature and authorize the fake request of the attacker. This is possible because the server does not check that the signed Body is the one he authorizes.

6.1.2 Need for a New Version of Generalized Horn Clauses

In order to model this protocol, we suppose that the XML parser parses the SOAP envelope as a pair. The first component is a list of triplets (tag, id, corresponding content) and the second component is the content of the body (that is, the request). The list in the first component is useful to retrieve the content of an element from its id by looking up the list. The content of the Signature header is modeled as a pair (*SignedInfo*, *SignatureValue*). *SignedInfo* is a list of pairs containing an id and the hash of the corresponding content. *SignatureValue* is the signature of *SignedInfo* with a secret key sk_C.

A given XML document can then be represented using lists of fixed length (and other standard functions). As in the previous part of the thesis, we denote by $\langle p_1, \ldots, p_h \rangle$ a list of fixed length h; such lists are modeled as a family of constructors, one for each length.

However, the receiver of a SOAP envelope accepts messages containing any number of headers. Moreover, the SignedInfo element may contain references to any number of elements of the document. Hence, we use the construct $list(i \leq L, p_i)$ introduced in Section 4.2: $(list(i \leq L, (tag_i, id_i, cont_i)), r)$, where tag_i, id_i, and $cont_i$ are variables representing tags, identifiers, and contents respectively and r is the variable for the request.

The server has to check the signature, before authorizing the request r. He has to verify that the list contains a tag tag_j equal to Signature and that $cont_j$ contains a correct signature. These checks can be represented by equations: $tag_j \doteq \mathsf{Signature}$, $cont_j \doteq (sinfo, sign(sinfo, \mathsf{sk}_C))$, $sinfo \doteq list(k \leq L', (id_{\phi(k)}, sha1(cont_{\phi(k)})))$, where the function ϕ is a mapping from the index k of each element in the *sinfo* list to its index $\phi(k)$ in the list that represents the whole message. Additionally, one of the signed elements must have tag Body, that is, $\phi(m) \doteq d$, $tag_d \doteq \mathsf{Body}$. In the list $list(i \leq L, (tag_i, id_i, cont_i))$ all elements tag_i, id_i, and $cont_i$ need to have the same form, therefore we cannot replace directly the variables with the values given by the equations. So, we keep the equations in the clause for future use. Since the equations give different forms to different elements of a list, they allow us to handle protocols that treat elements of list non-uniformly.

After modeling the clauses, we need to perform resolution. Let $[1, L] = \{1, \ldots, L\}$ and suppose that we resolve $R_2 = \bigwedge_{i \in [1,L]} \mathrm{att}(sha1(cont_{\phi(i)})) \Rightarrow \mathrm{event}(e(r))$ with the clause $R_1 = \mathrm{att}(x) \Rightarrow \mathrm{att}(sha1(x))$, that is, we use R_1 to derive one hypothesis of R_2, say the one

of index $i = k_1 \in [1, L]$. So we unify $\text{att}(sha1(x))$ with $\text{att}(sha1(cont_{\phi(k_1)}))$, yielding the equality $x \doteq cont_{\phi(k_1)}$. The obtained clause would then be:

$$\text{att}(x_1) \wedge \bigwedge_{i \in [1,L] \setminus \{k_1\}} \text{att}(sha1(cont_{\phi(i)})) \wedge \{x_1 \doteq cont_{\phi(k_1)}\} \Rightarrow \text{event}(e(r))$$

The variable x is renamed into x_1 to distinguish it from the variable x in R_1. The obtained clause can then again be resolved with R_1 for some $i = k_2 \in [1, L] \setminus \{k_1\}$, yielding

$$\text{att}(x_1) \wedge \text{att}(x_2) \wedge \bigwedge_{i \in [1,L] \setminus \{k_1,k_2\}} \text{att}(sha1(cont_{\phi(i)})) \wedge$$
$$\{x_1 \doteq cont_{\phi(k_1)}, x_2 \doteq cont_{\phi(k_2)}\} \Rightarrow \text{event}(e(r))$$

which can again be resolved with R_1, and so on. Since L is not bounded, such resolution steps yield an infinite loop. To avoid this loop, we define a resolution step that simultaneously resolves R with several instances of R' for i in any non-empty subset $I \subseteq [1, L]$. We name such a step *hyperresolution* by analogy with the hyperresolution rule that allows one to resolve one clause with several clauses [27]. To perform hyperresolution, we first transform R_1 into a clause R'_1 corresponding to the combination of several instances of R_1, one for each $i \in I$. This step is named *immersion* and yields $R'_1 = \bigwedge_{i \in I} \text{att}(x_i) \Rightarrow \text{att}(sha1(x_i))$. We can then perform hyperresolution with R_2 and obtain:

$$\bigwedge_{i \in I} \text{att}(x_i) \wedge \bigwedge_{i \in [1,L] \setminus I} \text{att}(sha1(cont_{\phi(i)})) \wedge \{\bigwedge_{i \in I} x_i \doteq cont_{\phi(i)}\} \Rightarrow \text{event}(e(r)).$$

When $I = [1, L]$, the second hypothesis is removed and the substitution of $cont_{\phi(i)}$ for x_i is performed for all i, so we obtain

$$\bigwedge_{i \in [1,L]} \text{att}(cont_{\phi(i)}) \Rightarrow \text{event}(e(r)).$$

When $I \neq [1, L]$, to have a more symmetric notation and to simplify the language of sets used in conjunctions, we introduce a symbol I' for $[1, L] \setminus I$, and keep the constraint that $I \uplus I' = [1, L]$: I and I' are disjoint and their union is $[1, L]$. The clause is then denoted by

$$I \uplus I' = [1, L], \bigwedge_{i \in I} \text{att}(cont_{\phi(i)}) \wedge \bigwedge_{i \in I'} \text{att}(sha1(cont_{\phi(i)})) \Rightarrow \text{event}(e(r))$$

where I and I' represent non-empty subsets of $[1, L]$.

6.2 Syntax

This section formally defines the syntax and semantics of the extension of generalized Horn clauses, which were motivated informally previously.

$\iota ::=$	index term
$\quad i$	index variable
$\quad \phi(\iota_1, \ldots, \iota_h)$	function application
$p^G, p'^G ::=$	clause terms
$\quad x_{\iota_1,\ldots,\iota_h}$	variable ($h \geq 0$)
$\quad f(p_1^G, \ldots, p_l^G)$	function application
$\quad a^{L_1,\ldots,L_h}_{\iota_1,\ldots,\iota_h}[p_1^G, \ldots, p_l^G]$	indexed names
$\quad list(i \leq M, p^G)$	list constructor
$J ::=$	set computation
$\quad I$	set symbol
$\quad \{()\}$	singleton
$\quad J \times [1, L]$	product
$C ::= \bigwedge_{(i_1,\ldots i_h)\in J}$	conjunction
$F^G ::= C\, pred(p_1^G, \ldots, p_l^G)$	fact
$E ::= C\, p^G \doteq p'^G$	equation over terms
$E' ::= C\, \iota \doteq \iota'$	equation over indices
$\mathcal{E} ::= \{E_1, \ldots, E_n, E'_1, \ldots, E'_{n'}\}$	set of equations
$\mathcal{I} ::= I_1 \uplus \cdots \uplus I_h = J$	constraint over sets
$Cts ::= \{\mathcal{I}_1, \ldots, \mathcal{I}_n\}$	set of constraints
$R^G ::= Cts,\ F_1^G \wedge \cdots \wedge F_n^G \wedge \mathcal{E} \Rightarrow pred(p_1^G, \ldots, p_l^G)$	generalized Horn clause

Figure 6.1: Syntax of generalized Horn clauses

The syntax of these new clauses is defined in Figure 6.1. The syntax of clause terms p^G is adapted with the introduction of the new indices ι. The variables may have indices $x_{\iota_1,\ldots,\iota_h}$. We consider tuples $(p_1, \ldots, p_l^G)$ and lists of fixed length $\langle p_1^G, \ldots, p_l^G \rangle$ as particular constructors. We suppose that the implementation of the protocol uses distinct encodings for tuples and for lists, so that they cannot be confused with each other. The clause term $a^{L_1,\ldots,L_h}_{\iota_1,\ldots,\iota_h}[p_1^G, \ldots, p_n^G]$ represents a fresh name a indexed by ι_1 in $[1, L_1]$, …, ι_h in $[1, L_h]$. Differently from Section 4.2, we consider names without indices a as indexed names with $h = 0$. Finally, as in the syntax introduced in Part 1, we find the constructor $list(i \leq L, p^G)$ to represent lists of length L, where L is an unknown bound.

Facts depending on indices become $\bigwedge_{(i_1,\ldots,i_h)\in J} pred(p_1^G, \ldots, p_l^G)$. The set J can be a product of a set symbol I or a singleton $\{()\}$ and of different sets $[1, L]$, depending on different bounds L. The symbol $[1, L]$ represents the set $\{1, \ldots, L\}$. The symbol $[1, L]$ allows us to keep the information that an index ranges over the full set of indices of bound L, while other set symbols I represent an unknown, non-empty set of indices. We write $[1, L]$ instead of $\{()\} \times [1, L]$. The conjunction $C = \bigwedge_{(i_1,\ldots,i_h)\in J}$ with $J = \{()\}$ and $h = 0$ is omitted: the fact

$C\,pred(p_1^G, \ldots, p_l^G)$ is then simply $pred(p_1^G, \ldots, p_l^G)$.

In the generalized Horn clause Cts, $F_1^G \wedge \cdots \wedge F_n^G \wedge \mathcal{E} \Rightarrow pred(p_1^G, \ldots, p_l^G)$, there are two new members: a *set of constraints* Cts and a *set of equations* $\mathcal{E}$. The first one is a set of constraints on sets used in conjunctions: the constraint $I_1 \uplus \ldots \uplus I_h = J$ means that $I_1, \ldots, I_h$ are pairwise disjoint and their union is J. The second one is a set of equations that represent the substitutions that cannot be done because the equations hold for some but not all values of the indices. These equations can be equations over clause terms E and equations over indices E'. The clause Cts, $F_1^G \wedge \cdots \wedge F_n^G \wedge \mathcal{E} \Rightarrow pred(p_1^G, \ldots, p_l^G)$ means that, if the constraints in Cts are satisfied, and the facts $F_1^G, \ldots, F_n^G$ and the equations in $\mathcal{E}$ hold, then the fact $pred(p_1^G, \ldots, p_l^G)$ also holds. The conclusion of a clause does not contain a conjunction C: we can simply leave the indices of $pred(p_1^G, \ldots, p_l^G)$ free to mean that $pred(p_1^G, \ldots, p_l^G)$ can be concluded for any value of these indices.

The clauses are required to satisfy the following invariants:

1. In a conjunction $\bigwedge_{(i_1,\ldots,i_h)\in J}$, the indices $i_1, \ldots, i_h$ are pairwise distinct.
2. Each set symbol I occurs at most once in each constraint in Cts.
3. Each set symbol I occurs in at most one left-hand side of a constraint in Cts.
4. If a set symbol I occurs in a conjunction or in the right-hand side of a constraint in Cts, then it occurs in exactly one left-hand side of a constraint in Cts.

We use H^G for hypothesis and C^G for conclusions. When Cts or $\mathcal{E}$ are empty, we omit them in the clause.

6.3 Representation of the Protocol

The representation of the abilities of the attacker includes the clauses given in Section 3.3.1. For our running example, att($pk(\mathsf{sk}_C)$), att(Signature), att(Body) represent that the attacker initially knows the public key $pk(\mathsf{sk}_C)$ and the constants Signature and Body, and the clauses

$$\begin{array}{ll}
\text{att}(b[x]) & \text{att}((x,y)) \Rightarrow \text{att}(x) \\
\text{att}(x) \wedge \text{att}(y) \Rightarrow \text{att}(sign(x,y)) & \text{att}((x,y)) \Rightarrow \text{att}(y) \\
\text{att}(x) \Rightarrow \text{att}(pk(x)) & \text{att}(x) \Rightarrow \text{att}(\langle x \rangle) \\
\text{att}(x) \Rightarrow \text{att}(sha1(x)) & \text{att}(\langle x \rangle) \Rightarrow \text{att}(x) \\
\text{att}(x) \wedge \text{att}(y) \Rightarrow \text{att}((x,y)) &
\end{array}$$

represent that the attacker can create fresh names, sign messages, create its own public keys, apply hash functions, compose and decompose pairs and lists of length one. We have similar clauses for triples and lists of length two. (These arities are the only ones used in our example.)

Additionally, as in the other part of the thesis, we have clauses for *list*:

$$\bigwedge_{i\in[1,M]} \text{att}(x_i) \Rightarrow \text{att}(\mathit{list}(j \leq M, x_j)) \tag{6.1}$$

$$\text{att}(\mathit{list}(j \leq M, x_j)) \Rightarrow \text{att}(x_i) \tag{6.2}$$

Let us now give clauses that represent the protocol itself. To model the authentication, we use two events b and e. The event $b(r)$ means that the client sends the request r; the event $e(r)$ means that the server authorizes the request r. Our goal is to prove that, if $e(r)$ is executed, then $b(r)$ has been executed. We suppose that the only element signed by the client is the Body. Hence the document can be represented as follows: $(\langle(\mathsf{Signature}, \mathsf{ids}, (\langle(\mathsf{idb}, \mathit{sha1}(\mathsf{Req}))\rangle, \mathit{sign}((\langle(\mathsf{idb}, \mathit{sha1}(\mathsf{Req}))\rangle, \mathsf{sk}_C)))), (\mathsf{Body}, \mathsf{idb}, \mathsf{Req})\rangle, \mathsf{Req})$, where ids is the identifier of the Signature and idb the one of the Body. The client executes the event $b(\mathsf{Req})$, then sends the SOAP envelope on the network and the attacker intercepts it, so we have the clause:

$$\begin{aligned}\text{m-event}(b(\mathsf{Req})) \Rightarrow \text{att}((\langle(\mathsf{Signature}, \mathsf{ids}, (\langle(\mathsf{idb}, \mathit{sha1}(\mathsf{Req}))\rangle,\\ \mathit{sign}((\langle(\mathsf{idb}, \mathit{sha1}(\mathsf{Req}))\rangle, \mathsf{sk}_C)))), (\mathsf{Body}, \mathsf{idb}, \mathsf{Req})\rangle, \mathsf{Req})).\end{aligned} \tag{6.3}$$

The server expects a document $(\mathit{list}(i \leq L, (\mathit{tag}_i, \mathit{id}_i, \mathit{cont}_i)), r)$; he checks the signature, and if the check succeeds, he executes the event $e(r)$:

$$\begin{aligned}&\text{att}((\mathit{list}(i \leq L, (\mathit{tag}_i, \mathit{id}_i, \mathit{cont}_i)), r)) \wedge\\ &\{\mathit{tag}_j \doteq \mathsf{Signature}, \mathit{cont}_j \doteq (\mathit{sinfo}, \mathit{sign}(\mathit{sinfo}, \mathsf{sk}_C)),\\ &\ \mathit{sinfo} \doteq \mathit{list}(k \leq L', (\mathit{id}_{\phi(k)}, \mathit{sha1}(\mathit{cont}_{\phi(k)}))),\\ &\ \phi(m) \doteq d, \mathit{tag}_d \doteq \mathsf{Body}\} \Rightarrow\\ &\qquad \text{event}(e(r))\end{aligned} \tag{6.4}$$

This clause uses the equations explained in Section 6.1.2.

As already mentioned in Section 6.1.1, this protocol has a wrapping attack. The corrected version of this protocol additionally requires that the signed body is the one that the server authorizes. This additional check can be modeled by adding the equation $\mathit{cont}_d \doteq r$ in the clause (6.4).

6.4 Type System

In this section, we define a simple type system for the new version of generalized Horn clauses.

Definition 9. *An index i is bound if:*

- *it appears as an index of a conjunction defining a fact, so, for instance, in the fact $\bigwedge_{(i_1,\ldots,i_h)\in J} pred(p_1^G,\ldots,p_l^G)$, $i_1,\ldots,i_h$ are bound in $pred(p_1^G,\ldots,p_l^G)$.*
- *it appears as an index for a list constructor, that is, in the clause term $list(i \leq L, p^G)$, i is bound in p^G.*

We identity facts, equations, and clauses up to renaming of bound indices. For simplicity, we suppose that the bound indices of clauses have been renamed so that they have pairwise distinct names, and names distinct from the names of free indices. The set of free indices of a fact F^G, of a clause R^G, or of an hypothesis H^G is denoted by $fi(F^G)$, $fi(R^G)$, $fi(H^G)$ respectively.

In the type system, the type environment Γ is a list of type declarations:

- $i : [1, L]$ means that i is of type $[1, L]$, that is, intuitively, the value of index i can vary between 1 and the value of the bound L;
- $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L]$ means that the function ϕ expects as input h indices of types $[1, L_j]$, for $j = 1, \ldots, h$ and computes an index of type $[1, L]$;
- $x\_ : [1, L_1] \times \cdots \times [1, L_h]$ means that the variable x expects indices of types $[1, L_1]$, $\ldots$, $[1, L_h]$;
- $I : [1, L_1] \times \cdots \times [1, L_h]$ means that the elements of set I are tuples of h indices, each of type $[1, L_j]$ for $j = 1, \ldots, h$.

The type rules are given in Figure 6.2. The type system defines the judgments:

- $\Gamma \vdash \iota : [1, L]$, which means that ι has type $[1, L]$ in the type environment Γ, by rules (EnvIndex) and (Index);
- $\Gamma \vdash J : [1, L_1] \times \cdots \times [1, L_h]$, which means that the elements of set J are tuples of h indices, each of type $[1, L_j]$ for $j = 1, \ldots, h$ in the type environment Γ, by rules (EmptySet), (EnvSet), and (Set);

$$\frac{i : [1, L] \in \Gamma}{\Gamma \vdash i : [1, L]}(\text{EnvIndex})$$

$$\frac{\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma \qquad \Gamma \vdash \iota_1 : [1, L_1] \ldots \Gamma \vdash \iota_h : [1, L_h]}{\Gamma \vdash \phi(\iota_1, \ldots, \iota_h) : [1, L]}(\text{Index})$$

$$\Gamma \vdash \{()\} : () \,(\text{EmptySet}) \qquad \frac{I : [1, L_1] \times \cdots \times [1, L_h] \in \Gamma}{\Gamma \vdash I : [1, L_1] \times \cdots \times [1, L_h]}(\text{EnvSet})$$

$$\frac{\Gamma \vdash J : [1, L_1] \times \cdots \times [1, L_h]}{\Gamma \vdash J \times [1, L] : [1, L_1] \times \cdots \times [1, L_h] \times [1, L]}(\text{Set})$$

$$\frac{x\_ : [1, L_1] \times \cdots \times [1, L_h] \in \Gamma \qquad \Gamma \vdash \iota_1 : [1, L_1] \ldots \Gamma \vdash \iota_h : [1, L_h]}{\Gamma \vdash x_{\iota_1, \ldots, \iota_h}}(\text{Var})$$

$$\frac{\Gamma \vdash p_1^G \ldots \Gamma \vdash p_h^G}{\Gamma \vdash f(p_1^G, \ldots, p_h^G)}(\text{Fun}) \qquad \frac{\Gamma, i : [1, L] \vdash p^G}{\Gamma \vdash \mathit{list}(i \leq L, p^G)}(\text{List})$$

$$\frac{\Gamma \vdash p_1^G \ldots \Gamma \vdash p_h^G \qquad \Gamma \vdash \iota_1 : [1, L_1] \ldots \Gamma \vdash \iota_h : [1, L_h]}{\Gamma \vdash a^{L_1, \ldots, L_h}_{\iota_1, \ldots, \iota_h}[p_1^G, \ldots, p_h^G]}(\text{Name})$$

$$\frac{\begin{array}{c}\Gamma, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash p_1^G \\ \cdots \\ \Gamma \vdash J : [1, L_1] \times \cdots \times [1, L_h] \qquad \Gamma, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash p_l^G\end{array}}{\Gamma \vdash \bigwedge_{(i_1, \ldots, i_h) \in J} \mathit{pred}(p_1^G, \ldots, p_l^G)}\,(\text{Fact})$$

$$\frac{\begin{array}{c}\Gamma, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash p^G \\ \Gamma \vdash J : [1, L_1] \times \cdots \times [1, L_h] \qquad \Gamma, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash p'^G\end{array}}{\Gamma \vdash \bigwedge_{(i_1, \ldots, i_h) \in J} p^G \doteq p'^G}\,(\text{TermEq})$$

$$\frac{\begin{array}{c}\Gamma, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash \iota : [1, L] \\ \Gamma \vdash J : [1, L_1] \times \cdots \times [1, L_h] \qquad \Gamma, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash \iota' : [1, L]\end{array}}{\Gamma \vdash \bigwedge_{(i_1, \ldots, i_h) \in J} \iota \doteq \iota'}\,(\text{IndEq})$$

$$\frac{\begin{array}{c}\Gamma \vdash I_1 : [1, L_1] \times \cdots \times [1, L_k] \\ \cdots \\ \Gamma \vdash I_h : [1, L_1] \times \cdots \times [1, L_k] \qquad \Gamma \vdash J : [1, L_1] \times \cdots \times [1, L_k]\end{array}}{\Gamma \vdash I_1 \uplus \cdots \uplus I_h = J}\,(\text{SetConstr})$$

$$\frac{\forall \mathcal{I} \in \mathit{Cts}, \Gamma \vdash \mathcal{I} \quad \forall j \leq n, \Gamma \vdash F_j^G \quad \forall j \leq m, \Gamma \vdash E_j \quad \forall j \leq m', \Gamma \vdash E'_j \quad \Gamma \vdash F^G}{\Gamma \vdash \mathit{Cts}, F_1^G \wedge \cdots \wedge F_n^G \wedge \{E_1, \ldots, E_m, E'_1, \ldots, E'_{m'}\} \Rightarrow F^G}(\text{Clause})$$

Figure 6.2: Type system for generalized Horn clauses

- $\Gamma \vdash p^G, \Gamma \vdash F^G, \Gamma \vdash E, \Gamma \vdash E', \Gamma \vdash \mathcal{I}, \Gamma \vdash R^G$, which mean that p^G, F^G, E, E', $\mathcal{I}$, R^G, respectively, are well-typed in the type environment Γ.

Most type rules are straightforward. For instance, the rule (Var) means that $x_{i_1,\ldots,i_h}$ is well-typed when the types expected by x for its indices match the types of $i_1, \ldots, i_h$. In the rule (Name), the type of the indices $\iota_1, \ldots, \iota_h$ of $a^{L_1,\ldots,L_h}_{\iota_1,\ldots,\iota_h}$ is $[1, L_1], \ldots, [1, L_h]$ respectively.

We suppose that all clauses are well-typed, and we consider that each clause R^G comes with its type environment Γ such that $\Gamma \vdash R^G$. It is easy to verify that the clauses of Section 6.3 are well-typed. Clause (6.1) is well-typed in the environment $x\_ : [1, L]$, (6.2) in the environment $x\_ : [1, L], i : [1, L]$, and Clause (6.4) in the following environment $tag\_ : [1, L], id\_ : [1, L], cont\_ : [1, L], sinfo\_ : [\,], r\_ : [\,], j : [1, L], \phi : [1, L'] \to [1, L], m : [1, L'], d : [1, L]$. The notation $x\_ : [\,]$ means that the variable x has no index.

6.5 From Generalized Horn Clauses to Horn Clauses

A generalized Horn clause represents several Horn clauses: for each value of the bounds L, set symbols I, functions ϕ, and free indices i that occur in a generalized Horn clause R^G, R^G corresponds to a certain Horn clause. This section formally defines this correspondence.

Definition 10. *Given a well-typed generalized Horn clause $\Gamma \vdash R^G$, an environment T for $\Gamma \vdash R^G$ is a function that associates:*

- *to each bound L that appears in R^G or Γ an integer L^T;*
- *to each index i such that $i : [1, L] \in \Gamma$, an index $i^T \in \{1, \ldots, L^T\}$;*
- *to each index function ϕ such that $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma$, a function $\phi^T : \{1, \ldots, L_1^T\} \times \cdots \times \{1, \ldots, L_h^T\} \to \{1, \ldots, L^T\}$.*
- *to each set symbol I such that $\mathcal{I} : [1, L_1] \times \cdots \times [1, L_h] \in \Gamma$, a set I^T such that $\emptyset \subset I^T \subseteq \{1, \ldots, L_1^T\} \times \cdots \times \{1, \ldots, L_h^T\}$;*

We define similarly environments for $\Gamma \vdash p^G, \Gamma \vdash F^G, \Gamma \vdash E, \Gamma \vdash E', \Gamma \vdash \mathcal{I}$.

Given an environment T and values $v_1, \ldots, v_h$, we write $T[i_1 \mapsto v_1, \ldots, i_h \mapsto v_h]$ for the environment that associates to indices $i_1, \ldots, i_h$ the values $v_1, \ldots, v_h$ respectively and that maps all other values like T.

Given an environment T for $\Gamma \vdash R^G$, the generalized Horn clause R^G is translated into the standard Horn clause R^{GT} defined as follows. We denote respectively $p^{GT}, E^T, \ldots$ the translation of $p^G, E, \ldots$ using the environment T.

The translation of an index term ι such that $\Gamma \vdash \iota : [1, L]$ is an integer $\iota^T \in \{1, \ldots, L^T\}$ defined as follows:

$$\iota^T = \begin{cases} i^T & \text{if } \iota = i \\ \phi^T(\iota_1^T, \ldots, \iota_h^T) & \text{if } \iota = \phi(\iota_1, \ldots, \iota_h) \end{cases}$$

The translation of a clause term p^G is defined as follows:

$$\begin{aligned} (x_{\iota_1,\ldots,\iota_h})^T &= x_{\iota_1^T,\ldots,\iota_h^T} \\ f(p_1^G, \ldots, p_l^G)^T &= f(p_1^{GT}, \ldots, p_l^{GT}) \\ a^{L_1,\ldots L_h}_{\iota_1,\ldots,\iota_h}[p_1^G, \ldots, p_l^G]^T &= a^{L_1^T,\ldots,L_h^T}_{\iota_1^T,\ldots,\iota_h^T}[p_1^{GT}, \ldots, p_l^{GT}] \\ \mathit{list}(i \le L, p^G)^T &= \langle p^{GT[i \mapsto 1]}, \ldots, p^{GT[i \mapsto L^T]} \rangle \end{aligned}$$

The translation of $\mathit{list}(i \le L, p^G)$ is a list of length L^T.

The translation of a set computation J is a set of tuples defined by:

$$J^T = \begin{cases} I^T & \text{if } J = I \\ \{\,()\,\} & \text{if } J = \{()\} \\ J'^T \times \{1, \ldots, L^T\} & \text{if } J = J' \times [1, L] \end{cases}$$

In this definition, the cross product $\times$ decomposes tuples before building the whole tuple, so that $J'^T \times \{1, \ldots, L^T\} = \{(v_1, \ldots, v_h, v) \mid (v_1, \ldots, v_h) \in J'^T, v \in \{1, \ldots, L^T\}\}$.

Given a conjunction $C = \bigwedge_{(i_1,\ldots,i_h)\in J}$ and an environment T, we define the set of environments $T^C = \{T[i_1 \mapsto v_1, \ldots, i_h \mapsto v_h] \mid (v_1, \ldots, v_h) \in J^T\}$: these environments map the indices $i_1, \ldots, i_h$ of the conjunction to all their possible values in J^T and map all other values like T.

The translation of a fact $F = C\,\mathit{pred}(p_1^G, \ldots, p_l^G)$ is

$$(C\,\mathit{pred}(p_1^G, \ldots, p_l^G))^T = F_1 \wedge \cdots \wedge F_k$$

where $\{F_1, \ldots, F_k\} = \{\mathit{pred}(p_1^{GT'}, \ldots, p_l^{GT'}) \mid T' \in T^C\}$ and $(F_1^G \wedge \cdots \wedge F_n^G)^T = F_1^{GT} \wedge \cdots \wedge F_n^{GT}$.

The translation of a set of equations $\mathcal{E}$ is the set $\mathcal{E}^T$ obtained by translating the equations E, E' as follows:

- $(C\,\iota \doteq \iota')^T = \begin{cases} \mathit{true} & \text{if for all } T' \in T^C,\ \iota^{T'} = \iota'^{T'} \\ \mathit{false} & \text{otherwise.} \end{cases}$
- $(C\,p^G \doteq p'^G)^T = \{p^{GT'} = p'^{GT'} \mid T' \in T^C\}$.
- If $\forall E' \in \mathcal{E},\ E'^T = \mathit{true}$, then $\mathcal{E}^T = \bigcup_{E \in \mathcal{E}} E^T$; otherwise, $\mathcal{E}^T$ is undefined.

The equations over indices can be evaluated to *true* or *false* knowing the environment T. The equations over generalized clause terms are translated into equations over clause terms. A set of equations $\mathcal{E}$ is translated into a set of equations over clause terms if all equations over indices are true. Otherwise, the translation of $\mathcal{E}$ is undefined.

Given a set of constraints *Cts* and an environment T, we say that T satisfies *Cts* when, for all equations $I_1 \uplus \cdots \uplus I_m = J$ in *Cts*, we have that $I_1^T \uplus \cdots \uplus I_m^T = J^T$, that is, $I_1^T, \ldots, I_m^T$ are pairwise disjoint and their union is J^T.

Given a set of equations $\{p_1 = p'_1, \ldots, p_n = p'_n\}$ over standard clause terms, we define as usual its most general unifier $\textsc{mgu}\,(\{p_1 = p'_1, \ldots, p_n = p'_n\})$ as the most general substitution σ such that $\sigma p_i = \sigma p'_i$ for all $i \in \{1, \ldots, n\}$, $\mathit{dom}(\sigma) \cup \mathit{fv}(\mathit{im}(\sigma)) \subseteq \mathit{fv}(p_1, p'_1, \ldots, p_n, p'_n)$, and $\mathit{dom}(\sigma) \cap \mathit{fv}(\mathit{im}(\sigma)) = \emptyset$, where $\mathit{fv}(p)$ designates the (free) variables of p, $\mathit{dom}(\sigma)$ is the domain of σ: $\mathit{dom}(\sigma) = \{x \mid \sigma x \neq x\}$, and $\mathit{im}(\sigma)$ is the image of σ: $\mathit{im}(\sigma) = \{\sigma x \mid \sigma x \neq x\}$. We denote by $\{x_1 \mapsto p_1, \ldots, x_n \mapsto p_n\}$ the substitution that maps x_i to p_i for all $i = 1, \ldots, n$.

Finally, we define the translation of the generalized Horn clause $R^G = \mathit{Cts}, H^G \wedge \mathcal{E} \Rightarrow \mathit{pred}(p_1^G, \ldots, p_l^G)$ as follows. If T does not satisfy *Cts*, $\mathcal{E}^T$ is undefined, or the unification of $\mathcal{E}^T$ fails, then R^{GT} is undefined. Otherwise,

$$R^{GT} = \textsc{mgu}\,(\mathcal{E}^T)H^{GT} \Rightarrow \textsc{mgu}\,(\mathcal{E}^T)\mathit{pred}(p_1^{GT}, \ldots, p_l^{GT}).$$

When $\mathcal{R}^G$ is a set of well-typed generalized Horn clauses (i.e., a set of pairs of a type environment Γ and a clause R^G such that $\Gamma \vdash R^G$), we define $\mathcal{R}^{G\mathcal{T}} = \{R^{GT} \mid \Gamma \vdash R^G \in \mathcal{R}^G,\ T$ is an environment for $\Gamma \vdash R^G$ and R^{GT} is defined$\}$. In terms of abstract interpretation, the sets of generalized Horn clauses ordered by inclusion constitute the abstract domain, the sets of Horn clauses ordered by inclusion the concrete domain, and $\mathcal{R}^{G\mathcal{T}}$ is the concretization of $\mathcal{R}^G$.

Example 1. *Given the clause*

$$R^G = \{I' \uplus I'' = [1, L]\}, \bigwedge_{i \in I'} \mathrm{att}(\mathsf{a}_i^L) \wedge \bigwedge_{i \in I''} \mathrm{att}(\mathit{list}(j \leq L', x_{i,j})) \wedge \{\bigwedge_{i \in I''} i \doteq \phi(i)\} \Rightarrow \mathrm{att}(\mathsf{a}_{\phi(k)}^L)$$

and the type environment $\Gamma = \{I' : [1, L], I'' : [1, L], x\_ : [1, L] \times [1, L'], \phi : [1, L] \to [1, L], k : [1, L]\}$*, if we consider the environment* $T' = \{L \mapsto 4, I' \mapsto \{1, 3\}, I'' \mapsto \{2\}, k \mapsto 4, \phi \mapsto \{1 \mapsto 3, 2 \mapsto 2, 3 \mapsto 4, 4 \mapsto 1\}\}$*, the constraint* $I'^{T'} \uplus I''^{T'} = \{1, \ldots, L^{T'}\}$ *is*

not satisfied ($\{1,3\} \uplus \{2\} \neq \{1,2,3,4\}$*), so* $R^{GT'}$ *is not defined. For* $T'' = \{L \mapsto 4, I' \mapsto \{1,3\}, I'' \mapsto \{2,4\}, k \mapsto 4, \phi \mapsto \{1 \mapsto 3, 2 \mapsto 2, 3 \mapsto 4, 4 \mapsto 1\}\}$*,* $R^{GT''}$ *is also undefined, because when translating the equation, we find that* $4 = \phi(4)$ *is false. For* $T = \{L \mapsto 4, I' \mapsto \{1,3,4\}, I'' \mapsto \{2\}, k \mapsto 4, \phi \mapsto \{1 \mapsto 3, 2 \mapsto 2, 3 \mapsto 4, 4 \mapsto 1\}\}$*, the translated clause is defined:*

$$R^{GT} = \mathrm{att}(\mathsf{a}_1^4) \wedge \mathrm{att}(\mathsf{a}_3^4) \wedge \mathrm{att}(\mathsf{a}_4^4) \wedge \mathrm{att}(\langle x_{2,1}, x_{2,2}, x_{2,3}, x_{2,4} \rangle) \Rightarrow \mathrm{att}(\mathsf{a}_1^4)$$

CHAPTER 7

New Definitions for Generalized Horn Clauses

Contents

In this chapter, we adapt the notions of subsumption and resolution to generalized Horn clauses. In Section 7.1, we define adapted notions of substitutions, then define subsumption and prove its soundness. We then provide an adapted algorithm that computes the substitutions for the subsumption test. In Section 7.2, we give a formal definition of hyperresolution and then prove its soundness. Finally, in Section 7.3, we provide several additional transformations in order to simplify clauses and prove their soundness. All these operations will be used in the next chapter to build a new resolution algorithm for generalized Horn clauses.

7.1 Subsumption

To adapt the subsumption test to generalized Horn clauses, we first define adapted notions of substitutions, then define subsumption and prove its soundness.

Definition 11 (Substitutions)**.** *We denote by ρ a substitution that maps:*

- *bounds to bounds $\rho(L) = L'$;*
- *index variables to index terms: $\rho(i) = \iota$;*
- *set symbols to set symbols: $\rho(I) = I'$;*
- *function symbols ϕ to function symbols: $\rho(\phi) = \phi'$.*

We denote by σ^G a substitution that maps variables to clause terms: $\sigma^G(x_{i_1,\dots,i_h}) = p^G$ where $fi(p^G) \subseteq \{i_1, \dots, i_h\}$.

As usual, substitutions are extended from variables to index terms, clause terms, facts, equations, constraints, and clauses as homomorphisms: for instance, $\rho(\phi(\iota_1, \dots, \iota_k)) = (\rho(\phi))(\rho(\iota_1), \dots, \rho(\iota_k))$. However, since the grammar of generalized Horn clauses is richer than usual, we need to notice two points:

- Substitutions avoid the capture of bound indices: if $i \notin dom(\rho)$ and $i \notin fi(im(\rho))$, $\rho(list(i \leq L, p^G)) = list(i \leq \rho(L), \rho(p^G))$, which can be guaranteed by renaming i if necessary. We have similar formulas for indices bound by conjunctions as well as for substitutions σ^G.
- The substitutions σ^G are extended from variables $x_{i_1,\dots,i_h}$ with indices $i_1, \dots, i_h$ to variables $x_{\iota_1,\dots,\iota_h}$ with terms $\iota_1, \dots, \iota_h$ as indices: if $\sigma^G(x_{i_1,\dots,i_h}) = p^G$, then $\sigma^G(x_{\iota_1,\dots,\iota_h}) = \rho p^G$ where $\rho = \{i_1 \mapsto \iota_1, \dots, i_h \mapsto \iota_h\}$.

Given a substitution σ^G, we say that $\Gamma \vdash \sigma^G$ when, for each mapping $x_{i_1,\dots,i_h} \mapsto p^G \in \sigma^G$, we have $x\_ : [1, L_1] \times \cdots \times [1, L_h] \in \Gamma$ for some $L_1, \dots, L_h$ and $\Gamma, i_1 : [1, L_1], \dots, i_h : [1, L_h] \vdash p^G$. Given an environment T for $\Gamma \vdash \sigma^G$, we define the translation of a substitution σ^G as the substitution σ^{GT} obtained by replacing the mapping $x_{i_1,\dots,i_h} \mapsto p^G$ with the mappings:

$$x_{v_1,\dots,v_h} \mapsto (p^G)^{T[i_1 \mapsto v_1,\dots,i_h \mapsto v_h]}$$

for all $v_1 \in \{1, \dots, L_1^T\}, \dots, v_h \in \{1, \dots, L_h^T\}$, where $\Gamma \vdash x\_ : [1, L_1] \times \cdots \times [1, L_h]$.

We need a notion of subsumption between type environments, which is intuitively $\rho\Gamma_1 \subseteq \Gamma_2$. However, we cannot use exactly $\rho\Gamma_1 \subseteq \Gamma_2$ because ρ maps indices to index terms and not to indices, so we define $\Gamma_1 \leq_\rho \Gamma_2$ by:

- for each $i : [1, L] \in \Gamma_1$, we have $\Gamma_2 \vdash \rho i : [1, \rho L]$;
- for each $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma_1$, we have $\rho\phi : [1, \rho L_1] \times \cdots \times [1, \rho L_h] \to [1, \rho L] \in \Gamma_2$;
- for each $I : [1, L_1] \times \cdots \times [1, L_h] \in \Gamma_1$, we have $\rho I : [1, \rho L_1] \times \cdots \times [1, \rho L_h] \in \Gamma_2$;
- for each $x\_ : [1, L_1] \times \cdots \times [1, L_h] \in \Gamma_1$, we have $x\_ : [1, \rho L_1] \times \cdots \times [1, \rho L_h] \in \Gamma_2$.

Definition 12 (Subsumption)**.** *Given two well-typed generalized Horn clauses,* $\Gamma_1 \vdash R_1^G$ *and* $\Gamma_2 \vdash R_2^G$*, with* $R_1^G = Cts_1, H_1^G \wedge \mathcal{E}_1 \Rightarrow C_1^G$ *and* $R_2^G = Cts_2, H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$*, we say that* $\Gamma_1 \vdash R_1^G$ *subsumes* $\Gamma_2 \vdash R_2^G$*, and we write* $\Gamma_1 \vdash R_1^G \sqsupseteq \Gamma_2 \vdash R_2^G$*, if there exist substitutions* ρ *and* σ^G *such that* $\sigma^G \rho C_1^G = C_2^G$*,* $\sigma^G \rho H_1^G \subseteq H_2^G$ *(multiset inclusion),* $\sigma^G \rho \mathcal{E}_1 \subseteq \mathcal{E}_2$ *(modulo commutativity of* $\doteq$*),* $\rho Cts_1 \subseteq Cts_2$ *(modulo associativity and commutativity of* $\uplus$*), and* $\Gamma_1 \leq_\rho \Gamma_2$*.*

This definition is similar to subsumption for standard clauses but uses the richer substitutions introduced above.

Example 2. *The clause*

$$R_1^G = \{I \uplus I_1 = [1, L]\}, \bigwedge_{i \in I} \mathrm{att}(sign(x_i, \mathsf{sk}^L_{\phi(l)})) \wedge \{\bigwedge_{m \in I_1} y_m \doteq \mathsf{c}^L_m\} \Rightarrow \mathrm{att}(y_l)$$

typed by the type environment $\Gamma_1 = \{I : [1, L], I_1 : [1, L], x\_ : [1, L], \phi : [1, L] \to [1, L'], y_\cdot[1, L], l : [1, L]\}$*, subsumes the clause*

$$R_2^G = \{I' \uplus I'_1 = [1, L_1]\}, \bigwedge_{j \in I'} \mathrm{att}(sign((\mathsf{r}^{L_1}_{\psi(j)}, \mathsf{s}^{L_1}_{\psi(j)}), \mathsf{sk}^{L'_1}_{\varphi(\psi(k))})) \wedge \{\bigwedge_{n \in I'_1} z_n \doteq \mathsf{c}^{L_1}_n\} \Rightarrow \mathrm{att}(z_{\psi(k)}).$$

typed by the environment $\Gamma_2 = \{I' : [1, L_1], I'_1 : [1, L_1], \psi : [1, L_1] \to [1, L_1], \varphi : [1, L_1] \to [1, L'_1], k : [1, L_1], z\_ : [1, L_1]\}$

With $\rho = \{L \mapsto L_1, L' \mapsto L'_1, I \mapsto I', I_1 \mapsto I'_1, \phi \mapsto \varphi, l \mapsto \psi(k)\}$ *and renaming the bound indices i into j and m into n, we obtain:*

$$\rho R_1^G = \{I' \uplus I'_1 = [1, L_1]\}, \bigwedge_{j \in I'} \mathrm{att}(sign(x_j, \mathsf{sk}^{L'_1}_{\varphi(\psi(k))})) \wedge \{\bigwedge_{n \in I'_1} y_n \doteq \mathsf{c}^{L_1}_n\} \Rightarrow \mathrm{att}(y_{\psi(k)})$$

and conclude that R_1^G *subsumes* R_2^G *using the substitution*

$$\sigma^G = \{x_i \mapsto (\mathsf{r}^{L_1}_{\psi(i)}, \mathsf{s}^{L_1}_{\psi(i)}), y_i \mapsto z_i\}.$$

The following theorem shows the soundness of subsumption for generalized Horn clauses.

Theorem 13 (Subsumption)**.** *Let* $\Gamma_1 \vdash R_1^G$ *and* $\Gamma_2 \vdash R_2^G$ *be two well-typed clauses. If* $\Gamma_1 \vdash R_1^G \sqsupseteq \Gamma_2 \vdash R_2^G$ *then, for all environments* T_2 *for* $\Gamma_2 \vdash R_2^G$ *such that* $R_2^{GT_2}$ *is defined, there exists an environment* T_1 *for* $\Gamma_1 \vdash R_1^G$ *such that* $R_1^{GT_1}$ *is defined and* $R_1^{GT_1} \sqsupseteq R_2^{GT_2}$.

As for standard clauses, our resolution algorithm for generalized Horn clauses will eliminate subsumed clauses. This theorem shows that, if R_2^G is eliminated in the algorithm for generalized Horn clauses because it is subsumed by R_1^G, then all corresponding clauses $R_2^{GT_2}$ would be eliminated in the algorithm for standard clauses: $R_2^{GT_2}$ is subsumed by some $R_1^{GT_1}$.

The subsumption test is approximate, because clauses that have the same meaning may sometimes be written in different forms. For instance, one can compose two functions ϕ and ψ, always writing $\phi(\psi(i, j))$ instead of a single function $\psi(i, j)$. These two formulas yield the same translation into standard Horn clauses, but are considered different by the subsumption test. However, this approximation does not contradict the soundness of subsumption.

7.1.1 Proof of Theorem 13

We generalize the notion of environment as follows. An environment T is a partial function that associates:

- to bounds L, integers L^T;
- to set symbols I, non-empty sets I^T of tuples of integers;
- to indices i, integer indices i^T;
- to index functions ϕ, partial functions ϕ^T from tuples of integers to integers.

Obviously, an environment for $\Gamma \vdash R^G$ is also an environment as defined above. The notations ι^T, p^{GT}, F^{GT}, ... can be used with any environment T, but obviously they are not defined when T does not define the elements required to compute them.

To prove the theorem, we need to state several lemmas. The proofs of these lemmas are detailed in Appendix A.2.1.

Lemma 14. *Given a substitution* ρ *and an environment* T*, let* T' *be defined by* $L^{T'} = (\rho L)^T$, $i^{T'} = (\rho i)^T$, $\phi^{T'} = (\rho\phi)^T$, *and* $I^{T'} = (\rho I)^T$. *Then*

- $(\rho\iota)^T = \iota^{T'}$ *for all index terms* ι*;*
- $(\rho J)^T = J^{T'}$ *for all sets* J*;*

- $(\rho p^G)^T = p^{GT'}$ *for all clause terms* p^G*;*
- $(\rho F^G)^T = F^{GT'}$ *for all facts* F^G*;*
- $(\rho E)^T = E^{T'}$ *for all equations* E*;*
- $(\rho E')^T = E'^{T'}$ *for all equations* E'*;*
- $(\rho \mathcal{E})^T = \mathcal{E}^{T'}$ *for all sets of equations* $\mathcal{E}$*;*
- T *satisfies* $\rho \mathcal{I}$ *if and only if* T' *satisfies* $\mathcal{I}$*, for all constraints* $\mathcal{I}$*.*

In all equalities above, the left-hand side is defined if and only if the right-hand side is defined.

Lemma 15. *Given a substitution* σ^G *and an environment* T*, we have*

- $(\sigma^G p^G)^T = \sigma^{GT} p^{GT}$ *for all clause terms* p^G*;*
- $(\sigma^G F^G)^T = \sigma^{GT} F^{GT}$ *for all facts* F^G*;*
- $(\sigma^G E)^T = \sigma^{GT} E^T$ *for all equations* E*;*
- $(\sigma^G \mathcal{E})^T = \sigma^{GT} \mathcal{E}^T$ *for all sets of equations* $\mathcal{E}$*.*

In all equalities above, the left-hand side is defined if and only if the right-hand side is defined.

Lemma 16. *Let* $\mathcal{E}_1$ *and* $\mathcal{E}_2$ *be two sets of equations and* σ^G *a substitution such that* $\sigma^G \mathcal{E}_1 \subseteq \mathcal{E}_2$ *(modulo commutativity of* $\doteq$*). Then, for all environments* T *such that* $\textsc{mgu}\,(\mathcal{E}_2^T)$ *is defined, we have that* $\textsc{mgu}\,(\mathcal{E}_1^T)$ *is defined and*

$$\textsc{mgu}\,(\mathcal{E}_2^T)\sigma^{GT}\,\textsc{mgu}\,(\mathcal{E}_1^T) = \textsc{mgu}\,(\mathcal{E}_2^T)\sigma^{GT}.$$

Proof of Theorem 13. Let $R_1^G = Cts_1,\ H_1^G \wedge \mathcal{E}_1 \Rightarrow C_1^G$ and $R_2^G = Cts_2,\ H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$ be two clauses, well-typed in the type environments Γ_1 and Γ_2 respectively, such that $\Gamma_1 \vdash R_1^G \sqsupseteq \Gamma_2 \vdash R_2^G$. There exist ρ and σ^G such that:

- $\sigma^G \rho C_1^G = C_2^G$,
- $\sigma^G \rho H_1^G \subseteq H_2^G$ (multiset inclusion),
- $\sigma^G \rho \mathcal{E}_1 \subseteq \mathcal{E}_2$ (modulo commutativity of $\doteq$),
- $\rho Cts_1 \subseteq Cts_2$ (modulo associativity and commutativity of $\uplus$),

- $\Gamma_1 \leq_\rho \Gamma_2$.

Let T_2 be an environment for $\Gamma_2 \vdash R_2^G$ such that $R_2^{GT_2}$ is defined. We have to prove that there exists an environment T_1 for $\Gamma_1 \vdash R_1^G$ such that $R_1^{GT_1}$ is defined and $R_1^{GT_1} \sqsupseteq R_2^{GT_2}$. We define T_1 by $L^{T_1} = (\rho L)^{T_2}$, $i^{T_1} = (\rho i)^{T_2}$, $\phi^{T_1} = (\rho\phi)^{T_2}$, and $I^{T_1} = (\rho I)^{T_2}$.

Let us prove that T_1 is an environment for $\Gamma_1 \vdash R_1^G$. For all bounds L that occur in R_1^G, since $\sigma^G \rho C_1^G = C_2^G$, $\sigma^G \rho H_1^G \subseteq H_2^G$, and $\sigma^G \rho \mathcal{E}_1 \subseteq \mathcal{E}_2$, we have that ρL occurs in R_2^G, so $(\rho L)^{T_2}$ is defined since T_2 is an environment for $\Gamma_2 \vdash R_2^G$. Hence, L^{T_1} is defined. The other properties needed to show that T_1 is an environment for $\Gamma_1 \vdash R_1^G$ come from $\Gamma_1 \leq_\rho \Gamma_2$ and T_2 is an environment for $\Gamma_2 \vdash R_2^G$. For each bound L that occurs in Γ_1, ρL occurs in Γ_2, so $(\rho L)^{T_2}$ is defined, hence L^{T_1} is defined. For each $i : [1, L] \in \Gamma_1$, we have $\Gamma_2 \vdash \rho i : [1, \rho L]$, so $(\rho i)^{T_2} \in \{1, \ldots, (\rho L)^{T_2}\}$, hence $i^{T_1} \in \{1, \ldots, L^{T_1}\}$. We show similar properties for set symbols I and for index functions ϕ.

Since T_2 satisfies Cts_2, it satisfies ρCts_1, so by Lemma 14, T_1 satisfies Cts_1. By Lemma 16, since $\text{MGU}(\mathcal{E}_2^{T_2})$ is defined, $\text{MGU}((\rho\mathcal{E}_1)^{T_2})$ is defined, that is, by Lemma 14, $\text{MGU}(\mathcal{E}_1^{T_1})$ is defined. Hence $R_1^{GT_1}$ is defined.

To show that $R_1^{GT_1} \sqsupseteq R_2^{GT_2}$, we prove:

$$\sigma \text{MGU}(\mathcal{E}_1^{T_1}) H_1^{GT_1} \subseteq \text{MGU}(\mathcal{E}_2^{T_2}) H_2^{GT_2} \text{ (multiset inclusion)}$$
$$\sigma \text{MGU}(\mathcal{E}_1^{T_1}) C_1^{GT_1} = \text{MGU}(\mathcal{E}_2^{T_2}) C_2^{GT_2}$$

where $\sigma = \text{MGU}(\mathcal{E}_2^{T_2})\sigma^{GT_2}$. We have

$$\begin{aligned}
\sigma \text{MGU}(\mathcal{E}_1^{T_1}) H_1^{GT_1} &= \text{MGU}(\mathcal{E}_2^{T_2}) \sigma^{GT_2} \text{MGU}((\rho\mathcal{E}_1)^{T_2}) (\rho H_1^G)^{T_2} && \text{by Lemma 14} \\
&= \text{MGU}(\mathcal{E}_2^{T_2}) \sigma^{GT_2} (\rho H_1^G)^{T_2} && \text{by Lemma 16} \\
&= \text{MGU}(\mathcal{E}_2^{T_2}) (\sigma^G \rho H_1^G)^{T_2} && \text{by Lemma 15} \\
&\subseteq \text{MGU}(\mathcal{E}_2^{T_2}) H_2^{GT_2} && \text{since } \sigma^G \rho H_1^G \subseteq H_2^G
\end{aligned}$$

Similarly,

$$\begin{aligned}
\sigma \text{MGU}(\mathcal{E}_1^{T_1}) C_1^{GT_1} &= \text{MGU}(\mathcal{E}_2^{T_2}) \sigma^{GT_2} \text{MGU}((\rho\mathcal{E}_1)^{T_2}) (\rho C_1^G)^{T_2} && \text{by Lemma 14} \\
&= \text{MGU}(\mathcal{E}_2^{T_2}) \sigma^{GT_2} (\rho C_1^G)^{T_2} && \text{by Lemma 16} \\
&= \text{MGU}(\mathcal{E}_2^{T_2}) (\sigma^G \rho C_1^G)^{T_2} && \text{by Lemma 15} \\
&= \text{MGU}(\mathcal{E}_2^{T_2}) C_2^{GT_2} && \text{since } \sigma^G \rho C_1^G = C_2^G
\end{aligned}$$

□

7.1.2 Subsumption Algorithm

Because of the presence of indices, the algorithm for computing the substitutions ρ and σ^G required in Definition 12 is more complex than for standard clauses. It uses the functions defined in Figures 7.1 and 7.2. These functions use partly defined values of ρ and σ^G. At the beginning of the subsumption algorithm, ρ and σ^G are not defined at all, and they get progressively defined as the algorithm runs. In these functions, an assignment such as $\rho(i) \leftarrow \iota'$ means that, if $\rho(i)$ is not defined yet, we set $\rho(i)$ to ι'; if $\rho(i)$ is defined and equal to ι', we do nothing; if $\rho(i)$ is defined and different from ι', we fail.

The functions of Figure 7.1 work as follows. Compρind$(\iota, \iota', \Gamma, \Gamma')$ updates ρ such that $\rho\iota = \iota'$ and ρ satisfies the two following conditions:

- for each $i \in \mathit{fi}(\iota)$, if $\Gamma \vdash i : [1, L]$ then $\Gamma' \vdash \rho i : [1, \rho L]$;
- for each ϕ that appears in ι, if $\Gamma \vdash \phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L]$ then $\Gamma \vdash \rho\phi : [1, \rho L_1] \times \cdots \times [1, \rho L_h] \to [1, \rho L]$.

Compρset(J, J', Γ, Γ') updates ρ such that $\rho J = J'$ and for each set I that appears in J, if $\Gamma \vdash I : [1, L_1] \times \cdots \times [1, L_h]$ then $\Gamma' \vdash \rho I : [1, \rho L_1] \times \cdots \times [1, \rho L_h]$. Comp$\rho$subsind$(\iota, \iota', i_1, \ldots, i_h, \iota''_1, \ldots, \iota''_h, \Gamma, \Gamma')$ updates ρ such that $\iota\{i_1 \mapsto \rho\iota''_1, \ldots, i_h \mapsto \rho\iota''_h\} = \iota'$. Comp$\rho$subs$(p^G, p'^G, i_1, \ldots, i_h, \iota''_1, \ldots, \iota''_h, \Gamma, \Gamma')$ updates ρ such that $p^G\{i_1 \mapsto \rho\iota''_1, \ldots, i_h \mapsto \rho\iota''_h\} = p'^G$. If it is impossible to extend ρ to satisfy the desired equality, these functions fail.

In Figure 7.2, Subs$(F_1^G, F_2^G, \Gamma, \Gamma')$ updates ρ and σ^G such that $\sigma^G\rho F_1^G = F_2^G$. If $F_1^G = \bigwedge_{(i_1,\ldots,i_h)\in J} \mathit{pred}(p_1^G, \ldots, p_l^G)$ and $F_2^G = \bigwedge_{(j_1,\ldots,j_h)\in J'} \mathit{pred}(p_1'^G, \ldots, p_l'^G)$, then the algorithm calls the function Compρset on J and J' to update ρ such that $\rho J = J'$. Then, after renaming the two tuples of indices $i_1, \ldots, i_h$ and $j_1, \ldots, j_h$ into fresh symbols $\tilde{j}_1, \ldots, \tilde{j}_h$, it calls TermSubs on $p_k^G\{i_1 \mapsto \tilde{j}_1, \ldots, i_h \mapsto \tilde{j}_h\}$, $p_K'^G\{j_1 \mapsto \tilde{j}_1, \ldots, j_h \mapsto \tilde{j}_h\}$) for each $k = 1, \ldots, l$. TermSubs$(p^G, p'^G, \Gamma, \Gamma')$ updates ρ and σ^G such that $\sigma^G\rho p^G = p'^G$. When p^G is a variable $x_{\iota_1,\ldots,\iota_h}$, two cases may happen. If $\sigma^G x_{i_1,\ldots,i_h}$ is already defined, say $\sigma^G x_{i_1,\ldots,i_h} = p_1^G$, we update ρ such that $\sigma^G\rho x_{\iota_1,\ldots,\iota_h} = p'^G$, that is $p_1^G\{i_1 \mapsto \rho\iota_1, \ldots, i_h \mapsto \rho\iota_h\} = p'^G$, by calling Comp$\rho$subs. If $\sigma^G x_{i_1,\ldots,i_h}$ is not defined yet, we define it and update ρ if needed by calling ComputeSubs. ComputeSubs$(p^G, i_1, \ldots, i_h, \iota_1, \ldots, \iota_h, \Gamma, \Gamma')$ returns p'^G and updates ρ such that $\mathit{fi}(p'^G) \subseteq \{i_1, \ldots, i_h\}$ and $p'^G\{i_1 \mapsto \rho\iota_1, \ldots, i_h \mapsto \rho\iota_h\} = p^G$. ComputeSubs first replaces all index terms $\rho(\iota_m)$ with i_m in p^G, when $\rho(\iota_m)$ is defined. (Note that $\{\rho(\iota_m) \mapsto i_m \mid m \in \{1, \ldots, h\}, \rho(\iota_m) \text{ is defined}\}$ is not a substitution since it replaces terms $\rho(\iota_m)$ and not variables.) If the obtained result p'^G is such that $\mathit{fi}(p'^G) \subseteq \{i_1, \ldots, i_h\}$, then the required constraint is satisfied and we return p'^G. Otherwise, we try to extend ρ in such a way that the constraint is satisfied. If $\rho(\iota_m)$ is already defined for all $m \in \{1, \ldots, h\}$, this is not possible. (Extending ρ will not change the value of $p'^G\{i_1 \mapsto \rho\iota_1, \ldots, i_h \mapsto \rho\iota_h\}$.) So we fail. Otherwise, we choose an index term $\tilde{\iota}$ in p'^G such that $\mathit{fi}(\tilde{\iota}) \not\subseteq \{i_1, \ldots, i_h\}$ and

$\textsc{Comp}\rho\textsc{ind}(\iota, \iota', \Gamma, \Gamma')$:
let $\Gamma \vdash \iota : [1, L]$ and $\Gamma' \vdash \iota' : [1, L']$
$\rho(L) \leftarrow L'$
case ι, ι' **of**
i, ι': $\rho(i) \leftarrow \iota'$
$\phi(\iota_1, \ldots, \iota_h), \psi(\iota'_1, \ldots, \iota'_h)$:
for each $k = 1, \ldots, h$, $\textsc{Comp}\rho\textsc{ind}(\iota_k, \iota'_k, \Gamma, \Gamma')$
$\rho(\phi) \leftarrow \psi$
other cases: **fail**

$\textsc{Comp}\rho\textsc{set}(J, J', \Gamma, \Gamma')$:
let $\Gamma \vdash J : [1, L_1] \times \cdots \times [1, L_h]$ and $\Gamma' \vdash J' : [1, L'_1] \times \cdots \times [1, L'_h]$
case J, J' **of**
I, I':
for each $k = 1, \ldots, h, \rho(L_k) \leftarrow L'_k$
$\rho(I) \leftarrow I'$
$\{()\}, \{()\}$: **return**
$J_1 \times [1, L_h], J'_1 \times [1, L'_h]$:
$\rho(L_h) \leftarrow L'_h$
$\textsc{Comp}\rho\textsc{set}(J_1, J'_1, \Gamma, \Gamma')$
other cases: **fail**

$\textsc{Comp}\rho\textsc{subsind}(\iota, \iota', i_1, \ldots, i_h, \iota''_1, \ldots, \iota''_h, \Gamma, \Gamma')$:
case ι, ι' **of**
i_k, ι': $\textsc{Comp}\rho\textsc{ind}(\iota''_k, \iota', \Gamma, \Gamma')$
i, i: **return**
$\phi(\iota_1, \ldots, \iota_k), \phi(\iota'_1, \ldots, \iota'_k)$:
for each $j = 1, \ldots, k$, $\textsc{Comp}\rho\textsc{subsind}(\iota_j, \iota'_j, i_1, \ldots, i_h, \iota''_1, \ldots, \iota''_h, \Gamma, \Gamma')$
other cases: **fail**

$\textsc{Comp}\rho\textsc{subs}(p^G, p'^G, i_1, \ldots, i_h, \iota''_1, \ldots, \iota''_h, \Gamma, \Gamma')$:
case p^G, p'^G **of**
$x_{\iota_1,\ldots,\iota_k}, x_{\iota'_1,\ldots,\iota'_k}$:
for each $j = 1, \ldots, k$, $\textsc{Comp}\rho\textsc{subsind}(\iota_j, \iota'_j, i_1, \ldots, i_h, \iota''_1, \ldots, \iota''_h, \Gamma, \Gamma')$
$f(p^G_1, \ldots, p^G_k), f(p'^G_1, \ldots, p'^G_k)$:
for each $j = 1, \ldots, k$, $\textsc{Comp}\rho\textsc{subs}(p^G_j, p'^G_j, i_1, \ldots, i_h, \iota''_1, \ldots, \iota''_h, \Gamma, \Gamma')$
$a^{L_1,\ldots,L_n}_{\iota_1,\ldots,\iota_n}[p^G_1, \ldots, p^G_k], a^{L_1,\ldots,L_n}_{\iota'_1,\ldots,\iota'_n}[p'^G_1, \ldots, p'^G_k]$:
for each $j = 1, \ldots, n$, $\textsc{Comp}\rho\textsc{subsind}(\iota_j, \iota'_j, i_1, \ldots, i_h, \iota''_1, \ldots, \iota''_h, \Gamma, \Gamma')$
for each $j = 1, \ldots, k$, $\textsc{Comp}\rho\textsc{subs}(p^G_j, p'^G_j, i_1, \ldots, i_h, \iota''_1, \ldots, \iota''_h, \Gamma, \Gamma')$
$list(i \le L, p^G_1), list(j \le L, p'^G_1)$:
let $\tilde{\jmath}$ be a fresh symbol; $\rho(\tilde{\jmath}) \leftarrow \tilde{\jmath}$
$\textsc{Comp}\rho\textsc{subs}(p^G_1\{i \mapsto \tilde{\jmath}\}, p'^G_1\{j \mapsto \tilde{\jmath}\},$
$i_1, \ldots, i_h, \iota''_1, \ldots, \iota''_h, \Gamma, (\Gamma', \tilde{\jmath} : [1, L]))$
other cases: **fail**

Figure 7.1: Functions that compute ρ

ComputeSubs($p^G, i_1, \ldots, i_h, \iota_1, \ldots, \iota_h, \Gamma, \Gamma'$)
  while true **do**
    $p'^G \leftarrow p^G\{\rho(\iota_m) \mapsto i_m \mid m \in \{1, \ldots, h\}, \rho(\iota_m)$ is defined$\}$
    if $fi(p'^G) \subseteq \{i_1, \ldots, i_h\}$ **then**
      return p'^G
    if for all $m \in \{1, \ldots, h\}$, $\rho(\iota_m)$ is defined **then**
      fail
    choose
      an index term $\tilde{\iota}$ in p'^G such that $fi(\tilde{\iota}) \not\subseteq \{i_1, \ldots, i_h\}$
      and $m \in \{1, \ldots, h\}$ such that $\rho(\iota_m)$ is not defined
    Compρind($\iota_m, \tilde{\iota}\{i_{m'} \mapsto \rho(\iota_{m'}) \mid$
      $m' \in \{1, \ldots, h\}, \rho(\iota_{m'})$ is defined$\}, \Gamma, \Gamma'$)

TermSubs(p^G, p'^G, Γ, Γ')
  case p^G, p'^G **of**
  $x_{\iota_1,\ldots,\iota_h}, p'^G$:
    if $\sigma^G x_{i_1,\ldots,i_h}$ is defined **then**
      Compρsubs($\sigma^G x_{i_1,\ldots,i_h}, p'^G, i_1, \ldots, i_h, \iota_1, \ldots, \iota_h, \Gamma, \Gamma'$)
    else
      $\sigma^G x_{i_1,\ldots,i_h} \leftarrow$ ComputeSubs($p'^G, i_1, \ldots, i_h, \iota_1, \ldots, \iota_h, \Gamma, \Gamma'$)
  $f(p_1^G, \ldots, p_k^G), f(p_1'^G, \ldots, p_k'^G)$:
    for each $j = 1, \ldots, k$, TermSubs(p_j^G, $p_j'^G, \Gamma, \Gamma'$)
  $a^{L_1,\ldots,L_n}_{\iota_1,\ldots,\iota_n}[p_1^G, \ldots, p_k^G], a^{L'_1,\ldots,L'_n}_{\iota'_1,\ldots,\iota'_n}[p_1'^G, \ldots, p_k'^G]$:
    for each $j = 1, \ldots, n, \rho(L_j) \leftarrow L'_j$
    for each $j = 1, \ldots, n$, Compρind(ι_j, ι'_j)
    for each $j = 1, \ldots, k$, TermSubs(p_j^G, $p_j'^G, \Gamma, \Gamma'$)
  $list(i \le L, p_1^G), list(j \le L', p_1'^G)$:
    $\rho(L) \leftarrow L'$
    let $\tilde{\jmath}$ be a fresh symbol; $\rho(\tilde{\jmath}) \leftarrow \tilde{\jmath}$
    TermSubs($p_1^G\{i \mapsto \tilde{\jmath}\}$, $p_1'^G\{j \mapsto \tilde{\jmath}\}, (\Gamma, \tilde{\jmath} : [1, L]), (\Gamma', \tilde{\jmath} : [1, L'])$)
  other cases: **fail**

Subs($F_1^G, F_2^G, \Gamma, \Gamma'$) {
  let $F_1^G = \bigwedge_{(i_1,\ldots,i_h)\in J} pred(p_1^G, \ldots, p_l^G)$.
  if $F_2^G = \bigwedge_{(j_1,\ldots,j_h)\in J'} pred(p_1'^G, \ldots, p_l'^G)$ **then**
    let $\Gamma \vdash J : [1, L_1] \times \cdots \times [1, L_h]$ and $\Gamma' \vdash J' : [1, L'_1] \times \cdots \times [1, L'_h]$
    Compρset(J, J', Γ, Γ')
    for each $k = 1, \ldots, h$,
      let $\tilde{\jmath}_k$ be a fresh symbol; $\rho(\tilde{\jmath}_k) \leftarrow \tilde{\jmath}_k$
    for each $k = 1, \ldots, l$,
    TermSubs($p_k^G\{i_1 \mapsto \tilde{\jmath}_1, \ldots, i_h \mapsto \tilde{\jmath}_h\}$,
      $p_k'^G\{j_1 \mapsto \tilde{\jmath}_1, \ldots, j_h \mapsto \tilde{\jmath}_h\}$)
      $(\Gamma, \tilde{\jmath}_1 : [1, L_1], \ldots, \tilde{\jmath}_h : [1, L_h]), (\Gamma', \tilde{\jmath}_1 : [1, L'_1], \ldots, \tilde{\jmath}_h : [1, L'_h])$)
  else fail

Figure 7.2: Subs algorithm

$m \in \{1, \ldots, h\}$ such that $\rho(\iota_m)$ is not defined, and try to extend ρ such that $\rho(\iota_m) = \tilde{\iota}\{i_{m'} \mapsto \rho(\iota_{m'}) \mid m' \in \{1, \ldots, h\}, \rho(\iota_{m'})$ is defined$\}$ by calling CompρIND. If this succeeds, we retry: the initial replacement of $\rho(\iota_m)$ with i_m will then additionally replace $\tilde{\iota}$ with i_m. In case the subsumption algorithm subsequently fails, we backtrack on the choice on $\tilde{\iota}$ and m.

It is easy to define functions similar to Subs for equations. A function updates ρ and σ^G such that $\sigma^G\rho(C\, p_1^G \doteq p_2^G) = (C'\, p_1'^G \doteq p_2'^G)$ by calling TermSubs to relate p_1^G and $p_1'^G$, as well as p_2^G and $p_2'^G$. In case the subsumption algorithm subsequently fails and p_2^G is a variable, we commute the first equation and try to update ρ and σ^G such that $\sigma^G\rho(C\, p_2^G \doteq p_1^G) = (C'\, p_1'^G \doteq p_2'^G)$. (Because of the simplifications made in the unification algorithm, the clause terms p_1^G and $p_1'^G$ are always variables.) Another function updates ρ such that $\rho(C\, \iota_1 \doteq \iota_2) = (C'\, \iota_1' \doteq \iota_2')$ by calling CompρIND to relate ι_1 and ι_1', as well as ι_2 and ι_2'. In case the subsumption algorithm subsequently fails, we also commute the first equation.

For testing subsumption between clauses R_1^G and R_2^G, we can then proceed as in the subsumption algorithm for standard clauses. We call Subs$(C_1^G, C_2^G, \Gamma, \Gamma')$ where C_1^G is the conclusion of R_1^G and C_2^G the conclusion of R_2^G. For each hypothesis F_1^G in R_1^G, we choose a hypothesis F_2^G in R_2^G not used yet (since we prove multiset inclusion), and call Subs$(F_1^G, F_2^G, \Gamma, \Gamma')$. For each equation E_1 in R_1^G, we choose an equation E_2 in R_2^G and relate it to E_1. For each constraint $I_1 \uplus \cdots \uplus I_h = J$ in R_1^G, we choose a constraint $I_1' \uplus \cdots \uplus I_h' = J'$ in R_2^G and relate it to the former constraint by calling CompρSET(J, J'); for the $\rho(I_k)$ that are defined, we check that each $\rho(I_k)$ is equal to a distinct $I_{k'}'$; for the $\rho(I_k)$ that are not defined, we choose an $I_{k'}'$ not mapped yet, and define $\rho(I_k) \leftarrow I_{k'}'$. We backtrack on all these choices in case of subsequent failure of the subsumption algorithm.

7.2 Resolution and Hyperresolution

As introduced in Section 6.1.2, resolution becomes hyperresolution for generalized clauses. We first give the formal definition of hyperresolution, then explain it.

Definition 13 (Hyperresolution)**.** *Let $R_1^G = Cts_1, H_1^G \wedge \mathcal{E}_1 \Rightarrow C_1^G$ and $R_2^G = Cts_2,\ F_0^G \wedge H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$ be two clauses. We rename R_1^G and R_2^G so that they do not use common names for bounds M, variables x, indices i, functions on indices ϕ, and set symbols I.*

First case: $F_0^G = \bigwedge_{(i_1,\ldots,i_h)\in J} pred_2(p_{1,2}^G, \ldots p_{l,2}^G)$ *($h \neq 0$). Let $\Gamma_1 \vdash R_1^G$ and J such that $\Gamma_2 \vdash J : [1, L_1] \times \cdots \times [1, L_h]$. The immersion of R_1^G into $(i_1, \ldots, i_h) \in J$ is the clause $imm(R_1^G, (i_1, \ldots, i_h) \in J)$ obtained by replacing:*

1. *all free indices i of R_1^G with $\phi'(i_1, \ldots, i_h)$, where a new function symbol ϕ' is associated to each free index i of R_1^G;*

2. *all index terms* $\phi(\iota_1, \ldots, \iota_k)$ *with* $\phi'(i_1, \ldots, i_h, \iota_1, \ldots, \iota_k)$*, where a new function symbol* ϕ' *is associated to each function symbol* ϕ *in* R_1^G*;*

3. *all variables* $x_{\iota_1,\ldots,\iota_k}$ *in* R_1^G *with* $x_{i_1,\ldots,i_h,\iota_1,\ldots,\iota_k}$*;*

4. *all conjunctions* $\bigwedge_{(j_1,\ldots,j_k)\in[1,L_1']\times\cdots\times[1,L_k']}$ *in* H_1^G *and* $\mathcal{E}_1$ *with* $\bigwedge_{(i_1,\ldots,i_h,j_1,\ldots,j_k)\in J\times[1,L_1']\times\cdots\times[1,L_k']}$*;*

5. *all conjunctions* $\bigwedge_{(j_1,\ldots,j_k)\in I\times[1,L_1']\times\cdots\times[1,L_l']}$ *in* H_1^G *and* $\mathcal{E}_1$ *with* $\bigwedge_{(i_1,\ldots,i_h,j_1,\ldots j_k)\in I'\times[1,L_1']\times\cdots\times[1,L_l']}$*, where a new set symbol* I' *is associated to each set symbol* I *in* R_1^G*;*

6. *all constraints* $I_1 \uplus \cdots \uplus I_n = [1, L_1'] \times \cdots \times [1, L_k']$ *in* Cts_1 *with* $I_1' \uplus \cdots \uplus I_n' = J \times [1, L_1'] \times \cdots \times [1, L_k']$ *and all constraints* $I_1 \uplus \cdots \uplus I_n = I \times [1, L_1'] \times \cdots \times [1, L_k']$ *in* Cts_1 *with* $I_1' \uplus \cdots \uplus I_n' = I' \times [1, L_1'] \times \cdots \times [1, L_k']$.

Let $imm(R_1^G, (i_1, \ldots, i_h) \in J) = Cts_J, H_J^G \wedge \mathcal{E}_J \Rightarrow pred_1(p_{1,1}^G, \ldots p_{l',1}^G)$*. If* $pred_1 = pred_2$ *(hence* $l = l'$*), we define:*

$$R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G = Cts_J \cup Cts_2, H_J^G \wedge H_2^G \wedge$$
$$(\{\bigwedge_{(i_1\ldots,i_h)\in J} p_{1,1}^G \doteq p_{1,2}^G, \ldots, \bigwedge_{(i_1\ldots,i_h)\in J} p_{l,1}^G \doteq p_{l,2}^G\} \cup \mathcal{E}_J \cup \mathcal{E}_2) \Rightarrow C_2^G$$

Let I' *and* I'' *be fresh set symbols. Let* $imm(R_1^G, (i_1, \ldots, i_h) \in I') = Cts_{I'}, H_{I'}^G \wedge \mathcal{E}_{I'} \Rightarrow pred_1(p_{1,1}^G, \ldots p_{l',1}^G)$*. If* $pred_1 = pred_2$ *(hence* $l = l'$*), we define:*

$$R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G = Cts_{I'} \cup Cts_2 \cup \{I' \uplus I'' = J\},$$
$$(H_{I'}^G \wedge \bigwedge_{(i_1,\ldots,i_h)\in I''} pred_2(p_{1,2}^G, \ldots, p_{l,2}^G) \wedge H_2^G) \wedge$$
$$(\{\bigwedge_{(i_1,\ldots,i_h)\in I'} p_{1,1}^G \doteq p_{1,2}^G, \ldots, \bigwedge_{(i_1,\ldots,i_h)\in I'} p_{l,1}^G \doteq p_{l,2}^G\} \cup \mathcal{E}_{I'} \cup \mathcal{E}_2) \Rightarrow C_2^G.$$

Second case: $F_0^G = pred_2(p_{1,2}^G, \ldots, p_{l,2}^G)$ *and* $C_1^G = pred_1(p_{1,1}^G, \ldots p_{l',1}^G)$ *(ordinary resolution). If* $pred_1 = pred_2$ *(hence* $l = l'$*), we define* $R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G = Cts_1 \cup Cts_2$, $H_1^G \wedge H_2^G \wedge (\{p_{1,1}^G \doteq p_{1,2}^G, \ldots, p_{l,1}^G \doteq p_{l,2}^G\} \cup \mathcal{E}_1 \cup \mathcal{E}_2) \Rightarrow C_2^G$*.* $R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G$ *is undefined.*

The immersion of $R^G = H^G \Rightarrow C^G$ into $(i_1, \ldots, i_h) \in J$ corresponds to $\bigwedge_{(i_1,\ldots,i_h)\in J} H^G(i_1, \ldots, i_h) \Rightarrow C^G(i_1, \ldots, i_h)$, which represents the combination of a distinct instance of R^G for each $(i_1, \ldots, i_h) \in J$, as outlined in Section 6.1.2. To represent distinct instances of R^G, the free indices i, variables x, and functions ϕ use $(i_1, \ldots, i_h)$ as additional indices or arguments. Sets I should also become functions of the indices $(i_1, \ldots, i_h)$; however, to

simplify the syntax of generalized Horn clauses, we avoid sets that depend on indices: when $\bigwedge_{(j_1,\dots,j_k)\in I\times[1,M_1]\times\cdots\times[1,M_l]} pred(p_1^G,\dots,p_l^G) \in H^G$, instead of writing

$$\bigwedge_{(i_1,\dots,i_h)\in J} \bigwedge_{(j_1,\dots,j_k)\in I(i_1,\dots,i_h)\times[1,M_1]\times\cdots\times[1,M_l]} pred(p_1^G,\dots,p_l^G)$$

we write $\bigwedge_{(i_1,\dots,i_h,j_1,\dots,j_k)\in I'\times[1,M_1]\times\cdots\times[1,M_l]} pred(p_1^G,\dots,p_l^G)$ where the symbol I' stands for the set of tuples $\{(i_1,\dots,i_h,j_1,\dots,j_{k-l}) \mid (i_1,\dots,i_h)\in J, (j_1,\dots,j_{k-l})\in I(i_1,\dots,i_h)\}$. This leads to point 5 of the definition of immersion, and the corresponding transformation of constraints in point 6. This transformation introduces a minor approximation, since the obtained clause does not keep the information on how the set I' is built.

In order to resolve $R_2^G = Cts_2, F_0^G \wedge H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$ with R_1^G upon $F_0^G = \bigwedge_{(i_1,\dots,i_h)\in J} pred_2(p_{1,2}^G, \dots p_{l,2}^G)$, we perform the resolution on any non-empty subset I' of J. We distinguish two cases: either I' is the full set J, and we produce the full hyperresolution $R_1^G \circ^{\text{Full}}_{F_0^G} R_2^G$, or I' is a strict subset of J, and we produce the partial hyperresolution $R_1^G \circ^{\text{Part}}_{F_0^G} R_2^G$. In the latter case, using the set I'' such that $I' \uplus I'' = J$, R_2^G can also be written $R_2^G = Cts_2 \cup \{I' \uplus I'' = J\}$, $\bigwedge_{(i_1,\dots,i_h)\in I'} pred_2(p_{1,2}^G, \dots p_{l,2}^G) \wedge \bigwedge_{(i_1,\dots,i_h)\in I''} pred_2(p_{1,2}^G, \dots p_{l,2}^G) \wedge H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$. We can then perform resolution upon $F_{I'}^G = \bigwedge_{(i_1,\dots,i_h)\in I'} pred_2(p_{1,2}^G, \dots p_{l,2}^G)$ for the full set I'. We use the immersion to compute a clause $R_{I'}^G$ that corresponds to instances of R_1^G for all $(i_1,\dots,i_h)\in I'$, and then perform a fairly standard resolution step: we require that the conclusion of $R_{I'}^G$ equals the hypothesis $F_{I'}^G$ of R_2^G and generate the combined clause. (The unification of the conclusion of $R_{I'}^G$ with the hypothesis $F_{I'}^G$ of R_2^G is delayed until the simplification of clauses, described in Section 7.3.)

When $h = 0$ and $J = \{()\}$, $F_0^G = pred_2(p_{1,2}^G, \dots p_{l,2}^G)$ and J is a singleton, so $R_1^G \circ^{\text{Part}}_{F_0^G} R_2^G$ does not exist, and we perform an ordinary resolution step. (Immersion is not necessary.)

Theorem 17 (Resolution)**.** *Let $\Gamma_1 \vdash R_1^G$ and $\Gamma_2 \vdash R_2^G$ be two well-typed clauses, such that $R_2^G = Cts_2, F_0^G \wedge H_2^G \wedge \mathcal{E}_2 \Rightarrow C_2^G$. We have $\Gamma_{\text{Full}} \vdash R_1^G \circ^{\text{Full}}_{F_0^G} R_2^G$ and $\Gamma_{\text{Part}} \vdash R_1^G \circ^{\text{Part}}_{F_0^G} R_2^G$ for some type environments Γ_{Full} and Γ_{Part} (when these clauses are defined).*

Let T_i be an environment for $\Gamma_i \vdash R_i^G$ such that $R_i^{GT_i}$ is defined, for $i = 1, 2$. Let $F_0 \in \text{MGU}\,(\mathcal{E}_2^{T_2}) F_0^{GT_2}$ be a fact in the hypothesis of $R_2^{GT_2}$ that comes from the translation of F_0^G. Suppose that $R_1^{GT_1}$ and $R_2^{GT_2}$ can be resolved upon F_0 into the clause R. Then R is equal to $(R_1^G \circ^{\text{Full}}_{F_0^G} R_2^G)^T$ or $(R_1^G \circ^{\text{Part}}_{F_0^G} R_2^G)^T$ up to renaming of variables, for some environment T for $\Gamma_{\text{Full}} \vdash R_1^G \circ^{\text{Full}}_{F_0^G} R_2^G$ (resp. $\Gamma_{\text{Part}} \vdash R_1^G \circ^{\text{Part}}_{F_0^G} R_2^G$).

Example 3. *Let us consider the following two clauses:*

$$R_1^G = \text{att}(x) \Rightarrow \text{att}(sha1(x)) \tag{7.1}$$

$$R_2^G = \bigwedge_{i\in[1,L]} \text{att}(sha1(cont_{\phi(i)})) \wedge \text{att}(sign(list(i \le L, sha1(cont_{\phi(i)})), \mathsf{sk}_C)) \Rightarrow \text{event}(e(r)) \tag{7.2}$$

To compute the partial hyperresolution of (7.2) *with* (7.1) *for a set I, we compute the immersion of* (7.1) *into $i \in I$: we add to x the index i and we add a conjunction $\bigwedge_{i\in I}$ in the hypothesis (replacing the omitted empty conjunction $\bigwedge_{()\in\{()\}}$):*

$$\bigwedge_{i\in I} \text{att}(x_i) \Rightarrow \text{att}(sha1(x_i)) \tag{7.3}$$

Partial hyperresolution yields

$$R_1^G \circ_{F_0}^{\text{Part}} R_2^G = \{I \uplus I' = [1, L]\},\ \bigwedge_{i\in I} \text{att}(x_i) \wedge \bigwedge_{i\in I'} \text{att}(sha1(cont_{\phi(i)})) \wedge$$
$$\text{att}(sign(list(i \le L, sha1(cont_{\phi(i)})), \mathsf{sk}_C)) \wedge \{\bigwedge_{i\in I} sha1(x_i) \doteq sha1(cont_{\phi(i)})\} \Rightarrow \text{event}(e(r)) \tag{7.4}$$

This clause will be further simplified by the transformations given in Section 7.3.

7.2.1 Proof of Theorem 17

To prove the theorem, we need to state a couple of lemmas. The proofs of these lemmas are detailed in Appendix A.2.2. We use the following standard result.

Lemma 18. *Suppose that $p_{1,1}^G, \ldots, p_{l,1}^G$ and $\mathcal{E}_1$ contain no variable common with $p_{1,2}^G, \ldots, p_{l,2}^G$ and $\mathcal{E}_2$. Let $\mathcal{E} = \{C\, p_{1,1}^G \doteq p_{1,2}^G, \ldots, C\, p_{l,1}^G \doteq p_{l,2}^G\} \cup \mathcal{E}_1 \cup \mathcal{E}_2$. Let T be an environment such that*

$$\sigma = \text{MGU}\,\{\text{MGU}\,(\mathcal{E}_1^T) pred(p_{1,1}^{GT'}, \ldots, p_{l,1}^{GT'}) = \text{MGU}\,(\mathcal{E}_2^T) pred(p_{1,2}^{GT'}, \ldots, p_{l,2}^{GT'}) \mid T' \in T^C\}.$$

is defined. Then $\text{MGU}\,(\mathcal{E}^T) = \sigma\text{MGU}\,(\mathcal{E}_1^T)\text{MGU}\,(\mathcal{E}_2^T)$.

Lemma 19. *Given a well-typed generalized Horn clause $\Gamma_1 \vdash R_1^G$, a set J, and a type environment Γ_2 such that $\Gamma_2 \vdash J : [1, L_1] \times \cdots \times [1, L_h]$, there exists a type environment Γ such that $\Gamma \vdash imm(R_1^G, (i_1, \ldots, i_h) \in J)$.*

Proof of Theorem 17. First case: $F_0^G = \bigwedge_{(i_1,\ldots,i_h)\in J} pred_2(p_{1,2}^G, \ldots p_{l,2}^G)$ with $h \neq 0$.

Let us first show that $R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G$ and $R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G$ are well-typed.

- First consider the clause $R^G = R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G$. Let $R_J^G = imm(R_1^G, (i_1, \ldots, i_h) \in J) = Cts_J$, $H_J^G \wedge \mathcal{E}_J \Rightarrow pred_1(p_{1,1}^G, \ldots p_{l',1}^G)$ and $\mathcal{E} = \{\bigwedge_{(i_1\ldots,i_h)\in J} p_{1,1}^G \doteq p_{1,2}^G, \ldots, \bigwedge_{(i_1\ldots,i_h)\in J} p_{l,1}^G \doteq p_{l,2}^G\} \cup \mathcal{E}_J \cup \mathcal{E}_2$. We have

$$R^G = Cts_J \cup Cts_2,\ H_J^G \wedge H_2^G \wedge \mathcal{E} \Rightarrow C_2^G$$

Suppose that $\Gamma_2 \vdash J : [1, L_1] \times \cdots \times [1, L_h]$. Let Γ_J be the type environment defined in the proof of Lemma 19, such that $\Gamma_J \vdash R_J^G$. We define the type environment $\Gamma_{\text{Full}} = (\Gamma_J \setminus \{i_1 : [1, L_1], \ldots, i_h : [1, L_h]\}) \cup \Gamma_2$. This type environment is well defined, since Γ_J and Γ_2 give the same type to their common element, the set symbol I_0 that occurs in J in case $J = I_0 \times [1, L_{k+1}] \times \cdots \times [1, L_h]$.

We have $\Gamma_{\text{Full}} \vdash R^G$. Indeed, we have for all $k = 1, \ldots, l$, $\Gamma_J \vdash p_{k_1}^G$ and $\Gamma_2, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash p_{k_2}^G$, so $\Gamma_{\text{Full}}, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash p_{k_1}^G$ and $\Gamma_{\text{Full}}, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash p_{k_2}^G$, so $\Gamma_{\text{Full}} \vdash \bigwedge_{(i_1,\ldots,i_h)\in J} p_{k_1}^G \doteq p_{k_2}^G$ by (PatEq). Each other constraint, fact, or equation in R^G comes either from R_J^G or from R_2^G, so it is well-typed in either Γ_J or Γ_2, and does not contain $i_1, \ldots, i_h$ as free indices, so it is also well-typed in Γ_{Full}.

- Next, consider $R^G = R_1^G \circ_{F_0^G}^{\text{Part}} R_2^G$. Let $R_{I'}^G = imm(R_1^G, (i_1, \ldots, i_h) \in I') = Cts_{I'}, H_{I'}^G \wedge \mathcal{E}_{I'} \rightarrow pred_1(p_{1,1}^G, \ldots p_{l',1}^G)$ and $\mathcal{E} = \{\bigwedge_{(i_1,\ldots,i_h)\in I'} p_{1,1}^G \doteq p_{1,2}^G, \ldots, \bigwedge_{(i_1,\ldots,i_h)\in I'} p_{l,1}^G \doteq p_{l,2}^G\} \cup \mathcal{E}_{I'} \cup \mathcal{E}_2$. We have

$$R^G = Cts_{I'} \cup Cts_2 \cup \{I' \uplus I'' = J\}, H_{I'}^G \wedge \bigwedge_{(i_1,\ldots,i_h)\in I''} pred_2(p_{1,2}^G, \ldots p_{l,2}^G) \wedge H_2^G \wedge \mathcal{E} \Rightarrow C_2^G$$

Suppose that $\Gamma_2 \vdash J : [1, L_1] \times \cdots \times [1, L_h]$. Let $\Gamma_2' = \Gamma_2, I' : [1, L_1] \times \cdots \times [1, L_h]$, $I'' : [1, L_1] \times \cdots \times [1, L_h]$. Let $\Gamma_{I'}$ be the type environment defined in the proof of Lemma 19, such that $\Gamma_{I'} \vdash R_{I'}^G$. We define the type environment $\Gamma_{\text{Part}} = (\Gamma_{I'} \setminus \{i_1 : [1, L_1], \ldots, i_h : [1, L_h]\}) \cup \Gamma_2'$. This type environment is well defined since $\Gamma_{I'}$ and Γ_2' give the same type to their common element I', and $\Gamma_{\text{Part}} \vdash R^G$ as in the previous case.

Since $F_0 \in \text{MGU}(\mathcal{E}_2^{T_2})(\bigwedge_{(i_1,\ldots,i_h)\in J} pred_2(p_{1,2}^G, \ldots p_{l,2}^G))^{T_2}$, there exists a tuple $(v_1, \ldots, v_h) \in J^{T_2}$ such that $F_0 = \text{MGU}(\mathcal{E}_2^{T_2}) pred_2(p_{1,2}^{GT_2[i_1\mapsto v_1,\ldots,i_h\mapsto v_h]}, \ldots, p_{l,2}^{GT_2[i_1\mapsto v_1,\ldots,i_h\mapsto v_h]})$. Since $R_1^{GT_1}$ and $R_2^{GT_2}$ can be resolved upon F_0 into the clause R, F_0 unifies with the conclusion of $R_1^{GT_1}$, so $pred_1 = pred_2$. We have two cases:

- Case $J^{T_2} = \{(v_1, \ldots, v_h)\}$. Let $R^G = R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G$. We show that there exists an environment T for $\Gamma_{\text{Full}} \vdash R^G$ such that R^{GT} is equal to R up to renaming.

  Let $R_J^G = imm(R_1^G, (i_1, \ldots, i_h) \in J) = Cts_J, H_J^G \wedge \mathcal{E}_J \Rightarrow pred_1(p_{1,1}^G, \ldots p_{l',1}^G)$ and $\mathcal{E} = \{\bigwedge_{(i_1\ldots,i_h)\in J} p_{1,1}^G \doteq p_{1,2}^G, \ldots, \bigwedge_{(i_1\ldots,i_h)\in J} p_{l,1}^G \doteq p_{l,2}^G\} \cup \mathcal{E}_J \cup \mathcal{E}_2$. We have

$$R^G = Cts_J \cup Cts_2,\ H_J^G \wedge H_2^G \wedge \mathcal{E} \Rightarrow C_2^G$$

  We define the environment T for $\Gamma_{\text{Full}} \vdash R^G$ as the extension of T_2 such that

  - $L^T = L^{T_1}$ for all bounds L that occur in $\Gamma_1 \vdash R_1^G$;
  - $i_1^T = v_1, \ldots, i_h^T = v_h$;

- $\phi'^T(v_1,\ldots,v_h) = i^{T_1}$, where ϕ' in R_J^G is associated to the free index i in R_1^G, by Item 1 of the definition of immersion (ϕ'^T can take any value on arguments different from $v_1,\ldots,v_h$);
- $\phi'^T(v_1,\ldots,v_h,v'_1,\ldots,v'_k) = \phi^{T_1}(v'_1,\ldots,v'_k)$ for all $(v'_1,\ldots,v'_k) \in \{1,\ldots,{L'_1}^{T_1}\} \times \cdots \times \{1,\ldots,{L'_k}^{T_1}\}$, where $\Gamma_1 \vdash \phi : [1,L'_1] \times \cdots \times [1,L'_k] \to [1,L']$ and ϕ' in R_J^G is associated to ϕ in R_1^G by Item 2 of the definition of immersion;
- $I'^T = \{(v_1,\ldots,v_h)\} \times I^{T_1}$, where I' in R_J^G is associated to I in R_1^G by Item 5 of the definition of immersion.

Then $R_1^{GT_1} =^\alpha R_J^{GT}$ where $=^\alpha$ stands for equality modulo renaming of variables.
So $R = R_1^{GT_1} \circ_{F_0} R_2^{GT_2} =^\alpha R_J^{GT} \circ_{F_0} R_2^{GT}$ is defined, with

$$R_J^{GT} = \text{MGU}(\mathcal{E}_J^T)H_J^{GT} \Rightarrow \text{MGU}(\mathcal{E}_J^T)pred_1(p_{1,1}^{GT},\ldots p_{l',1}^{GT})$$
$$R_2^{GT} = F_0 \wedge \text{MGU}(\mathcal{E}_2^T)H_2^{GT} \Rightarrow \text{MGU}(\mathcal{E}_2^T)C_2^{GT}.$$

So $\text{MGU}(\mathcal{E}_J^T)pred_1(p_{1,1}^{GT},\ldots p_{l',1}^{GT})$ and $F_0 = \text{MGU}(\mathcal{E}_2^T)pred_2(p_{1,2}^{GT},\ldots p_{l,2}^{GT})$ unify, let σ be their most general unifier. We have

$$R =^\alpha \sigma\text{MGU}(\mathcal{E}_J^T)H_J^{GT} \wedge \sigma\text{MGU}(\mathcal{E}_2^T)H_2^{GT} \Rightarrow \sigma\text{MGU}(\mathcal{E}_2^T)C_2^{GT}$$

Since $J^T = \{(v_1,\ldots,v_h)\}$, for $C = \bigwedge_{(i_1,\ldots,i_h)\in J}$, $T^C = \{T[i_1 \mapsto v_1,\ldots,i_h \mapsto v_h]\} = \{T\}$, so $\sigma = \text{MGU}(\{\text{MGU}(\mathcal{E}_J^T)pred_1(p_{1,1}^{GT'},\ldots p_{l,1}^{GT'}) = \text{MGU}(\mathcal{E}_2^T)pred_2(p_{1,2}^{GT'},\ldots p_{l,2}^{GT'}) \mid T' \in T^C\})$. By Lemma 18, $\text{MGU}(\mathcal{E}^T) = \sigma\text{MGU}(\mathcal{E}_J^T)\text{MGU}(\mathcal{E}_2^T)$. The environment T_1 satisfies Cts_1, so by construction of T and Cts_J, T satisfies Cts_J. Moreover, T_2 satisfies Cts_2, so T satisfies Cts_2. Hence, T satisfies $Cts_J \cup Cts_2$. So, translating R^G, we obtain:

$$\begin{aligned}
R^{GT} &= \text{MGU}(\mathcal{E}^T)(H_J^{GT} \wedge H_2^{GT}) \Rightarrow \text{MGU}(\mathcal{E}^T)C_2^{GT} \\
&= \sigma\text{MGU}(\mathcal{E}_J^T)\text{MGU}(\mathcal{E}_2^T)(H_J^{GT} \wedge H_2^{GT}) \Rightarrow \sigma\text{MGU}(\mathcal{E}_J^T)\text{MGU}(\mathcal{E}_2^T)C_2^{GT} \\
&= \sigma\text{MGU}(\mathcal{E}_J^T)H_J^{GT} \wedge \sigma\text{MGU}(\mathcal{E}_2^T)H_2^{GT} \Rightarrow \sigma\text{MGU}(\mathcal{E}_2^T)C_2^{GT} \\
&=^\alpha R.
\end{aligned}$$

- Case $J^{T_2} \neq \{(v_1,\ldots,v_h)\}$. Let $R^G = R_1^G \circ^{\text{Part}}_{F_0^G} R_2^G$. We show that there exists an environment T for $\Gamma_{\text{Part}} \vdash R^G$ such that R^{GT} is equal to R up to renaming.

  Let $R_{I'}^G = imm(R_1^G,(i_1,\ldots,i_h) \in I') = Cts_{I'}, H_{I'}^G \wedge \mathcal{E}_{I'} \to pred_1(p_{1,1}^G,\ldots p_{l',1}^G)$ and $\mathcal{E} = \{\bigwedge_{(i_1,\ldots,i_h)\in I'} p_{1,1}^G \doteq p_{1,2}^G,\ldots,\bigwedge_{(i_1,\ldots,i_h)\in I'} p_{l,1}^G \doteq p_{l,2}^G\} \cup \mathcal{E}_{I'} \cup \mathcal{E}_2$. We have

$$R^G = Cts_{I'} \cup Cts_2 \cup \{I' \uplus I'' = J\}, H_{I'}^G \wedge \bigwedge_{(i_1,\ldots,i_h)\in I''} pred_2(p_{1,2}^G,\ldots,p_{l,2}^G) \wedge H_2^G \wedge \mathcal{E} \Rightarrow C_2^G$$

  We also define the environment T for $\Gamma_{\text{Part}} \vdash R^G$ as in the previous case, with in addition $I'^T = \{(v_1,\ldots,v_h)\}$ and $I''^T = J^{T_2} \setminus \{(v_1,\ldots,v_h)\}$. Then $R_1^{GT_1} =^\alpha R_{I'}^{GT}$. So $R = R_1^{GT_1} \circ_{F_0} R_2^{GT_2} =^\alpha R_{I'}^{GT} \circ_{F_0} R_2^{GT}$ is defined, with $R_{I'}^{GT} = \text{MGU}(\mathcal{E}_{I'}^T)H_{I'}^{GT} \Rightarrow$

$\text{MGU}(\mathcal{E}_{I'}^T)pred_1(p_1^{GT})$ and $R_2^{GT} = F_0 \wedge \text{MGU}(\mathcal{E}_2^T)(\bigwedge_{(i_1,\ldots,i_h)\in I''} pred_2(p_{1,2}^G,\ldots,p_{l,2}^G) \wedge H_2^G)^T \Rightarrow \text{MGU}(\mathcal{E}_2^T)C_2^{GT}$.

So $\text{MGU}(\mathcal{E}_{I'}^T)pred_1(p_{1,1}^{GT},\ldots p_{l',1}^{GT})$ and $F_0 = \text{MGU}(\mathcal{E}_2^T)pred_2(p_{1,2}^{GT},\ldots,p_{l,2}^{GT})$ unify, let σ be their most general unifier. We have

$$R =^{\alpha} \sigma\text{MGU}(\mathcal{E}_2^T)(\bigwedge_{(i_1,\ldots,i_h)\in I''} pred_2(p_{1,2}^G,\ldots,p_{l,2}^G) \wedge H_2^G)^T$$
$$\wedge\, \sigma\text{MGU}(\mathcal{E}_{I'}^T)H_{I'}^{GT} \Rightarrow \sigma\text{MGU}(\mathcal{E}_2^T)C_2^{GT}$$

By Lemma 18, $\text{MGU}(\mathcal{E}^T) = \sigma\text{MGU}(\mathcal{E}_{I'}^T)\text{MGU}(\mathcal{E}_2^T)$. The environment T satisfies $Cts_{I'} \cup Cts_2 \cup \{I' \uplus I'' = J\}$. So, translating R^G and letting $\overline{H}_2^G = \bigwedge_{(i_1,\ldots,i_h)\in I''} pred_2(p_{1,2}^G,\ldots,p_{l,2}^G) \wedge H_2^G$, we obtain:

$$\begin{aligned}
R^{GT} &= \text{MGU}(\mathcal{E}^T)(H_{I'}^{GT} \wedge \overline{H}_2^{GT}) \Rightarrow \text{MGU}(\mathcal{E}^T)C_2^{GT} \\
&= \sigma\text{MGU}(\mathcal{E}_{I'}^T)\text{MGU}(\mathcal{E}_2^T)(H_{I'}^{GT} \wedge \overline{H}_2^{GT}) \Rightarrow \sigma\text{MGU}(\mathcal{E}_{I'}^T)\text{MGU}(\mathcal{E}_2^T)C_2^{GT} \\
&= \sigma\text{MGU}(\mathcal{E}_{I'}^T)H_{I'}^{GT} \wedge \sigma\text{MGU}(\mathcal{E}_2^T)\overline{H}_2^{GT} \Rightarrow \sigma\text{MGU}(\mathcal{E}_2^T)C_2^{GT} \\
&=^{\alpha} R.
\end{aligned}$$

Second case: $F_0^G = pred_2(p_{1,2}^G,\ldots,p_{l,2}^G)$ and $C_1^G = pred_1(p_{1,1}^G,\ldots p_{l',1}^G)$. Since $R_1^{GT_1}$ and $R_2^{GT_2}$ can be resolved upon F_0 into the clause R, F_0 unifies with the conclusion of $R_1^{GT_1}$, so $pred_1 = pred_2$ and $l = l'$. The clauses R_1^G and R_2^G resolve into the clause $R^G = R_1^G \circ_{F_0^G}^{\text{Full}} R_2^G$:

$$R^G = Cts_1 \cup Cts_2,\ H_1^G \wedge H_2^G \wedge \mathcal{E} \Rightarrow C_2^G$$

where $\mathcal{E} = \{p_{1,1}^G \doteq p_{1,2}^G,\ldots,p_{l,1}^G \doteq p_{l,2}^G\} \cup \mathcal{E}_1 \cup \mathcal{E}_2$.

It is easy to see that R^G is well-typed in type environment $\Gamma = \Gamma_1 \cup \Gamma_2$. The environment $T = T_1 \cup T_2$ is an environment for $\Gamma \vdash R^G$. Let us show that R^{GT} is equal to R up to renaming.

The clause $R = R_1^{GT_1} \circ_{F_0} R_2^{GT_2} = R_1^{GT} \circ_{F_0} R_2^{GT}$ is defined, where $F_0 = \text{MGU}(\mathcal{E}_2^T)pred_2(p_{1,2}^{GT},\ldots,p_{l,2}^{GT})$, $R_1^{GT} = \text{MGU}(\mathcal{E}_1^T)H_1^{GT} \Rightarrow \text{MGU}(\mathcal{E}_1^T)pred_1(p_{1,1}^{GT},\ldots,p_{l,1}^{GT})$, and $R_2^{GT} = F_0 \wedge \text{MGU}(\mathcal{E}_2^T)H_2^{GT} \Rightarrow \text{MGU}(\mathcal{E}_2^T)C_2^{GT}$.

So $F_0 = \text{MGU}(\mathcal{E}_2^T)pred_2(p_{1,2}^{GT},\ldots,p_{l,2}^{GT})$ and $\text{MGU}(\mathcal{E}_1^T)pred_1(p_{1,1}^{GT},\ldots,p_{l,1}^{GT})$ unify with most general unifier σ. By Lemma 18, $\text{MGU}(\mathcal{E}^T) = \sigma\text{MGU}(\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_2^T)$ The environment T satisfies Cts_1 and Cts_2, so the translation of R^G is

$$\begin{aligned}
R^{GT} &= \text{MGU}(\mathcal{E}^T)(H_1^{GT} \wedge H_2^{GT}) \Rightarrow \text{MGU}(\mathcal{E}^T)C_2^{GT} \\
&= \sigma\text{MGU}(\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_2^T)(H_1^{GT} \wedge H_2^{GT}) \Rightarrow \sigma\text{MGU}(\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_2^T)C_2^{GT} \\
&= \sigma(\text{MGU}(\mathcal{E}_1^T)H_1^{GT} \wedge \text{MGU}(\mathcal{E}_2^T)H_2^{GT}) \Rightarrow \sigma\text{MGU}(\mathcal{E}_2^T)C_2^{GT} \\
&= R.
\end{aligned}$$

□

7.3 Simplification of Clauses

We use several additional transformations in order to simplify clauses. Some of these transformations are easily adapted from similar transformations already used in ProVerif. We prove the soundness of these transformations in Section 7.3.8.

7.3.1 Unification

Let us first define when two clause terms may unify. This notion is used in Item 17 below. We say that p^G and p'^G *may unify* when one the following cases holds:

- $p^G = x_{\iota_1,\ldots,\iota_h}$;
- $p'^G = x_{\iota_1,\ldots,\iota_h}$;
- $p^G = f(p_1^G, \ldots, p_h^G)$, $p'^G = f(p_1'^G, \ldots, p_h'^G)$, and p_j^G and $p_j'^G$ may unify for all $j \leq h$;
- $p^G = a^{L_1,\ldots,L_k}_{\iota_1,\ldots,\iota_k}[p_1^G, \ldots, p_h^G]$, $p'^G = a^{L'_1,\ldots,L'_k}_{\iota'_1,\ldots,\iota'_k}[p_1'^G, \ldots, p_h'^G]$, and p_j^G and $p_j'^G$ may unify for all $j \leq h$;
- $p^G = list(i \leq L, p_1^G)$, $p'^G = list(i' \leq L', p_1'^G)$, and p_1^G and $p_1'^G$ may unify;
- $p^G = list(i \leq L, p_1^G)$, $p'^G = \langle p_1'^G, \ldots, p_h'^G \rangle$, and p_1^G and $p_j'^G$ may unify for all $j \leq h$ (or the symmetric case obtained by swapping p^G and p'^G).

In all other cases, we say that p^G and p'^G *cannot unify*.

Instead of performing unification as part of resolution, we simplify equations a posteriori. To eliminate clauses in which the facts do not unify and to simplify the remaining clauses, we use the following algorithm, inspired by the algorithm of [33]. In a clause $R^G = Cts, H^G \wedge \mathcal{E} \Rightarrow C^G$, if some equation in $\mathcal{E}$ matches the following cases, the corresponding transformations are applied:

1. $C\, f(p_1^G, \ldots, p_k^G) \doteq f(p_1'^G, \ldots, p_k'^G)$: replace the equation with $C\, p_1^G \doteq p_1'^G$, $\ldots$, $C\, p_k^G \doteq p_k'^G$.
2. $C\, a^{L_1,\ldots,L_h}_{\iota_1,\ldots,\iota_h}[p_1^G, \ldots, p_k^G] \doteq a^{L'_1,\ldots,L'_h}_{\iota'_1,\ldots,\iota'_h}[p_1'^G, \ldots, p_k'^G]$: replace the equation with $C\, p_1^G \doteq p_1'^G, \ldots, C\, p_k^G \doteq p_k'^G, C\, \iota_1 \doteq \iota'_1, \ldots, C\, \iota_h \doteq \iota'_h$ and replace every occurrence of L'_j in the clause with L_j for each $j = 1, \ldots, h$.

3. $\bigwedge_{(i_1,\ldots,i_h)\in J} list(i \leq L, p^G) \doteq list(i \leq L', p'^G)$: replace this equation with the following $\bigwedge_{(i_1,\ldots,i_h,i)\in J\times[1,L]} p^G \doteq p'^G$ and replace every occurrence of L' in the clause with L.

4. $\bigwedge_{(i_1,\ldots,i_{h'})\in J} list(i \leq L, p^G) \doteq \langle p_1^G, \ldots, p_h^G\rangle$ or $\bigwedge_{(i_1,\ldots,i_{h'})\in J} \langle p_1^G, \ldots, p_h^G\rangle \doteq list(i \leq L, p^G)$: this equation can be handled in two ways.

   The preferred solution is to instantiate L into the integer h, and apply unification again on the obtained clause(s). This solution is applied when the clause contains no function ϕ with a result of type $[1, L]$, no set symbol I with a type that contains $[1, L]$, and no name with an index of type $[1, L]$. (Otherwise, the instantiated clause may not fit in our language of clauses.) The instantiation function $\Im$ takes as argument a mapping T from indices of type $[1, L]$ to integers in $\{1, \ldots, h\}$. We generates clauses $\Im_{T_0}(R^G)$ for all mappings T_0 from the free indices of type $[1, L]$ of R^G to integers in $\{1, \ldots, h\}$. The instantiation function is defined by induction on the syntax, as follows:

   - We define $\Im_T(i) = i^T$ if i is of type $[1, L]$, and $\Im_T(i) = i$ otherwise.
   - Functions over indices ϕ with k arguments of type $[1, L]$ are mapped to h^k functions $\phi\_v_1\_\ldots\_v_k$ with $(v_1, \ldots, v_k) \in \{1, \ldots, h\}^k$; thus, $\Im_T(\phi(\iota_1, \ldots, \iota_{k+k'})) = \phi\_v_1\_\ldots\_v_k(\Im_T(\iota'_1), \ldots, \Im_T(\iota'_{k'}))$ where $\iota_1, \ldots, \iota_{k+k'}$ consists of k indices $i_1, \ldots, i_k$ of type $[1, L]$ such that $i_l^T = v_l$ for all $l \leq k$, and k' indices $\iota'_1, \ldots, \iota'_{k'}$ of other types. (When $k' = 0$, $\phi\_v_1\_\ldots\_v_k$ is in fact a free index rather than a function, since it has no argument.)
   - Variables x with k indices of type $[1, L]$ are mapped to h^k variables $x\_v_1\_\ldots\_v_k$ with $(v_1, \ldots, v_k) \in \{1, \ldots, h\}^k$; thus, $\Im_T(x_{\iota_1,\ldots,\iota_{k+k'}}) = x\_v_1\_\ldots\_v_{k(\Im_T(\iota'_1),\ldots,\Im_T(\iota'_{k'}))}$ where $\iota_1, \ldots, \iota_{k+k'}$ consists of k indices $i_1, \ldots, i_k$ of type $[1, L]$ such that $i_l^T = v_l$ for all $l \leq k$, and k' indices $\iota'_1, \ldots, \iota'_{k'}$ of other types.
   - We define $\Im_T(list(i \leq L, p^G)) = \langle \Im_{T[i\mapsto 1]}(p^G), \ldots, \Im_{T[i\mapsto h]}(p^G)\rangle$.
   - We define

$$\Im_T(\bigwedge_{(i_1,\ldots,i_{k+k'})\in J} pred(p_1^G, \ldots, p_l^G)) = \bigwedge_{(i'_1,\ldots,i'_{k'})\in J'} pred(\Im_{T'_1}(p_1^G), \ldots, \Im_{T'_1}(p_l^G)) \wedge \ldots \wedge \bigwedge_{(i'_1,\ldots,i'_{k'})\in J'} pred(\Im_{T'_{h^k}}(p_1^G), \ldots, \Im_{T'_{h^k}}(p_l^G))$$

   where $i_1, \ldots, i_{k+k'}$ consists of k indices of type $[1, L]$ and k' indices $i'_1, \ldots, i'_{k'}$ of other types, J' is obtained from J by removing the factors $[1, L]$, $T'_1, \ldots, T'_{h^k}$ are the extensions of T that map the k indices of type $[1, L]$ among $i_1, \ldots, i_{k+k'}$ to integers in $\{1, \ldots, h\}$.

   - The case of equations is handled similarly to the case of facts. In case we instantiate $\bigwedge_{(i_1,\ldots,i_{k+k'})\in J} \iota \doteq \iota'$, and ι and ι' are of type $[1, L]$, we obtain equations $v \doteq v'$ where v and v' are integers in $\{1, \ldots, h\}$. When there is such an equation $v \doteq v'$ with $v \neq v'$, we remove the clause. We remove the equations $v \doteq v$.
   - In all other cases, the instantiation is just applied recursively to the subelements.

Example 4. *Let us consider the following clause:*

$$R^G = \bigwedge_{(i,j)\in[1,L]\times[1,L']} \mathrm{att}(senc(s_{\phi(i,j)}, y_i)) \wedge \mathrm{att}(a_1) \wedge \mathrm{att}(a_2) \wedge \mathrm{att}(a_3) \wedge$$
$$\{list(i \leq L, y_i) \doteq \langle a_1, a_2, a_3\rangle\} \Rightarrow \mathrm{att}(s_{\phi(m,k)})$$

We first instantiate L to 3, since the list $list(i \leq L, y_i)$ must have length 3 for the unification to succeed. Then we generate the clauses $\Im_{T_0}(R^G)$ for all mapping T_0 from m to $\{1,2,3\}$, that is $\{m \mapsto 1\}, \{m \mapsto 2\}, \{m \mapsto 3\}$ and obtain the following clauses:

$$\Im_{\{m\mapsto 1\}}(R^G) = \bigwedge_{j\in[1,L']} \mathrm{att}(senc(s_{\phi\_1(j)}, y_1)) \wedge \bigwedge_{j\in[1,L']} \mathrm{att}(senc(s_{\phi\_2(j)}, y_2)) \wedge$$
$$\bigwedge_{j\in[1,L']} \mathrm{att}(senc(s_{\phi\_3(j)}, y_3)) \wedge \mathrm{att}(a_1) \wedge \mathrm{att}(a_2) \wedge \mathrm{att}(a_3) \wedge$$
$$\{\langle y_1, y_2, y_3\rangle \doteq \langle a_1, a_2, a_3\rangle\} \Rightarrow \mathrm{att}(s_{\phi\_1(k)})$$
$$\Im_{\{m\mapsto 2\}}(R^G) = \bigwedge_{j\in[1,L']} \mathrm{att}(senc(s_{\phi\_1(j)}, y_1)) \wedge \bigwedge_{j\in[1,L']} \mathrm{att}(senc(s_{\phi\_2(j)}, y_2)) \wedge$$
$$\bigwedge_{j\in[1,L']} \mathrm{att}(senc(s_{\phi\_3(j)}, y_3)) \wedge \mathrm{att}(a_1) \wedge \mathrm{att}(a_2) \wedge \mathrm{att}(a_3) \wedge$$
$$\{\langle y_1, y_2, y_3\rangle \doteq \langle a_1, a_2, a_3\rangle\} \Rightarrow \mathrm{att}(s_{\phi\_2(k)})$$
$$\Im_{\{m\mapsto 3\}}(R^G) = \bigwedge_{j\in[1,L']} \mathrm{att}(senc(s_{\phi\_1(j)}, y_1)) \wedge \bigwedge_{j\in[1,L']} \mathrm{att}(senc(s_{\phi\_2(j)}, y_2)) \wedge$$
$$\bigwedge_{j\in[1,L']} \mathrm{att}(senc(s_{\phi\_3(j)}, y_3)) \wedge \mathrm{att}(a_1) \wedge \mathrm{att}(a_2) \wedge \mathrm{att}(a_3) \wedge$$
$$\{\langle y_1, y_2, y_3\rangle \doteq \langle a_1, a_2, a_3\rangle\} \Rightarrow \mathrm{att}(s_{\phi\_3(k)})$$

When the instantiation described above cannot be applied, we replace the equation with

$$\bigwedge_{(i_1,\ldots,i_{h'},i)\in J\times[1,L]} x_{i_1,\ldots,i_{h'},i} \doteq p^G$$
$$\bigwedge_{(i_1,\ldots,i_{h'},i)\in I_1} x_{i_1,\ldots,i_{h'},i} \doteq p_1^G$$
$$\ldots$$
$$\bigwedge_{(i_1,\ldots,i_{h'},i)\in I_h} x_{i_1,\ldots,i_{h'},i} \doteq p_h^G$$

and add the constraint $I_1 \uplus \ldots \uplus I_h = J\times[1,L]$, where x is a fresh variable and $I_1, \ldots, I_h$ are fresh set symbols. Intuitively, the variable $x_{i_1,\ldots,i_{h'},i}$ contains the i-th element of the list. This solution introduces an approximation: we remember that the elements of the list are $p_1^G, \ldots, p_h^G$ but we forget their order and their number of repetitions.

Example 5. *Let us consider the following clause:*

$$R^G = \bigwedge_{i\in[1,L]} \mathrm{att}(senc(s_i, y_{\phi(i)})) \wedge \mathrm{att}(a_1) \wedge \mathrm{att}(a_2) \wedge \mathrm{att}(a_3) \wedge$$
$$\{list(i \leq L, y_i) \doteq \langle a_1, a_2, a_3\rangle\} \Rightarrow \mathrm{att}(s_k)$$

In this case we cannot apply the instantiation described above because the result type of ϕ is $[1, L]$ and the name s has an index of type $[1, L]$. Hence we create a fresh variable x_i and three fresh symbols I_1, I_2, I_3 and then replace the equation with

$$\bigwedge_{i\in[1,L]} x_i \doteq y_i, \bigwedge_{i\in I_1} x_i \doteq a_1, \bigwedge_{i\in I_2} x_i \doteq a_2, \bigwedge_{i\in I_3} x_i \doteq a_3$$

and finally we add the constraint $I_1 \uplus I_2 \uplus I_3 = [1, L]$ to the set of constraints.

5. $C\, f(p_1^G, \dots p_k^G) \doteq g(p_1'^G, \dots, p_m'^G)$ where $f \neq g$: delete the clause. This case also applies to $a\_$ or *list* instead of f and/or g.
6. $C\, x_{\iota_1,\dots,\iota_h} \doteq x_{\iota_1,\dots,\iota_h}$: delete the equation.
7. $C\, p^G \doteq x_{\iota_1,\dots,\iota_h}$ where p^G is not variable: replace the equation with $C\, x_{\iota_1,\dots,\iota_h} \doteq p^G$.
8. $C\, x_{\iota_1,\dots,\iota_h} \doteq p^G$ where $p^G \neq x_{\iota_1,\dots,\iota_h}$ and $x_{\iota_1,\dots,\iota_h}$ occurs in p^G: delete the clause.
9. $C\, \iota \doteq \iota$: delete the equation.
10. $\bigwedge_{(i_1\dots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} x_{\iota_1,\dots,\iota_{h'}} \doteq p'^G$ and p'^G does not contain $x_{\iota_1,\dots,\iota_{h'}}\{i_1 \mapsto \iota'_1, \dots, i_h \mapsto \iota'_h\}$ for any $\iota'_1, \dots, \iota'_h$: replace $x_{\iota_1,\dots,\iota_{h'}}\{i_1 \mapsto \iota'_1, \dots, i_h \mapsto \iota'_h\}$ (for any $\iota'_1, \dots, \iota'_h$) with $p'^G\{i_1 \mapsto \iota'_1, \dots, i_h \mapsto \iota'_h\}$ everywhere else in the clause. (The condition "p'^G does not contain $x_{\iota_1,\dots,\iota_{h'}}\{i_1 \mapsto \iota'_1, \dots, i_h \mapsto \iota'_h\}$ for any $\iota'_1, \dots, \iota'_h$" serves to avoid loops in which we would infinitely replace $x\_$ with p'^G.)
11. $\bigwedge_{(i_1\dots,i_{k+h})\in I\times[1,L_1]\times\cdots\times[1,L_h]} x_{\iota_1,\dots,\iota_{h'}} \doteq p'^G$ and p'^G does not contain $x_{\iota_1,\dots,\iota_{h'}}\{i_{k+1} \mapsto \iota'_1, \dots, i_{k+h} \mapsto \iota'_h\}$ for any $\iota'_1, \dots, \iota'_h$: replace $x_{\iota_1,\dots,\iota_{h'}}\{i_1 \mapsto i'_1, \dots, i_h \mapsto i'_k, i_{k+1} \mapsto \iota'_1, \dots, i_{k+h} \mapsto \iota'_h\}$ (for any $\iota'_1, \dots, \iota'_h$) with $p'^G\{i_1 \mapsto i'_1, \dots, i_h \mapsto i'_k, i_{k+1} \mapsto \iota'_1, \dots, i_{k+h} \mapsto \iota'_h\}$ under a conjunction $\bigwedge_{(i'_1,\dots,i'_{k+h'})\in I\times[1,L_1]\times\cdots\times[1,L_{h'}]}$ everywhere else in the clause.
12. $i \doteq \iota$ or $\iota \doteq i$, where i does not occur in ι: replace i with ι everywhere else in the clause, and delete the equation.
13. $\bigwedge_{(i_1\dots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} \iota \doteq \iota'$ or $\bigwedge_{(i_1\dots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} \iota' \doteq \iota$: replace $\iota'\{i_1 \mapsto \iota_1, \dots, i_h \mapsto \iota_h\}$ (for any $\iota_1, \dots, \iota_h$) with $\iota\{i_1 \mapsto \iota_1, \dots, i_h \mapsto \iota_h\}$ everywhere else in the clause. This replacement is performed only when one of the following two conditions holds:
    - $\iota' = \phi(\iota'_1, \dots, \iota'_{h'})$, ϕ does not occur in ι, ι'_1, …, $\iota'_{h'}$, all indices $i_1, \dots, i_h$ that occur in ι also occur in ι', and this replacement removes all other occurrences of ϕ. In this situation, we also remove the equation. Indeed, it is very likely that we can find a function ϕ that satisfies the equation. In case we cannot find one, the transformation introduces an approximation, but remains sound.
    - $|\iota| < |\iota'|$ (that is, ι has fewer symbols than ι'). In this case, we reduce the size of the indices, thus simplifying the clause.
14. $\bigwedge_{(i_1\dots,i_{k+h})\in I\times[1,L_1]\times\dots[1,L_h]} \iota \doteq \iota'$ or $\bigwedge_{(i_1\dots,i_{k+h})\in I\times[1,L_1]\times\dots[1,L_h]} \iota' \doteq \iota$: replace $\iota'\{i_1 \mapsto i'_1, \dots, i_h \mapsto i'_k, i_{k+1} \mapsto \iota_1, \dots, i_{k+h} \mapsto \iota_h\}$ (for any $\iota_1, \dots, \iota_h$) with $\iota\{i_1 \mapsto i'_1, \dots, i_h \mapsto i'_k, i_{k+1} \mapsto \iota_1, \dots, i_{k+h} \mapsto \iota_h\}$ under a conjunction $\bigwedge_{(i'_1,\dots,i'_{k+h'})\in I\times[1,L_1]\times\dots[1,L_{h'}]}$ everywhere else in the clause. This replacement is performed under the same conditions as the previous one.

15. $C\, x_{i_1,\ldots,i_h} \doteq p^G$ or $C\, p^G \doteq x_{i_1,\ldots,i_h}$, $fi(p^G) \subseteq \{i_1, \ldots, i_h\}$, and x does not occur anywhere else in R^G (in particular, x does not occur in p^G): delete the equation.

16. If in $\bigwedge_{(i_1,\ldots,i_h)\in J} X$, some indices $i_1, \ldots, i_h$ do not occur in X, where $X = pred(p_1^G, \ldots, p_l^G)$, $p^G \doteq p'^G$, or $\iota \doteq \iota'$, then we remove the unused indices from the conjunction. More precisely:

    - If $J = [1, L_1]\times\cdots\times[1, L_h]$, then we remove the indices i_k ($k \in \{1, \ldots, h\}$) that do not occur in X from the conjunction and we remove the corresponding factors $[1, L_k]$ from J.
    - If $J = I'\times[1, L_{l+1}]\times\cdots\times[1, L_h]$, then we remove the indices i_k ($k \in \{l+1, \ldots, h\}$) that do not occur in X from the conjunction and remove the corresponding factors $[1, L_k]$ from J. Moreover, if $i_1, \ldots, i_l$ do not occur in X, then we remove the indices $i_1, \ldots, i_l$ from the conjunction and remove the corresponding factor I' from J.

17. Let sel^G be a generalized selection function, that is a function from generalized Horn clauses to sets of facts, such that $sel^G(Cts, H^G \wedge \mathcal{E} \Rightarrow C^G) \subseteq H^G$. (We will give the formal definition of it in Section 8.1). Suppose that the following conditions are satisfied: H^G contains a fact $\bigwedge_{(i_1,\ldots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} \mathrm{att}(x_{i_1,\ldots,i_h})$ (the indices in the conjunction may be reordered); $sel^G(R^G) = \emptyset$; if $C^G = \mathrm{att}(p^G)$, then x does not occur in p^G; and $\mathcal{E}$ contains equations $C_k\, x_{\iota_{k,1},\ldots,\iota_{k,h}} \doteq p_k^G$ for $k = 1, \ldots, K$, where p_k^G and $p_{k'}^G$ cannot unify for all $k \neq k'$ and, for each index i bound by C_k, there exists $l \leq h$ such that $\iota_{k,l} = i$. Then we replace the fact $\bigwedge_{(i_1,\ldots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} \mathrm{att}(x_{i_1,\ldots,i_h})$ with the conjunction of the facts $C_k\, \mathrm{att}(p_k^G)$ for $k = 1, \ldots, K$.

    Intuitively, the hypothesis of the clause contains $\mathrm{att}(x_{i_1,\ldots,i_h})$ for all $i_1, \ldots, i_h$, so in particular, $C_k\, \mathrm{att}(x_{\iota_{k,1},\ldots,\iota_{k,h}})$ for all k, that is, $C_k\, \mathrm{att}(p_k^G)$ for all k, given the equations $\mathcal{E}$.

    Formally, for this transformation to be sound, the translation of the obtained generalized Horn clause must subsume the translation of the initial generalized Horn clause. Since the translation of initial clause contains one fact $\mathrm{att}(x_{i_1,\ldots,i_h})$ for each value of $i_1, \ldots, i_h$, the transformation must not introduce several hypotheses $\mathrm{att}(x_{\iota_1,\ldots,\iota_h}) = \mathrm{att}(p^G)$ for indices $\iota_1, \ldots, \iota_h$ that have the same value. (Recall that hypotheses form a multiset, and that the definition of subsumption uses multiset inclusion.) The hypothesis that p_k^G and $p_k'^G$ do not unify for $k \neq k'$ guarantees that distinct facts $C_k\, \mathrm{att}(p_k^G)$ and $C_{k'}\, \mathrm{att}(p_{k'}^G)$ correspond to different indices of x. (We could also use other criteria to obtain this information, for example by relying on constraints on sets. The current criterion is sufficient in our examples.) The hypothesis that for each index i bound by C_k, there exists $l \leq h$ such that $\iota_{k,l} = i$ guarantees that a single fact $C_k\, \mathrm{att}(p_k^G) = C_k\, \mathrm{att}(x_{\iota_1,\ldots,\iota_h})$ does not yield $\mathrm{att}(x_{i_1,\ldots,i_h})$ several times for the same value of $i_1, \ldots, i_h$.

    When there are occurrences of x in addition to the K equations of $\mathcal{E}$ above, this transformation may introduce an approximation, since we may forget that the attacker must have $x_{i_1,\ldots,i_h}$ for some values of $i_1, \ldots, i_h$. That is why we perform this transformation only when there is little chance of obtaining a proof without it, which is

checked by the hypothesis that $sel^G(R^G) = \emptyset$ and, if $C^G = \text{att}(p^G)$, then x does not occur in p^G. Indeed, when $sel^G(R^G) = \emptyset$, we will further resolve upon some hypothesis of the clause, and the clause is not included in the final result of the resolution algorithm. When $sel^G(R^G) = \emptyset$ and the conclusion $C^G = att(p^G)$ contains x, the clause is included in the final result of the resolution algorithm, but it does not prevent proving secrecy or authentication; furthermore, the value of x may be instantiated by future resolution. On the other hand, when $sel^G(R^G) = \emptyset$ and the conclusion is not of the form $C^G = \text{att}(p^G)$ where x occurs in p^G, the clause is present in final result of the resolution algorithm, and either it may prevent proving secrecy or authentication, or x will stay uninstantiated in it. In this case, the transformation improves precision by allowing us to take into account the information on x that we have in the equations.

Example 6. *Consider the following clause:*

$$R^G = \{I_1 \uplus I_2 = [1, L] \times [1, L']\}, \bigwedge_{(i,j)\in[1,L]\times[1,L']} \text{att}(x_{i,j}) \wedge \bigwedge_{j\in[1,L']} \text{att}(b_j) \wedge$$
$$\{\bigwedge_{(i,j)\in I_1} x_{i,j} \doteq senc(a_i, b_j), \bigwedge_{(i,j)\in I_2} x_{i,j} \doteq (a_i, b_j)\} \Rightarrow \text{att}(a_i)$$

Clearly $senc(a_i, b_j)$ and (a_i, b_j) cannot unify. We can then apply this last transformation and replace the fact $\bigwedge_{(i,j)\in[1,L]\times[1,L']} \text{att}(x_{i,j})$ with the conjunction of facts $\bigwedge_{(i,j)\in I_1} \text{att}(senc(a_i, b_j)) \wedge \bigwedge_{(i,j)\in I_2} \text{att}((a_i, b_j))$ and obtain the following clause:

$$R^G = \{I_1 \uplus I_2 = [1, L] \times [1, L']\}, \bigwedge_{(i,j)\in I_1} \text{att}(senc(a_i, b_j)) \wedge$$
$$\bigwedge_{(i,j)\in I_2} \text{att}((a_i, b_j)) \wedge \bigwedge_{j\in[1,L']} \text{att}(b_j) \wedge$$
$$\{\bigwedge_{(i,j)\in I_1} x_{i,j} \doteq senc(a_i, b_j), \bigwedge_{(i,j)\in I_2} x_{i,j} \doteq (a_i, b_j)\} \Rightarrow \text{att}(a_i)$$

The unification algorithm is not complete, in that for some complex equations, it may be unable to use the equations to simplify the clause or to remove it, even though that would be sound. This limitation may be lead to false attacks. In practice, the algorithm is still precise enough to be able the prove all desired properties of our case studies (Section 8.3).

Example 7. *Applying these transformations, the clause* (7.4) *becomes*

$$R_1^G \circ_{F_0}^{\text{Part}} R_2^G = \{I \uplus I' = [1, L]\}, \bigwedge_{i\in I} \text{att}(cont_{\phi(i)}) \wedge$$
$$\bigwedge_{i\in I'} \text{att}(sha1(cont_{\phi(i)})) \wedge \tag{7.5}$$
$$\text{att}(sign(list(i \le L, sha1(cont_{\phi(i)})), \mathsf{sk}_C)) \Rightarrow \text{event}(e(r))$$

Indeed, the equation $\bigwedge_{i\in I} sha1(x_i) \doteq sha1(cont_{\phi(i)})$ first becomes $\bigwedge_{i\in I} x_i \doteq cont_{\phi(i)}$, then $cont_{\phi(i)}$ is substituted for x_i, and the equation is removed.

7.3.2 Merging of Sets

When a clause uses two disjoint sets I and I' in the same facts and equations (up to renaming), we merge these two sets into a single set I''. This transformation is key to obtain termination of the algorithm: when a clause R^G is obtained by partial hyperresolution of two clauses R_1^G and R_2^G, it can be resolved again with R_1^G into R'^G, and so on, which would yield an infinite loop. However, by merging sets in R'^G, we can build a clause that is subsumed by R^G, and so removed, which stops the loop. We illustrate this point on an example. The clause (7.5) can be resolved again by partial hyperresolution with the clause (7.2). We obtain:

$$\{I \uplus I' = [1, L], I_1 \uplus I'_1 = I'\}, \bigwedge_{i \in I} \mathrm{att}(cont_{\phi(i)}) \wedge$$
$$\bigwedge_{i \in I_1} \mathrm{att}(cont_{\phi(i)}) \wedge$$
$$\bigwedge_{i \in I'_1} \mathrm{att}(sha1(cont_{\phi(i)})) \wedge$$
$$\mathrm{att}(sign(list(i \leq L, sha1(cont_{\phi(i)})), \mathsf{sk}_C)) \Rightarrow \mathrm{event}(e(r))$$

which could be resolved again with (7.2). However, after replacing the two constraints with $I \uplus I_1 \uplus I'_1 = [1, L]$, we merge I and I_1 together into the set I_2:

$$\{I_2 \uplus I'_1 = [1, L]\}, \bigwedge_{i \in I_2} \mathrm{att}(cont_{\phi(i)}) \wedge$$
$$\bigwedge_{i \in I'_1} \mathrm{att}(sha1(cont_{\phi(i)})) \wedge$$
$$\mathrm{att}(sign(list(i \leq L, sha1(cont_{\phi(i)})), \mathsf{sk}_C)) \Rightarrow \mathrm{event}(e(r))$$

This clause will be removed by the algorithm of Section 8.1, because it is subsumed by (7.5), so the loop is avoided.

The merging of sets is not always able to avoid loops that come from partial hyperresolution, in particular when a fresh bound L is created at each partial hyperresolution step. This limitation is the main new cause of non-termination, but its impact is limited in practice since the selection function is tuned to avoid partial hyperresolution when possible.

Next we formalize this transformation in details. If a clause contains constraints $I_1 \uplus \cdots \uplus I_h = I$ and $I'_1 \uplus \cdots \uplus I'_{h'} = J$ with $I = I'_k$, and I does not occur elsewhere, then we replace these constraints with $I'_1 \uplus \cdots \uplus I'_{k-1} \uplus I_1 \uplus \cdots \uplus I_h \uplus I'_{k+1} \uplus \cdots \uplus I'_{h'} = J$.

If a clause R^G, well-typed in Γ, contains a constraint $I_1 \uplus \cdots \uplus I_h = J$, we try to merge any pair of sets among $I_1, \ldots, I_h$, say I_1 and I_2. We let S_1 be the smallest set of set symbols such that S_1 contains I_1, and if R^G contains a constraint $I'_1 \uplus \cdots \uplus I'_h = I' \times [1, L_1] \times \cdots \times [1, L_n]$ with $I' \in S_1$, then S_1 also contains $I'_1, \ldots, I'_h$. We define S_2 similarly with I_2 instead of I_1. We split the

set of constraints of R^G into the following components: the constraint $I_1 \uplus \cdots \uplus I_h = J$, the constraints Cts_1 that contain a set symbol in S_1 in their right-hand side, the constraints Cts_2 that contain a set symbol in S_2 in their right-hand side, and the remaining constraints Cts. (Because of Invariant 3 of Section 6.2, S_1 and S_2 are disjoint, and Cts_1 and Cts_2 are disjoint.) Similarly, we split the hypotheses of R^G into the hypotheses H_1^G that contain a set symbol in S_1, the hypotheses H_2^G that contain a set symbol in S_2, and the remaining hypotheses H^G. We split similarly the equations of R^G into $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}$. Now $R^G = \{I_1 \uplus \cdots \uplus I_h = J\} \cup Cts_1 \cup Cts_2 \cup Cts, H_1^G \wedge H_2^G \wedge H^G \wedge (\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}) \Rightarrow C^G$. Suppose that there exists a renaming α such that $\alpha I_1 = I_2$, $\alpha Cts_1 = Cts_2$, $\alpha H_1^G = H_2^G$, and $\alpha\mathcal{E}_1 = \mathcal{E}_2$, where α can rename variables, functions ϕ, set symbols I, and bounds L, and equality is modulo renaming of bound names and modulo commutativity of $\doteq$ in equations. Moreover, we suppose that the elements in the image of α do not occur in Cts_1, H_1^G, $\mathcal{E}_1$. We suppose that the elements in the domain or image of α do not occur in Cts, H^G, $\mathcal{E}$, C^G. We suppose that the domain of α does not contain bounds L that occur in lists, such as $list(i \leq L, p^G)$, or in intervals $[1, L]$ in Cts_1, Cts_2 or in indices of conjunctions in H_1^G, H_2^G, $\mathcal{E}_1$, $\mathcal{E}_2$. We also suppose that, if α renames x and x is used under a conjunction $\bigwedge_{(i_1,\ldots,i_k,j_1,\ldots,j_n)\in I\times[1,L_1]\times\cdots\times[1,L_n]}$, then x has $i_1, \ldots, i_k$ as first indices $(x_{i_1,\ldots,i_k,\ldots})$, and similarly for αx. Similarly, if α renames ϕ and ϕ is used under a conjunction $\bigwedge_{(i_1,\ldots,i_k,j_1,\ldots,j_n)\in I\times[1,L_1]\times\cdots\times[1,L_n]}$, then ϕ has $i_1, \ldots, i_k$ as first arguments $(\phi(i_1, \ldots, i_k, \ldots))$, and similarly for $\alpha\phi$. Then we replace R^G with $R'^G = \{I_1 \uplus I_3 \uplus \cdots \uplus I_h = J\} \cup Cts_1 \cup Cts, H_1^G \wedge H^G \wedge (\mathcal{E}_1 \cup \mathcal{E}) \Rightarrow C^G$, where I_1 now stands for the union $I_1 \uplus I_2$.

In case $h = 2$, the constraint $I_1 \uplus I_3 \uplus \cdots \uplus I_h = J$ is just $I_1 = J$, so we replace I_1 with J in the obtained clause, and delete the constraint.

7.3.3 Decomposition of Tuples

For tuples, ProVerif generates the following two clauses:

$$\mathrm{att}(x_1) \wedge \cdots \wedge \mathrm{att}(x_n) \Rightarrow \mathrm{att}((x_1, \ldots, x_n))$$
$$\mathrm{att}((x_1, \ldots, x_n)) \Rightarrow \mathrm{att}(x_i)$$

Hence, $\mathrm{att}((p_1, \ldots, p_n))$ is derivable if and only if $\forall i \in \{1, \ldots, n\}$, $\mathrm{att}(p_i)$ is derivable, so $C\,\mathrm{att}((p_1, \ldots, p_n))$ can be replaced with the conjunction $C\,\mathrm{att}(p_1) \wedge \cdots \wedge C\,\mathrm{att}(p_n)$. If this replacement is done in the conclusion of a clause $H \Rightarrow \mathrm{att}(f(p_1, \ldots, p_n))$, then n clauses are created: $H \Rightarrow \mathrm{att}(p_i)$ for each $i \in \{1, \ldots, n\}$.

The same decomposition also applies to lists of fixed length. We extend the decomposition of tuples to *list*: we replace $\bigwedge_{(i_1,\ldots,i_h)\in J} \mathrm{att}(list(i \leq L, p^G))$ with $\bigwedge_{(i_1,\ldots,i_h,i)\in J\times[1,L]} \mathrm{att}(p^G)$ in hypotheses of clauses, and $\mathrm{att}(list(i \leq L, p^G))$ with $\mathrm{att}(p^G)$ in conclusions (i becomes a free

index). In fact, the attacker has the list $list(i \leq L, p^G)$ if and only if it has all components of the list, that is, $\bigwedge_{i\in[1,L]} \text{att}(p^G)$.

7.3.4 Secrecy Assumptions

The user can give to ProVerif the information that some clause term p remains secret; then, ProVerif removes all clauses that have $\text{att}(p)$ in their hypotheses, since $\text{att}(p)$ should not be derivable. In the end, ProVerif verifies that $\text{att}(p)$ indeed is not derivable, so that the user cannot obtain proofs of insecure protocols by giving incorrect secrecy assumptions.

7.3.5 Elimination of Tautologies

In ProVerif, when a clause is a tautology, that is the conclusion is already in the hypotheses, we remove that clause. We extend the elimination of tautologies to the new facts: we remove a clause when the conclusion is $pred(p_1^G, \ldots, p_l^G)$ and the hypotheses contain a fact $\bigwedge_{(i_1,\ldots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} pred(p_1'^G, \ldots, p_l'^G)$ such that $p_1^G = \rho p_1'^G, \ldots, p_l^G = \rho p_l'^G$ for some mapping ρ from the indices $i_1, \ldots, i_h$ to index terms.

7.3.6 Elimination of Duplicate Hypotheses

When a fact occurs several times in the hypotheses of a clause modulo renaming of bound indices, the duplicates of the fact are removed, so that only one occurrence remains.

7.3.7 Elimination of Hypotheses $\text{att}(x)$

In ProVerif, when a clause contains in the hypotheses a fact $\text{att}(x)$, where x is a variable that does not occur anywhere else in the clause, then the hypothesis $\text{att}(x)$ is removed. We extend this transformation to generalized Horn clauses as follows: when a clause contains in the hypotheses a fact $C\,\text{att}(x_{\iota_1,\ldots,\iota_k})$ such that $x_{\iota'_1,\ldots,\iota'_k}$ does not occur anywhere else in the clause, then we remove $C\,\text{att}(x_{\iota_1,\ldots,\iota_k})$. Indeed, by the clause $\text{att}(a)$, the adversary has at least one term, so $C\,\text{att}(x_{\iota_1,\ldots,\iota_k})$ is always derivable.

7.3.8 Soundess of the Transformations

This section shows the soundness of the new simplifications presented in Sections 7.3.1 and 7.3.2. Let $\textsc{simp}(\Gamma \vdash R^G)$ be the set of well-typed generalized Horn clauses obtained by simplifying R^G as described in Sections 7.3.1 and 7.3.2. This function is naturally extended to sets of clauses. The following theorem shows the soundness of SIMP.

Theorem 20. *Let $\Gamma \vdash R^G$ be a well-typed generalized Horn clause. For all environments T for $\Gamma \vdash R^G$, if R^{GT} is defined, then there exist a clause $\Gamma' \vdash R'^G \in \textsc{simp}(\Gamma \vdash R^G)$ and an environment T' for $\Gamma' \vdash R'^G$ such that $R'^{GT'} \sqsupseteq R^{GT}$.*

Theorem 20 is an immediate consequence of the two following lemmas. The proofs of these lemmas are detailed in Appendix A.2.3.

Lemma 21. *Let $\Gamma \vdash R^G = \mathit{Cts}, H^G \wedge \mathcal{E} \Rightarrow C^G$ be a well-typed generalized Horn clause. For all environments T for $\Gamma \vdash R^G$, if R^{GT} is defined, then there exist a clause $\Gamma' \vdash R'^G = \mathit{Cts}', H'^G \wedge \mathcal{E}' \Rightarrow C'^G$ obtained after applying one step of the unification algorithm and an environment T' for $\Gamma' \vdash R'^G$ such that $R'^{GT'} \sqsupseteq R^{GT}$.*

Lemma 22. *Let $\Gamma \vdash R^G = \mathit{Cts}, H^G \wedge \mathcal{E} \Rightarrow C^G$ be a well-typed generalized Horn clause. For all environments T for $\Gamma \vdash R^G$, if R^{GT} is defined, then there exist a clause $\Gamma \vdash R'^G = \mathit{Cts}', H'^G \wedge \mathcal{E}' \Rightarrow C'^G$ obtained after applying the merging of sets and an environment T' for $\Gamma \vdash R'^G$ such that $R'^{GT'} =^\alpha R^{GT}$, where $=^\alpha$ stands for equality modulo renaming of variables.*

Chapter 8

Extension of the Resolution Algorithm

Contents

In this chapter, we use the new definitions of subsumption, hyperresolution and simplification algorithm to build a new resolution algorithm for generalized Horn clauses. First, we adapt the notion of selection function to generalized Horn clauses and we give the selection function we choose. Then, we illustrate the new resolution algorithm and prove its correctness. Finally we apply this result to our running example.

8.1 The New Algorithm

The resolution algorithm for generalized Horn clauses simulates the one for standard Horn clauses. We need to define a generalized selection function sel^G.

Definition 14 (Generalized selection function)**.** *A generalized selection function is a function from generalized Horn clauses to sets of facts, such that* $sel^G(Cts,\ H^G \wedge \mathcal{E} \Rightarrow C^G) \subseteq H^G$. *If* $F^G \in sel^G(R^G)$, *we say that* F^G *is selected in* R^G. *If* $sel^G(R^G) = \emptyset$, *we say that no hypothesis is selected in* R^G, *or that the conclusion of* R^G *is selected.*

Similarly to the case of standard Horn clauses, a good generalized selection function does not select m-event(p^G) nor att($x_{\iota_1,\ldots,\iota_h}$). Furthermore, resolution is much simpler for facts att(p^G) without conjunction than for facts with conjunction, so we select a fact without

$\textsc{saturate}^G(\mathcal{R}_0^G) =$

1. $\mathcal{R}^G \leftarrow \textsc{elim}^G(\textsc{simp}(\mathcal{R}_0^G))$.
2. Repeat until a fixpoint is reached:
   for each $\Gamma \vdash R^G \in \mathcal{R}^G$ such that $sel^G(R^G) = \emptyset$,
   for each $\Gamma' \vdash R'^G \in \mathcal{R}^G$, for each $F_0^G \in sel^G(R'^G)$,
   $\mathcal{R}^G \leftarrow \textsc{elim}^G(\textsc{simp}(\{\Gamma_{\text{Full}} \vdash R^G \circ^{\text{Full}}_{F_0^G} R'^G,$
   $\Gamma_{\text{Part}} \vdash R^G \circ^{\text{Part}}_{F_0^G} R'^G\}) \cup \mathcal{R}^G)$.
3. Return $\{\Gamma \vdash R^G \in \mathcal{R}^G \mid sel^G(R^G) = \emptyset\}$.

Figure 8.1: New Resolution Algorithm

conjunction when possible. Hence, we use the following selection function:

$$sel^G(Cts,\, H^G \wedge \mathcal{E} \Rightarrow C^G) = \begin{cases} \{\text{att}(p^G)\} & \text{if att}(p^G) \in H^G \text{ for some non-variable } p^G \\ \emptyset & \text{if the first case does not apply and} \\ & C^G = \text{att}(p^G) \text{ for some non-variable } p^G \\ \{C\,\text{att}(p^G)\} & \text{if the previous cases do not apply and} \\ & C\,\text{att}(p^G) \in H^G \text{ for some non-variable } p^G \\ \emptyset & \text{otherwise} \end{cases}$$

The resolution algorithm is shown in Figure 8.1. It mimics the algorithm of Figure 3.7: $\textsc{saturate}^G(\mathcal{R}_0^G)$ first inserts in $\mathcal{R}^G$ the clauses in $\mathcal{R}_0^G$ after elimination of subsumed clauses by $\textsc{elim}^G$: when $\Gamma' \vdash R'^G$ subsumes $\Gamma \vdash R^G$ and both $\Gamma \vdash R^G$ and $\Gamma' \vdash R'^G$ are in $\mathcal{R}^G$, $\Gamma \vdash R^G$ is removed by $\textsc{elim}^G(\mathcal{R}^G)$. Next, the resolution algorithm performs hyperresolution until a fixpoint is reached. Finally, it returns the clauses in $\mathcal{R}^G$ with no selected hypothesis.

8.2 Soundness of the Resolution Algorithm

Next, we show the soundness of our algorithm. Let $\mathcal{F}_{\text{me}}$ be any set of facts of the form m-event(p).

Theorem 23. *Let F be a well-typed closed fact and $\mathcal{R}_0^G$ a set of well-typed generalized Horn clauses. If F is derivable from $\mathcal{R}_0^{G\mathcal{T}} \cup \mathcal{F}_{\text{me}}$, then F is derivable from*

$$(\textsc{saturate}^G(\mathcal{R}_0^G))^{\mathcal{T}} \cup \mathcal{F}_{\text{me}}.$$

Its proof is adapted from the proof of SATURATE (Theorem 3) and uses the soundness theorems for resolution, simplification, and subsumption. We detail the proof of Theorem 23 below.

Proof of Theorem 23. Let $\mathcal{R}_1^G$ be the value of $\mathcal{R}^G$ after the first two steps of $\textsc{saturate}^G(\mathcal{R}_0^G)$, so that

$$\textsc{saturate}^G(\mathcal{R}_0^G) = \{\Gamma \vdash R^G \in \mathcal{R}_1^G \mid sel^G(R^G) = \emptyset\}.$$

For each $R \in \mathcal{R}_1^{G\mathcal{T}}$, there exist $\Gamma \vdash R^G \in \mathcal{R}_1^G$ and an environment T for $\Gamma \vdash R^G$ such that $R = R^{GT}$. We choose one such T and R^G for each R, and define the selection function by

$$sel(R) = \textsc{mgu}\,(\mathcal{E}^T)(sel^G(R^G))^T$$

where $R^G = Cts, H^G \wedge \mathcal{E} \Rightarrow C^G$.

We show that:

1. For all $R \in \mathcal{R}_0^{G\mathcal{T}}$, R is subsumed by a clause in $\mathcal{R}_1^{G\mathcal{T}}$;
2. Let $R, R' \in \mathcal{R}_1^{G\mathcal{T}}$. Assume that $sel(R) = \emptyset$ and there exists $F_0 \in sel(R')$ such that $R \circ_{F_0} R'$ is defined. In this case, $R \circ_{F_0} R'$ is subsumed by a clause in $\mathcal{R}_1^{G\mathcal{T}}$.

To prove the first property, we show that, at each step during $\textsc{saturate}^G$, for all $R \in \mathcal{R}_0^{G\mathcal{T}}$, R is subsumed by a clause in $\mathcal{R}^{G\mathcal{T}}$.

We first show that, if R is subsumed by a clause in $\mathcal{R}^{G\mathcal{T}}$, then R is subsumed by a clause in $(\textsc{elim}^G(\mathcal{R}^G))^{\mathcal{T}}$. Suppose that R is subsumed by a clause R' in $\mathcal{R}^{G\mathcal{T}}$. So there exist $\Gamma' \vdash R'^G \in \mathcal{R}^G$ and an environment T' for $\Gamma' \vdash R'^G$ such that $R'^{GT'} = R'$. If $\Gamma' \vdash R'^G$ is eliminated by $\textsc{elim}^G(\mathcal{R}^G)$, then there exists a well-typed clause $\Gamma \vdash R^G \in \textsc{elim}^G(\mathcal{R}^G)$ such that $\Gamma \vdash R^G \sqsupseteq \Gamma' \vdash R'^G$. By Theorem 13, there exists an environment T for $\Gamma \vdash R^G$ such that $R^{GT} \sqsupseteq R'^{GT'} = R' \sqsupseteq R$, and $R^{GT} \in (\textsc{elim}^G(\mathcal{R}^G))^{\mathcal{T}}$. Hence R is subsumed by a clause in $(\textsc{elim}^G(\mathcal{R}^G))^{\mathcal{T}}$.

For all $R \in \mathcal{R}_0^{G\mathcal{T}}$, by Theorem 20, R is subsumed by a clause in $\textsc{simp}(\mathcal{R}_0^G)^{\mathcal{T}}$, so R is subsumed by a clause in $\mathcal{R}^{G\mathcal{T}}$, for $\mathcal{R}^G = \textsc{elim}^G(\textsc{simp}(\mathcal{R}_0^G))$.

Suppose that for all $R \in \mathcal{R}_0^{G\mathcal{T}}$, R is subsumed by a clause in $\mathcal{R}^{G\mathcal{T}}$. Then, a fortiori, R is subsumed by a clause in $(\{R^G \circ^{\text{Full}}_{F_0^G} R'^G, R^G \circ^{\text{Part}}_{F_0^G} R'^G\} \cup \mathcal{R}^G)^{\mathcal{T}}$, so by Theorem 20, R is subsumed by a clause in $(\textsc{simp}(\{R^G \circ^{\text{Full}}_{F_0^G} R'^G, R^G \circ^{\text{Part}}_{F_0^G} R'^G\} \cup \mathcal{R}^G))^{\mathcal{T}}$, so R is subsumed by a clause in $(\textsc{elim}^G(\textsc{simp}(\{R^G \circ^{\text{Full}}_{F_0^G} R'^G, R^G \circ^{\text{Part}}_{F_0^G} R'^G\} \cup \mathcal{R}^G)))^{\mathcal{T}}$.

So we have the invariant that, at each step during $\textsc{saturate}^G$, for all $R \in \mathcal{R}_0^{G\mathcal{T}}$, R is subsumed by a clause in $\mathcal{R}^{G\mathcal{T}}$. Therefore, the first property holds.

To prove the second property, we rely on the fact that the fixpoint is reached at the end of $\textsc{saturate}^G$. Suppose that $R, R' \in \mathcal{R}_1^{G\mathcal{T}}$, $sel(R) = \emptyset$, and $F_0 \in sel(R')$ is such that $R \circ_{F_0} R'$ is defined. Then there exist $\Gamma \vdash R^G \in \mathcal{R}_1^G$ and an environment T for $\Gamma \vdash R^G$ such that $R = R^{GT}$ and $sel^G(R^G) = \emptyset$, by definition of sel. Similarly, there exist $\Gamma' \vdash R'^G \in \mathcal{R}_1^G$ and an environment T' for $\Gamma' \vdash R'^G$ such that $R' = R'^{GT'}$ and $F_0 \in \textsc{mgu}\,(\mathcal{E}'^{T'})(sel^G(R'^G))^{T'}$, where $R'^G = Cts', H'^G \wedge \mathcal{E}' \Rightarrow C'^G$. Hence, there exists $F_0^G \in sel^G(R'^G)$ such that $F_0 \in \textsc{mgu}\,(\mathcal{E}'^{T'})F_0^{GT'}$.

By Theorem 17, there exists an environment T_1 for $\Gamma_{\text{Full}} \vdash R^G \circ^{\text{Full}}_{F_0^G} R'^G$ or $\Gamma_{\text{Part}} \vdash R^G \circ^{\text{Part}}_{F_0^G} R'^G$ such that $(R^G \circ^{\text{Full}}_{F_0^G} R'^G)^{T_1}$ or $(R^G \circ^{\text{Part}}_{F_0^G} R'^G)^{T_1}$ is equal to $R \circ_{F_0} R'$ up to renaming of variables. That is, we have a clause $\Gamma_1 \vdash R_1^G$, equal to $\Gamma_{\text{Full}} \vdash R^G \circ^{\text{Full}}_{F_0^G} R'^G$ or $\Gamma_{\text{Part}} \vdash R^G \circ^{\text{Part}}_{F_0^G} R'^G$, such that $R_1^{GT_1} =^\alpha R \circ_{F_0} R'$. By Theorem 20, there exist a clause $\Gamma_1' \vdash R_1'^G \in \textsc{simp}(\Gamma_1 \vdash R_1^G)$ and an environment T_1' for $\Gamma_1' \vdash R_1'^G$ such that $R_1'^{GT_1'} \sqsupseteq R_1^{GT_1}$. Since the fixpoint is reached in $\textsc{saturate}^G$, the clauses in $\textsc{simp}(\{\Gamma_{\text{Full}} \vdash R^G \circ^{\text{Full}}_{F_0^G} R'^G, \Gamma_{\text{Part}} \vdash R^G \circ^{\text{Part}}_{F_0^G} R'^G\})$ are subsumed (as generalized Horn clauses) by clauses in $\mathcal{R}_1^G$. Hence, the clause $\Gamma_1' \vdash R_1'^G$ is subsumed by a clause in $\mathcal{R}_1^G$, so there exists a clause $\Gamma_2 \vdash R_2^G \in \mathcal{R}_1^G$ such that $\Gamma_2 \vdash R_2^G \sqsupseteq \Gamma_1' \vdash R_1'^G$. By Theorem 13, there exists an environment T_2 for $\Gamma_2 \vdash R_2^G$ such that $R_2^{GT_2} \sqsupseteq R_1'^{GT_1'} \sqsupseteq R_1^{GT_1} =^\alpha R \circ_{F_0} R'$. Hence, $R \circ_{F_0} R'$ is subsumed by a clause in $\mathcal{R}_1^{G\mathcal{T}}$, namely $R_2^{GT_2}$. This concludes the proof of the second property.

Therefore, we have proved that $\mathcal{R}_1^{G\mathcal{T}}$ satisfies the properties of Lemma 5 of [16]. Following the proof of Lemma 1 in [16], we can show that, if F is derivable from $\mathcal{R}_0^{G\mathcal{T}} \cup \mathcal{F}_{\text{me}}$, then F is derivable from $\mathcal{R}_2 \cup \mathcal{F}_{\text{me}}$, where $\mathcal{R}_2 = \{R \in \mathcal{R}_1^{G\mathcal{T}} \mid sel(R) = \emptyset\}$. (The proof of [16] does not consider m-event facts. It can easily be extended to such facts. A proof with m-event facts can also be found in the long version of [15] available at `http://arxiv.org/pdf/0802.3444v1.pdf`; however, the latter proof is more complicated because it considers additional simplifications.) If $R \in \mathcal{R}_2$, then there exist $\Gamma \vdash R^G \in \mathcal{R}_1^G$ and an environment T for $\Gamma \vdash R^G$ such that $R = R^{GT}$ and $sel^G(R^G) = \emptyset$. So $\Gamma \vdash R^G \in \{\Gamma \vdash R^G \in \mathcal{R}_1^G \mid sel^G(R^G) = \emptyset\} = \textsc{saturate}^G(\mathcal{R}_0^G)$, so $R \in (\textsc{saturate}^G(\mathcal{R}_0^G))^{\mathcal{T}}$. Therefore, $\mathcal{R}_2 \subseteq (\textsc{saturate}^G(\mathcal{R}_0^G))^{\mathcal{T}}$, hence F is derivable from $(\textsc{saturate}^G(\mathcal{R}_0^G))^{\mathcal{T}} \cup \mathcal{F}_{\text{me}}$. □

To prove that a closed fact $\text{att}(p^{GT})$ is not derivable from $\mathcal{R}_1^{G\mathcal{T}} \cup \mathcal{F}_{\text{me}}$, we use the following result, where att' is a new predicate:

Corollary 24. *If* $\textsc{saturate}^G(\mathcal{R}_1^G \cup \{\text{att}(p^G) \Rightarrow \text{att}'(p^G)\})$ *contains no clause of the form* $\Gamma \vdash Cts, H^G \wedge \mathcal{E} \Rightarrow \text{att}'(p'^G)$*, then, for all environments* T*,* $\text{att}(p^{GT})$ *is not derivable from* $\mathcal{R}_1^{G\mathcal{T}} \cup \mathcal{F}_{\text{me}}$.

This corollary is proved like Corollary 4, and makes it possible to prove that the protocol preserves the secrecy of p^{GT}, for lists of any length.

Corollary 25. *Suppose that all clauses of* $\textsc{saturate}^G(\mathcal{R}_1^G)$ *that conclude* $\text{event}(e(p^G))$ *for some* p^G *are of the form* $\Gamma \vdash Cts, \bigwedge_{(i_1,\ldots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} \text{m-event}(e'(p'^G)) \wedge H^G \wedge \mathcal{E} \Rightarrow \text{event}(e(\rho p'^G))$ *for some* Γ*,* Cts*,* $i_1, \ldots, i_h$*,* $L_1, \ldots, L_h$*,* H^G*,* $\mathcal{E}$*,* p'^G*, and some substitution* ρ *that maps indices* $i_1, \ldots, i_h$ *to index terms. Then, for all* $\mathcal{F}_{\text{me}}$*, for all* p*, if* $\text{event}(e(p))$ *is derivable from* $\mathcal{R}_1^{G\mathcal{T}} \cup \mathcal{F}_{\text{me}}$*, then* $\text{m-event}(e'(p)) \in \mathcal{F}_{\text{me}}$.

This corollary makes it possible to prove that the protocol satisfies the correspondence "if $e(x)$ has been executed, then $e'(x)$ has been executed".

8.3 Experimental Results

Our algorithm cannot prove authentication for our running example. Indeed, the result of $\textsc{saturate}^G$ contains a clause of the form

$$\text{att}(r) \wedge \text{m-event}(b(\mathsf{Req})) \wedge \mathcal{E} \Rightarrow \text{event}(e(r))$$

where r does not occur in $\mathcal{E}$. This clause does not satisfy Corollary 25: the event $e(r)$ may be executed for any r that the adversary has, even though only $b(\mathsf{Req})$ has been executed. This clause corresponds to the wrapping attack described in Section 6.1.1.

In contrast, our algorithm proves authentication for the corrected version of our running example. The only clause in the result of $\textsc{saturate}^G$ that concludes $\text{event}(e(p^G))$ is of the form

$$\text{m-event}(e(\mathsf{Req})) \wedge \mathcal{E} \Rightarrow \text{event}(e(\mathsf{Req}))$$

Hence, using Corollary 25, we can conclude that, if event $e(r)$ has been executed, then event $b(r)$ has been executed as well. (In our case, the only possible value of r is the request Req of the client.) Notice that an approach similar to [42] for authentication cannot be applied: when translating the list in clause (6.4) to a list of length one there would not be any wrapping attack possible.

In addition to this protocol, we tested our tool on the XML protocols studied in [12]: password digest, password-based signature, X.509 signature, and firewall-based authentication. These are web services security protocols, whose goal is to authenticate a client to a web service. The first two protocols depend on password-based authentication: a password is shared between the user and the server. In the first protocol, a digest of the password, a nonce, and a timestamp is sent by the client to the server. In the second protocol, the client signs its request using a signature key generated from the password. The X.509 signature protocol uses public-key signatures based on X.509 certificates. Finally, the firewall-based

authentication protocol uses a SOAP-level firewall in addition to the server and the client. The client uses a password-based signature; the firewall verifies this signature, and when this authentication succeeds, adds a new `firewall` header, signed using its X.509 certificate, indicating that it has authenticated the client. For all these protocols, we automatically proved the authentication property proved manually in [12].

Finally, we also modeled the Asokan-Ginzboorg group protocol, introduced in Part 1, and proved the secrecy of the session key. For all these examples, our tool terminates in less than 0.5 s on a MacBook Air 1.4 Ghz. The input files for all these case studies can be found at `http://prosecco.gforge.inria.fr/personal/mpaiola/publications/Thesis.html`.

CHAPTER 9

Representation of protocols with a process calculus

Contents

To complete our automatic verifier for lists of unbounded length, we still need to adapt the process calculus introduced in Section 3.1 to model protocols with lists of unbounded length. In this chapter, we present an extension of the process calculus, give its translation into generalized Horn clauses and prove its soundness.

9.1 Need for a New Process Calculus

In this chapter, we use as a running example the correct version of the protocol introduced in Section 6.1.1. As in Section 6.1.2, we represent an XML message as a pair, containing first component a list of triplets (tag, identifier, corresponding content) and as a second component the content of the body. We would like to model this running example with the process calculus introduced in Section 3.1. However since the length of the header and the

length of the list of references of the signature can be different from a document to another, we encounter several problems.

First, since the receiver of the SOAP envelope accepts messages containing any number of headers, we need lists of variable length in order to model the expected message. We solve this problem by adding a new construct to the syntax of terms: $list(i \leq L, M_i)$ for the list of elements M_i with index i in the set $\{1, \ldots, L\}$.

Suppose now to have the following process $\mathsf{let}\ list(i \leq L, y_i) = x\ \mathsf{in}\ P\ \mathsf{else}\ \mathbf{0}$: we would like to bind y_i $(i \leq L)$ to the element of the list x, without knowing the length of the list in advance. To do this, we introduce a new construct $\mathsf{choose}\ L\ \mathsf{in}\ P$ that chooses non-deterministically a bound L and then executes P.

Hence the beginning of the process P_S, that describes the receiver of the SOAP envelope, will be:

$$
\begin{array}{ll}
P_S := & \mathsf{in}(c, x).\mathsf{choose}\ L\ \mathsf{in} \\
& \mathsf{let}\ (list(j \leq L, (tag_j, id_j, cont_j)), w) = x\ \mathsf{in} \\
& \ldots
\end{array}
$$

Next, the server has to check the signature, before authorizing the request he receives. He has to verify that the list contains a tag tag_k equal to Signature and that $cont_k$ contains a correct signature. In other words, the server has to choose a k and test whether tag_k is equal to Signature and $cont_k$ contains a correct signature. We introduce a new process $\mathsf{choose}\ k \leq L\ \mathsf{in}\ P$ that chooses non-deterministically an index $k \in \{1, \ldots, L\}$ and then executes P. This construct allows us to handle protocols that treat elements of lists non-uniformly: we can in fact perform a look-up in a list.

Hence, we can represent the beginning of the check of the signature as:

$$
\begin{array}{l}
\ldots \\
\mathsf{choose}\ k \leq L\ \mathsf{in} \\
\mathsf{if}\ tag_k = \mathsf{Signature}\ \mathsf{then} \\
\mathsf{let}\ (sinfo, sinfosign) = cont_k\ \mathsf{in} \\
\ldots
\end{array}
$$

We will give the final representation of this protocol with the new process calculus in Section 9.3.3.

Suppose now, that we want to model the Asokan-Ginzboorg protocol, introduced in Section 4.1.1, with the process calculus. Since we have L participants we would like to describe each participant with a process A_i and replicate A_i L times. Moreover we may need to create L identifiers a_i, each for one participant A_i. We solve these two issues by introducing two

new constructs: $\Pi_{i \leq L} P$ and (for all $i \leq L, \nu a_i) P$. The first represents L copies of P running in parallel; the second creates L names $a_1, \ldots, a_L$ and then executes P.

Finally, suppose to apply a destructor $g(r_i, s_i)$ to each element y_i of a list $list(i \leq L, y_i)$. Since L is not fixed we cannot model this destructor application as let $y_1 = g(r_1, s_1)$ in $\ldots$ let $y_L = g(r_L, s_L)$ in P else $Q \ldots$ else Q. Hence we introduce a new destructor application let for all $i_1 \leq L_1, \ldots, i_h \leq L_h, x_{i_1,\ldots,i_h} = g(M_1, \ldots, M_n)$ in P else Q: it tries to evaluate $g(M_1, \ldots, M_n)$ for each $i_1 \in \{1, \ldots, L_1\}, \ldots, i_h \in \{1, \ldots, L_h\}$; if this succeeds, then $x_{i_1,\ldots,i_h}$ is bound to the result and P is executed, else Q is executed. This construct allows us to perform a map on the list: the destructor g is in fact applied to all the elements of the list.

9.2 Reminder

9.2.1 Adding the Internal Choice

Before giving the syntax of the new process calculus for list of unbounded length, we need to extend the process calculus introduced in Section 3.1 with an internal choice operator $P + Q$. This process behaves either as P or as Q, non-deterministically. It will be helpful for defining the extension to lists. The semantics of the internal choice is defined by adding the following rules to the semantics of [15].

$$E, \mathcal{P} \cup \{ P + Q \} \rightarrow E, \mathcal{P} \cup \{ P \} \qquad \text{(Red Choice 1)}$$

$$E, \mathcal{P} \cup \{ P + Q \} \rightarrow E, \mathcal{P} \cup \{ Q \} \qquad \text{(Red Choice 2)}$$

We weaken the definition of suitable renaming, given at the beginning of Section 3.3.2: in fact, in the process $P + Q$ the bound names in P need not be distinct from those in Q. Using the same symbols for both names in P and Q does not cause problems because P and Q cannot be both executed.

The translation of the process $[\![P + Q]\!]\rho H$ is defined as follows:

$$[\![P + Q]\!]\rho H = [\![P]\!]\rho H \cup [\![Q]\!]\rho H.$$

The translation of the choice $P + Q$ is the union of the translation of P and Q, since $P + Q$ behaves either as P or as Q.

We have easily extended the proofs of the results of [15] to the internal choice. The definition of *Label* in Appendix A is extended by $Label(P + Q) = \max(Label(P), Label(Q))$. Then Lemma 5 holds even if bound names are not renamed to be distinct in P and Q inside the

process $P + Q$. The type system of Figure 7 is extended with a rule for the internal choice:

$$\frac{E \vdash P \qquad E \vdash Q}{E \vdash P + Q}.$$

In the proof of Theorem 1, the typability of the adversary and of P'_0 follow immediately by induction as well as subject reduction.

Notice that it is also possible to encode $P + Q$ as $(\nu a)(a\langle a\rangle \mid a(x).P \mid a(x).Q)$ where a and x do not occur in P and Q. This encoding leads to more complex clauses so we preferred defining $P + Q$ as a new construct.

9.2.2 Restriction of Generalized Horn Clauses

The syntax of generalized Horn clauses that we use in this chapter is presented in Figure 9.1. These clauses are simplified with respect to Chapter 6: in that chapter, we considered conjunctions over arbitrary subsets of $[1, L_1] \times \cdots \times [1, L_h]$ and equations on indices. These two features appear during the resolution algorithm on generalized Horn clauses, but are not needed in the initial clauses, so we omit them here.

$\iota ::=$	index terms
$\quad i$	index variable
$\quad \phi(\iota_1, \ldots, \iota_h)$	function application
$p^G ::=$	clause terms
$\quad x_{\iota_1,\ldots,\iota_h}$	variable ($h \geq 0$)
$\quad f(p_1^G, \ldots, p_n^G)$	function application
$\quad a^{L_1,\ldots,L_h}_{\iota_1,\ldots,\iota_h}[p_1^G, \ldots, p_n^G]$	indexed names
$\quad list(i \leq L, p^G)$	list constructor
$C ::= \bigwedge_{(i_1,\ldots i_h)\in[1,L_1]\times\cdots\times[1,L_h]}$	conjunctions
$F^G = C\, pred(p_1^G, \ldots, p_l^G)$	facts
$E ::= C\, p^G \doteq p'^G$	equations
$\mathcal{E} ::= \{E_1, \ldots, E_n\}$	set of equations
$R^G ::= F_1^G \wedge \cdots \wedge F_n^G \wedge \mathcal{E} \Rightarrow pred(p_1^G, \ldots, p_l^G)$	generalized Horn clauses

Figure 9.1: Syntax of generalized Horn clauses

9.3 Generalized Process Calculus

This section formally defines the syntax and the semantics of the new process calculus that we introduce to represent protocols with lists of unbounded length. We will refer to this new process calculus as *generalized process calculus*.

9.3.1 Syntax

$\iota ::=$	index terms
i	index variable
$\phi(\iota_1, \ldots, \iota_h)$	function application
$M^G, N^G ::=$	terms
$x_{\iota_1,\ldots,\iota_h}$	variable ($h \geq 0$)
$f(M_1^G, \ldots, M_n^G)$	function application
a	name
a_ι	indexed name
$list(i \leq L, M^G)$	list constructor
$pat^G :=$	patterns
$x_{i_1,\ldots,i_h}$	variable
$f(pat_1^G, \ldots, pat_n^G)$	data constructor
$list(i \leq L, pat^G)$	list pattern
$P^G, Q^G ::=$	processes
$\mathsf{out}(M^G, N^G).P^G$	output
$\mathsf{in}(M^G, x).P^G$	input
$\mathbf{0}$	nil
$P^G \mid Q^G$	parallel composition
$!P^G$	replication
$\Pi_{i \leq L} P^G$	indexed replication
$(\nu a)P^G$	restriction
$(\mathsf{for\ all}\ i \leq L, \nu a_i)P^G$	restriction
$\mathsf{let\ for\ all}\ i_1 \leq L_1, \ldots, i_h \leq L_h, x_{i_1,\ldots,i_h} = g(M_1^G, \ldots, M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$	destructor application
$\mathsf{let\ for\ all}\ i_1 \leq L_1, \ldots, i_h \leq L_h, pat^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$	pattern matching
$\mathsf{event}(e(M^G)).P^G$	event
$\mathsf{choose}\ L\ \mathsf{in}\ P^G$	bound choice
$\mathsf{choose}\ k \leq L\ \mathsf{in}\ P^G$	index choice
$\mathsf{choose}\ \phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L']\ \mathsf{in}\ P^G$	function choice

Figure 9.2: Syntax of the generalized process calculus

The syntax of the generalized process calculus is described in Figure 9.2. Terms are enriched with several new constructs. Variables may have indices $x_{\iota_1,\ldots,\iota_h}$, and so do names a_ι. We use the construct $list(i \leq L, M^G)$ to represent lists of variable length L. Lists of fixed length are represented by a data constructor $\langle M_1^G, \ldots, M_n^G \rangle$ for each length n.

We use $\widetilde{i}$ to represent a tuple of indices $i_1, \ldots, i_h$, and we use the notation $x_{\widetilde{i}}$ for $x_{i_1,\ldots,i_h}$ and $list(\widetilde{i} \leq \widetilde{L}, M^G)$ for $list(i_1 \leq L_1, list(i_2 \leq L_2, \ldots, list(i_h \leq L_h, M^G)\ldots))$.

Processes are also enriched with new constructs:

- The indexed replication $\Pi_{i \leq L} P^G$ represents L copies of P^G in parallel. It may represent L participants of a group protocol, where L is not fixed.
- The restriction (for all $i \leq L, \nu a_i) P^G$ creates L names $a_1, \ldots, a_L$ and then executes P^G. The names $a_1, \ldots, a_L$ may for instance be a secret key for each member of a group of L participants.
- The destructor application let for all $i_1 \leq L_1, \ldots, i_h \leq L_h, x_{i_1,\ldots,i_h} = g(M_1^G, \ldots, M_n^G)$ in P^G else Q^G tries to evaluate $g(M_1^G, \ldots, M_n^G)$ for each $i_1 \in \{1, \ldots, L_1\}, \ldots, i_h \in \{1, \ldots, L_h\}$; if this succeeds, then $x_{i_1,\ldots,i_h}$ is bound to the result and P^G is executed, else Q^G is executed. In this chapter, we suppose that each destructor has a single rewrite rule.
- The pattern matching let for all $i_1 \leq L_1, \ldots, i_h \leq L_h, pat^G = M^G$ in P^G else Q^G matches M^G with the pattern pat^G for each $i_1 \in \{1, \ldots, L_1\}, \ldots, i_h \in \{1, \ldots, L_h\}$ and executes P^G when the matching succeeds, Q^G otherwise. The pattern pat^G can be a variable $x_{i_1,\ldots,i_h}$, a data constructor application $f(pat_1^G, \ldots, pat_h^G)$, or a list of variable length $list(i \leq L, pat^G)$; the latter pattern is essential to be able to decompose lists without fixing their length, since we do not have destructors to perform this decomposition. When a variable occurs in the pattern pat^G in the process let for all $i_1 \leq L_1, \ldots, i_{h'} \leq L_{h'}, list(i_{h'+1} \leq L_{h'+1}, \ldots list(i_h \leq L_h, pat^G)\ldots) = M^G$ in P^G else Q^G, its indices must be $i_1, \ldots, i_h$. Patterns are linear.
- The bound choice choose L in P^G chooses non-deterministically a bound L and then executes P^G. For example, in the process choose L in let $list(i \leq L, y_i) = x$ in P^G else $\mathbf{0}$, the non-deterministic choice of the bound L allows us to bind y_i ($i \leq L$) to the elements of the list x, without knowing the length of the list in advance.
- The index choice choose $k \leq L$ in P^G chooses non-deterministically an index $k \in \{1, \ldots, L\}$ and then executes P^G. In particular, this construct allows us to perform a lookup in a list. For example, let $list(i \leq L, x_i) = z$ in choose $k \leq L$ in if $f(x_k) = M^G$ then P^G else $\mathbf{0}$ looks for an element x_k of the list z such that $f(x_k) = M^G$.
- The function choice choose $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L']$ in P^G chooses non-deterministically an index function $\phi : \{1, \ldots, L_1\} \times \cdots \times \{1, \ldots, L_h\} \to \{1, \ldots, L\}$ and

then executes P^G. For instance, this construct allows us to verify that the elements of a list are a subset of the elements of another list, by non-deterministically choosing the mapping between the indices of the two lists, as we do in Section 9.3.3.

We will use the notation for all $\widetilde{i} \leq \widetilde{L}$ instead of for all $i_1 \leq L_1, \ldots, i_h \leq L_h$, and simply omit "for all " when $h = 0$. As for the process calculus of Section 3.1, we can encode the process if for all $i_1 \leq L_1, \ldots, i_h \leq L_h, M^G = N^G$ then P^G else Q^G in the generalized process calculus as let $x = equal(list(\widetilde{i} \leq \widetilde{L}, M^G), list(\widetilde{i} \leq \widetilde{L}, N^G))$ in P^G else Q^G, where x does not occur in P^G. The "else" branches can be omitted when they are "else $\mathbf{0}$".

9.3.2 Type System

In this section we define a simple type system for the generalized process calculus, to guarantee that the indices of all variables vary in the appropriate interval. In the type system defined in Figure 9.3, the type environment Γ is a list of type declarations:

- $i : [1, L]$ means that i is of type $[1, L]$, that is, intuitively, the value of index i can vary between 1 and the value of the bound L;
- $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L]$ means that the function ϕ expects as input h indices of types $[1, L_j]$, for $j = 1, \ldots, h$ and computes an index of type $[1, L]$;
- $x_{\_} : [1, L_1] \times \cdots \times [1, L_h]$ means that the variable x expects indices of types $[1, L_1]$, $\ldots, [1, L_h]$; we write $x_{\_} : [\,]$ when x expects no index (that is, $h = 0$);
- $a_{\_} : [1, L]$ means that the name a expects an index of type $[1, L]$, and $a_{\_} : [\,]$ means that the name a expects no index.

The type rules are given in Figure 6.2. The type system defines the following judgments:

- $\Gamma \vdash \iota : [1, L]$, which means that ι has type $[1, L]$ in the type environment Γ;
- $\Gamma \vdash M^G$, $\Gamma \vdash P^G$, which mean that M^G, P^G, respectively, are well-typed in the type environment Γ.
- $i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash pat^G \rightsquigarrow \Gamma$, which means that the pattern pat^G has free indices $i_1, \ldots, i_h$ of types $[1, L_1], \ldots, [1, L_h]$ respectively, and binds the variables in Γ.

Most type rules are straightforward. For instance, the rule (Var) means that $x_{i_1,\ldots,i_h}$ is well-typed when the types expected by x for its indices match the types of $i_1, \ldots, i_h$. The

$$\frac{i : [1, L] \in \Gamma}{\Gamma \vdash i : [1, L]}$$

$$\frac{\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma \qquad \Gamma \vdash \iota_1 : [1, L_1] \ldots \Gamma \vdash \iota_h : [1, L_h]}{\Gamma \vdash \phi(\iota_1, \ldots, \iota_h) : [1, L]}$$

$$\frac{x\_ : [1, L_1] \times \cdots \times [1, L_h] \in \Gamma \qquad \Gamma \vdash \iota_1 : [1, L_1] \ldots \Gamma \vdash \iota_h : [1, L_h]}{\Gamma \vdash x_{\iota_1, \ldots, \iota_h}} \text{(Var)}$$

$$\frac{\Gamma \vdash M_1^G \ldots \Gamma \vdash M_n^G}{\Gamma \vdash f(M_1^G, \ldots, M_n^G)}$$

$$\frac{a\_ : [\,] \in \Gamma}{\Gamma \vdash a} \qquad \frac{a\_ : [1, L] \in \Gamma \qquad \Gamma \vdash \iota : [1, L]}{\Gamma \vdash a_\iota} \qquad \frac{\Gamma, i : [1, L] \vdash M^G}{\Gamma \vdash list(i \le L, M^G)}$$

$$\frac{\Gamma \vdash M^G \qquad \Gamma \vdash N^G \qquad \Gamma \vdash P^G}{\Gamma \vdash \mathsf{out}(M^G, N^G).P^G}$$

$$\frac{\Gamma \vdash M^G \qquad \Gamma, x\_ : [\,] \vdash P^G}{\Gamma \vdash \mathsf{in}(M^G, x).P^G}$$

$$\Gamma \vdash \mathbf{0} \qquad \frac{\Gamma \vdash P^G \qquad \Gamma \vdash Q^G}{\Gamma \vdash P^G \mid Q^G} \qquad \frac{\Gamma \vdash P^G}{\Gamma \vdash !P^G}$$

$$\frac{\Gamma, i : [1, L] \vdash P^G}{\Gamma \vdash \Pi_{i \le L} P^G} \qquad \frac{\Gamma, a\_ : [\,] \vdash P^G}{\Gamma \vdash (\nu a) P^G} \qquad \frac{\Gamma, a\_ : [1, L] \vdash P^G}{\Gamma \vdash (\mathsf{for\ all}\ i \le L, \nu a_i) P^G}$$

$$\frac{\begin{array}{c}\Gamma, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash M_1^G \\ \ldots \\ \Gamma, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash M_n^G\end{array} \qquad \begin{array}{c}\Gamma, x\_ : [1, L_1] \times \cdots \times [1, L_h] \vdash P^G \\ \Gamma \vdash Q^G\end{array}}{\mathsf{let\ for\ all}\ i_1 \le L_1, \ldots, i_h \le L_h, x_{i_1, \ldots, i_h} = g(M_1^G, \ldots M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G}$$

$$i_1 : [1, L_1], \ldots, i_h [1, L_h] \vdash x_{i_1, \ldots, i_h} \rightsquigarrow (x\_ : [1, L_1] \times \cdots \times [1, L_h]) \text{ (PatVar)}$$

$$\frac{\text{for all } j \le n, \text{ we have } i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash pat_j^G \rightsquigarrow \Gamma_j}{i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash f(pat_1^G, \ldots, pat_n^G) \rightsquigarrow \Gamma_1, \ldots, \Gamma_n} \text{(PatData)}$$

$$\frac{i_1 : [1, L_1], \ldots, i_h : [1, L_h], i : [1, L] \vdash pat^G \rightsquigarrow \Gamma}{i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash list(i \le L, pat^G) \rightsquigarrow \Gamma} \text{(PatList)}$$

$$\frac{\begin{array}{c}i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash pat^G \rightsquigarrow \Gamma' \qquad \Gamma, \Gamma' \vdash P^G \\ \Gamma, i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash M^G \qquad \Gamma \vdash Q^G\end{array}}{\mathsf{let\ for\ all}\ i_1 \le L_1, \ldots, i_h \le L_h, pat^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G}$$

$$\frac{\Gamma \vdash M^G \qquad \Gamma \vdash P^G}{\Gamma \vdash \mathsf{event}(e(M^G)).P^G} \qquad \frac{\Gamma \vdash P^G}{\Gamma \vdash \mathsf{choose}\ L\ \mathsf{in}\ P^G}$$

$$\frac{\Gamma, k : [1, L] \vdash P^G}{\Gamma \vdash \mathsf{choose}\ k \le L\ \mathsf{in}\ P^G} \qquad \frac{\Gamma, \phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \ \vdash P^G}{\Gamma \vdash \mathsf{choose}\ \phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L]\ \mathsf{in}\ P^G}$$

Figure 9.3: Type system for the generalized process calculus

rules (PatVar), (PatData), and (PatList) deal with the patterns $x_{i_1,\ldots,i_h}$, $f(pat_1^G, \ldots, pat_n^G)$, and $list(i \leq L, pat^G)$, respectively. They build the type environment that gives types to the variables bound in the pattern.

Notice that the type system for generalized Horn clauses and the one for the generalized process calculus are different. In particular, in type environments for processes we find type declarations for names, since it is expected that the index of a certain name a always has the same type. However, in type environments for generalized Horn clauses we have no information on the type of the indices of a name a. As explained in Section 4.2.3, in generalized Horn clauses we allow the same name a to have indices of different types: the name a exists for indices up to the value of the largest bound.

We have notions of bound indices i, functions ϕ, variables x, names a, and bounds L. For example, the index k is bound in P^G in the process $\mathsf{choose}\ k \leq L\ \mathsf{in}\ P^G$. In the pattern matching $\mathsf{let\ for\ all}\ i_1 \leq L_1, \ldots, i_h \leq L_h, pat^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$, the indices $i_1, \ldots, i_h$ are bound in $pat^G = M^G$, but not in P^G or Q^G. The bound L is bound in P^G in the process $\mathsf{choose}\ L\ \mathsf{in}\ P^G$. A closed process has no free bounds, indices, functions, and variables. It may have free names.

We suppose that all processes are well-typed. A closed process P_0^G is well-typed as follows: $\Gamma_0 \vdash P^G$ where $\Gamma_0 = \{a_\_ : [\,] \mid a \in fn(P^G)\}$.

9.3.3 Example

The representation of our running example in our process calculus is given in Figure 9.4. As in Chapter 6, we represent an XML message as a pair, containing as first component a list of triplets (tag, identifier, corresponding content) and as second component the content of the body. The client process P_C first executes an event $b(\mathsf{Req})$, meaning that he starts the protocol with a request Req. Then he builds his message and sends it on the public channel c. We suppose that the only element signed by the client is the Body. Since the receiver of the SOAP envelope accepts messages containing any number of headers, we need lists of variable length in order to model the expected message. This is why we model a generic XML message as $(list(j \leq L, (tag_j, id_j, cont_j)), w)$, where tag_j, id_j, and $cont_j$ are variables representing tags, identifiers, and contents respectively and w is the variable for the body. Therefore, the server process P_S receives on channel c the document x consisting of $list(j \leq L, (tag_j, id_j, cont_j))$ together with the body w. Then he looks for an element with tag $tag_k = \mathsf{Signature}$ and tries to match the corresponding content $cont_k$ to $(sinfo, sinfosign)$, where $sinfosign$ is the signature of $sinfo$ under the secret key sk_C. He checks that $sinfo$ is a list of references to elements of the message $list(l \leq L', (id_{\phi(l)}, sha1(cont_{\phi(l)})))$, and that in this list, there is an element with tag $tag_{\phi(d)} = \mathsf{Body}$ and with content $cont_{\phi(d)}$ equal to the content of the body w. When all checks succeed, he authorizes the request w, which is

$$
\begin{array}{ll}
P_C := & \mathsf{event}(b(\mathsf{Req})).\mathsf{out}(c, (\langle(\mathsf{Signature}, \mathsf{ids}, (\langle(\mathsf{idb}, sha1(\mathsf{Req}))\rangle, \\
& \quad sign(\langle(\mathsf{idb}, sha1(\mathsf{Req}))\rangle, \mathsf{sk}_C))), (\mathsf{Body}, \mathsf{idb}, \mathsf{Req})\rangle, \mathsf{Req})) \\
P_S := & \mathsf{in}(c, x).\mathsf{choose}\ L\ \mathsf{in} \\
& \mathsf{let}\ (list(j \leq L, (tag_j, id_j, cont_j)), w) = x\ \mathsf{in} \\
& \mathsf{choose}\ k \leq L\ \mathsf{in} \\
& \mathsf{if}\ tag_k = \mathsf{Signature}\ \mathsf{then} \\
& \mathsf{let}\ (sinfo, sinfosign) = cont_k\ \mathsf{in} \\
& \mathsf{let}\ z = checksign(sinfosign, pk_C, sinfo)\ \mathsf{in} \\
& \mathsf{choose}\ L'\ \mathsf{in}\ \mathsf{choose}\ \phi : [1, L'] \to [1, L]\ \mathsf{in} \\
& \mathsf{if}\ sinfo = list(l \leq L', (id_{\phi(l)}, sha1(cont_{\phi(l)})))\ \mathsf{then} \\
& \mathsf{choose}\ d \leq L'\ \mathsf{in} \\
& \mathsf{if}\ tag_{\phi(d)} = \mathsf{Body}\ \mathsf{then}\ \mathsf{if}\ cont_{\phi(d)} = w\ \mathsf{then}\ \mathsf{event}(e(w)) \\
P := & (\nu \mathsf{sk}_C)\mathsf{let}\ pk_C = pk(\mathsf{sk}_C)\ \mathsf{in}\ \mathsf{out}(c, pk_C).(!P_C \mid !P_S)
\end{array}
$$

Figure 9.4: Representation of our running example

modeled by the event $e(w)$. Our goal is to show that, if the server authorizes a request w, then the client has sent this request, that is, if event $e(w)$ is executed, then event $b(w)$ has been executed.

9.3.4 Semantics

We define the semantics of a generalized process by translating it into a corresponding standard process. To define this translation, we need an environment that gives a value to each free bound, index, and index function of the process.

Definition 15. *Given a generalized process $\Gamma \vdash P^G$, an environment T for $\Gamma \vdash P^G$ is a function that associates:*

- *to each bound L free in P^G or that appears in Γ an integer L^T;*
- *to each index i such that $i : [1, L] \in \Gamma$, an index $i^T \in \{1, \ldots, L^T\}$;*
- *to each index function ϕ such that $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma$, a function $\phi^T : \{1, \ldots, L_1^T\} \times \cdots \times \{1, \ldots, L_h^T\} \to \{1, \ldots, L^T\}$.*

In order to abbreviate notations, we write:

- $T[\widetilde{i} \mapsto \widetilde{v}]$ instead of $T[i_1 \mapsto v_1, \ldots, i_h \mapsto v_h]$;
- $T[\widetilde{i} \mapsto \widetilde{1}]$ instead of $T[i_1 \mapsto 1, \ldots, i_h \mapsto 1]$;

- $T[\widetilde{i} \mapsto \widetilde{L^T}]$ instead of $T[i_1 \mapsto L_1^T, \ldots, i_h \mapsto L_h^T]$;
- $\widetilde{v} \leq \widetilde{L^T}$ instead of $\forall j \in \{1, \ldots, h\}, v_j \in \{1, \ldots, L_j^T\}$;
- $\widetilde{i} : \widetilde{L}$ instead of $i_1 : [1, L_1], \ldots, i_h : [1, L_h]$;
- $x_\_ : \widetilde{L}$ instead of $x_\_ : [1, L_1] \times \cdots \times [1, L_h]$;
- $\bigwedge_{\widetilde{i} \in \widetilde{L}}$ instead of $\bigwedge_{(i_1, \ldots, i_h) \in [1, L_1] \times \cdots \times [1, L_h]}$.

Given an environment T for $\Gamma \vdash P^G$, the generalized process P^G is translated into the standard process P^{GT} defined as follows. The translation ι^T of an index term ι is defined exactly as in Section 6.5. The translation M^{GT} of a term M^G is defined as follows:

$$\begin{aligned}
(x_{\iota_1, \ldots, \iota_h})^T &= x_{\iota_1^T, \ldots, \iota_h^T} \\
f(M_1^G, \ldots, M_n^G)^T &= f(M_1^{GT}, \ldots, M_n^{GT}) \\
a^T &= a \\
a_\iota^T &= a_{\iota^T} \\
\mathit{list}(i \leq L, M^G)^T &= \langle M^{GT[i \mapsto 1]}, \ldots, M^{GT[i \mapsto L^T]} \rangle
\end{aligned}$$

The translation of $\mathit{list}(i \leq L, M^G)$ is a list of length L^T. Patterns pat^G are translated exactly in the same way as terms M^G.

Finally the translation of a generalized process is defined as follows and explained below.

- $(\mathsf{out}(M^G, N^G).P^G)^T = \mathsf{out}(M^{GT}, N^{GT}).P^{GT}$.
- $(\mathsf{in}(M^G, x).P^G)^T = \mathsf{in}(M^{GT}, x).P^{GT}$.
- $\mathbf{0}^T = \mathbf{0}$.
- $(P^G \mid Q^G)^T = P^{GT} \mid Q^{GT}$.
- $(!P^G)^T = !P^{GT}$.
- $(\Pi_{i \leq L} P^G)^T = P^{GT[i \mapsto 1]} \mid \cdots \mid P^{GT[i \mapsto L^T]}$.
- $((\nu a)P^G)^T = (\nu a)P^{GT}$.
- $((\mathsf{for\ all}\ i \leq L, \nu a_i)P^G)^T = (\nu a_1^{L^T}) \ldots (\nu a_{L^T}^{L^T})P^{GT}$.
- $(\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \ldots M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G)^T = \mathsf{let}\ E_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ E_l\ \mathsf{in}\ P^T\ \mathsf{else}\ Q^T\ \ldots\ \mathsf{else}\ Q^T$, where $\{E_1, \ldots E_l\} = \{x_{\widetilde{i}}^{T'} = g(M_1^{GT'}, \ldots, M_n^{GT'}) \mid T' = T[\widetilde{i} \mapsto \widetilde{v}], \widetilde{v} \leq \widetilde{L^T}\}$.
- $(\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, pat^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G)^T = \mathsf{let}\ E_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ E_l\ \mathsf{in}\ P^T\ \mathsf{else}\ Q^T\ \ldots\ \mathsf{else}\ Q^T$, where $\{E_1, \ldots E_l\} = \{pat^{GT'} = M^{GT'} \mid T' = T[\widetilde{i} \mapsto \widetilde{v}], \widetilde{v} \leq \widetilde{L^T}\}$.

- $(\mathsf{event}(e(M^G)).P^G)^T = \mathsf{event}(e(M^{GT})).P^{GT}$.
- $(\mathsf{choose}\ L\ \mathsf{in}\ P^G)^T = P^{GT[L\mapsto 1]} + \cdots + P^{GT[L\mapsto n]} + \cdots$.
- $(\mathsf{choose}\ k \le L\ \mathsf{in}\ P^G)^T = P^{GT[k\mapsto 1]} + \cdots + P^{GT[k\mapsto L^T]}$.
- $(\mathsf{choose}\ \phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L']\ \mathsf{in}\ P^G)^T = P^{GT[\phi\mapsto\phi_1]} + \cdots + P^{GT[\phi\mapsto\phi_l]}$, where $\{\phi_1, \ldots, \phi_l\} = \{\phi \mid \phi : \{1, \ldots, L_1^T\} \times \cdots \times \{1, \ldots, L_h^T\} \to \{1, \ldots, L^T\}\}$.

In most cases, a construct of the generalized process calculus is translated into the corresponding construct of the standard process calculus. The translation of $(\mathsf{for\ all}\ i \le L, \nu a_i)P^G$ creates L^T names and then executes P^{GT}. The translation of the process $\mathsf{let}\ \widetilde{i} \le \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \ldots, M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$ computes $g(M_1^G, \ldots, M_h^G)$ and stores it in $x_{\widetilde{i}}$, for all values of the indices $\widetilde{i}$. We define the translation of the pattern matching similarly. The choice processes are translated into a non-deterministic choice between all the values that L, i, or ϕ can assume. The translation of the process $\mathsf{choose}\ L\ \mathsf{in}\ P^G$ is an infinite process: this translation cannot be handled by ProVerif and our work solves this problem. When P^G is a closed process, it can be translated in the empty environment, which we denote by T_0.

9.4 Translation into Generalized Horn Clauses

As for the standard process calculus, we define the translation of the generalized process calculus into generalized Horn clauses, by giving the clauses for the attacker and those for the protocol.

Clauses for the Attacker. The clauses for the attacker are the same as in ProVerif, that is, the clauses $\mathrm{att}(a[\,])$ for each $a \in S$ and the clauses (Rn), (Rf), (Rg), (Rl), (Rs), except that the following two clauses for lists are added:

$$\bigwedge_{i\in[1,M]} \mathrm{att}(x_i) \Rightarrow \mathrm{att}(\mathit{list}(j \le M, x_j)) \qquad \text{(Rf-list)}$$

$$\mathrm{att}(\mathit{list}(j \le M, x_j)) \Rightarrow \mathrm{att}(x_i) \qquad \text{(Rg-list)}$$

and the clauses (Rf) and (Rg) for lists of fixed length $\langle \cdots \rangle$ are removed. (The two clauses above are sufficient for all lists.)

Clauses for the Protocol. The protocol is represented by a closed process P_0^G. To compute the clauses, we assume that the bound names in P_0^G have been renamed so that they are pairwise distinct and distinct from free names of P_0^G.

Next, we instrument the process P_0^G by labeling each replication $!P^G$ with a distinct session identifier s, so that it becomes $!^s P^G$, and labeling each restriction (for all $i \leq L, va_i$) with the clause term that corresponds to the fresh name a_i, $a_{i,i_1,\ldots,i_h}^{L,L_1,\ldots,L_h}[x_1,\ldots,x_n,s_1,\ldots,s_{n'}]$, where $x_1,\ldots,x_n$ are the variables that store the messages received in inputs above (for all $i \leq L, va_i$) in P_0^G, $s_1,\ldots,s_{n'}$ are the session identifiers that label replications above (for all $i \leq L, va_i$) in the instrumentation of P_0^G and $i_1,\ldots,i_h$ and $L_1,\ldots,L_h$ are the indices that label indexed replications above (for all $i \leq L, va_i$) in P_0^G. The construct (va) is instrumented in the same way, so that it becomes $(va : a_{i_1,\ldots,i_h}^{L_1,\ldots,L_h}[x_1,\ldots,x_n,s_1,\ldots,s_{n'}])$. We denote the instrumentation of P_0^G by $\text{instr}^G(P_0^G)$.

The instrumentation of processes is formally defined by induction on the syntax of the processes, as follows.

Definition 16. *Given a generalized process P^G, a list of variables $Vars = x_1,\ldots,x_n$, a list of session identifiers $SessId = s_1,\ldots,s_{n'}$, and a list of indices $\widetilde{i} \leq \widetilde{L}$, we define the instrumented generalized process as follows:*

- $\text{instr}^G(\mathsf{in}(M^G,x).P^G, Vars, SessId, \widetilde{i} \leq \widetilde{L}) =$
  $\mathsf{in}(M^G,x).\text{instr}^G(P^G,(Vars,x),SessId,\widetilde{i} \leq \widetilde{L})$;
- $\text{instr}^G(!P^G, Vars, SessId, \widetilde{i} \leq \widetilde{L}) = !^s\text{instr}^G(P^G, Vars, (SessId,s), \widetilde{i} \leq \widetilde{L})$;
- $\text{instr}^G(\Pi_{i\leq L}P^G, Vars, SessId, \widetilde{i} \leq \widetilde{L}) =$
  $\Pi_{i\leq L}\text{instr}^G(P^G, Vars, SessId, (\widetilde{i}, i \leq \widetilde{L}, L))$;
- $\text{instr}^G((\text{for all } i \leq L, va_i)P^G, Vars, SessId, \widetilde{i} \leq \widetilde{L}) =$
  $(\text{for all } i \leq L, va_i : a_{i,\widetilde{i}}^{L,\widetilde{L}}[Vars, SessId])\text{instr}^G(P, Vars, SessId, \widetilde{i} \leq \widetilde{L})$;
- $\text{instr}^G((va)P^G, Vars, SessId, \widetilde{i} \leq \widetilde{L}) =$
  $(va : a_{\widetilde{i}}^{\widetilde{L}}[Vars, SessId])\text{instr}^G(P^G, Vars, SessId, \widetilde{i} \leq \widetilde{L})$;
- *In all other cases, the same instrumentation is applied recursively on the subprocesses. For instance,* $\text{instr}^G(P^G \mid Q^G, Vars, SessId, \widetilde{i} \leq \widetilde{L}) = \text{instr}^G(P^G, Vars, SessId, \widetilde{i} \leq \widetilde{L}) \mid \text{instr}^G(Q^G, Vars, SessId, \widetilde{i} \leq \widetilde{L})$.

We let $\text{instr}^G(P^G) = \text{instr}^G(P^G, \emptyset, \emptyset, \emptyset \leq \emptyset)$.

The translation $[\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma$ of a well-typed instrumented process $\Gamma_P \vdash P^G$ is a set of clauses, where the environment ρ^G is a mapping that associates each name and variable, possibly with indices, to a clause term, H^G is a sequence of facts $\text{message}(\cdot,\cdot)$ and $\text{m-event}(\cdot)$, $\mathcal{E}$ is a set of equations, and Γ is a type environment for generalized Horn clauses such that:

- $\Gamma \vdash H^G$;

- $\Gamma \vdash \mathcal{E}$;
- $\Gamma_P, \Gamma \vdash \rho^G$: for each mapping $x_{\widetilde{i}} \mapsto p^G$ in ρ^G, if $\Gamma_P \vdash x_\_ : \widetilde{L}$ then $\Gamma, \widetilde{i} : \widetilde{L} \vdash p^G$, for each mapping $a \mapsto p^G$ in ρ^G, then $\Gamma \vdash p^G$, for each mapping $a_i \mapsto p^G$, if $\Gamma_P \vdash a_\_ : [1, L]$ then $\Gamma, i : [1, L] \vdash p^G$, and for each declaration $i : [1, L] \in \Gamma_P$ (resp. $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma_P$) we have $i : [1, L] \in \Gamma$ (resp. $\phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma$).

The mapping ρ^G is then extended into a substitution that maps terms M^G to clause terms $p^G = \rho^G(M^G)$, by replacing each name and variable with the corresponding clause term, as follows:

$$\begin{aligned}
\rho^G(x_{\widetilde{\iota}}) &= p^G\{\widetilde{\iota}/\widetilde{i}\} \text{ if } \rho^G(x_{\widetilde{i}}) = p^G \\
\rho^G(f(M_1^G, \ldots, M_n^G)) &= f(\rho^G(M_1^G), \ldots, \rho^G(M_n^G)) \\
\rho^G(a_\iota) &= p^G\{\iota/i\} \text{ if } \rho^G(a_i) = p^G \\
\rho^G(list(i \le L, M^G)) &= list(i \le L, \rho^G(M^G)) \text{ if } i \notin fi(im(\rho^G))
\end{aligned}$$

The side condition $i \notin fi(im(\rho))$ in the last formula can be guaranteed by renaming i if needed; it avoids the capture of bound indices.

- $[\![\mathsf{out}(M^G, N^G).P^G]\!]\rho^G H^G \mathcal{E}\Gamma =$
  $[\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma \cup \{\Gamma \vdash H^G \wedge \mathcal{E} \Rightarrow \text{message}(\rho^G(M^G), \rho^G(N^G))\}$.
- $[\![\mathsf{in}(M^G, x).P^G]\!]\rho^G H^G \mathcal{E}\Gamma =$
  $[\![P^G]\!](\rho^G[x \mapsto x])(H^G \wedge \text{message}(\rho^G(M^G), x))\mathcal{E}(\Gamma, x_\_ : [\,])$.
- $[\![\mathbf{0}]\!]\rho^G H^G \mathcal{E}\Gamma = \emptyset$.
- $[\![P^G \mid Q^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma \cup [\![Q^G]\!]\rho^G H^G \mathcal{E}\Gamma$.
- $[\![!^s P^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!](\rho^G[s \mapsto s])H^G \mathcal{E}\Gamma$.
- $[\![\Pi_{i \le L} P^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, i : [1, L])$.
- $[\![(\nu a : a_{\widetilde{i}}^{\widetilde{L}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!](\rho^G[a \mapsto a_{\widetilde{i}}^{\widetilde{L}}[\rho^G(x_1), \ldots, \rho^G(x_n), \rho^G(s_1), \ldots, \rho^G(s_{n'})]\,])H^G \mathcal{E}\Gamma$.
- $[\![(\text{for all } i \le L, \nu a_i : a_{i,\widetilde{i}}^{L,\widetilde{L}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^G]\!]\rho^G H^G \mathcal{E}\Gamma =$
  $[\![P^G]\!](\rho^G[a_i \mapsto a_{i,\widetilde{i}}^{L,\widetilde{L}}[\rho^G(x_1), \ldots, \rho^G(x_n), \rho^G(s_1), \ldots, \rho^G(s_{n'})]\,])H^G \mathcal{E}\Gamma$.
- $[\![\text{let for all } \widetilde{i} \le \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \ldots M_n^G) \text{ in } P^G \text{ else } Q^G]\!]\rho^G H^G \mathcal{E}\Gamma =$
  $[\![Q^G]\!]\rho^G H^G \mathcal{E}\Gamma \cup [\![P^G]\!](\rho^G[x_{\widetilde{i}} \mapsto p'^G])H^G(\mathcal{E} \cup \mathcal{E}')\Gamma'$,
  where p'^G, $\mathcal{E}'$, and Γ' are defined as follows. Let $g(p_1, \ldots, p_n) \to p$ be the rewrite rule in $def(g)$. The rewrite rule $g(p_1'^G, \ldots, p_n'^G) \to p'^G$ is obtained from $g(p_1, \ldots, p_n) \to p$ by replacing all variables y of this rule with fresh variables with indices $\widetilde{i}$: $y'_{\widetilde{i}}$. Then

$\mathcal{E}' = \{\bigwedge_{\widetilde{i} \leq \widetilde{L}} p_1'^G \doteq \rho^G(M_1^G), \ldots, \bigwedge_{\widetilde{i} \leq \widetilde{L}} p_n'^G \doteq \rho^G(M_n^G)\}$ and Γ' is Γ extended with $x_{\_} : \widetilde{L}$ and $y'_{\_} : \widetilde{L}$ for each variable $y'_{\widetilde{i}}$ in $p_1'^G, \ldots, p_n'^G, p'^G$.

- $[\![\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, pat^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![Q^G]\!]\rho^G H^G \mathcal{E}\Gamma \cup [\![P^G]\!](\rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'}\ \text{occurs in}\ pat^G])H^G(\mathcal{E} \cup \{\bigwedge_{\widetilde{i} \leq \widetilde{L}} pat^G \doteq \rho^G(M^G)\})\Gamma'$, where Γ' is Γ extended for the variables in pat^G: if $\widetilde{i} : \widetilde{L} \vdash pat^G \rightsquigarrow \Gamma_1$, then $\Gamma' = \Gamma, \Gamma_1$.
- $[\![\mathsf{event}(e(M^G)).P^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!]\rho^G(H^G \wedge \text{m-event}(e(\rho^G(M^G))))\mathcal{E}\Gamma \cup \{\Gamma \vdash H^G \wedge \mathcal{E} \Rightarrow \text{event}(e(\rho^G(M^G)))\}$.
- $[\![\mathsf{choose}\ L\ \mathsf{in}\ P^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma$.
- $[\![\mathsf{choose}\ k \leq L\ \mathsf{in}\ P^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, k : [1, L])$.
- $[\![\mathsf{choose}\ \phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L']\ \mathsf{in}\ P^G]\!]\rho^G H^G \mathcal{E}\Gamma = [\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, \phi : [1, L_1] \times \cdots \times [1, L_h] \to [1, L'])$.

In most cases, the translation is similar to the one of the standard process calculus. The translation of the process $(\mathsf{for\ all}\ i \leq L, \nu a_i : a_{i,\widetilde{i}}^{L,\widetilde{L}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^G$ extends ρ^G to the name a_i for all possible values of $i \in \{1, \ldots, L\}$. The translation of the destructor application $\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, x_{\widetilde{i}} = g(M_1^G, \ldots M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$ is the union of the clauses for the case where the destructor succeeds and for the case where it fails. In particular, when the destructor succeeds, instead of performing unification, we add the equations $\rho(M_i^G) = p_i'^G$ for every $\widetilde{i} \leq \widetilde{L}$ to $\mathcal{E}$ and extend ρ^G to the variable $x_{\widetilde{i}}$. We define the translation of the pattern matching similarly. Finally, the type environment Γ is extended with the chosen index or function in the choice processes and in the indexed replication; this is sufficient since the chosen bound, index, or function can take any value in the generalized Horn clauses.

Summary. Let $\rho_0 = \{a \mapsto a[\,] \mid a \in fn(P_0^G)\}$. The set of generalized Horn clauses corresponding to the closed process P_0^G, well-typed in $\Gamma_0 = \{a_{\_} : [\,] \mid a \in fn(P^G)\}$, is defined as:

$$\mathcal{R}_{P_0^G,S}^G = [\![\text{instr}^G(P_0^G)]\!]\rho_0\emptyset\emptyset\Gamma_0 \cup \{\text{att}(a[\,]) \mid a \in S\} \cup \{(\text{Rn}), (\text{Rf}), (\text{Rg}), (\text{Rl}), (\text{Rs}), (\text{Rf-list}), (\text{Rg-list})\}$$

where S is the set of names initially known by the attacker.

For example, by translating the process P_S of our running example, we obtain the following clause:

$$\begin{aligned}&\text{message}(c, x) \wedge \{s \doteq pk(\mathsf{sk}_C), pk_C \doteq s, (list(j \leq L, (tag_j, id_j, cont_j)), w) \doteq x,\\ &tag_k \doteq \mathsf{Signature}, cont_k \doteq (sinfo, sinfosign), sinfosign \doteq sign(v, y),\\ &sinfo \doteq v, pk_C \doteq pk(y), sinfo \doteq list(l \leq L', (id_{\phi(l)}, sha1(cont_{\phi(l)}))),\\ &tag_{\phi(d)} \doteq \mathsf{Body}, cont_{\phi(d)} \doteq w\} \Rightarrow \text{event}(e(w))\end{aligned}$$

which means that the server process P_S executes event $e(w)$ when it has received a message x that satisfies all the checks. This clause will be simplified by the simplification algorithm presented in Chapter 8.

9.5 Soundness of the Generalized Horn Clauses

In this section, we relate the generalized Horn clauses generated from a closed well-typed generalized process $\Gamma_0 \vdash P_0^G$, to the Horn clauses generated from $P_0^{GT_0}$, to show that our generated Horn clauses are correct. We assume that the bound names in P_0^G have been renamed so that they are pairwise distinct and distinct from free names of P_0^G.

The bound names in $P_0^{GT_0}$ need not be pairwise distinct, so we first need to rename them, before generating the Horn clauses. Hence, we define a function Tren that combines the translation P^{GT} with that renaming of bound names.

Definition 17. *Given a well-typed generalized process $\Gamma \vdash P^G$, an environment T for $\Gamma \vdash P^G$, and a list of indices $\widetilde{i} \leq \widetilde{L}$, let* Tren *be defined by:*

- $\mathrm{Tren}(\Pi_{i \leq L} P^G, T, \widetilde{i} \leq \widetilde{L}) = \mathrm{Tren}(P^G, T[i \mapsto 1], (\widetilde{i}, i) \leq (\widetilde{L}, L)) \mid \cdots \mid \mathrm{Tren}(P^G, T[i \mapsto L^T], (\widetilde{i}, i) \leq (\widetilde{L}, L))$;
- $\mathrm{Tren}((\nu a)P^G, T, \widetilde{i} \leq \widetilde{L}) = (\nu a^{\widetilde{L}^T}_{\widetilde{i}^T})\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})\{a^{\widetilde{L}^T}_{\widetilde{i}^T}/a\}$;
- $\mathrm{Tren}((\text{for all } i \leq L, \nu a_i)P^G, T, \widetilde{i} \leq \widetilde{L}) = (\nu a^{L^T, \widetilde{L}^T}_{1, \widetilde{i}^T}) \ldots (\nu a^{L^T, \widetilde{L}^T}_{L^T, \widetilde{i}^T})\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})\{a^{L^T, \widetilde{L}^T}_{1, \widetilde{i}^T}/a_1, \ldots, a^{L^T, \widetilde{L}^T}_{L^T, \widetilde{i}^T}/a_{L^T}\}$;
- *In all other cases,* $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})$ *is defined like* P^{GT} *except that it recursively calls* $\mathrm{Tren}(P'^G, T, \widetilde{i} \leq \widetilde{L})$ *instead of* P'^{GT} *on the subprocesses. For instance,* $\mathrm{Tren}(\text{choose } k \leq L \text{ in } P^G, T, \widetilde{i} \leq \widetilde{L}) = \mathrm{Tren}(P^G, T[k \mapsto 1], \widetilde{i} \leq \widetilde{L}) + \cdots + \mathrm{Tren}(P^G, T[k \mapsto L^T], \widetilde{i} \leq \widetilde{L})$.

The next lemma shows that $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})$ renames the names of P^{GT} as desired. It is proved in Appendix A.2.4.

Lemma 26. $\mathrm{Tren}(P_0^G, T_0, \emptyset \leq \emptyset)$ *is a suitable renaming of* $P_0^{GT_0}$.

The following theorem shows the soundness of our generalized Horn clauses. Its main idea is summarized in Figure 9.5. We prove it in Section 9.6.

Theorem 27. *Let $\Gamma_0 \vdash P_0^G$ be a closed well-typed generalized process, and S be a set of names. Let $P'_0 = \mathrm{Tren}(P_0^G, T_0, \emptyset \leq \emptyset)$. We have $\mathcal{R}^{G\mathcal{T}}_{P_0^G, S} \supseteq \mathcal{R}_{P'_0, S}$.*

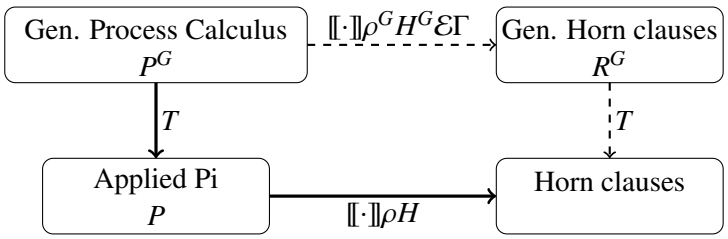

Figure 9.5: Basic idea of Theorem 27

Furthermore, if $\mathcal{R}_1 \sqsupseteq \mathcal{R}_2$ and a fact F is derivable from $\mathcal{R}_1$, then it is also derivable from $\mathcal{R}_2$. So, by Theorems 1, 2, and 27 and Lemma 26, we obtain the following results.

Corollary 28 (Secrecy)**.** *Let M^G be a term. Let p^G be the clause term obtained by replacing names a with $a[\,]$ and names a_i with $a_i[\,]$ in M^G. If for all environments T, $\mathrm{att}(p^{GT})$ is not derivable from $\mathcal{R}^{G\mathcal{T}}_{P_0^G,S} \cup \mathcal{F}_{\mathrm{me}}$ for any $\mathcal{F}_{\mathrm{me}}$, then for all environments T, $P_0^{GT_0}$ preserves the secrecy of M^{GT} from adversaries with initial knowledge S.*

Corollary 29 (Authentication)**.** *Suppose that, for all $\mathcal{F}_{\mathrm{me}}$, for all p, if $\mathrm{event}(e(p))$ is derivable from $\mathcal{R}^{G\mathcal{T}}_{P_0^G,S} \cup \mathcal{F}_{\mathrm{me}}$, then $\text{m-event}(e'(p)) \in \mathcal{F}_{\mathrm{me}}$. Then $P_0^{GT_0}$ satisfies the correspondence "if $e(x)$ has been executed, then $e'(x)$ has been executed" against adversaries with initial knowledge S.*

The hypotheses of these two corollaries are precisely those that can be proved using the resolution algorithm we proposed in Chapter 8, as shown by Corollaries 24 and 25. So by combining Corollaries 28 and 29 with Corollaries 24 and 25, we can prove secrecy and authentication for protocols that use lists of any length.

For example, after translating our running example into generalized Horn clauses, we can run the tool described in Chapter 8 and obtain that the hypothesis of Corollary 29 holds for events e and b. Therefore, by Corollary 29, the process of Section 9.3.3 satisfies the desired correspondence: if $e(x)$ is executed, then $b(x)$ has been executed.

9.6 Proof of Theorem 27

Theorem 27 comes from the combination of two different results. The first result (Lemma 30) shows that the translation from generalized processes to processes commutes with the instrumentation (provided the translation is suitably renamed using Tren). The second result (Lemma 31) shows the soundness of the translation from instrumented processes to generalized Horn clauses. The proofs of these lemmas are detailed in Appendix A.2.5.

Before stating the first result, we define the translation P^{GT} on instrumented processes. It is defined similarly to the translation on non-instrumented processes; the cases that differ are as follows:

- $(!^s P^G)^T = !^s P^{GT}$
- $((\nu a : a'\widetilde{\underset{i}{L}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^G)^T = (\nu a : a'\widetilde{\underset{i^T}{L^T}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^{GT}$
- $((\text{for all } i \leq L, \nu a_i : a'\widetilde{\underset{i,i}{L,L}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^G)^T = (\nu a_1 : a'\widetilde{\underset{1,i^T}{L^T,L^T}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}]) \ldots (\nu a_{L_T} : a'\widetilde{\underset{L^T,i^T}{L^T,L^T}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^{GT}$

We write $P \equiv_\alpha Q$ when the process P is equal to Q up to renaming of bound names: in an instrumented process $(\nu a : a'[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P$, the name a can be renamed, but the function symbol a' remains unchanged. This is why we may end up with instrumented processes in which the name a is different from the function symbol a'.

Lemma 30. *Given a well-typed generalized process $\Gamma_0 \vdash P_0^G$, we have:*

$$(\text{instr}^G(P_0^G))^{T_0} \equiv_\alpha \text{instr}(\text{Tren}(P_0^G, T_0, \emptyset \leq \emptyset)) .$$

We write $T' ext\, T$ to mean that T' is an extension of the environment T. Given a type environment Γ_P for processes and a type environment Γ for generalized Horn clauses, we define $\{x_{\widetilde{i}} \mapsto p^G\}^T = \{x_{\widetilde{v}} \mapsto p^{GT[\widetilde{i} \mapsto \widetilde{v}]} \mid \widetilde{v} \leq \widetilde{L}\}$ when $x_\_ : \widetilde{L} \in \Gamma$, $\{a \mapsto p^G\}^T = \{a \mapsto p^{GT}\}$, and $\{a_i \mapsto p^G\}^T = \{a_v \mapsto p^{GT[i \mapsto v]} \mid v \leq L\}$ when $a_\_ : [1, L] \in \Gamma_P$. We extend this definition naturally to ρ^{GT}.

Lemma 31. *Let $\Gamma_P \vdash P^G$ be a well-typed instrumented generalized process, ρ^G a function that associates a clause term with each name and variable, possibly with indices, H^G a sequence of facts, $\mathcal{E}$ a set of equations, and Γ is an environment for generalized Horn clauses such that:*

- $\Gamma \vdash H^G$;
- $\Gamma \vdash \mathcal{E}$;
- $\Gamma_P, \Gamma \vdash \rho^G$.

Then

$$[\![P^{GT}]\!](\text{MGU}\,(\mathcal{E}^T)\rho^{GT})(\text{MGU}\,(\mathcal{E}^T)H^{GT}) \sqsubseteq \bigcup_{T' ext\, T} ([\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}$$

and the clauses in the right hand side are well-typed.

From the previous results, we easily obtain Theorem 27.

Proof of Theorem 27. By Lemma 31, $([\![P_1^G]\!]\rho_0\emptyset\emptyset\Gamma_0)^{\mathcal{T}} = \bigcup_T([\![P_1^G]\!]\rho_0\emptyset\emptyset\Gamma_0)^T \sqsupseteq [\![P_1^{GT_0}]\!]\rho_0\emptyset$, where $P_1^G = \mathrm{instr}^G(P_0^G)$. By Lemma 30, $\mathrm{instr}(P'_0) = \mathrm{instr}(\mathrm{Tren}(P_0^G, T_0, \emptyset \leq \emptyset)) \equiv_\alpha \mathrm{instr}^G(P_0^G)^{T_0} = P_1^{GT_0}$, so we have $([\![\mathrm{instr}^G(P_0^G)]\!]\rho_0\emptyset\emptyset\Gamma_0)^{\mathcal{T}} \sqsupseteq [\![\mathrm{instr}(P'_0)]\!]\rho_0\emptyset$ since the translation to Horn clauses $[\![\cdot]\!]$ is invariant by renaming of bound names.

Moreover, for each clause R in $\{\mathrm{att}(a[\,]) \mid a \in S\} \cup \{(\mathrm{Rn}), (\mathrm{Rf}), (\mathrm{Rg}), (\mathrm{Rl}), (\mathrm{Rs})\}$ except the clauses (Rf) and (Rg) for lists of fixed length, R is also a generalized Horn clause and we have $\{R\}^{\mathcal{T}} = \{R\}$. The clauses (Rf) for lists of fixed length are in $\{R^G\}^{\mathcal{T}} = \{\mathrm{att}(x_1) \wedge \cdots \wedge \mathrm{att}(x_n) \Rightarrow \mathrm{att}(\langle x_1, \ldots, x_n\rangle) \mid n \in \mathbb{N}\}$, where $R^G =$ (Rf-list). The clauses (Rg) for lists of fixed length are in $\{R^G\}^{\mathcal{T}} = \{\mathrm{att}(\langle x_1, \ldots, x_n\rangle) \Rightarrow \mathrm{att}(x_v) \mid n \in \mathbb{N}, v \leq n\}$ where $R^G =$ (Rg-list).

So we obtain $\mathcal{R}^{G\mathcal{T}}_{P_0^G,S} \sqsupseteq \mathcal{R}_{P'_0,S}$. □

Conclusions

This thesis provides contributions to the field of automatic verification of security protocols. In particular it focuses on finding automatic techniques for verifying secrecy and authentication properties for protocols that manipulate lists of unbounded lengths.

In the first part of the thesis we have proposed a technique for verifying security properties for protocols that manipulate lists elements in a uniform way. We have proposed a new type of clauses, generalized Horn clauses, useful to represent protocols that manipulate lists of unbounded length, as well as group protocols with an unbounded number of participants. We have shown that, for a subclass of generalized Horn clauses, if secrecy is proved by the Horn clause technique for lists of length one, then we have secrecy for lists of any length. We have also provided an approximation algorithm that transforms a set of generalized Horn clauses for satisfying the hypothesis of our main theorem. Using these results, one can prove secrecy for lists of any length for some group protocols, as we did for the Asokan-Ginzboorg protocol.

The main limitation of this approach is that all elements of lists must be treated uniformly. By relaxing the condition on a uniform treatment of lists, we have developed a second technique, illustrated in Part 2 of this thesis. In order to support equality tests on certain components of lists, we have further extended generalized Horn clauses. We have adapted the definitions previously introduced for Horn clauses to these new clauses. We have thus obtained a new algorithm for verifying secrecy and authentication properties for protocols with lists, which we have proved correct and implemented. We have successfully tested our tool on several XML protocols.

Since manually modeling protocols with generalized Horn clauses is delicate and error-prone, we provided a more convenient input language for protocols. More precisely, we have proposed a new process calculus, useful to represent protocols that manipulate lists of unbounded length. We have defined its semantics and provided an automatic translation from this calculus into generalized Horn clauses. We have proved that this translation is sound. One can represent a protocol as a process from the generalized process calculus, translate the process into generalized Horn clauses and finally run the resolution algorithm on the set of clauses. In this way, we obtain an automatic technique for proving secrecy and

authentication properties of protocols that manipulate unbounded lists, for an unbounded number of sessions, represented in a process calculus.

These two techniques are a first step for proving protocols with lists of unbounded length for an unbounded number of sessions and any number of protocol participants. However there are still a few limitations. At first, we do not support cryptographic primitives such as Diffie-Hellman key agreements that are modeled using equational theories. This is why we could not apply our approaches to more examples on group protocols. We plan to overcome these limitations in the future. Indeed, some group protocols (e.g. A.GDH-2) use the Diffie-Hellman key agreement, which we cannot handle yet. We believe that it could be handled by combining our first result with [31]. Moreover it could also be interesting to adapt the second result of this thesis to analyze protocols with algebraic properties.

Concerning the technique proposed in Part 2, since one of the major applications is SOAP envelope, it would be useful to develop a specification language for representing protocols with XML messages. In particular, we would consider the style of the tool TulaFale [13] and adapt it with for unbounded lists.

Finally, it would be interesting to apply our verifier to protocols for electronic voting. However, this application has to face also other limitations of ProVerif, as for example that it cannot prove equivalences under homomorphic encryption and that it is not possible to represent mix nets.

Bibliography

[1] Martín Abadi and Bruno Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. *Journal of the ACM*, 52(1):102–146, 2005.

[2] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL'01*, pages 104–115, London, United Kingdom, January 2001. ACM Press.

[3] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

[4] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 20(3):395, 2007.

[5] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The avispa tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

[6] N. Asokan and Philip Ginzboorg. Key agreement in ad hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.

[7] L. Bachmair and H. Ganzinger. Resolution theorem proving. In *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–100. North Holland, 2001.

[8] Michael Backes, Sebastian Mödersheim, Birgit Pfitzmann, and Luca Viganò. Symbolic and cryptographic analysis of the secure WS-ReliableMessaging scenario. In *FoSSaCS'06*, volume 3921 of *Lecture Notes in Computer Science*, pages 428–445. Springer, 2006.

[9] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. XML signature syntax and processing (second edition). Available at `http://www.w3.org/TR/xmldsig-core/`.

[10] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011.

[11] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101. ACM, 2009.

[12] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. A semantics for web services authentication. In *POPL'04*, pages 198–209. ACM, 2004.

[13] Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Riccardo Pucella. Tulafale: A security tool for web services. In *FMCO'03*, volume 3188 of *Lecture Notes in Computer Science*, pages 197–222. Springer, 2004.

[14] Bruno Blanchet. CryptoVerif: A computationally sound mechanized prover for cryptographic protocols. In *Dagstuhl seminar "Formal Protocol Verification Applied"*, October 2007.

[15] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, July 2009.

[16] Bruno Blanchet. Using Horn clauses for analyzing security protocols. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*. IOS Press, March 2011. Available at `http://www.di.ens.fr/~blanchet/publications/BlanchetBook09.html`.

[17] Bruno Blanchet and Miriam Paiola. Automatic verification of protocols with lists of unbounded length. In *ACM Conference on Computer and Communications Security (CCS'13)*, pages 573–584, Berlin, Germany, November 2013. ACM.

[18] A. Brown, B. Fox, S. Hada, B. LaMacchia, and H. Maruyama. SOAP security extensions: Digital signature. Available at `http://www.w3.org/TR/SOAP-dsig/`.

[19] Jeremy Bryans and Steve Schneider. CSP, PVS and recursive authentication protocol. In *DIMACS Workshop on Formal Verification of Security Protocols*, September 1997.

[20] Najah Chridi, Mathieu Turuani, and Michaël Rusinowitch. Decidable analysis for a class of cryptographic group protocols with unbounded lists. In *CSF'09*, pages 277–289, Los Alamitos, 2009. IEEE.

[21] Hubert Comon-Lundh and Véronique Cortier. Computational soundness of observational equivalence. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 109–118, 2008.

[22] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *Journal of Automated Reasoning*, 46(3-4):225–259, April 2010.

[23] C.J.F. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, volume 5123/2008 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.

[24] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[25] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[26] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, 1983.

[27] Lilia Georgieva, Ullrich Hustadt, and Renate A. Schmidt. A new clausal class decidable by hyperresolution. In *CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2002.

[28] Jean Goubault-Larrecq. Une fois qu'on n'a pas trouvé de preuve, comment le faire comprendre à un assistant de preuve ? In *JFLA'04*, pages 1–20. INRIA, 2004.

[29] E. Kleiner and A. W. Roscoe. On the relationship between web services security and traditional protocols. In *MFPS 21*, volume 155 of *Electronic Notes in Theoretical Computer Science*, pages 583–603, 2006.

[30] Steve Kremer, Antoine Mercier, and Ralf Treinen. Proving group protocols secure against eavesdroppers. In *IJCAR '08*, volume 5195 of *LNAI*, pages 116–131, Heidelberg, 2008. Springer.

[31] R. Küsters and T. Truderung. Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In *CSF'09*, pages 157–171, Los Alamitos, 2009. IEEE.

[32] Ralf Küsters and Tomasz Truderung. On the automatic analysis of recursive security protocols with XOR. In W. Thomas and P. Weil, editors, *STACS'07*, volume 4393 of *Lecture Notes in Computer Science*, pages 646–657, Heidelberg, 2007. Springer.

[33] Jean-Louis Lassez, Michael J. Maher, and Kim Marriott. Unification revisited. In *Foundations of Deductive Databases and Logic Programming.*, pages 587–625. Morgan Kaufmann, 1988.

[34] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131 – 133, 1995.

[35] Gavin Lowe. A hierarchy of authentication specifications. In *CSFW'97*, pages 31–43. IEEE Computer Society, June 1997.

[36] Michael Mcintosh and Paula Austel. XML signature element wrapping attacks and countermeasures. In *SWS'05*, pages 20–27. ACM, 2005.

[37] Catherine Meadows. Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In *WITS'00*, 2000.

[38] Catherine Meadows, Paul Syverson, and Iliano Cervesato. Formal specification and analysis of the Group Domain of Interpretation protocol using NPATRL and the NRL protocol analyzer. *Journal of Computer Security*, 12(6):893–931, 2004.

[39] Cathy Meadows and Paliath Narendran. A unification algorithm for the group Diffie-Hellman protocol. In *WITS'02*, 2002.

[40] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In Moni Naor, editor, *Theory of cryptography conference - Proceedings of TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151, Cambridge, MA, USA, February 2004. Springer.

[41] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[42] Miriam Paiola and Bruno Blanchet. Verification of security protocols with lists: from length one to unbounded length. In Pierpaolo Degano and Joshua D. Guttman, editors, *POST'12*, volume 7215 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2012.

[43] Miriam Paiola and Bruno Blanchet. Verification of security protocols with lists: from length one to unbounded length. *Journal of Computer Security*, 21(6):781–816, December 2013. Special issue POST'12.

[44] Lawrence C. Paulson. Mechanized proofs for a recursive authentication protocol. In *CSFW'97*, pages 84–95. IEEE Computer Society Press, 1997.

[45] Olivier Pereira and Jean-Jacques Quisquater. Some attacks upon authenticated group key agreement protocols. *Journal of Computer Security*, 11(4):555–580, 2003.

[46] Olivier Pereira and Jean-Jacques Quisquater. Generic insecurity of cliques-type authenticated group key agreement protocols. In *CSFW'04*, pages 16–19, Los Alamitos, 2004. IEEE.

[47] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[48] A. W. Roscoe and E. Kleiner. Web Services Security: a preliminary study using Casper and FDR. In *Automated Reasoning for Security Protocol Analysis (ARSPA'04)*, 2004.

[49] Graham Steel and Alan Bundy. Attacking group protocols by refuting incorrect inductive conjectures. *Journal of Automated Reasoning*, 36(1-2):149–176, 2006.

[50] Michael Steiner, Gene Tsudik, and Michael Waidner. CLIQUES: A new approach to group key agreement. In *ICDCS'98*, pages 380–387, Los Alamitos, 1998. IEEE.

[51] Tomasz Truderung. Selecting theories and recursive protocols. In *CONCUR 2005*, volume 3653 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2005.

Appendix A

Appendix

A.1 Proofs of Part 1

A.1.1 Proof of Lemma 7

Proof. By induction on the clause term p^G.

- $p^G = x_{i_1,\dots,i_h}$: in this case, $T = T'[i_1 \mapsto \bar{\imath}_1, \dots, i_h \mapsto \bar{\imath}_h]$. As p^G is well typed and the only possibility for typing $\Gamma \vdash x_{i_1,\dots,i_h}$ is applying the type rule (Var), for each $j = 1, \dots, h$ we have $\Gamma \vdash i_j : [1, L_j]$ for some L_j. We can then define τ by $\tau([1, L_j]) = \bar{\imath}_j$, since the types of the free indices $i_1, \dots, i_h$ are pairwise distinct. Therefore, we have that

$$\mathbb{I}_{\Gamma,\tau}(\sigma) = \cup_{p' \in \mathbb{I}(\sigma x_{\bar{\imath}_1,\dots,\bar{\imath}_h})}\{\{x_{\underbrace{1,\dots,1}_{h}} \mapsto p'\}\} = \{\{x_{\underbrace{1,\dots,1}_{h}} \mapsto p' \mid p' \in \mathbb{I}(\sigma p^{GT})\}.$$

  Hence, for all $p \in \mathbb{I}(\sigma p^{GT})$, there exists $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$ such that $\sigma' p^{G1} = \sigma' x_{\underbrace{1,\dots,1}_{h}} = p$.

- $p^G = f(p_1^G, \dots, p_h^G)$: in this case

$$\sigma p^{GT} = \sigma(f(p_1^G, \dots, p_h^G))^T = f(\sigma p_1^{GT}, \dots, \sigma p_h^{GT}).$$

  As p^G is well typed in Γ and the only possibility for typing $\Gamma \vdash f(p_1^G, \dots, p_h^G)$ is applying the type rule (Fun), we have $\Gamma \vdash p_1^G, \dots, \Gamma \vdash p_h^G$. Now, for all $p \in \mathbb{I}(\sigma p^{GT}) = \mathbb{I}(f(\sigma p_1^{GT}, \dots, \sigma p_h^{GT}))$, there exist $p_j \in \mathbb{I}(\sigma p_j^{GT})$ for $j = 1, \dots, h$, such that $p = f(p_1, \dots, p_h)$. By induction, for all $p_j \in \mathbb{I}(\sigma p_j^{GT})$, there exist τ_j consistent with T for $\Gamma \vdash p_j^G$ and $\sigma'_j \in \mathbb{I}_{\Gamma,\tau_j}(\sigma_{|fv(p_j^{GT})})$ such that $\sigma'_j p_j^{G1} = p_j$. We can define $\tau = \cup_j \tau_j$: in fact, $\tau_1, \dots, \tau_h$ have the same value over types of free indices (because they are consistent with the same T) and are disjoint over types of bound indices (because the bound indices have pairwise distinct bounds, and distinct from the bounds of free indices). Since p^G is linear, $\sigma'_1, \dots, \sigma'_h$ have disjoint domain, so we can define $\sigma' = \cup_j \sigma'_j$. Hence, τ is consistent with T for $\Gamma \vdash p^G$, $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$, and $\sigma' p^{G1} = \sigma'(f(p_1^G, \dots, p_h^G))^1 = f(p_1, \dots, p_h) = p$.

- $p^G = a_i[p_1^G, \dots, p_h^G]$. Similar to the previous case.

- $p^G = list(i \le M, p'^G)$: in this case

$$\begin{aligned}\sigma p^{GT} &= \sigma(list(i \le M, p'^G))^T = \sigma\langle p'^{GT[i\mapsto 1]}, \dots, p'^{GT[i\mapsto M^T]}\rangle \\ &= \langle \sigma p'^{GT[i\mapsto 1]}, \dots, \sigma p'^{GT[i\mapsto M^T]}\rangle.\end{aligned}$$

  For all $p \in \mathbb{I}(\sigma p^{GT}) = \mathbb{I}(\langle \sigma p'^{GT[i\mapsto 1]}, \dots, \sigma p'^{GT[i\mapsto M^T]}\rangle)$, there exist $k \in \{1, \dots, M^T\}$ and $p_k \in \mathbb{I}(\sigma p'^{GT[i\mapsto k]})$ such that $p = \langle p_k \rangle$. As p^G is well typed in Γ and the only possibility for typing $\Gamma \vdash list(i \le M, p'^G)$ is applying the type rule (List), we have

$\Gamma, i : [1, M] \vdash p'^G$. Hence, by induction, for each $p_k \in \mathbb{I}(\sigma p'^{GT[i \mapsto k]})$, there exist τ_k consistent with $T' = T[i \mapsto k]$ for $\Gamma' = \Gamma, i : [1, M] \vdash p'^G$ and $\sigma'_k \in \mathbb{I}_{\Gamma', \tau_k}(\sigma)$ such that $\sigma'_k p'^{G1} = p_k$.

We show that τ_k is consistent with T for $\Gamma \vdash p^G$. Notice that τ_k is defined on $Types(\Gamma' \vdash p'^G)$ and that this is equal to $Types(\Gamma \vdash p^G)$. As τ_k is consistent with T' for $\Gamma' \vdash p'^G$, for each type $[1, L] \in Types(\Gamma' \vdash p'^G) = Types(\Gamma \vdash p^G)$, we have:

- $1 \leq \tau_k([1, L]) \leq L^{T'} = L^T$.
- $j^{T'} = \tau_k([1, L])$ if j is a free index of p'^G and $\Gamma \vdash j : [1, L]$. As $T' = T[i \mapsto k]$ and the free indices of p'^G are the free indices of p^G plus i, we have that $j^T = \tau_k([1, L])$ if j is a free index of p^G and $\Gamma \vdash j : [1, L]$.

Notice that $\mathbb{I}_{\Gamma,\tau}(\sigma) = \mathbb{I}_{\Gamma', \tau_k}(\sigma)$, as the variables typed in Γ and Γ' have the same type in both the type environments.

Hence we can choose $\tau = \tau_k$ and $\sigma' = \sigma'_k$ so that τ is consistent with T for $\Gamma \vdash p^G$, $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$, and $\sigma' p^{G1} = \sigma'_k list(i \leq M, p'^G)^1 = \langle p_k \rangle = p$. □

A.1.2 Proof of Lemma 11

Proof. Let $\sigma'_j = \sigma'_{|fv(p_j^{G1})}$ and $\sigma_j = \sigma_{|fv(p_j^{GT})}$.

First, we have $dom(\sigma'_j) = dom\mathbb{I}(\sigma_j)$. As σ'_j is the restriction of $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$, by definition of $\mathbb{I}_{\Gamma,\tau}(\sigma)$, for each $x_{\underbrace{1,\ldots,1}_{l}} \in dom(\sigma'_j)$, $\sigma'_j x_{\underbrace{1,\ldots,1}_{l}} \in \mathbb{I}(\sigma x_{\bar{\imath}_1,\ldots,\bar{\imath}_l})$, where $\Gamma \vdash x_\_ : [1, L_1] \times \cdots \times [1, L_l]$ and $\bar{\imath}_k = \tau([1, L_k])$ for all $k = 1, \ldots, l$. As in Remark 1, $x_{\bar{\imath}_1,\ldots,\bar{\imath}_l} \in fv(p_j^{GT}) = dom(\sigma_j)$, so $\sigma_j x_{\bar{\imath}_1,\ldots,\bar{\imath}_l} = \sigma x_{\bar{\imath}_1,\ldots,\bar{\imath}_l}$. Moreover, $\tau_j([1, L_k]) = \tau([1, L_k])$. Hence, for each $x_{\underbrace{1,\ldots,1}_{l}} \in dom(\sigma'_j)$, $\sigma'_j x_{\underbrace{1,\ldots,1}_{l}} \in \mathbb{I}(\sigma_j x_{\bar{\imath}_1,\ldots,\bar{\imath}_l})$, where $\Gamma \vdash x_\_ : [1, L_1] \times \cdots \times [1, L_l]$ and $\bar{\imath}_k = \tau_j([1, L_k])$ for all $k = 1, \ldots, l$. Hence, by definition of $\mathbb{I}_{\Gamma,\tau_j}(\sigma_j)$, we have $\sigma'_j \in \mathbb{I}_{\Gamma,\tau_j}(\sigma_j)$. □

A.1.3 Proof of Lemma 9

Proof. By induction on the clause term p^G.

- $p^G = x_{i_1,\ldots,i_h}$: in this case, $T = T'[i_1 \mapsto \bar{\imath}_1, \ldots, i_h \mapsto \bar{\imath}_h]$. As p^G is well typed the only possibility for typing $\Gamma \vdash x_{i_1,\ldots,i_h}$ is applying the type rule (Var), for each $j = 1, \ldots, h$ we have $\Gamma \vdash i_j : [1, L_j]$ for some L_j. For each τ consistent with T for $\Gamma \vdash p^G$, for

each $j = 1, \ldots, h$, we have $\tau([1, L_j]) = \bar{\imath}_j$. Hence, for each $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$, we have $\sigma' p^{G1} = \sigma' x_{\underbrace{1,\ldots,1}_{h}} \in \mathbb{I}(\sigma x_{\bar{\imath}_1,\ldots,\bar{\imath}_h}) = \mathbb{I}(\sigma p^{GT})$.

- $p^G = f(p_1^G, \ldots, p_h^G)$: in this case

$$\sigma p^{GT} = \sigma(f(p_1^G, \ldots, p_h^G))^T = f(\sigma p_1^{GT}, \ldots, \sigma p_h^{GT}).$$

As p^G is well typed in Γ and the only possibility for typing $\Gamma \vdash f(p_1^G, \ldots, p_h^G)$ is applying the type rule (Fun), we have $\Gamma \vdash p_1^G, \ldots, \Gamma \vdash p_h^G$. For all τ consistent with T for $\Gamma \vdash p^G$, for all $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$, for each $j = 1, \ldots, h$, by Lemma 11, we have that $\sigma'_{|fv(p_j^{G1})} \in \mathbb{I}_{\Gamma,\tau_j}(\sigma_{|fv(p_j^{GT})})$, where $\tau_j = \tau_{|Types(\Gamma \vdash p_j^G)}$. By induction, for each $j = 1, \ldots, h$, since τ_j is consistent with T for $\Gamma \vdash p_j^G$, and $\sigma'_{|fv(p_j^{G1})} \in \mathbb{I}_{\Gamma,\tau_j}(\sigma_{|fv(p_j^{GT})})$, we have $\sigma'_{|fv(p_j^{G1})} p_j^{G1} \in \mathbb{I}(\sigma_{|fv(p_j^{GT})} p_j^{GT})$. Therefore, $\sigma' p^{G1} = f(\sigma'_{|fv(p_1^{G1})} p_1^{GT}, \ldots, \sigma'_{|fv(p_h^{G1})} p_h^{GT}) \in \mathbb{I}(f(\sigma_{|fv(p_1^{GT})} p_1^{GT}, \ldots, \sigma_{|fv(p_h^{GT})} p_h^{GT})) = \mathbb{I}(\sigma p^{GT})$.

- $p^G = a_i[p_1^G, \ldots, p_h^G]$. Similar to the previous case.

- $p^G = list(i \leq L, p'^G)$: in this case

$$\sigma p^{GT} = \langle \sigma p'^{GT[i \mapsto 1]}, \ldots, \sigma p'^{GT[i \mapsto L^T]} \rangle.$$

As p^G is well typed in Γ and the only possibility for typing $\Gamma \vdash list(i \leq L, p'^G)$ is applying the type rule (List), we have $\Gamma, i : [1, L] \vdash p'^G$. Let τ be consistent with T for $\Gamma \vdash p^G$: we have $\tau([1, L]) = \bar{\imath}$, for some $\bar{\imath} \in \{1, \ldots, L^T\}$, and τ is consistent with $T' = T[i \mapsto \bar{\imath}]$ for $\Gamma, i : [1, L] \vdash p'^G$. Hence, for each $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma)$, we have $\sigma' \in \mathbb{I}_{\Gamma,\tau}(\sigma_{|fv(p'^{GT'})})$. Therefore, by induction, we have $\sigma' p'^{G1} \in \mathbb{I}(\sigma p'^{GT'})$, from which follows that $\sigma' p^{G1} = \langle \sigma' p'^{G1} \rangle \in \mathbb{I}(\sigma p^{GT})$. □

A.2 Proofs of Part 2

A.2.1 Proofs of Section 7.1.1

Proof of Lemma 14. This result is easily proved by induction over the syntax of ι, J, p^G, F^G, E, E', $\mathcal{E}$, $\mathcal{I}$. □

Proof of Lemma 15. This result is proved by induction over the syntax of p^G, F^G, E, $\mathcal{E}$. The only interesting case is the clause term $p^G = x_{\iota_1,\ldots,\iota_h}$.

Suppose that σ^G maps $x_{i_1,\ldots,i_h}$ to p'^G. Let $\rho = \{i_1 \mapsto \iota_1, \ldots, i_h \mapsto \iota_h\}$. Then $(\sigma^G x_{\iota_1,\ldots,\iota_h})^T = (\rho p'^G)^T = p'^{GT[i_1 \mapsto \iota_1^T,\ldots,i_h \mapsto \iota_h^T]}$ by Lemma 14, since the environment T' of Lemma 14 is $T' = T[i_1 \mapsto \iota_1^T, \ldots, i_h \mapsto \iota_h^T]$. Moreover, $\sigma^{GT}(x_{\iota_1,\ldots,\iota_h})^T = \sigma^{GT} x_{\iota_1^T,\ldots,\iota_h^T}$. Now in σ^{GT}, we have the mappings $x_{v_1,\ldots,v_h} \mapsto p'^{GT[i_1 \mapsto v_1,\ldots,i_h \mapsto v_h]}$ for all $j = 1, \ldots, h$, for all $v_j \in \{1, \ldots, L_j^T\}$, so $\sigma^{GT} x_{\iota_1^T,\ldots,\iota_h^T} = p'^{GT[i_1 \mapsto \iota_1^T,\ldots,i_h \mapsto \iota_h^T]}$. Therefore, we have $(\sigma^G x_{\iota_1,\ldots,\iota_h})^T = \sigma^{GT}(x_{\iota_1,\ldots,\iota_h})^T$.

If σ^G leaves $x_{i_1,\ldots,i_h}$ unchanged, then we have $(\sigma^G x_{\iota_1,\ldots,\iota_h})^T = x_{\iota_1^T,\ldots,\iota_h^T} = \sigma^{GT} x_{\iota_1^T,\ldots,\iota_h^T} = \sigma^{GT}(x_{\iota_1,\ldots,\iota_h})^T$. □

Proof of Lemma 16. Since $\mathcal{E}_2^T$ is defined and $\sigma^G \mathcal{E}_1 \subseteq \mathcal{E}_2$, $(\sigma^G \mathcal{E}_1)^T$ is also defined and $(\sigma^G \mathcal{E}_1)^T \subseteq \mathcal{E}_2^T$. By Lemma 15, we have $\sigma^{GT} \mathcal{E}_1^T = (\sigma^G \mathcal{E}_1)^T \subseteq \mathcal{E}_2^T$. Since $\text{MGU}\,(\mathcal{E}_2^T)$ unifies $\mathcal{E}_2^T$, it also unifies $\sigma^{GT} \mathcal{E}_1^T$ and, as $\text{MGU}\,(\mathcal{E}_2^T)(\sigma^{GT} \mathcal{E}_1^T) = (\text{MGU}\,(\mathcal{E}_2^T)\sigma^{GT})\mathcal{E}_1^T$, $\text{MGU}\,(\mathcal{E}_2^T)\sigma^{GT}$ unifies $\mathcal{E}_1^T$. Then $\text{MGU}\,(\mathcal{E}_1^T)$ is defined and there exists σ' such that:

$$\begin{aligned} \text{MGU}\,(\mathcal{E}_2^T)\sigma^{GT} &= \sigma' \,\text{MGU}\,(\mathcal{E}_1^T) \\ &= \sigma' \,\text{MGU}\,(\mathcal{E}_1^T)\,\text{MGU}\,(\mathcal{E}_1^T) \\ &= \text{MGU}\,(\mathcal{E}_2^T)\sigma^{GT}\,\text{MGU}\,(\mathcal{E}_1^T) \end{aligned}$$

since $\text{MGU}\,(\mathcal{E}_1^T)$ is idempotent. (The variables in its image do not occur in its domain, by our definition of most general unifiers.) □

A.2.2 Proofs of Section 7.2.1

A.2.2.1 Proof of Lemma 18

We use the following standard result.

Lemma 32. *Let $\mathcal{E}_1$, $\mathcal{E}_2$ be two sets of equations over standard clause terms. Then* $\text{MGU}(\mathcal{E}_1 \cup \mathcal{E}_2)$ *is defined if and only if* $\text{MGU}(\text{MGU}(\mathcal{E}_2)\mathcal{E}_1)\text{MGU}(\mathcal{E}_2)$ *is defined, and* $\text{MGU}(\mathcal{E}_1 \cup \mathcal{E}_2) = \text{MGU}(\text{MGU}(\mathcal{E}_2)\mathcal{E}_1)\text{MGU}(\mathcal{E}_2)$.

We can extend the following result for clause terms p_1^G and p_2^G to the corresponding facts $pred(p_1^G)$ and $pred(p_2^G)$ and obtain Lemma 18:

Lemma 33. *Suppose that p_1^G and $\mathcal{E}_1$ contain no variable common with p_2^G and $\mathcal{E}_2$. Let $\mathcal{E} = \{C\, p_1^G \doteq p_2^G\} \cup \mathcal{E}_1 \cup \mathcal{E}_2$. Let T be an environment such that*

$$\sigma = \text{MGU}\{\text{MGU}(\mathcal{E}_1^T)p_1^{GT'} = \text{MGU}(\mathcal{E}_2^T)p_2^{GT'} \mid T' \in T^C\}.$$

is defined. Then $\text{MGU}(\mathcal{E}^T) = \sigma\text{MGU}(\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_2^T)$.

Proof. We have

$$\text{MGU}(\mathcal{E}^T) = \text{MGU}(\{p_1^{GT'} = p_2^{GT'} \mid T' \in T^C\} \cup \mathcal{E}_1^T \cup \mathcal{E}_2^T).$$

By Lemma 32,

$$\begin{aligned}\text{MGU}(\mathcal{E}_1^T \cup \mathcal{E}_2^T) &= \text{MGU}(\text{MGU}(\mathcal{E}_2)^T\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_2^T)\\ &= \text{MGU}(\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_2^T)\end{aligned}$$

since $\mathcal{E}_1^T$ and $\mathcal{E}_2^T$ have no common variables. Again by Lemma 32,

$$\begin{aligned}&\text{MGU}(\{p_1^{GT'} = p_2^{GT'} \mid T' \in T^C\} \cup \mathcal{E}_1^T \cup \mathcal{E}_2^T)\\ &= \text{MGU}(\text{MGU}(\mathcal{E}_1^T \cup \mathcal{E}_2^T)\{p_1^{GT'} = p_2^{GT'} \mid T' \in T^C\})\text{MGU}(\mathcal{E}_1^T \cup \mathcal{E}_2^T)\\ &= \text{MGU}(\text{MGU}(\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_2^T)\{p_1^{GT'} = p_2^{GT'} \mid T' \in T^C\})\text{MGU}(\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_2^T)\\ &= \text{MGU}(\{\text{MGU}(\mathcal{E}_1^T)p_1^{GT'} = \text{MGU}(\mathcal{E}_2^T)p_2^{GT'} \mid T' \in T^C\})\text{MGU}(\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_2^T)\end{aligned}$$

since $p_1^{GT'}$ has no common variables with $\mathcal{E}_2^T$ and $p_2^{GT'}$ has no common variables with $\mathcal{E}_1^T$. So $\text{MGU}(\mathcal{E}^T) = \sigma\text{MGU}(\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_2^T)$. □

A.2.2.2 Proof of Lemma 19

Proof. We build the type environment Γ as follows:

- for all $j \le h$, $i_j : [1, L_j] \in \Gamma$;
- since $\Gamma_2 \vdash J : [1, L_1] \times \cdots \times [1, L_h]$ has been derived by rules (EmptySet), (EnvSet), and (Set), J is either $[1, L_1] \times \cdots \times [1, L_h]$ or $I \times [1, L_{k+1}] \times \cdots \times [1, L_h]$ with $I : [1, L_1] \times \cdots \times [1, L_k] \in \Gamma_2$. In the latter case, we include $I : [1, L_1] \times \cdots \times [1, L_k]$ in Γ, so that in all cases $\Gamma \vdash J : [1, L_1] \times \cdots \times [1, L_h]$;

- if $i : [1, L] \in \Gamma_1$, then $\phi' : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma$, where ϕ' is the function symbol associated to the free index i of R_1^G by Item 1 of the definition of immersion;
- if $\phi : [1, L'_1] \times \cdots \times [1, L'_k] \to [1, L] \in \Gamma_1$, then $\phi' : [1, L_1] \times \cdots \times [1, L_h] \times [1, L'_1] \times \cdots \times [1, L'_k] \to [1, L] \in \Gamma$, where ϕ' is the function symbol associated to ϕ by Item 2 of the definition of immersion;
- if $x\_ : [1, L'_1] \times \cdots \times [1, L'_k] \in \Gamma_1$, then $x\_ : [1, L_1] \times \cdots \times [1, L_h] \times [1, L'_1] \times \cdots \times [1, L'_k] \in \Gamma$;
- if $I : [1, L''_1] \times \cdots \times [1, L''_{k-l}] \in \Gamma_1$, then $I' : [1, L_1] \times \cdots \times [1, L_h] \times [1, L''_1] \times \cdots \times [1, L''_{k-l}] \in \Gamma$, where I' is the set symbol associated to I by Item 5 of the definition of immersion.

To show that $\Gamma \vdash imm(R_1^G, (i_1, \ldots, i_h) \in J)$, we build the derivation of $\Gamma \vdash imm(R_1^G, (i_1, \ldots, i_h) \in J)$ from the derivation of $\Gamma_1 \vdash R_1^G$ by induction. Let Γ'_1 be an extension of Γ_1 with bound indices: $\Gamma'_1 = \Gamma_1, i'_1 : [1, L''_1], \ldots, i'_l : [1, L''_l]$, and Γ' be the corresponding extension of Γ: $\Gamma' = \Gamma, i'_1 : [1, L''_1], \ldots, i'_l : [1, L''_l]$.

- If $\Gamma'_1 \vdash \iota : [1, L]$, then $\Gamma' \vdash \iota' : [1, L]$, where ι is transformed into ι' by Items 1 and 2 of the definition of immersion, by induction on the derivation of $\Gamma'_1 \vdash \iota : [1, L]$.

  If $\Gamma'_1 \vdash \iota : [1, L]$ is derived by (EnvIndex) and ι is a free index, then $\iota = i$, $\iota' = \phi'(i_1, \ldots, i_h)$, and we derive $\Gamma \vdash \iota' : [1, L]$ from $i_j : [1, L_j] \in \Gamma$ for all $j \leq h$ and $\phi' : [1, L_1] \times \cdots \times [1, L_h] \to [1, L] \in \Gamma$ by (EnvIndex) and (Index).

  If $\Gamma'_1 \vdash \iota : [1, L]$ is derived by (EnvIndex) and ι is a bound index, then $\iota = i'_j$ for some $j \leq l$, $\iota' = i'_j$, $i'_j : [1, L''_j] \in \Gamma'_1$ (with $L''_j = L$), $i'_j : [1, L''_j] \in \Gamma'$, so $\Gamma' \vdash \iota' : [1, L]$ by (EnvIndex).

  If $\Gamma_1 \vdash \iota : [1, L]$ is derived by (Index), then $\iota = \phi(\iota_1, \ldots, \iota_k)$, $\iota' = \phi'(i_1, \ldots, i_h, \iota'_1, \ldots, \iota'_k)$, with $\Gamma_1 \vdash \iota_j : [1, L'_j]$ for all $j \leq k$. By induction hypothesis, $\Gamma \vdash \iota'_j : [1, L'_j]$ for all $j \leq k$. We derive $\Gamma \vdash \iota' : [1, L]$ from $i_j : [1, L_j] \in \Gamma$ for all $j \leq h$ and $\phi' : [1, L_1] \times \cdots \times [1, L_h] \times [1, L'_1] \times \cdots \times [1, L'_k] \to [1, L] \in \Gamma$ by (EnvIndex) and (Index).

- If $\Gamma_1 \vdash J' : [1, L'_1] \times \cdots \times [1, L'_k]$, then $\Gamma \vdash J'' : [1, L_1] \times \cdots \times [1, L_h] \times [1, L'_1] \times \cdots \times [1, L'_k]$, where J' is transformed into J'' by immersion:
  - if $J' = [1, L'_1] \times \cdots \times [1, L'_k]$, then $J'' = J \times [1, L'_1] \times \cdots \times [1, L'_k]$;
  - if $J' = I \times [1, L'_{l+1}] \times \cdots \times [1, L'_k]$, then $J'' = I' \times [1, L'_{l+1}] \times \cdots \times [1, L'_k]$, where I' is the set symbol associated to I by Item 5 of the definition of immersion.

  This result is proved by induction on the derivation of $\Gamma_1 \vdash J' : [1, L'_1] \times \cdots \times [1, L'_k]$. If $\Gamma_1 \vdash J' : [1, L'_1] \times \cdots \times [1, L'_k]$ is derived by (EmptySet), then $J' = \{()\}$ and $J'' = J$; we use the property $\Gamma \vdash J : [1, L_1] \times \cdots \times [1, L_h]$. Otherwise, we apply the same type rule, (EnvSet) or (Set), in both derivations.

- In all other cases, we show that, if $\Gamma'_1 \vdash p^G$ (resp. $\Gamma_1 \vdash F^G$, $\Gamma_1 \vdash E$, $\Gamma_1 \vdash E'$, $\Gamma_1 \vdash \mathcal{I}$, $\Gamma_1 \vdash R^G$), then $\Gamma' \vdash p^G$ (resp. $\Gamma \vdash F^G$, $\Gamma \vdash E$, $\Gamma \vdash E'$, $\Gamma \vdash \mathcal{I}$, $\Gamma \vdash R^G$), by induction on the initial derivation, using the same type rule in both derivations.

□

A.2.3 Proof of Section 7.3.8

A.2.3.1 Proof of Lemma 21

To prove Lemma 21 we first need to prove the following result.

Lemma 34. *Let $\Gamma \vdash R^G$ and $\Gamma \vdash R'^G$ be two well-typed generalized Horn clauses such that $R^G = Cts, H^G \wedge \mathcal{E} \Rightarrow C^G$ and $R'^G = Cts, H^G \wedge \mathcal{E}' \Rightarrow C^G$. Let T be an environment for $\Gamma \vdash R^G$ and for $\Gamma \vdash R'^G$. Suppose that $\text{MGU}(\mathcal{E}^T) = \text{MGU}(\mathcal{E}'^T)$. If R^{GT} is defined, then R'^{GT} is defined and $R^{GT} = R'^{GT}$.*

Proof. Since R^{GT} is defined, T satisfies Cts and $\text{MGU}(\mathcal{E}^T) = \text{MGU}(\mathcal{E}'^T)$ is defined, so $R'^{GT} = \text{MGU}(\mathcal{E}'^T)H^{GT} \Rightarrow \text{MGU}(\mathcal{E}'^T)C^{GT} = \text{MGU}(\mathcal{E}^T)H^{GT} \Rightarrow \text{MGU}(\mathcal{E}^T)C^{GT} = R^{GT}$. □

Proof of Lemma 21. The proof proceeds by cases on the step applied by the unification algorithm. The cases are numbered as in the definition of the unification algorithm in Section 7.3.1.

1. The clause R'^G differs from R^G only by the set of equations: $\mathcal{E}' = \mathcal{E}_1 \cup \{C\, p_1^G \doteq p_1'^G, \ldots, C\, p_k^G \doteq p_k'^G\}$ where $\mathcal{E} = \mathcal{E}_1 \cup \{C\, f(p_1^G, \ldots, p_k^G) \doteq f(p_1'^G, \ldots, p_k'^G)\}$. We choose $T' = T$ as environment for $\Gamma \vdash R'^G$. If R^{GT} is defined, then $\mathcal{E}^T$ is defined and $\mathcal{E}'^T$ is defined too. We have that:

$$\mathcal{E}^T = \mathcal{E}_1^T \cup \{f(p_1^{GT''}, \ldots, p_k^{GT''}) = f(p_1'^{GT''}, \ldots, p_k'^{GT''}) \mid T'' \in T^C\}$$
$$\mathcal{E}'^T = \mathcal{E}_1^T \cup \{p_1^{GT''} = p_1'^{GT''}, \ldots, p_k^{GT''} = p_k'^{GT''} \mid T'' \in T^C\}$$

   $\mathcal{E}^T$ and $\mathcal{E}'^T$ have the same solutions: $\sigma f(p_1^{GT''}, \ldots, p_k^{GT''}) = \sigma f(p_1'^{GT''}, \ldots, p_k'^{GT''})$ if and only if for all $j = 1, \ldots, k$, $\sigma p_j^{GT''} = \sigma p_j'^{GT''}$. We apply Lemma 34 and conclude.

2. This case is similar to case 1, except that the algorithm adds to $\mathcal{E}'$ the equation $C\,\iota_j \doteq \iota'_j$ and replaces L'_j with L_j, for every $j = 1, \ldots, h$. Since R^{GT} is defined, $\text{MGU}(\mathcal{E}^T)$ is defined and for all $T'' \in T^C$, $\mathcal{E}^T$ contains the equation $a^{L_1^T, \ldots, L_h^T}_{\iota_1^{T''}, \ldots, \iota_h^{T''}}[\ldots] = a^{L_1'^T, \ldots, L_h'^T}_{\iota_1'^{T''}, \ldots, \iota_h'^{T''}}[\ldots]$, so $L_j^T = L_j'^T$ and $\iota_j^{T''} = \iota_j'^{T''}$, for each $j = 1, \ldots, h$. Therefore, $(C\,\iota_j \doteq \iota'_j)^T = \mathit{true}$, for each $j = 1, \ldots, h$. Let Γ' be the type environment obtained from Γ by replacing every

occurrence of L'_1 with $L_1, \ldots, L'_h$ with L_h. Let T' be the environment obtained from T by removing $\{L'_j \mapsto L'^T_j \mid j = 1, \ldots, h\}$: T' is an environment for $\Gamma' \vdash R'^G$. For every $j = 1, \ldots, h$, we can replace L'_j with L_j without changing the translation since $L^T_j = L'^T_j$. Then, we proceed as in case 1, using $a^{L^T_1,\ldots,L^T_h}_{\iota^{T''}_1,\ldots,\iota'^{T''}_h}$ as function symbol instead of f.

3. Since R^{GT} is defined, we have $L^T = L'^T$; otherwise, the translated lists would have different lengths and $\text{MGU}(\mathcal{E}^T)$ would not be defined. We have $\mathcal{E}' = \mathcal{E}_1 \cup \{\bigwedge_{(i_1,\ldots,i_h,i)\in J\times[1,L]} p^G \doteq p'^G\}$ where $\mathcal{E} = \mathcal{E}_1 \cup \{\bigwedge_{(i_1,\ldots,i_h)\in J} \mathit{list}(i \le L, p^G) \doteq \mathit{list}(i \le L', p'^G)\}$. Let Γ' be the type environment obtained from Γ by replacing every occurrence of L' with L. Let T' be the environment obtained from T by removing $\{L' \mapsto L'^T\}$: T' is an environment for $\Gamma' \vdash R'^G$. We can replace L' with L without changing the translation since $L^T = L'^T$. Since R^{GT} is defined,

$$\begin{aligned}
\mathcal{E}^T &= \mathcal{E}^T_1 \cup \{\langle p^{GT''[i\mapsto 1]}, \ldots, p^{GT''[i\mapsto L^T]}\rangle = \langle p'^{GT''[i\mapsto 1]}, \ldots, p'^{GT''[i\mapsto L'^T]}\rangle \mid \\
&\qquad T'' = T[i_1 \mapsto v_1, \ldots, i_h \mapsto v_h], (v_1, \ldots v_h) \in J^T\} \\
&= \mathcal{E}^T_1 \cup \{\langle p^{GT''[i\mapsto 1]}, \ldots, p^{GT''[i\mapsto L^T]}\rangle = \langle p'^{GT''[i\mapsto 1]}, \ldots, p'^{GT''[i\mapsto L^T]}\rangle \mid \\
&\qquad T'' = T[i_1 \mapsto v_1, \ldots, i_h \mapsto v_h], (v_1, \ldots v_h) \in J^T\} \\
\mathcal{E}'^T &= \mathcal{E}^T_1 \cup \{p^{GT''} = p'^{GT''} \mid T'' = T[i_1 \mapsto v_1, \ldots, i_h \mapsto v_h, i \mapsto v], \\
&\qquad (v_1, \ldots v_h, v) \in J^T \times \{1, \ldots, L^T\}.
\end{aligned}$$

Since $\langle \ldots \rangle$ is a function of arity L^T, we conclude as in case 1.

4. Let us first consider the case in which we instantiate L to the integer h. Let T be any environment such that R^{GT} is defined. Then $\text{MGU}(\mathcal{E}^T)$ is defined; since $\mathcal{E}$ contains the equation $\bigwedge_{(i_1,\ldots,i_{h'})\in J} \mathit{list}(i \le L, p^G) \doteq \langle p^G_1, \ldots, p^G_h\rangle$, this implies that $L^T = h$.

   Let T_0 be the mapping from free indices of R^G of type $[1, L]$ to integers in $\{1, \ldots, h\}$ defined by $i^{T_0} = i^T$.

   Let T' be the environment equal to T for all bounds except L, for all set symbols, and for all indices that are not of type $[1, L]$, and such that $\phi\_v'_1\_\ldots\_v'^{T'}_k(v'_{k+1}, \ldots, v'_{k+k'}) = \phi^T(v_1, \ldots, v_{k+k'})$, where ϕ takes as argument k indices of type $[1, L]$ and k' indices not of type $[1, L]$, $v'_1, \ldots, v'_k$ are the elements of $v_1, \ldots, v_{k+k'}$ that correspond to the indices of type $[1, L]$, and $v'_{k+1}, \ldots, v'_{k+k'}$ are the elements of $v_1, \ldots, v_{k+k'}$ that correspond to the other indices.

   Let α be the renaming of variables that maps $x_{v_1,\ldots,v_{k+k'}}$ to $x\_v'_1\_\ldots\_v'_{k(v'_{k+1},\ldots,v'_{k+k'})}$ where x is a variable that has k indices of type $[1, L]$ and k' indices not of type $[1, L]$, $v'_1, \ldots, v'_k$ are the elements of $v_1, \ldots, v_{k+k'}$ that correspond to the indices of type $[1, L]$, and $v'_{k+1}, \ldots, v'_{k+k'}$ are the elements of $v_1, \ldots, v_{k+k'}$ that correspond to the other indices.

   Then we can show that $(\Im_{T_0}(R^G))^{T'} = \alpha R^{GT}$. The proof proceeds by induction on the syntax of clauses.

- For all index terms ι, for all environments T for ι such that $L^T = h$, we can define T_0 and T' as above and show that $(\Im_{T_0}(\iota))^{T'} = \iota^T$. (For integers v, we define $v^{T'} = v$.)
- For all clause terms p^G, for all environments T for p^G such that $L^T = h$, we can define T_0 and T' as above and show that $(\Im_{T_0}(p^G))^{T'} = \alpha p^{GT}$.

It follows that $(\Im_{T_0}(R^G))^{T'} \sqsupseteq R^{GT}$.

Let us now consider the case in which the instantiation cannot be applied. Let T be any environment such that R^{GT} is defined. Then $\text{MGU}(\mathcal{E}^T)$ is defined; since $\mathcal{E}$ contains the equation $\bigwedge_{(i_1,\ldots,i_{h'})\in J} list(i \le L, p^G) \doteq \langle p_1^G, \ldots, p_h^G\rangle$, this implies that $L^T = h$. Let us define $T' = T[I_1 \mapsto J^T \times \{1\}, \ldots, I_h \mapsto J^T \times \{h\}]$. The environment T' satisfies the constraint $I_1 \uplus \ldots \uplus I_h = J \times [1, L]$, so it satisfies $Cts' = Cts \cup \{I_1 \uplus \ldots \uplus I_h = J \times [1, L]\}$ since T satisfies Cts. Since $\text{MGU}(\mathcal{E}^T)$ is a unifier for $(\bigwedge_{(i_1,\ldots,i_{h'})\in J} list(i \le M, p^G) \doteq \langle p_1^G, \ldots, p_h^G\rangle)^T$, we have $\text{MGU}(\mathcal{E}^T) list(i \le L, p^G)^{T''} = \text{MGU}(\mathcal{E}^T)\langle p_1^G, \ldots, p_h^G\rangle^{T''}$ for all $T'' = T[i_1 \mapsto v_1, \ldots, i_{h'} \mapsto v_{h'}]$ with $(v_1, \ldots, v_{h'}) \in J^T$, that is, $\text{MGU}(\mathcal{E}^T) p^{GT''[i\mapsto v]} = \text{MGU}(\mathcal{E}^T) p_v^{GT''}$ for all $v \in \{1, \ldots, L^T\}$. Let $\sigma_x = \{x_{v_1,\ldots,v_{h'},v} \mapsto p_v^{GT[i_1\mapsto v_1,\ldots,i_{h'}\mapsto v_{h'}]} \mid (v_1, \ldots, v_{h'}, v) \in (J \times [1, L])^T\}$. Then $\text{MGU}(\mathcal{E}^T)\sigma_x$ unifies the equations

$$\begin{aligned}\{ &\bigwedge_{(i_1,\ldots,i_{h'},i)\in J\times[1,L]} x_{i_1,\ldots,i_{h'},i} \doteq p^G,\\ &\bigwedge_{(i_1,\ldots,i_{h'},i)\in I_1} x_{i_1,\ldots,i_{h'},i} \doteq p_1^G,\\ &\ldots,\\ &\bigwedge_{(i_1,\ldots,i_{h'},i)\in I_h} x_{i_1,\ldots,i_{h'},i} \doteq p_h^G\}^{T'}\end{aligned}$$

so it unifies the equations $\mathcal{E}'^{T'}$, since $\text{MGU}(\mathcal{E}^T)$ unifies the other equations of $\mathcal{E}'^{T'}$, which are also in $\mathcal{E}^T$ and do not contain x. Therefore, $\text{MGU}(\mathcal{E}'^{T'})$ is more general than $\text{MGU}(\mathcal{E}^T)\sigma_x$, that is, there exists a substitution σ such that $\text{MGU}(\mathcal{E}^T)\sigma_x = \sigma\text{MGU}(\mathcal{E}'^{T'})$. Hence

$$\begin{aligned}\sigma\text{MGU}(\mathcal{E}'^{T'})C'^{GT'} &= \text{MGU}(\mathcal{E}^T)\sigma_x C^{GT} = \text{MGU}(\mathcal{E}^T)C^{GT}\\ \sigma\text{MGU}(\mathcal{E}'^{T'})H'^{GT'} &= \text{MGU}(\mathcal{E}^T)\sigma_x H^{GT} = \text{MGU}(\mathcal{E}^T)H^{GT}\end{aligned}$$

since $C'^G = C^G$ and $H'^G = H^G$, C^G and H^G do not contain $I_1, \ldots, I_h$ so $C^{GT} = C^{GT'}$ and $H^{GT} = H^{GT'}$, and x does not occur in H^G nor in C^G so $\sigma_x C^{GT} = C^{GT}$ and $\sigma_x H^{GT} = H^{GT}$. Therefore, $R'^{GT'} \sqsupseteq R^{GT}$.

5. We have $\mathcal{E} = \mathcal{E}_1 \cup \{C\, f(p_1^G, \ldots, p_k^G) \doteq g(p_1'^G, \ldots, p_m'^G)\}$ and $f \neq g$. Since no substitution can make terms with different root function symbols equal, the unification of $\mathcal{E}^T$ fails and R^{GT} is not defined, so the lemma holds.

6. The clause R'^G differs from R^G only by the set of equations: $\mathcal{E} = \mathcal{E}' \cup \{C\, x_{\iota_1,\ldots,\iota_k} \doteq x_{\iota_1,\ldots,\iota_k}\}$. We choose $T' = T$ as an environment for $\Gamma \vdash R'^G$. If R^{GT} is defined, then $\mathcal{E}^T$ is defined and $\mathcal{E}'^T$ is defined too. Hence,

$$\mathcal{E}^T = \mathcal{E}'^T \cup \{x_{\iota_1,\ldots,\iota_k}^{T''} = x_{\iota_1,\ldots,\iota_k}^{T''} \mid T'' \in T^C\}$$

$\text{MGU}(\mathcal{E}^T)$ and $\text{MGU}(\mathcal{E}'^T)$ have the same solutions: all substitutions are solutions of $x^{T''}_{\iota_1,\ldots,\iota_k} = x^{T''}_{\iota_1,\ldots,\iota_k}$. We can then apply Lemma 34 and conclude.

7. The clause R'^G differs from R^G only by the set of equations: $\mathcal{E}' = \mathcal{E}_1 \cup \{C\, x_{\iota_1,\ldots,\iota_k} \doteq p^G\}$ where $\mathcal{E} = \mathcal{E}_1 \cup \{C\, p^G \doteq x_{\iota_1,\ldots,\iota_k}\}$. We choose $T' = T$ as an environment for $\Gamma \vdash R'^G$. Then, if $\mathcal{E}_1^T$ is defined, we have:

$$\mathcal{E}^T = \mathcal{E}_1^T \cup \{p^{GT''} = x^{T''}_{\iota_1,\ldots,\iota_k} \mid T'' \in T^C\}$$
$$\mathcal{E}'^T = \mathcal{E}_1^T \cup \{x^{T''}_{\iota_1,\ldots,\iota_k} = p^{GT''} \mid T'' \in T^C\}$$

$\text{MGU}(\mathcal{E}^T)$ and $\text{MGU}(\mathcal{E}'^T)$ have the same solutions, so the result follows by Lemma 34.

8. We have

$$\{x_{\iota_1^{T''},\ldots,\iota_k^{T''}} = p^{GT''} \mid T'' \in T^C\} \subseteq \mathcal{E}^T$$

and, for all $T'' \in T^C$, the variable $x_{\iota_1^{T''},\ldots,\iota_k^{T''}}$ occurs in $p^{GT''}$ and $p^{GT''} \neq x_{\iota_1^{T''},\ldots,\iota_k^{T''}}$, so there is no substitution σ such that $\sigma x_{\iota_1^{T''},\ldots,\iota_k^{T''}} = \sigma p^{GT''}$. Therefore, $\text{MGU}(\mathcal{E}^T)$ is not defined and R^{GT} is not defined either, so the lemma holds.

9. The clause R'^G differs from R^G only by the set of equations: $\mathcal{E} = \mathcal{E}' \cup \{C\,\iota \doteq \iota\}$. We choose $T' = T$ as an environment for $\Gamma \vdash R'^G$. If R^{GT} is defined, then $\mathcal{E}^T$ is defined and $\mathcal{E}'^T$ is defined too. Moreover $\mathcal{E}^T = \mathcal{E}'^T$. Hence, we can apply Lemma 34 and conclude.

10. We have that $\mathcal{E} = \mathcal{E}_1 \cup \{\bigwedge_{(i_1,\ldots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} x_{\iota_1,\ldots,\iota_{h'}} \doteq p'^G\} = \mathcal{E}_1 \cup \mathcal{E}_0$. To obtain R'^G, we replace $x_{\iota_1,\ldots,\iota_{h'}}\{i_1 \mapsto \iota'_1,\ldots,i_h \mapsto \iota'_h\}$ with $p'^G\{i_1 \mapsto \iota'_1,\ldots,i_h \mapsto \iota'_h\}$ everywhere else in the clause and obtain: $R'^G = Cts, H'^G \wedge (\mathcal{E}_0 \cup \mathcal{E}'_1) \to C'^G$. Clearly, $T' = T$ is an environment for $\Gamma \vdash R'^G$. We have

$$\mathcal{E}_0^T = \{x_{\iota_1^{T''},\ldots,\iota_{h'}^{T''}} = p'^{GT''} \mid T'' = T[i_1 \mapsto v_1,\ldots,i_h \mapsto v_h],$$
$$(v_1,\ldots,v_h) \in \{1,\ldots,L_1^T\} \times \cdots \times \{1,\ldots L_h^T\}\}$$

Let $\sigma_0 = \text{MGU}(\mathcal{E}_0^T)$. For all $(v_1,\ldots,v_h) \in \{1,\ldots,L_1^T\} \times \cdots \times \{1,\ldots L_h^T\}$, we have $\sigma_0 x_{\iota_1^{T''},\ldots,\iota_{h'}^{T''}} = \sigma_0 p'^{GT''}$, where $T'' = T[i_1 \mapsto v_1,\ldots,i_h \mapsto v_h]$. When $x_{\iota_1,\ldots,\iota_{h'}}\{i_1 \mapsto \iota'_1,\ldots,i_h \mapsto \iota'_h\}$ occurs under a conjunction C in R^G, we have for all $T'' \in T^C$, $(\iota_1'^{T''},\ldots,\iota_h'^{T''}) \in \{1,\ldots,L_1^T\} \times \cdots \times \{1,\ldots L_h^T\}$, so $\sigma_0(x_{\iota_1,\ldots,\iota_{h'}}\{i_1 \mapsto \iota'_1,\ldots,i_h \mapsto \iota'_h\})^{T''} = (p'^G\{i_1 \mapsto \iota'_1,\ldots,i_h \mapsto \iota'_h\})^{T''}$. Hence, $\sigma_0 H'^{GT} = \sigma_0 H^{GT}$, $\sigma_0 \mathcal{E}_1'^T = \sigma_0 \mathcal{E}_1^T$, and $\sigma_0 C'^{GT} = \sigma_0 C^{GT}$. Now, we have that:

$$\begin{aligned}
\text{MGU}(\mathcal{E}_1'^T \cup \mathcal{E}_0^T) H'^{GT} &= \text{MGU}(\text{MGU}(\mathcal{E}_0^T)\mathcal{E}_1'^T)\text{MGU}(\mathcal{E}_0^T) H'^{GT} && \text{by Lemma 32}\\
&= \text{MGU}(\sigma_0 \mathcal{E}_1'^T)\sigma_0 H'^{GT}\\
&= \text{MGU}(\sigma_0 \mathcal{E}_1^T)\sigma_0 H^{GT}\\
&= \text{MGU}(\text{MGU}(\mathcal{E}_0^T)\mathcal{E}_1^T)\text{MGU}(\mathcal{E}_0^T) H^{GT}\\
&= \text{MGU}(\mathcal{E}_1^T \cup \mathcal{E}_0^T) H^{GT} && \text{by Lemma 32}
\end{aligned}$$

In the same way, we can prove that $\text{MGU}(\mathcal{E}_1'^T \cup \mathcal{E}_0^T) C'^{GT} = \text{MGU}(\mathcal{E}_1^T \cup \mathcal{E}_0^T) C^{GT}$ and then conclude.

11. Similar to the previous case.

12. We have that: $\mathcal{E} = \mathcal{E}_1 \cup \{i \doteq \iota\} = \mathcal{E}_1 \cup \mathcal{E}_0$. To obtain R'^G, we replace i with ι everywhere else in the clause, and obtain: $R'^G = Cts, H'^G \wedge (\mathcal{E}_0 \cup \mathcal{E}'_1) \rightarrow C'^G$. Clearly, $T' = T$ is an environment for $\Gamma \vdash R'^G$. If R^{GT} is defined, then $\mathcal{E}_0^T$ is defined, so $i^T = \iota^T$. Hence $R'^{GT} = R^{GT}$.

13. We have that: $\mathcal{E} = \mathcal{E}_1 \cup \{\bigwedge_{(i_1,\ldots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} \iota \doteq \iota'\} = \mathcal{E}_1 \cup \mathcal{E}_0$. To obtain R'^G, we replace $\iota'\{i_1 \mapsto \iota'_1, \ldots, i_h \mapsto \iota'_h\}$ with $\iota\{i_1 \mapsto \iota'_1, \ldots, i_h \mapsto \iota'_h\}$ everywhere else in the clause, and obtain: $R'^G = Cts, H'^G \wedge (\mathcal{E}_0 \cup \mathcal{E}'_1) \rightarrow C'^G$. Clearly, $T' = T$ is an environment for $\Gamma \vdash R'^G$. If R^{GT} is defined, then $\mathcal{E}_0^T$ is defined, so $\iota'^{T[i_1\mapsto v_1,\ldots,i_h\mapsto v_h]} = \iota'^{T[i_1\mapsto v_1,\ldots,i_h\mapsto v_h]}$ for any $(v_1, \ldots, v_h) \in \{1, \ldots, L_1^T\}\times\cdots\times\{1, \ldots, L_h^T\}$. When $\iota\{i_1 \mapsto \iota'_1, \ldots, i_h \mapsto \iota'_h\}$ occurs under a conjunction C in R^G, we have for all $T'' \in T^C$, $(\iota'^{T''}_1, \ldots, \iota'^{T''}_h) \in \{1, \ldots, L_1^T\} \times \cdots \times \{1, \ldots L_h^T\}$, so $(\iota'\{i_1 \mapsto \iota'_1, \ldots, i_h \mapsto \iota'_h\})^{T''} = (\iota\{i_1 \mapsto \iota'_1, \ldots, i_h \mapsto \iota'_h\})^{T''}$. Hence $R'^{GT} = R^{GT}$.

    In case we remove the equation $\bigwedge_{(i_1,\ldots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} \iota \doteq \phi(\iota'_1, \ldots, \iota'_{h'})$, the clause R'^G differs from R^G only by the set of equations: $\mathcal{E} = \mathcal{E}' \cup \{C\,\iota \doteq \phi(\iota_1, \ldots, \iota_h)\}$. We choose $T' = T$ as an environment for $\Gamma \vdash R'^G$. If R^{GT} is defined, then $\mathcal{E}^T$ is defined and $\mathcal{E}'^T$ is defined too. Clearly, $\text{MGU}(\mathcal{E}^T) = \text{MGU}(\mathcal{E}'^T)$: we can apply Lemma 34 and conclude.

14. Similar to the previous case.

15. We have that R'^G differs from R^G only by the set of equations: $\mathcal{E} = \mathcal{E}' \cup \{C\,x_{i_1,\ldots i_h} \doteq p'^G\}$. We choose $T' = T$ as an environment for $\Gamma \vdash R'^G$. If R^{GT} is defined, then $\mathcal{E}^T$ is defined and $\mathcal{E}'^T$ is defined too. We have that:

$$\mathcal{E}^T = \mathcal{E}'^T \cup \{x_{i_1^{T'},\ldots,i_h^{T'}} = p'^{GT'} \mid T' \in T^C\}.$$

    As x does not occur anywhere else in R^G (and R'^G), x does not occur in $\mathcal{E}'$ nor in p'^G, so $\text{MGU}(\mathcal{E}^T) = \text{MGU}(\mathcal{E}'^T)\{x_{i_1^{T'},\ldots,i_h^{T'}} \mapsto p'^{GT'} \mid T' \in T^C\}$. Since x does not occur in H^G nor in C^G, we have that:

$$\begin{aligned}\text{MGU}(\mathcal{E}^T)H^{GT} &= \text{MGU}(\mathcal{E}'^T)H^{GT} = \text{MGU}(\mathcal{E}'^T)H'^{GT}\\ \text{MGU}(\mathcal{E}^T)C^{GT} &= \text{MGU}(\mathcal{E}'^T)C^{GT} = \text{MGU}(\mathcal{E}'^T)C'^{GT}\end{aligned}$$

    so $R'^{GT} = R^{GT}$.

16. When there are unused indices in a conjunction, the translation generates several copies of the same fact or equation under that conjunction, one for each value of the unused indices. Removing the unused indices from the conjunction just removes these duplicate facts or equations. It does not change the value of $\text{MGU}(\mathcal{E}^T)$, but may remove duplicates from H^{GT}. Therefore, $R'^{GT} \sqsupseteq R^{GT}$.

17. We have

$$H^G = H_0^G \cup \{ \bigwedge_{(i_1,\ldots,i_h)\in[1,L_1]\times\cdots\times[1,L_h]} \mathrm{att}(x_{i_1,\ldots,i_h})\}$$
$$H'^G = H_0^G \cup \{C_k\, \mathrm{att}(p_k^G) \mid k = 1,\ldots,K\}$$
$$\mathcal{E} = \mathcal{E}' \supseteq \{C_k\, x_{\iota_{k,1},\ldots,\iota_{k,h}} \doteq p_k^G \mid k = 1,\ldots,K\}$$

We choose $T' = T$ as an environment for $\Gamma \vdash R'^G$. Therefore

$$\begin{aligned}
&\mathrm{MGU}\,(\mathcal{E}^T)H^{GT} = \mathrm{MGU}\,(\mathcal{E}^T)H_0^{GT} \cup \\
&\qquad \{\mathrm{att}(\mathrm{MGU}\,(\mathcal{E}^T)x_{v_1,\ldots,v_h}) \mid (v_1,\ldots,v_h) \in [1,L_1^T]\times\cdots\times[1,L_h^T]\} \\
&\mathrm{MGU}\,(\mathcal{E}'^{T'})H'^{GT'} \\
&\quad = \mathrm{MGU}\,(\mathcal{E}^T)H_0^{GT} \cup \{\mathrm{att}(\mathrm{MGU}\,(\mathcal{E}^T)p_k^{GT''}) \mid T'' \in T^{C_k}, k = 1,\ldots,K\} \\
&\quad = \mathrm{MGU}\,(\mathcal{E}^T)H_0^{GT} \cup \{\mathrm{att}(\mathrm{MGU}\,(\mathcal{E}^T)x_{\iota_{k,1}^{T''},\ldots,\iota_{k,h}^{T''}}) \mid T'' \in T^{C_k}, k = 1,\ldots,K\}
\end{aligned}$$

since $\mathrm{MGU}\,(\mathcal{E}^T)$ is a unifier for the equations $x_{\iota_{k,1}^{T''},\ldots,\iota_{k,h}^{T''}} \doteq p_k^{GT''}$ for $T'' \in T^{C_k}$, $k = 1,\ldots,K$. In order to show that $R'^{GT'} \sqsupseteq R^{GT}$, we notice that $\mathrm{MGU}\,(\mathcal{E}'^{T'})C'^{GT'} = \mathrm{MGU}\,(\mathcal{E}^T)C^{GT}$, since $\mathcal{E}' = \mathcal{E}$, $C'^G = C^G$, and $T' = T$, and we show that $\mathrm{MGU}\,(\mathcal{E}'^{T'})H'^{GT'} \subseteq \mathrm{MGU}\,(\mathcal{E}^T)H^{GT}$ (multiset inclusion). The latter inclusion holds as soon as the multiset $\{(\iota_{k,1}^{T''},\ldots,\iota_{k,h}^{T''}) \mid T'' \in T^{C_k}, k = 1,\ldots,K\}$ does not contain duplicate elements. This multiset is equal to the multiset union $I_1 \cup \ldots \cup I_K$ where $I_k = \{(\iota_{k,1}^{T''},\ldots,\iota_{k,h}^{T''}) \mid T'' \in T^{C_k}\}$ for $k = 1,\ldots,K$.

Suppose that I_k contains a duplicate element. Thus, there exist $T_1'' \neq T_2''$ in T^{C_k} such that $(\iota_{k,1}^{T_1''},\ldots,\iota_{k,h}^{T_1''}) = (\iota_{k,1}^{T_2''},\ldots,\iota_{k,h}^{T_2''})$. For each index i bound by C_k, there exists $l \leq h$ such that $\iota_{k,l} = i$. Therefore, $i^{T_1''} = i^{T_2''}$. T_1'' and T_2'' are two extensions of T with values of indices bound by C_k and nothing else, so $T_1'' = T_2''$. Contradiction. Hence for $k = 1,\ldots,K$, I_k does not contain duplicate elements.

Suppose that I_k and $I_{k'}$ contain a common element. Thus, there exist $T_1'' \in T^{C_k}$ and $T_2'' \in T^{C_{k'}}$ such that $(\iota_{k,1}^{T_1''},\ldots,\iota_{k,h}^{T_1''}) = (\iota_{k',1}^{T_2''},\ldots,\iota_{k',h}^{T_2''})$. Hence $\mathrm{MGU}\,(\mathcal{E}^T)x_{\iota_{k,1}^{T_1''},\ldots,\iota_{k,h}^{T_1''}} = \mathrm{MGU}\,(\mathcal{E}^T)x_{\iota_{k',1}^{T_2''},\ldots,\iota_{k',h}^{T_2''}}$, so $\mathrm{MGU}\,(\mathcal{E}^T)p_k^{GT_1''} = \mathrm{MGU}\,(\mathcal{E}^T)p_{k'}^{GT_2''}$. Contradiction since p_k^G and $p_{k'}^G$ cannot unify. Therefore, I_k and $I_{k'}$ contain no common element.

Hence $I_1 \cup \ldots \cup I_K$ does not contain duplicate elements, which concludes the proof. □

A.2.3.2 Proof of Lemma 22

Proof. The merging of sets actually consists of three transformations:

1. When R^G contains constraints $I_1 \uplus \cdots \uplus I_h = I$ and $I_1' \uplus \cdots \uplus I_{h'}' = J$ with $I = I_k'$, and I does not occur elsewhere, we replace these constraints with $I_1' \uplus \cdots \uplus I_{k-1}' \uplus I_1 \uplus$

$\cdots \uplus I_h \uplus I'_{k+1} \uplus \cdots \uplus I'_{h'} = J$. Since R^{GT} is defined, we have $I_1^T \uplus \cdots \uplus I_h^T = I^T$ and $I_1'^T \uplus \cdots \uplus I_{h'}'^T = J^T$, so $I_1'^T \uplus \cdots \uplus I_{k-1}'^T \uplus I_1^T \uplus \cdots \uplus I_h^T \uplus I_{k+1}'^T \uplus \cdots \uplus I_{h'}'^T = J^T$. Hence R'^{GT} is also defined and $R'^{GT'} = R^{GT}$.

2. When R^G contains the constraint $I_1 = J$, we replace I_1 with J in the obtained clause, and delete the constraint. Since R^{GT} is defined, we have $I_1^T = J^T$, so replacing I_1 with J does not change the translation, and $R'^{GT} = R^{GT}$.

3. Finally, consider the actual merging of I_1 and I_2 when R^G contains a constraint $I_1 \uplus \cdots \uplus I_h = J$. Clearly, for the obtained clause R'^G, we have $\Gamma \vdash R'^G$. Let T be an environment for $\Gamma \vdash R^G$ such that R^{GT} is defined. Let us construct the environment T' for $\Gamma \vdash R'^G$ such that $R'^{GT'} =^\alpha R^{GT}$. We define T' exactly as T for the functions ϕ, sets I, and bounds L not renamed by α, and for free indices. For $I \in S_1$, we define $I^{T'} = I^T \cup (\alpha I)^T$. When a bound L is renamed by α, we define $L^{T'} = \max(L^T, (\alpha L)^T)$. When a variable renamed by α occurs under a conjunction $\bigwedge_{(i_1,\ldots,i_k,j_1,\ldots,j_n) \in I \times [1,L_1] \times \cdots \times [1,L_n]}$ in H_1^G or $\mathcal{E}_1$, it is of the form $x_{i_1,\ldots,i_k,\ldots}$, so that $x_{i_1^T,\ldots,i_k^T,\ldots}$ is used only for $(i_1^T, \ldots, i_k^T) \in I^T$ in R^{GT}. Moreover, $I \in S_1$, and an easy induction shows that for all $I \in S_1$, $I^T \subseteq I_1^T$, so $x_{i_1^T,\ldots,i_k^T,\ldots}$ is used only for $(i_1^T, \ldots, i_k^T) \in I_1^T$ in R^{GT}. Similarly, in H_2^G and $\mathcal{E}_2$, $(\alpha x)_{i_1^T,\ldots,i_k^T,\ldots}$ is used only for $(i_1^T, \ldots, i_k^T) \in I_2^T$. Moreover, I_1^T and I_2^T are disjoint by the constraint $I_1 \uplus \cdots \uplus I_h = J$, so we can replace αx with x without introducing a clash of variables in R^{GT}. Similarly, when ϕ is renamed by α, ϕ and $\alpha\phi$ are also used on disjoint sets of arguments (ϕ is used with its first k arguments in I_1^T and $\alpha\phi$ is used with its first k arguments in I_2^T), so we can define $\phi^{T'}$ as ϕ^T when its first k arguments are in I_1^T and as $(\alpha\phi)^T$ when its first k arguments are in I_2^T, and $\phi^{T'}$ can take any value on the rest of its domain. (The domain of $\phi^{T'}$ may be larger than the domain of ϕ^T since the translation of renamed bounds may be larger. The values of $\phi^{T'}$ outside the domain of ϕ^T are in fact not used. Since the bounds $L^{T'}$ are larger than L^T and $(\alpha L)^T$, the result of $\phi^{T'}$ remains in the interval corresponding to its type.) We can then replace $\alpha\phi$ with ϕ. Then, it is easy to check that $R'^{GT'} =^\alpha R^{GT}$.

It is then easy to see that the lemma holds for any composition of these three transformations. □

A.2.4 Proofs of Section 9.5

Lemma 26 is an immediate consequence of the following lemma.

Lemma 35. *Let $\Gamma \vdash P^G$ be a well-typed generalized process, such that the bound names of P^G are pairwise distinct and distinct from free names of P^G. Given an environment T for $\Gamma \vdash P^G$, and a list of indices $\widetilde{i} \leq \widetilde{L}$, we have:*

$$\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}) \equiv_\alpha P^{GT} .$$

Furthermore, we have the following two properties:

P1. The bound names in $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})$ are pairwise distinct and distinct from free names, except that in processes $P+Q$, the bound names in P need not be distinct from those in Q.

P2. All bound names in $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})$ are of the form $a^{\widetilde{L^T},\ldots}_{\widetilde{i^T},\ldots}$ when they come from (νa) in P^G and of the form $a^{L^T,\widetilde{L^T},\ldots}_{v,\widetilde{i^T},\ldots}$ when they come from $(\mathsf{for\ all}\ i \leq L, \nu a_i)$ in P^G.

Proof. The property $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}) \equiv_\alpha P^{GT}$ is proved by an easy induction on the syntax of P^G.

Properties P1 and P2 are proved by simultaneous induction on the syntax of P^G.

- Case $\Pi_{i \leq L} P^G$: for each $v \leq L^T$, by induction hypothesis, the bound names in $\mathrm{Tren}(P^G, T[i \mapsto v], (\widetilde{i}, i) \leq (\widetilde{L}, L))$ are pairwise distinct (except that in processes $P + Q$, the bound names in P need not be distinct from those in Q) and distinct from free names. Furthermore, they are of the form $a^{\widetilde{L^T},L^T,\ldots}_{\widetilde{i^T},v,\ldots}$ when they come from (νa) in P^G and of the form $a^{L'^T,\widetilde{L^T},L^T,\ldots}_{v',\widetilde{i^T},v,\ldots}$ when they come from $(\mathsf{for\ all}\ i' \leq L', \nu a_{i'})$ in P^G, so P2 holds. Hence the names $\mathrm{Tren}(P^G, T[i \mapsto v], (\widetilde{i}, i) \leq (\widetilde{L}, L))$ are distinct from the names in $\mathrm{Tren}(P^G, T[i \mapsto v'], (\widetilde{i}, i) \leq (\widetilde{L}, L))$ when $v \neq v'$, so P1 holds.
- Case $(\mathsf{for\ all}\ i \leq L, \nu a_i)P^G$: by induction hypothesis, the bound names in $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})$ are pairwise distinct (except that in processes $P + Q$, the bound names in P need not be distinct from those in Q) and distinct from free names. Furthermore, they are of the form $a'^{\widetilde{L^T},\ldots}_{\widetilde{i^T},\ldots}$ when they come from $(\nu a')$ in P^G and of the form $a'^{L^T,\widetilde{L^T},\ldots}_{v,\widetilde{i^T},\ldots}$ when they come from $(\mathsf{for\ all}\ i \leq L, \nu a'_i)$ in P^G. The new bound names $a^{L^T,\widetilde{L^T}}_{v,\widetilde{i^T}}$ for $v \leq L^T$ are of the required form, so P2 holds. They are distinct from the free names and from the bound names of $\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})$, since the bound names in $(\mathsf{for\ all}\ i \leq L, \nu a_i)P^G$ are pairwise distinct and distinct from free names, so they do not use the same symbol a. So P1 holds.

- The case $(va)P^G$ is similar to the previous one. All other cases follow easily using the induction hypothesis. We use the property that the bound names of P^G are pairwise distinct and distinct from free names of P^G. In the cases "choose", we also use that in processes $P + Q$, the bound names in P need not be distinct from those in Q, so the induction hypothesis already guarantees that names are distinct when desired. □

A.2.5 Proofs of Section 9.6

A.2.5.1 Proof of Lemma 30

Lemma 30 comes immediately from the following lemma applied to $\text{instr}^G(P_0^G) = \text{instr}^G(P_0^G, \emptyset, \emptyset, \emptyset \leq \emptyset)$.

Lemma 36. *Given a well-typed generalized process $\Gamma \vdash P^G$, an environment T for $\Gamma \vdash P^G$, a list of variables $Vars = x_1, \ldots x_n$, a list of session identifiers $SessId = s_1, \ldots, s_{n'}$, and a list of indices $\widetilde{i} \leq \widetilde{L}$, we have:*

$$(\text{instr}^G(P^G, Vars, SessId, \widetilde{i} \leq \widetilde{L}))^T \equiv_\alpha \text{instr}(\text{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), Vars, SessId)\,.$$

Proof of Lemma 36. This proof is done by structural induction on the process P^G. We detail here the most interesting cases.

- Case $\mathsf{in}(M^G, x).P^G$:

$$\begin{aligned}
&(\text{instr}^G(\mathsf{in}(M^G, x).P^G, Vars, SessId, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = (\mathsf{in}(M^G, x).\text{instr}^G(P^G, (Vars, x), SessId, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = \mathsf{in}(M^{GT}, x).(\text{instr}^G(P^G, (Vars, x), SessId, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad \equiv_\alpha \mathsf{in}(M^{GT}, x).\text{instr}(\text{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), (Vars, x), SessId) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by induction hypothesis} \\
&\quad \equiv_\alpha \text{instr}(\mathsf{in}(M^{GT}, x).\text{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), Vars, SessId) \\
&\quad \equiv_\alpha \text{instr}(\text{Tren}(\mathsf{in}(M^G, x).P^G, T, \widetilde{i} \leq \widetilde{L}), Vars, SessId)
\end{aligned}$$

- Case $!P^G$:

$$\begin{aligned}
&(\text{instr}^G(!P^G, Vars, SessId, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = (!^s\text{instr}^G(P^G, Vars, (SessId, s), \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = !^s(\text{instr}^G(P^G, Vars, (SessId, s), \widetilde{i} \leq \widetilde{L}))^T \\
&\quad \equiv_\alpha !^s\text{instr}(\text{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), Vars, (SessId, s)) \qquad \text{by induction hypothesis} \\
&\quad \equiv_\alpha \text{instr}(!\text{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), Vars, SessId) \\
&\quad \equiv_\alpha \text{instr}(\text{Tren}(!P^G, T, \widetilde{i} \leq \widetilde{L}), Vars, SessId)
\end{aligned}$$

- Case $\Pi_{i\leq L}P^G$:

$$\begin{aligned}
&(\mathrm{instr}^G(\Pi_{i\leq L}P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = (\Pi_{i\leq L}\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, (\widetilde{i}, i) \leq (\widetilde{L}, L)))^T \\
&\quad = (\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, (\widetilde{i}, i) \leq (\widetilde{L}, L)))^{T[i\mapsto 1]} \mid \cdots \mid \\
&\qquad (\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, (\widetilde{i}, i) \leq (\widetilde{L}, L)))^{T[i\mapsto L^T]}
\end{aligned}$$

  For each $v \leq L^T$, we have by induction hypothesis:

$$\begin{aligned}
&\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, (\widetilde{i}, i) \leq (\widetilde{L}, L))^{T[i\mapsto v]} \equiv_\alpha \\
&\quad \mathrm{instr}(\mathrm{Tren}(P^G, T[i \mapsto v], (\widetilde{i}, i) \leq (\widetilde{L}, L)), \mathit{Vars}, \mathit{SessId}).
\end{aligned}$$

  Hence:

$$\begin{aligned}
&(\mathrm{instr}^G(\Pi_{i\leq L}P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(P^G, T[i \mapsto 1], (\widetilde{i}, i) \leq (\widetilde{L}, L)), \mathit{Vars}, \mathit{SessId}) \mid \cdots \mid \\
&\qquad \mathrm{instr}(\mathrm{Tren}(P^G, T[i \mapsto L^T], (\widetilde{i}, i) \leq (\widetilde{L}, L)), \mathit{Vars}, \mathit{SessId}) \\
&\quad \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(P^G, T[i \mapsto 1], (\widetilde{i}, i) \leq (\widetilde{L}, L)) \mid \cdots \mid \\
&\qquad\quad \mathrm{Tren}(P^G, T[i \mapsto L^T], (\widetilde{i}, i) \leq (\widetilde{L}, L)), \mathit{Vars}, \mathit{SessId}) \\
&\quad \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(\Pi_{i\leq L}P^G, T, \widetilde{i} \leq \widetilde{L}), \mathit{Vars}, \mathit{SessId})
\end{aligned}$$

- Case $(\mathsf{for\ all}\ i \leq L, va_i)P^G$:

$$\begin{aligned}
&(\mathrm{instr}^G((\mathsf{for\ all}\ i \leq L, va_i)P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = ((\mathsf{for\ all}\ i \leq L, va_i : a^{L,\widetilde{L}}_{i,\widetilde{i}}[\mathit{Vars}, \mathit{SessId}]) \\
&\qquad \mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\quad = (va_1 : a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \ldots (va_{L^T} : a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \\
&\qquad (\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T
\end{aligned}$$

  Moreover, by induction hypothesis, $(\mathrm{instr}^G(P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \equiv_\alpha \mathrm{instr}(\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), \mathit{Vars}, \mathit{SessId})$. Therefore,

$$\begin{aligned}
&(\mathrm{instr}^G((\mathsf{for\ all}\ i \leq L, va_i)P^G, \mathit{Vars}, \mathit{SessId}, \widetilde{i} \leq \widetilde{L}))^T \\
&\equiv_\alpha (va_1 : a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \ldots (va_{L^T} : a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \\
&\qquad \mathrm{instr}(\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), \mathit{Vars}, \mathit{SessId}) \\
&\equiv_\alpha (va^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T} : a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \ldots (va^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T} : a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[\mathit{Vars}, \mathit{SessId}]) \\
&\qquad (\mathrm{instr}(\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L}), \mathit{Vars}, \mathit{SessId})\{a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}/a_1, \ldots, a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}/a_{L^T}\}) \\
&\qquad\qquad \text{by renaming bound names} \\
&\equiv_\alpha \mathrm{instr}((va^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}) \ldots (va^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T})\mathrm{Tren}(P^G, T, \widetilde{i} \leq \widetilde{L})\{a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}/a_1, \ldots, a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}/a_{L^T}\}, \\
&\qquad \mathit{Vars}, \mathit{SessId}) \\
&\equiv_\alpha \mathrm{instr}(\mathrm{Tren}((\mathsf{for\ all}\ i \leq L, va_i)P^G, T, \widetilde{i} \leq \widetilde{L}), \mathit{Vars}, \mathit{SessId})
\end{aligned}$$

- The case $(va)P^G$ can be handled similarly to the previous case. All other cases follow easily from the induction hypothesis. □

A.2.5.2 Proof of Lemma 31

Lemma 37. *Let P be an instrumented process, ρ a function that associates a clause term with each name and variable, and H a sequence of facts. Given a substitution σ over the variables in ρ, we have that:*

$$[\![P]\!](\sigma\rho)(\sigma H) \sqsubseteq [\![P]\!]\rho H\,.$$

Proof of Lemma 37. The proof of this lemma is done by structural induction on the process P. We detail here the most interesting cases.

- Case $M(x).P$:

$$\begin{aligned}
&[\![M(x).P]\!](\sigma\rho)(\sigma H)\\
&\quad = [\![P]\!]((\sigma\rho)[x \mapsto x])(\sigma H \wedge \text{message}(\sigma\rho(M), x))\\
&\quad = [\![P]\!](\sigma'(\rho[x \mapsto x]))(\sigma'(H \wedge \text{message}(\rho(M), x)))\\
&\qquad\qquad \text{where we define the substitution } \sigma' = \sigma[x \mapsto x]\\
&\quad \sqsubseteq [\![P]\!](\rho[x \mapsto x])(H \wedge \text{message}(\rho(M), x)) \qquad \text{by induction hypothesis}\\
&\quad \sqsubseteq [\![M(x).P]\!]\rho H
\end{aligned}$$

- Case $\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q$:

$$\begin{aligned}
&[\![\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!](\sigma\rho)(\sigma H)\\
&\quad = [\![Q]\!](\sigma\rho)(\sigma H) \cup \bigcup\{[\![P]\!](\sigma_1\sigma\rho[x \mapsto \sigma'_1 p'])(\sigma_1\sigma H) \mid g(p'_1, \ldots, p'_n) \to p'\\
&\qquad \text{is in } def(g) \text{ and } (\sigma_1, \sigma'_1) \text{ is a most general pair of substitutions}\\
&\qquad \text{such that } \sigma_1\sigma\rho(M_i) = \sigma'_1 p'_i, \text{for each } i = 1, \ldots n\}
\end{aligned}$$

By induction hypothesis, we have $[\![Q]\!](\sigma\rho)(\sigma H) \sqsubseteq [\![Q]\!]\rho H$. Let $g(p'_1, \ldots, p'_n) \to p'$ be a rule in $def(g)$, and (σ_1, σ'_1) be a most general pair of substitutions such that $\sigma_1\sigma\rho(M_i) = \sigma'_1 p'_i$, for each $i = 1, \ldots n$. Let $\sigma_2 = \sigma_1\sigma$ and $\sigma'_2 = \sigma'_1$. For each $i = 1, \ldots, n$, we have $\sigma_2\rho(M_i) = \sigma'_2 p'_i$. Let (σ_3, σ'_3) be a most general pair of substitutions such that for each $i = 1, \ldots, n$: $\sigma_3\rho(M_i) = \sigma'_3 p'_i$. As (σ_2, σ'_2) is such a pair (but maybe not a most general one), there exists a substitution σ_4 such that $\sigma_2 = \sigma_4\sigma_3$ and $\sigma'_2 = \sigma_4\sigma'_3$. Hence we have that

$$\begin{aligned}
&[\![P]\!](\sigma_1\sigma\rho[x \mapsto \sigma'_1 p'])(\sigma_1\sigma H)\\
&\quad = [\![P]\!](\sigma_4\sigma_3\rho[x \mapsto \sigma_4\sigma'_3 p'])(\sigma_4\sigma_3\sigma H)\\
&\quad = [\![P]\!](\sigma_4(\sigma_3\rho[x \mapsto \sigma'_3 p']))(\sigma_4(\sigma_3\sigma H))\\
&\quad \sqsubseteq [\![P]\!](\sigma_3\rho[x \mapsto \sigma'_3 p'])(\sigma_3\sigma H)
\end{aligned}$$

by induction hypothesis. Therefore,

$$\begin{aligned}
&[\![\mathsf{let}\ x = g(M_1,\ldots,M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!](\sigma\rho)(\sigma H)\\
&\quad\sqsubseteq [\![Q]\!]\rho H \cup \bigcup\{[\![P]\!](\sigma_3\rho[x \mapsto \sigma'_3 p'])(\sigma_3\sigma H) \mid g(p'_1,\ldots,p'_n) \to p'\\
&\qquad\text{is in } \mathit{def}(g) \text{ and } (\sigma_3,\sigma'_3) \text{ is a most general pair of substitutions}\\
&\qquad\text{such that } \sigma_3\rho(M_i) = \sigma'_3 p'_i, \text{for each } i = 1,\ldots n\}\\
&\quad\sqsubseteq [\![\mathsf{let}\ x = g(M_1,\ldots,M_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H.
\end{aligned}$$

- The other cases are straightforward using the induction hypothesis. □

Lemma 38. *We have*

$$\begin{aligned}
&[\![\mathsf{let}\ E_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ E_l\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho H\\
&\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\textsc{mgu}\,(\mathcal{E})(\rho[x_1 \mapsto p'_1,\ldots,x_l \mapsto p'_l]))(\textsc{mgu}\,(\mathcal{E})H)
\end{aligned}$$

where

- *for each $i \le l$, E_i is $x_i = g_i(M_{i,1},\ldots,M_{i,n_i})$;*
- *for each $i \le l$, x_i does not occur in Q nor in $M_{k,j}$ for all $k = 1,\ldots,l$ and $j = 1,\ldots,n_k$;*
- *for each $i \le l$, $g_i(p^0_{i,1},\ldots,p^0_{i,n_i}) \to p'^0_i$ is the rewriting rule of g_i and $p_{i,1},\ldots,p_{i,n_i},p'_i$ are obtained by renaming $p^0_{i,1},\ldots,p^0_{i,n_i},p'^0_i$ with fresh variables;*
- *$\mathcal{E} = \{\rho(M_{k,j}) = p_{k,j} \mid k = 1,\ldots,l \text{ and } j = 1,\ldots,n_k\}$.*

When the equations in $\mathcal{E}$ cannot be unified, $\textsc{mgu}\,(\mathcal{E})$ is not defined, and the second component of the union is omitted.

Proof. The proof is done by induction on l.

- Base case: $l = 1$.

  $$[\![\mathsf{let}\ E_1\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H = [\![Q]\!]\rho H \cup [\![P]\!](\sigma\rho[x_1 \mapsto \sigma p'_1])(\sigma H)$$

  where σ is a most general substitution such that $\sigma\rho(M_{1,j}) = \sigma p_{1,j}$ for each $j = 1,\ldots,n_1$, assuming that σ exists. (Finding such a σ is equivalent to finding a most general pair of substitutions (σ',σ'') such that $\sigma'\rho(M_{1,j}) = \sigma'' p^0_{1,j}$: we can define σ by $\sigma x = \sigma''\alpha^{-1}x$ where α is the renaming of $p^0_{i,j}$ into $p_{i,j}$ and x is a fresh variable introduced by this renaming, and $\sigma x = \sigma' x$ otherwise.) Hence $\sigma = \textsc{mgu}\,(\mathcal{E})$ where $\mathcal{E} = \{\rho(M_{1,j}) = p_{1,j} \mid j = 1,\ldots,n_1\}$ and we can conclude that

  $$[\![\mathsf{let}\ E_1\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H = [\![Q]\!]\rho H \cup [\![P]\!](\textsc{mgu}\,(\mathcal{E})(\rho[x_1 \mapsto p'_1]))(\textsc{mgu}\,(\mathcal{E})H)$$

  When $\textsc{mgu}\,(\mathcal{E})$ is not defined, that is, σ does not exist, the second component of the union is omitted.

- Inductive step. We have

$$\begin{aligned}
&[\![\mathsf{let}\ E_1\ \mathsf{in}\ \mathsf{let}\ E_2\ \mathsf{in}\ \ldots\ \mathsf{let}\ E_l\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q\ \mathsf{else}\ Q]\!]\rho H\\
&\quad = [\![Q]\!]\rho H \cup [\![\mathsf{let}\ E_2\ \mathsf{in}\ \ldots\ \mathsf{let}\ E_l\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho_1 H_1\\
&\qquad \text{where } \rho_1 = \textsc{mgu}\,(\mathcal{E}_1)(\rho[x_1 \mapsto p'_1]),\ H_1 = \textsc{mgu}\,(\mathcal{E}_1)H, \text{ and}\\
&\qquad \mathcal{E}_1 = \{\rho(M_{1,j}) = p_{1,j} \mid j = 1,\ldots,n_1\}, \text{ by the base case}\\
&\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![Q]\!]\rho_1 H_1 \cup\\
&\qquad [\![P]\!](\textsc{mgu}\,(\mathcal{E}_2)(\rho_1[x_2 \mapsto p'_2,\ldots,x_l \mapsto p'_l]))(\textsc{mgu}\,(\mathcal{E}_2)H_1)
\end{aligned}$$

  where $\mathcal{E}_2 = \{\rho_1(M_{k,j}) = p_{k,j} \mid k = 2,\ldots,l \text{ and } j = 1,\ldots,n_k\}$, by induction hypothesis, assuming that $\textsc{mgu}\,(\mathcal{E}_1)$ and $\textsc{mgu}\,(\mathcal{E}_2)$ are defined. We have $[\![Q]\!]\rho_1 H_1 = [\![Q]\!](\textsc{mgu}\,(\mathcal{E}_1)\rho)(\textsc{mgu}\,(\mathcal{E}_1)H)$ since x_1 does not occur in Q, so $[\![Q]\!]\rho_1 H_1 \sqsubseteq [\![Q]\!]\rho H$ by Lemma 37.

  Let $\mathcal{E}'_2 = \{\rho(M_{k,j}) = p_{k,j} \mid k = 2,\ldots,l \text{ and } j = 1,\ldots,n_k\}$. The variables of $p_{k,j}$ ($k \geq 2$) are fresh, so they are untouched by $\textsc{mgu}\,(\mathcal{E}_1)$, so we have $\mathcal{E}_2 = \textsc{mgu}\,(\mathcal{E}_1)\mathcal{E}'_2$ and $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}'_2$, so

$$\textsc{mgu}\,(\mathcal{E}_2)\textsc{mgu}\,(\mathcal{E}_1) = \textsc{mgu}\,(\textsc{mgu}\,(\mathcal{E}_1)\mathcal{E}'_2)\textsc{mgu}\,(\mathcal{E}_1) = \textsc{mgu}\,(\mathcal{E}_1 \cup \mathcal{E}'_2) = \textsc{mgu}\,(\mathcal{E})$$

  by Lemma 32. Moreover, the variables of $p'_2,\ldots,p'_l$ are fresh, so they are untouched by $\textsc{mgu}\,(\mathcal{E}_1)$. Hence

$$\begin{aligned}
&\textsc{mgu}\,(\mathcal{E}_2)(\rho_1[x_2 \mapsto p'_2,\ldots,x_l \mapsto p'_l])\\
&\quad = \textsc{mgu}\,(\mathcal{E}_2)\textsc{mgu}\,(\mathcal{E}_1)(\rho[x_1 \mapsto p'_1, x_2 \mapsto p'_2,\ldots,x_l \mapsto p'_l])\\
&\quad = \textsc{mgu}\,(\mathcal{E})(\rho[x_1 \mapsto p'_1,\ldots,x_l \mapsto p'_l])
\end{aligned}$$

  and $\textsc{mgu}\,(\mathcal{E}_2)H_1 = \textsc{mgu}\,(\mathcal{E}_2)\textsc{mgu}\,(\mathcal{E}_1)H = \textsc{mgu}\,(\mathcal{E})H$. Therefore,

$$\begin{aligned}
&[\![\mathsf{let}\ E_1\ \mathsf{in}\ \mathsf{let}\ E_2\ \mathsf{in}\ \ldots\ \mathsf{let}\ E_l\ \mathsf{in}\ P\ \mathsf{else}\ Q \ldots\ \mathsf{else}\ Q\ \mathsf{else}\ Q]\!]\rho H\\
&\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\textsc{mgu}\,(\mathcal{E})(\rho[x_1 \mapsto p'_1,\ldots,x_l \mapsto p'_l]))(\textsc{mgu}\,(\mathcal{E})H)
\end{aligned}$$

  As before, when $\textsc{mgu}\,(\mathcal{E})$ is not defined, that is, $\textsc{mgu}\,(\mathcal{E}_2)\textsc{mgu}\,(\mathcal{E}_1)$ is not defined, the second component of the union is omitted. □

From this lemma, we obtain the following result for the special case of the decomposition of data constructors.

Corollary 39. *Let f be a data constructor of arity n and f_1^{-1}, …, f_n^{-1} be its associated destructors.*

$$\begin{aligned}
&[\![\mathsf{let}\ x_1 = f_1^{-1}(M)\ \mathsf{in}\ \ldots\ \mathsf{let}\ x_n = f_n^{-1}(M)\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho H\\
&\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\textsc{mgu}\,(\mathcal{E})(\rho[x_1 \mapsto v_1,\ldots,x_n \mapsto v_n]))(\textsc{mgu}\,(\mathcal{E})H)
\end{aligned}$$

where $x_1,\ldots,x_n$ do not occur in Q nor in M, and $\mathcal{E} = \{f(v_1,\ldots,v_n) = \rho(M)\}$. When $\textsc{mgu}\,(\mathcal{E})$ is not defined, the second component of the union is omitted.

Proof. By Lemma 38, we obtain

$$\begin{aligned}&[\![\mathsf{let}\ x_1 = f_1^{-1}(M)\ \mathsf{in}\ \ldots\ \mathsf{let}\ x_n = f_n^{-1}(M)\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho H\\ &\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\mathrm{MGU}\,(\mathcal{E}')(\rho[x_1 \mapsto v_{1,1}, \ldots, x_n \mapsto v_{n,n}]))(\mathrm{MGU}\,(\mathcal{E}')H)\end{aligned}$$

where $\mathcal{E}' = \{\rho(M) = f(v_{k,1}, \ldots, v_{k,n}) \mid k = 1, \ldots, n\}$ and the variables $v_{k,j}$ ($k = 1, \ldots, n$, $j = 1, \ldots, n$) are fresh. We have $\mathrm{MGU}\,(\mathcal{E}')v_{k,j} = \mathrm{MGU}\,(\mathcal{E}')v_{k',j}$ for all k, k', j, so for all $j = 1, \ldots, n$, we can rename the variables $v_{k,j}$ for all k into the same variable v_j. After this renaming, we obtain the announced result. □

Lemma 40. *Suppose that the variables of $pat_1, \ldots, pat_n$ are pairwise distinct and fresh (that is, they do not occur in ρ, H, M_1, …, M_n, and Q).*

$$\begin{aligned}&[\![\mathsf{let}\ pat_1 = M_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ pat_n = M_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho H\\ &\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\mathrm{MGU}\,(\mathcal{E})(\rho[x \mapsto x \mid x\ \textit{occurs in}\ pat_1, \ldots, pat_n]))(\mathrm{MGU}\,(\mathcal{E})H)\end{aligned}$$

where $\mathcal{E} = \{pat_i = \rho(M_i) \mid i = 1, \ldots, n\}$.

Proof. The proof is done by induction on the total size of the patterns $pat_1, \ldots, pat_n$.

- Case 1: there is a single pattern $pat = x$.

$$\begin{aligned}&[\![\mathsf{let}\ x = M\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H\\ &\quad= [\![\mathsf{let}\ x = id(M)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H\\ &\quad= [\![Q]\!]\rho H \cup [\![P]\!](\mathrm{MGU}\,(\{\rho(M) = y\})(\rho[x \mapsto y]))(\mathrm{MGU}\,(\{\rho(M) = y\})H)\\ &\qquad\text{where } y \text{ is a fresh variable and the rewrite rule for destructor}\\ &\qquad id \text{ is renamed into } id(y) \to y \text{ (see the base case of Lemma 38).}\\ &\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\mathrm{MGU}\,(\{\rho(M) = x\})(\rho[x \mapsto x]))(\mathrm{MGU}\,(\{\rho(M) = x\})H)\\ &\qquad\text{by renaming } x \text{ into } y \text{ since } x \text{ and } y \text{ do not occur in } \rho,\ \rho(M),\\ &\qquad\text{and } H.\end{aligned}$$

- Case 2: there is a single pattern $pat = f(pat_1, \ldots, pat_n)$.

$$\begin{aligned}&[\![\mathsf{let}\ f(pat_1, \ldots, pat_n) = M\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H\\ &\quad= [\![\mathsf{let}\ x_1 = f_1^{-1}(M)\ \mathsf{in}\ \ldots\ \mathsf{let}\ x_n = f_n^{-1}(M)\ \mathsf{in}\\ &\qquad\mathsf{let}\ pat_1 = x_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ pat_n = x_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q\\ &\qquad\mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho H\\ &\qquad\text{where } x_1, \ldots, x_n \text{ are fresh variables}\\ &\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![\mathsf{let}\ pat_1 = x_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ pat_n = x_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\\ &\qquad(\mathrm{MGU}\,(\mathcal{E})(\rho[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]))(\mathrm{MGU}\,(\mathcal{E})H)\\ &\qquad\text{where } \mathcal{E} = \{f(v_1, \ldots, v_n) = \rho(M)\}, \text{ by Corollary 39}\\ &\quad\sqsubseteq [\![Q]\!]\rho H \cup [\![Q]\!]\rho' H' \cup\\ &\qquad[\![P]\!](\mathrm{MGU}\,(\mathcal{E}')(\rho'[x \mapsto x \mid x \text{ occurs in } pat_1, \ldots, pat_n]))(\mathrm{MGU}\,(\mathcal{E}')H')\end{aligned}$$

where $\rho' = \text{MGU}(\mathcal{E})(\rho[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n])$, $H' = \text{MGU}(\mathcal{E})H$, and $\mathcal{E}' = \{pat_1 = \rho'(x_1), \ldots, pat_n = \rho'(x_n)\}$, by induction hypothesis (since the total size of $pat_1, \ldots, pat_n$ is less than the size of $f(pat_1, \ldots, pat_n)$).

As $x_1, \ldots, x_n$ do not appear in Q, $[\![Q]\!]\rho'H' = [\![Q]\!](\text{MGU}(\mathcal{E})\rho)(\text{MGU}(\mathcal{E})H) \sqsubseteq [\![Q]\!]\rho H$, by Lemma 37.

We have $\mathcal{E}' = \{pat_i = \text{MGU}(\mathcal{E})v_i \mid i = 1, \ldots, n\} = \text{MGU}(\mathcal{E})\{pat_i = v_i \mid i = 1, \ldots, n\}$, so by Lemma 32, $\text{MGU}(\mathcal{E}')\text{MGU}(\mathcal{E}) = \text{MGU}(\{f(v_1, \ldots, v_n) = \rho(M)\} \cup \{pat_i = v_i \mid i = 1, \ldots, n\}) = \text{MGU}(\{f(pat_1, \ldots, pat_n) = \rho(M)\} \cup \{pat_i = v_i \mid i = 1, \ldots, n\})$. Let $\mathcal{E}'' = \{f(pat_1, \ldots, pat_n) = \rho(M)\}$. Then we have $\text{MGU}(\mathcal{E}')\text{MGU}(\mathcal{E}) = (\text{MGU}(\mathcal{E}''))[v_i \mapsto \text{MGU}(\mathcal{E}'')pat_i]$. Therefore we obtain that:

$$[\![\mathsf{let}\ f(pat_1, \ldots, pat_n) = M\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]\rho H \sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\text{MGU}(\mathcal{E}''))(\rho[x \mapsto x \mid x \text{ occurs in } pat_1, \ldots, pat_n]))(\text{MGU}(\mathcal{E}''))H)$$

since the variables $v_1, \ldots, v_n$ do not occur in ρ and H, and the variables $x_1, \ldots, x_n$ can be removed from the environment since they do not occur in P.

- Case 3: there are several patterns.

$$\begin{aligned}
&[\![\mathsf{let}\ pat_1 = M_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ pat_n = M_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho H \\
&\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![\mathsf{let}\ pat_2 = M_2\ \mathsf{in}\ \ldots\ \mathsf{let}\ pat_n = M_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!] \\
&\qquad (\text{MGU}(\mathcal{E}_1)(\rho[x \mapsto x \mid x \text{ occurs in } pat_1]))(\text{MGU}(\mathcal{E}_1)H) \\
&\qquad \text{where } \mathcal{E}_1 = \{pat_1 = \rho(M_1)\}\text{, by induction hypothesis applied to } pat_1 \\
&\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![Q]\!]\rho' H' \cup \\
&\qquad [\![P]\!](\text{MGU}(\mathcal{E}_2)(\rho'[x \mapsto x \mid x \text{ occurs in } pat_2, \ldots, pat_n]))(\text{MGU}(\mathcal{E}_2)H')
\end{aligned}$$

where $\rho' = \text{MGU}(\mathcal{E}_1)(\rho[x \mapsto x \mid x \text{ occurs in } pat_1])$, $H' = \text{MGU}(\mathcal{E}_1)H$, and $\mathcal{E}_2 = \{pat_i = \rho'(M_i) \mid i = 2, \ldots, n\}$, by induction hypothesis applied to $pat_2, \ldots, pat_n$.

Since the variables of pat_1 do not occur in the process Q, we have $[\![Q]\!]\rho'H' = [\![Q]\!](\text{MGU}(\mathcal{E}_1)\rho)(\text{MGU}(\mathcal{E}_1)H) \sqsubseteq [\![Q]\!]\rho H$ by Lemma 37.

Let $\mathcal{E}'_2 = \{pat_i = \rho(M_i) \mid i = 2, \ldots, n\}$ and $\mathcal{E} = \{pat_i = \rho(M_i) \mid i = 1, \ldots, n\}$. Since the variables of pat_i for $i \geq 2$ do not occur in $\mathcal{E}_1$, we have $\text{MGU}(\mathcal{E}_2)\text{MGU}(\mathcal{E}_1) = \text{MGU}(\text{MGU}(\mathcal{E}_1)\mathcal{E}'_2)\text{MGU}(\mathcal{E}_1) = \text{MGU}(\mathcal{E}_1 \cup \mathcal{E}'_2) = \text{MGU}(\mathcal{E})$ by Lemma 32. So

$$\begin{aligned}
&\text{MGU}(\mathcal{E}_2)(\rho'[x \mapsto x \mid x \text{ occurs in } pat_2, \ldots, pat_n]) \\
&\quad = \text{MGU}(\mathcal{E}_2)\text{MGU}(\mathcal{E}_1)(\rho[x \mapsto x \mid x \text{ occurs in } pat_1, \ldots, pat_n]) \\
&\quad = \text{MGU}(\mathcal{E})(\rho[x \mapsto x \mid x \text{ occurs in } pat_1, \ldots, pat_n])
\end{aligned}$$

and $\text{MGU}(\mathcal{E}_2)H' = \text{MGU}(\mathcal{E}_2)\text{MGU}(\mathcal{E}_1)H = \text{MGU}(\mathcal{E})H$. Therefore,

$$[\![\mathsf{let}\ pat_1 = M_1\ \mathsf{in}\ \ldots\ \mathsf{let}\ pat_n = M_n\ \mathsf{in}\ P\ \mathsf{else}\ Q\ \ldots\ \mathsf{else}\ Q]\!]\rho H \sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!](\text{MGU}(\mathcal{E})(\rho[x \mapsto x \mid x \text{ occurs in } pat_1, \ldots, pat_n]))(\text{MGU}(\mathcal{E})H)$$

□

Lemma 41. *Let $\Gamma_P \vdash M^G$ be a well-typed pattern, ρ^G a function that associates a clause term with each name and variable, possibly with indices, and Γ an environment for generalized Horn clauses such that $\Gamma_P, \Gamma \vdash \rho^G$. Then $\Gamma \vdash \rho^G(M^G)$*

Proof. We detail here the three interesting cases.

- Case $M^G = x_{\widetilde{\iota}}$. Since Γ_P types $x_{\widetilde{\iota}}$, we have two judgments $x_{\_} : \widetilde{L} \in \Gamma_P$ and $\Gamma_P \vdash \widetilde{\iota} : \widetilde{L}$. From the definition of $\Gamma_P, \Gamma \vdash \rho^G$, if $\{x_{\widetilde{i}} \mapsto p^G\} \in \rho^G$, then $\Gamma, \widetilde{i} : \widetilde{L} \vdash p^G$. Moreover, as $\Gamma_P \vdash \widetilde{\iota} : \widetilde{L}$, we have $\Gamma \vdash \widetilde{\iota} : \widetilde{L}$. Hence $\rho^G(M^G) = p^G\{\widetilde{\iota}/\widetilde{i}\}$ and $\Gamma \vdash \rho^G(M^G)$.
- Case $M^G = a$. From the definition of $\Gamma \vdash \rho^G$, if $\{a \mapsto p^G\} \in \rho^G$, then $\Gamma \vdash p^G = \rho^G(M^G)$.
- Case $M^G = a_\iota$. Since Γ_P types a_ι, we have two judgments $a_{\_} : [1, L] \in \Gamma_P$ and $\Gamma_P \vdash i : [1, L]$. From the definition of $\Gamma_P, \Gamma \vdash \rho^G$, if $\{a_i \mapsto p^G\} \in \rho^G$, then $\Gamma, i : [1, L] \vdash p^G$. Moreover, as $\Gamma_P \vdash i : [1, L]$, we have $\Gamma \vdash i : [1, L]$. Hence $\rho^G(M^G) = p^G\{\iota/i\}$ and $\Gamma \vdash \rho^G(M^G)$.

□

Proof of Lemma 31. The proof is done by structural induction on the process P^G. Let $\rho = \text{MGU}(\mathcal{E}^T)\rho^{GT}$ and $H = \text{MGU}(\mathcal{E}^T)H^{GT}$, and let us show that

$$[\![P^{GT}]\!]\rho H \sqsubseteq \bigcup_{T' ext T} ([\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}.$$

- Case $\mathsf{out}(M^G, N^G).P^G$:

$$\begin{aligned}
&[\![(\mathsf{out}(M^G, N^G).P^G)^T]\!]\rho H \\
&\quad = [\![\mathsf{out}(M^{GT}, N^{GT}).P^{GT}]\!]\rho H \\
&\quad = [\![P^{GT}]\!]\rho H \cup \{(\text{MGU}(\mathcal{E}^T)H^{GT} \\
&\qquad\qquad \Rightarrow \text{message}(\text{MGU}(\mathcal{E}^T)\rho^{GT}(M^{GT}), \text{MGU}(\mathcal{E}^T)\rho^{GT}(N^{GT}))\} \\
&\quad = [\![P^{GT}]\!]\rho H \cup (\{\Gamma \vdash H^G \wedge \mathcal{E} \Rightarrow \text{message}(\rho^G(M^G), \rho^G(N^G))\})^T \\
&\quad \sqsubseteq \bigcup_{T' ext T} ([\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup \\
&\qquad\qquad \bigcup_{T' ext T} (\{\Gamma \vdash H^G \wedge \mathcal{E} \Rightarrow \text{message}(\rho^G(M^G), \rho^G(N^G))\})^{T'} \\
&\qquad\qquad \text{by induction hypothesis and using that } T \text{ is an extension of itself} \\
&\quad \sqsubseteq \bigcup_{T' ext T} ([\![\mathsf{out}(M^G, N^G).P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}
\end{aligned}$$

- Case $\mathsf{in}(M^G, x).P^G$:

$$\begin{aligned}[[(\mathsf{in}(M^G, x).P^G)^T]]\rho H &= [[\mathsf{in}(M^{GT}, x).P^{GT}]]\rho H \\ &= [[P^{GT}]](\rho[x \mapsto x])(H \wedge \text{message}(\rho(M^{GT}), x))\end{aligned}$$

  The right-hand side of the theorem develops in

$$\bigcup_{T' ext T} ([[\mathsf{in}(M^G, x).P^G]]\rho^G H^G \mathcal{E}\Gamma)^{T'} = \bigcup_{T' ext T} ([[P^G]]\rho_1^G H_1^G \mathcal{E}\Gamma_1)^{T'}$$

  where $\rho_1^G = \rho^G[x \mapsto x]$, $H_1^G = H^G \wedge \text{message}(\rho^G(M^G), x)$, and $\Gamma_1 = \Gamma, x\_ : [\,]$. We show that $\rho[x \mapsto x] = \text{MGU}\,(\mathcal{E}^T)\rho_1^{GT}$:

$$\text{MGU}\,(\mathcal{E}^T)\rho_1^{GT} = \text{MGU}\,(\mathcal{E}^T)\rho^{GT}[x \mapsto x] = \rho[x \mapsto x]$$

  and $H \wedge \text{message}(\rho(M^{GT}), x) = \text{MGU}\,(\mathcal{E}^T)H_1^{GT}$:

$$\begin{aligned}\text{MGU}\,(\mathcal{E}^T)H_1^{GT} &= \text{MGU}\,(\mathcal{E}^T)(H^G \wedge \text{message}(\rho^G(M^G), x))^T \\ &= \text{MGU}\,(\mathcal{E}^T)H^{GT} \wedge \text{MGU}\,(\mathcal{E}^T)(\text{message}(\rho^{GT}(M^{GT}), x) \\ &= H \wedge \text{message}(\text{MGU}\,(\mathcal{E}^T)\rho^{GT}(M^{GT}), x) \\ &= H \wedge \text{message}(\rho(M^{GT}), x)\end{aligned}$$

  Let Γ'_P the environment that types P^G, $\Gamma'_P = \Gamma_P, x\_ : [\,]$. Before applying the induction hypothesis we need to show that $\Gamma'_P, \Gamma_1 \vdash \rho_1^G$ and $\Gamma_1 \vdash H_1^G$ (clearly, $\Gamma_1 \vdash \mathcal{E}$). Since $\Gamma_P, \Gamma \vdash \rho^G$, we have $\Gamma'_P, \Gamma_1 \vdash \rho^G$. For the new map $[x \mapsto x] \in \rho_1^G$ we have that $x\_ : [\,] \in \Gamma'_P$ and $\Gamma_1 \vdash x$. Hence $\Gamma'_P, \Gamma_1 \vdash \rho_1^G$.

  Since $\Gamma \vdash H^G$, we have $\Gamma_1 \vdash H^G$. From Lemma 41 we have that $\Gamma \vdash \rho^G(M^G)$, as $\Gamma_P, \Gamma \vdash \rho^G$ and $\Gamma_P \vdash M^G$. Finally $\Gamma_1 \vdash x$. Hence $\Gamma_1 \vdash \text{message}(\rho^G(M^G), x)$, and thus $\Gamma_1 \vdash H_1^G$. Therefore, we can apply the induction hypothesis and conclude.

- Case $\mathbf{0}$: $[[\mathbf{0}^T]]\rho H = \emptyset = \bigcup_{T' ext T}([[\mathbf{0}]]\rho^G H^G \mathcal{E}\Gamma)^{T'}$.

- Case $P^G \mid Q^G$:

$$\begin{aligned}[[(P^G \mid Q^G)^T]]\rho H = [[P^{GT} \mid Q^{GT}]]\rho H &= [[P^{GT}]]\rho H \cup [[Q^{GT}]]\rho H \\ &\sqsubseteq \bigcup_{T' ext T} ([[P^G]]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup \bigcup_{T' ext T} ([[Q^G]]\rho^G H^G \mathcal{E}\Gamma)^{T'} \\ &\sqsubseteq \bigcup_{T' ext T} ([[P^G \mid Q^G]]\rho^G H^G \mathcal{E}\Gamma)^{T'}\end{aligned}$$

- Case $!^s P^G$:

$$\begin{aligned}[[(!^s P^G)^T]]\rho h = [[!^s P^{GT}]]\rho H &= [[P^{GT}]](\rho[s \mapsto s])H \\ &\sqsubseteq \bigcup_{T' ext T} ([[P^G]](\rho^G[s \mapsto s])H^G \mathcal{E}\Gamma)^{T'} \\ &\sqsubseteq \bigcup_{T' ext T} ([[!^s P^G]]\rho^G H^G \mathcal{E}\Gamma)^{T'}\end{aligned}$$

- Case $\Pi_{i\leq L}P$:

$$\begin{aligned}[\![(\Pi_{i\leq L}P)^T]\!]\rho H &= [\![P^{GT[i\mapsto 1]} \mid \ldots \mid P^{GT[i\mapsto L^T]}]\!]\rho H \\ &= [\![P^{GT[i\mapsto 1]}]\!]\rho H \cup \cdots \cup [\![P^{GT[i\mapsto L^T]}]\!]\rho H\end{aligned}$$

By induction hypothesis, $[\![P^{GT[i\mapsto v]}]\!]\rho H \sqsubseteq \bigcup_{T' ext\, T[i\mapsto v]}([\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, i : [1, L]))^{T'}$ for each $v \in \{1, \ldots, L^T\}$. Therefore

$$\begin{aligned}[\![(\Pi_{i\leq L}P)^T]\!]\rho H &\sqsubseteq \bigcup_{T' ext\, T[i\mapsto 1]} ([\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, i : [1, L]))^{T'} \cup \cdots \cup \\ &\qquad \bigcup_{T' ext\, T[i\mapsto L^T]} ([\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, i : [1, L]))^{T'} \\ &\sqsubseteq \bigcup_{T' ext\, T} ([\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, i : [1, L]))^{T'} \\ &\sqsubseteq \bigcup_{T' ext\, T} ([\![\Pi_{i\leq L}P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}\end{aligned}$$

since $T[i \mapsto v]$ is an extension of T for each $v \in \{1, \ldots, L^T\}$.

- Case (for all $i \leq L, va_i : a^{L,\widetilde{L}}_{i,\widetilde{i}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^G$:

$$\begin{aligned}&[\![((\text{for all } i \leq L, va_i : a^{L,\widetilde{L}}_{i,\widetilde{i}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^G)^T]\!]\rho H \\ &\quad = [\![(va_1 : a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}]) \ldots \\ &\qquad (va_{L^T} : a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^{GT}]\!]\rho H \\ &\quad = [\![P^{GT}]\!]\rho_1 H\end{aligned}$$

where

$$\begin{aligned}\rho_1 = \rho[a_1 \mapsto a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[\rho(x_1), \ldots, \rho(x_n), \rho(s_1), \ldots, \rho(s_{n'})], \ldots, \\ a_{L^T} \mapsto a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[\rho(x_1), \ldots, \rho(x_n), \rho(s_1), \ldots, \rho(s_{n'})]]\,.\end{aligned}$$

The right-hand side of the theorem develops in

$$\begin{aligned}&\bigcup_{T' ext\, T} ([\![(\text{for all } i \leq L, va_i : a^{L,\widetilde{L}}_{i,\widetilde{i}}[x_1, \ldots, x_n, s_1, \ldots, s_{n'}])P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \\ &\quad = \bigcup_{T' ext\, T} ([\![P^G]\!]\rho_1^G H^G \mathcal{E}\Gamma)^{T'}\end{aligned}$$

where $\rho_1^G = \rho^G[a_i \mapsto a^{L,\widetilde{L}}_{i,\widetilde{i}}[\rho^G(x_1), \ldots, \rho^G(x_n), \rho^G(s_1), \ldots, \rho^G(s_{n'})]]$. We show that

$\rho_1 = \text{MGU}(\mathcal{E}^T)\rho_1^{GT}$:

$$\begin{aligned}
\rho_1 &= \rho[a_1 \mapsto a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[\rho(x_1),\dots,\rho(x_n),\rho(s_1),\dots,\rho(s_{n'})],\dots,\\
&\qquad a_{L^T} \mapsto a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[\rho(x_1),\dots,\rho(x_n),\rho(s_1),\dots,\rho(s_{n'})]]\\
&= \text{MGU}(\mathcal{E}^T)\\
&\qquad (\rho^{GT}[a_1 \mapsto a^{L^T,\widetilde{L}^T}_{1,\widetilde{i}^T}[\rho^{GT}(x_1),\dots,\rho^{GT}(x_n),\rho^{GT}(s_1),\dots,\rho^{GT}(s_{n'})],\\
&\qquad \dots, a_{L^T} \mapsto a^{L^T,\widetilde{L}^T}_{L^T,\widetilde{i}^T}[\rho^{GT}(x_1),\dots,\rho^{GT}(x_n),\rho^{GT}(s_1),\dots,\rho^{GT}(s_{n'})]])\\
&= \text{MGU}(\mathcal{E}^T)\rho_1^{GT}
\end{aligned}$$

Let Γ'_P the environment that types P^G, $\Gamma'_P = \Gamma_P, a_\_ : [1, L]$. Before applying the induction hypothesis we need to show that $\Gamma'_P, \Gamma \vdash \rho_1^G$. Since $\Gamma_P, \Gamma \vdash \rho^G$, we have $\Gamma'_P, \Gamma \vdash \rho^G$. For the new map $[a_i \mapsto a^{L,\widetilde{L}}_{i,\widetilde{i}}[\rho^G(x_1),\dots,\rho^G(x_n),\rho^G(s_1),\dots,\rho^G(s_{n'})]] \in \rho_1^G$ we have that $a_\_ : [1, L] \in \Gamma'_P$ and $\Gamma \vdash a^{L,\widetilde{L}}_{i,\widetilde{i}}[\rho^G(x_1),\dots,\rho^G(x_n),\rho^G(s_1),\dots,\rho^G(s_{n'})$ by Lemma 41. Hence $\Gamma'_P, \Gamma \vdash \rho_1^G$.

We can then apply the induction hypothesis and conclude.

- Case $(\nu a)P^G$: this case is similar to the previous one.
- Case $\mathsf{let\ for\ all}\ \widetilde{i} \le \widetilde{L}, x_{\widetilde{i}} = g(M_1^G,\dots,M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$: let $g(p_1,\dots,p_n) \to p'$ be the rewrite rule for the destructor g. We suppose that the tuples of indices $\widetilde{v} \le \widetilde{L}^T$ are indexed by $1,\dots,l$, that is, we define $\{\widetilde{v}_1,\dots,\widetilde{v}_l\} = \{\widetilde{1},\dots,\widetilde{L}^T\}$. We let $T'_k = T[\widetilde{i} \mapsto \widetilde{v}_k]$ for $k = 1,\dots,l$.

$$\begin{aligned}
&[\![(\mathsf{let\ for\ all}\ \widetilde{i} \le \widetilde{L}, x_{\widetilde{i}} = g(M_1^G,\dots,M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G)^T]\!]\rho H\\
&\quad = [\![\mathsf{let}\ E_1\ \mathsf{in}\ \dots\ \mathsf{let}\ E_l\ \mathsf{in}\ P^{GT}\ \mathsf{else}\ Q^{GT}\ \dots\ \mathsf{else}\ Q^{GT}]\!]\rho H\\
&\qquad \text{where } E_k \text{ is } x^{T'_k}_{\widetilde{i}} = g(M_1^{GT'_k},\dots,M_n^{GT'_k}) \text{ for } k = 1,\dots,l.\\
&\quad \sqsubseteq [\![Q^{GT}]\!]\rho H \cup [\![P^{GT}]\!](\text{MGU}(\mathcal{E}_1)(\rho[x_{\widetilde{v}_1} \mapsto p'_1,\dots,x_{\widetilde{v}_l} \mapsto p'_l]))(\text{MGU}(\mathcal{E}_1)H)
\end{aligned}$$

by Lemma 38, where $p_{k,1},\dots,p_{k,n},p'_k$ are the patterns $p_1,\dots,p_n,p'$ renamed with distinct fresh variables for each $k = 1,\dots,l$ and $\mathcal{E}_1 = \{\rho(M_j^{GT'_k}) = p_{k,j} \mid k = 1,\dots,l \text{ and } j = 1,\dots,n\}$, assuming that $\text{MGU}(\mathcal{E}_1)$ exists. (When $\text{MGU}(\mathcal{E}_1)$ does not exist, the second component of the union is omitted, and the rest of the proof can easily be adapted.) The right-hand side of the theorem develops in

$$\begin{aligned}
&\bigcup_{T' ext\ T} ([\![\mathsf{let\ for\ all}\ \widetilde{i} \le \widetilde{L}, x_{\widetilde{i}} = g(M_1^G,\dots,M_n^G)\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}\\
&\quad = \bigcup_{T' ext\ T} ([\![Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup \bigcup_{T' ext\ T} ([\![P^G]\!](\rho^G[x_{\widetilde{i}} \mapsto p'^G])H^G(\mathcal{E} \cup \mathcal{E}')\Gamma')^{T'}
\end{aligned}$$

where $\mathcal{E}'$ and Γ' are defined as follows. The rewrite rule $g(p_1'^G,\dots,p_n'^G) \to p'^G$ is obtained from $g(p_1,\dots,p_n) \to p'$ by replacing all variables y of this rule with fresh variables with indices $\widetilde{i}$: $y'_{\widetilde{i}}$. Then $\mathcal{E}'$ is the set of equations $\{\bigwedge_{\widetilde{i} \in \widetilde{L}} p_1'^G \doteq \rho^G(M_1^G),\dots,$

$\bigwedge_{\widetilde{i}\in\widetilde{L}} p'^G_n \doteq \rho^G(M^G_n)\}$ and Γ' is Γ extended with $x_\_ : \widetilde{L}$ and $y'_\_ : \widetilde{L}$ for each variable $y'_{\widetilde{i}}$ in $p'^G_1, \ldots p'^G_n, p'^G$.

We analyze now the relation between $\text{MGU}(\mathcal{E}_1)$ and $\mathcal{E}'^T$. We have

$$\mathcal{E}'^T = \{\rho^{GT}(M_j^{GT[\widetilde{i}\mapsto\widetilde{v}]}) = p_j'^{GT[\widetilde{i}\mapsto\widetilde{v}]} \mid \widetilde{v} \leq \widetilde{L}^T \text{ and } j = 1, \ldots n\}.$$

Given the construction of $p_{k,j}$, p'_k, p'^G_j, p'^G, there is a renaming α such that, for all $k = 1, \ldots, l$, we have $\alpha p_{k,j} = p_j^{'GT'_k}$ for each $j = 1, \ldots, n$ and $\alpha p'_i = p'^{GT'_k}$. Hence we have

$$\begin{aligned}
\text{MGU}(\mathcal{E}^T)\mathcal{E}'^T &= \{\text{MGU}(\mathcal{E}^T)\rho^{GT}(M_j^{GT[\widetilde{i}\mapsto\widetilde{v}]}) = \text{MGU}(\mathcal{E}^T)p_j'^{GT[\widetilde{i}\mapsto\widetilde{v}]} \\
&\qquad \mid \widetilde{v} \leq \widetilde{L}^T \text{ and } j = 1, \ldots n\} \\
&= \{\rho(M_j^{GT[\widetilde{i}\mapsto\widetilde{v}]}) = p_j'^{GT[\widetilde{i}\mapsto\widetilde{v}]} \mid \widetilde{v} \leq \widetilde{L}^T \text{ and } j = 1, \ldots n\} \\
&\qquad \text{since the variables of } p_j'^{GT[\widetilde{i}\mapsto\widetilde{v}]} \text{ are fresh,} \\
&\qquad \text{so they are not touched by } \text{MGU}(\mathcal{E}^T) \\
&= \{\rho(M_j^{GT'_k}) = \alpha p_{k,j} \mid k = 1, \ldots, l \text{ and } j = 1, \ldots n\} \\
&= \alpha\mathcal{E}_1
\end{aligned}$$

So, by Lemma 32,

$$\text{MGU}(\alpha\mathcal{E}_1)\text{MGU}(\mathcal{E}^T) = \text{MGU}(\text{MGU}(\mathcal{E}^T)\mathcal{E}'^T)\text{MGU}(\mathcal{E}^T) = \text{MGU}((\mathcal{E}\cup\mathcal{E}')^T)$$

Hence

$$\begin{aligned}
&\text{MGU}(\alpha\mathcal{E}_1)(\rho[x_{\widetilde{v}_1} \mapsto \alpha p'_1, \ldots, x_{\widetilde{v}_l} \mapsto \alpha p'_l]) \\
&\quad = \text{MGU}(\alpha\mathcal{E}_1)\text{MGU}(\mathcal{E}^T)(\rho^{GT}[x_{\widetilde{1}} \mapsto p'^{GT[\widetilde{i}\mapsto\widetilde{1}]}, \ldots, x_{\widetilde{L}^T} \mapsto p'^{GT[\widetilde{i}\mapsto\widetilde{L}^T]}]) \\
&\quad = \text{MGU}((\mathcal{E}\cup\mathcal{E}')^T)(\rho^G[x_{\widetilde{i}} \mapsto p'^G])^T
\end{aligned}$$

Similarly, $\text{MGU}(\alpha\mathcal{E}_1)H = \text{MGU}(\alpha\mathcal{E}_1)\text{MGU}(\mathcal{E}^T)H^{GT} = \text{MGU}((\mathcal{E}\cup\mathcal{E}')^T)H^{GT}$.

Let Γ'_P the environment that types P^G: by the typing rules we have that $\Gamma'_P = \Gamma_P, x_\_ : \widetilde{L}$. Before applying induction we need to show that $\Gamma'_P, \Gamma' \vdash \rho^G[x_{\widetilde{i}} \mapsto p'^G]$ and $\Gamma' \vdash \mathcal{E}\cup\mathcal{E}'$. At first, notice that $p'^G_1, \ldots, p'^G_n, p'^G$ are obtained from $p_1, \ldots, p_n, p'$ by replacing all variables y with fresh variables with indices $y'_{\widetilde{i}}$ and that Γ' types each variable $y'_\_$ with type $\widetilde{L}$. Hence all variables in $p'^G_1, \ldots, p'^G_n, p'^G$ are typed by Γ'.

We have that $\Gamma'_P, \Gamma' \vdash \rho^G$ because Γ' extends Γ, Γ'_P extends Γ_P, and $\Gamma_P, \Gamma \vdash \rho^G$ by hypothesis. Since $x_\_ : \widetilde{L} \in \Gamma'_P$ and $\Gamma', \widetilde{i} : \widetilde{L} \vdash p'^G$ (all variables in p'^G are typed by Γ'), we have $\Gamma'_P, \Gamma' \vdash \rho^G[x_{\widetilde{i}} \mapsto p'^G]$.

For each equation $\bigwedge_{\widetilde{i}\in\widetilde{L}} p'^G_j \doteq \rho^G(M^G_j)$, $j = 1, \ldots n$ we have that:

- $\Gamma', \widetilde{i} : \widetilde{L} \vdash \rho^G(M^G_j)$: this comes from Lemma 41 applied to $\Gamma_P, \widetilde{i} : \widetilde{L} \vdash M^G_j$ and $(\Gamma_P, \widetilde{i} : \widetilde{L}), \Gamma \vdash \rho^G$ and from the fact that Γ' extends Γ.

– $\Gamma', \widetilde{i} : \widetilde{L} \vdash p'^{G}_{j}$: all variables in p'^{G}_{j} are typed in Γ'.

This means that each equation in $\mathcal{E}'$ is well typed in Γ'; moreover $\Gamma' \vdash \mathcal{E}$ because Γ' extends Γ and $\Gamma \vdash \mathcal{E}$ by hypothesis. Thus $\Gamma' \vdash \mathcal{E} \cup \mathcal{E}'$.

We can then apply the induction hypothesis, which yields

$$\begin{aligned}&[\![P^{GT}]\!](\text{MGU}\,(\alpha\mathcal{E}_1)(\rho[x_{\widetilde{v}_1} \mapsto \alpha p'_1, \ldots, x_{\widetilde{v}_l} \mapsto \alpha p'_l]))(\text{MGU}\,(\alpha\mathcal{E}_1)H)\\ &\quad \sqsubseteq \bigcup_{T' ext\, T} ([\![P^G]\!](\rho^G[x_{\widetilde{i}} \mapsto p'^G])H^G(\mathcal{E} \cup \mathcal{E}')\Gamma')^{T'}\end{aligned}$$

and $[\![Q^{GT}]\!]\rho H \sqsubseteq \bigcup_{T' ext\, T}([\![Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}$.

Moreover,

$$\begin{aligned}&[\![P^{GT}]\!](\text{MGU}\,(\mathcal{E}_1)(\rho[x_{\widetilde{v}_1} \mapsto p'_1, \ldots, x_{\widetilde{v}_l} \mapsto p'_l]))(\text{MGU}\,(\mathcal{E}_1)H)\\ &\quad \sqsubseteq [\![P^{GT}]\!](\text{MGU}\,(\alpha\mathcal{E}_1)(\rho[x_{\widetilde{v}_1} \mapsto \alpha p'_1, \ldots, x_{\widetilde{v}_l} \mapsto \alpha p'_l]))(\text{MGU}\,(\alpha\mathcal{E}_1)H)\end{aligned}$$

(These two sets of clauses are in fact equal up to renaming of variables, by construction.)

Hence we can conclude that:

$$\begin{aligned}&[\![(\textsf{let for all}\ \widetilde{i} \le \widetilde{L}, x_{\widetilde{i}} = g(M^G_1, \ldots, M^G_n)\ \textsf{in}\ P^G\ \textsf{else}\ Q^G)^T]\!]\rho H \sqsubseteq\\ &\bigcup_{T' ext\, T} ([\![\textsf{let for all}\ \widetilde{i} \le \widetilde{L}, x_{\widetilde{i}} = g(M^G_1, \ldots, M^G_n)\ \textsf{in}\ P^G\ \textsf{else}\ Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}\end{aligned}$$

- Case $\textsf{let for all}\ \widetilde{i} \le \widetilde{L}, pat^G = M^G\ \textsf{in}\ P^G\ \textsf{else}\ Q^G$: as in the previous case, we suppose that the tuples of indices $\widetilde{v} \le \widetilde{L}^T$ are indexed by $1, \ldots, l$, that is, we define $\{\widetilde{v}_1, \ldots, \widetilde{v}_l\} = \{\widetilde{1}, \ldots, \widetilde{L}^T\}$.

$$\begin{aligned}&[\![(\textsf{let for all}\ \widetilde{i} \le \widetilde{L}, pat^G = M^G\ \textsf{in}\ P^G\ \textsf{else}\ Q^G)^T]\!]\rho H\\ &\quad = [\![\textsf{let}\ E_1\ \textsf{in}\ \ldots\ \textsf{let}\ E_l\ \textsf{in}\ P^{GT}\ \textsf{else}\ Q^{GT}\ \ldots\ \textsf{else}\ Q^{GT}]\!]\rho H\\ &\qquad \text{where } E_i \text{ is the equation } pat^{GT[\widetilde{i} \mapsto \widetilde{v}_i]} = M^{GT[\widetilde{i} \mapsto \widetilde{v}_i]}.\\ &\quad \sqsubseteq [\![Q]\!]\rho H \cup [\![P]\!]\rho' H'\end{aligned}$$

by Lemma 40, where $\mathcal{E}_1 = \{pat^{GT''} = \rho(M^{GT''}) \mid T'' = T[\widetilde{i} \mapsto \widetilde{v}], \widetilde{v} \le \widetilde{L}^T\}$, $\rho' = \text{MGU}\,(\mathcal{E}_1)(\rho[x \mapsto x \mid x \text{ occurs in } pat^{GT[\widetilde{i} \mapsto \widetilde{v}]}, \widetilde{v} \le \widetilde{L}^T])$, and $H' = \text{MGU}\,(\mathcal{E}_1)H$, assuming that $\text{MGU}\,(\mathcal{E}_1)$ exists. (When $\text{MGU}\,(\mathcal{E}_1)$ does not exist, the second component of the union is omitted, and the rest of the proof can easily be adapted.)

The right-hand side of the theorem develops in:

$$\begin{aligned}&\bigcup_{T' ext\, T} ([\![\textsf{let for all}\ \widetilde{i} \le \widetilde{L}, pat^G = M^G\ \textsf{in}\ P^G\ \textsf{else}\ Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}\\ &\quad = \bigcup_{T' ext\, T} ([\![Q]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup\\ &\qquad [\![P]\!](\rho^G[x_{\widetilde{i'}} \mapsto x_{\widetilde{i'}} \mid x_{\widetilde{i'}} \text{ occurs in } pat^G])H^G(\mathcal{E} \cup \mathcal{E}')\Gamma')^{T'}\end{aligned}$$

where $\mathcal{E}' = \bigwedge_{\widetilde{i} \leq \widetilde{L}} pat^G \doteq \rho^G(M^G)$ and Γ' is Γ extended for the variables occurring in pat^G. More precisely, if the typing rule for the process $\mathsf{let\ for\ all}\ \widetilde{i} \leq \widetilde{L}, pat^G = M^G\ \mathsf{in}\ P^G\ \mathsf{else}\ Q^G$ has $i_1 : [1, L_1], \ldots, i_h : [1, L_h] \vdash pat^G \rightsquigarrow \Gamma''$ as a premise, then $\Gamma' = \Gamma, \Gamma''$. Hence $\mathcal{E}'^T = \{pat^{GT''} = \rho^{GT}(M^{GT''}) \mid T'' = T[\widetilde{i} \mapsto \widetilde{v}], \forall \widetilde{v} \leq \widetilde{L}^T\}$. Hence we have that:

$$\begin{aligned}
\textsc{mgu}(\mathcal{E}^T)\mathcal{E}'^T &= \{\textsc{mgu}(\mathcal{E}^T)pat^{GT[\widetilde{i} \mapsto \widetilde{v}]} = \textsc{mgu}(\mathcal{E}^T)\rho^{GT}(M^{GT[\widetilde{i} \mapsto \widetilde{v}]}) \mid \forall \widetilde{v} \leq \widetilde{L}^T\} \\
&= \{pat^{GT[\widetilde{i} \mapsto \widetilde{v}]} = \rho(M^{GT[\widetilde{i} \mapsto \widetilde{v}]}) \mid \forall \widetilde{v} \leq \widetilde{L}^T\} \\
&= \mathcal{E}_1
\end{aligned}$$

By Lemma 32,

$$\textsc{mgu}(\mathcal{E}_1)\textsc{mgu}(\mathcal{E}^T) = \textsc{mgu}(\textsc{mgu}(\mathcal{E}^T)\mathcal{E}'^T)\textsc{mgu}(\mathcal{E}^T) = \textsc{mgu}((\mathcal{E} \cup \mathcal{E}')^T)$$

so

$$\begin{aligned}
\rho' &= \textsc{mgu}(\mathcal{E}_1)(\rho[x \mapsto x \mid x \text{ occurs in } pat^{GT[\widetilde{i} \mapsto \widetilde{v}_i]}, \widetilde{v}_i \leq \widetilde{L}^T]) \\
&= \textsc{mgu}(\mathcal{E}_1)\textsc{mgu}(\mathcal{E}^T)(\rho^{GT}[x \mapsto x \mid x \text{ occurs in } pat^{GT[\widetilde{i} \mapsto \widetilde{v}_i]}, \widetilde{v}_i \leq \widetilde{L}^T]) \\
&= \textsc{mgu}((\mathcal{E} \cup \mathcal{E}')^T)(\rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'} \text{ occurs in } pat^G])^T
\end{aligned}$$

Similarly,

$$H' = \textsc{mgu}(\mathcal{E}_1)H = \textsc{mgu}(\mathcal{E}_1)\textsc{mgu}(\mathcal{E}^T)H^{GT} = \textsc{mgu}((\mathcal{E} \cup \mathcal{E}')^T)H^{GT}\,.$$

Let Γ'_P the environment that types P^G: by the typing rules we have that $\Gamma'_P = \Gamma_P, \Gamma''_P$, where $\widetilde{i} : \widetilde{L} \vdash pat^G \rightsquigarrow \Gamma''_P$. Before applying induction we need to show that $\Gamma'_P, \Gamma' \vdash \rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'} \text{ occurs in } pat^G]$ and $\Gamma' \vdash \mathcal{E} \cup \mathcal{E}'$.

We have that $\Gamma'_P, \Gamma' \vdash \rho^G$ because Γ' extends Γ, Γ'_P extends Γ_P, and $\Gamma_P, \Gamma \vdash \rho^G$ by hypothesis. Clearly $x\_ : \widetilde{L} \in \Gamma''_P$ (that is $x\_ : \widetilde{L} \in \Gamma'_P$) and $\Gamma', \widetilde{i} : \widetilde{L} \vdash x_{\widetilde{i}}$ (all variables in pat^G are typed by Γ') then $\Gamma'_P, \Gamma' \vdash \rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'} \text{ occurs in } pat^G]$.

For the equation $\bigwedge_{\widetilde{i} \in \widetilde{L}} pat^G \doteq \rho^G(M^G)$ we have that:

- $\Gamma', \widetilde{i} : \widetilde{L} \vdash \rho^G(M^G)$: this comes from Lemma 41 applied to $\Gamma_P, \widetilde{i} : \widetilde{L} \vdash M^G$ and $(\Gamma_P, \widetilde{i} : \widetilde{L}), \Gamma \vdash \rho^G$ and from the fact that Γ' extends Γ.
- $\Gamma', \widetilde{i} : \widetilde{L} \vdash pat^G$: all variables in pat'^G are typed in Γ'.

This means that the equation in $\mathcal{E}'$ is well typed in Γ'; moreover $\Gamma' \vdash \mathcal{E}$ because Γ' extends Γ and $\Gamma \vdash \mathcal{E}$ by hypothesis. Thus $\Gamma' \vdash \mathcal{E} \cup \mathcal{E}'$.

We can then apply the induction hypothesis:

$$[\![P^{GT}]\!]\rho' H' \sqsubseteq \bigcup_{T' ext\, T} [\![P]\!](\rho^G[x_{\widetilde{i}'} \mapsto x_{\widetilde{i}'} \mid x_{\widetilde{i}'} \text{ occurs in } pat^G])H^G(\mathcal{E} \cup \mathcal{E}')\Gamma')^{T'}$$

and $[\![Q^{GT}]\!]\rho H \sqsubseteq \bigcup_{T' ext\, T}([\![Q^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}$. Therefore we can conclude.

- Case choose L in P^G:

$$
\begin{aligned}
&[\![(\text{choose } L \text{ in } P^G)^T]\!]\rho H \\
&\quad = [\![P^{GT[L\mapsto 1]} + \cdots + P^{GT[L\mapsto n]} + \cdots]\!]\rho H \\
&\quad = [\![P^{GT[L\mapsto 1]}]\!]\rho H \cup \cdots \cup [\![P^{GT[L\mapsto n]}]\!]\rho H \cup \cdots \\
&\quad \sqsubseteq \bigcup_{T' ext\, T[L\mapsto 1]} ([\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup \cdots \cup \bigcup_{T' ext\, T[L\mapsto n]} ([\![P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'} \cup \cdots \\
&\quad \sqsubseteq \bigcup_{T' ext\, T} ([\![\text{choose } L \text{ in } P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}
\end{aligned}
$$

- Case choose $k \leq L$ in P^G:

$$
\begin{aligned}
&[\![(\text{choose } k \leq L \text{ in } P^G)^T]\!]\rho H \\
&\quad = [\![P^{GT[k\mapsto 1]} + \cdots + P^{GT[k\mapsto L^T]}]\!]\rho H \\
&\quad = [\![P^{GT[k\mapsto 1]}]\!]\rho H \cup \cdots \cup [\![P^{GT[k\mapsto L^T]}]\!]\rho H \\
&\quad \sqsubseteq \bigcup_{T' ext\, T[k\mapsto 1]} ([\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, k : [1, L]))^{T'} \cup \cdots \cup \\
&\qquad\qquad \bigcup_{T' ext\, T[k\mapsto L^T]} ([\![P^G]\!]\rho^G H^G \mathcal{E}(\Gamma, k : [1, L]))^{T'} \\
&\quad \sqsubseteq \bigcup_{T' ext\, T} ([\![\text{choose } k \leq L \text{ in } P^G]\!]\rho^G H^G \mathcal{E}\Gamma)^{T'}
\end{aligned}
$$

- Case choose $\phi : L_1 \times \cdots \times L_h \to L'$ in P^G: this case is similar to previous one. □

Résumé

Dans cette thèse, nous proposons deux techniques différentes pour la vérification de protocoles de sécurité qui utilisent des listes de taille arbitraire.

Dans la première partie de la thèse, nous présentons une technique simple pour vérifier le secret pour des protocoles qui traitent tous les éléments des listes de façon uniforme, pour un nombre non borné de sessions. Plus précisément, cette technique est fondée sur l'approche à base de clauses de Horn utilisée par le vérificateur automatique ProVerif. Nous montrons que si un protocole est prouvé sûr par notre technique avec des listes de longueur un, alors il est sûr pour des listes de longueur arbitraire. Curieusement, ce théorème repose sur des approximations faites par notre technique de vérification : en général, le secret pour des listes de longueur un n'implique pas le secret pour des listes de taille arbitraire.

La deuxième technique présentée dans cette thèse permet de prouver automatiquement des propriétés de secret et d'authentification pour des protocoles qui traitent des éléments différents de façon différente. Ce résultat est obtenu en étendant l'approche par clauses de Horn utilisée par ProVerif. Nous étendons la syntaxe des clauses de Horn pour représenter des listes de taille arbitraire. Nous adaptons l'algorithme de résolution pour qu'il traite la nouvelle classe de clauses de Horn et nous prouvons la correction de cet algorithme. Nous avons implémenté notre algorithme et nous l'avons testé avec succès sur plusieurs protocoles, en particulier sur des protocoles pour les services web. Finalement, nous présentons une extension du langage d'entrée de ProVerif, une variante du pi calcul appliqué, pour modéliser les protocoles avec des listes de longueur non bornée, nous en donnons la sémantique et une traduction automatique en clauses de Horn.

Abstract

In this thesis, we propose two different techniques for verifying security protocols that manipulate lists of unbounded length.

In the first part of the thesis, we present a simple technique for proving secrecy properties for protocols that manipulate list elements in a uniform way, for an unbounded number of sessions. More specifically, this technique relies on the Horn clause approach used in the automatic verifier ProVerif: we show that if a protocol is proven secure by our technique with lists of length one, then it is secure for lists of unbounded length. Interestingly, this theorem relies on approximations made by our verification technique: in general, secrecy for lists of length one does not imply secrecy for lists of unbounded length.

The second technique presented in this thesis automatically proves secrecy and authentication properties for security protocols that manipulate different list elements in different ways. This result is achieved by extending the Horn clause approach of the automatic protocol verifier ProVerif. We extend the Horn clauses to be able to represent lists of unbounded length. We adapt the resolution algorithm to handle the new class of Horn clauses, and prove the soundness of this new algorithm. We have implemented our algorithm and successfully tested it on several protocol examples, including XML protocols coming from web services. Finally, we present an extension of the input language of ProVerif, a variant of the applied pi calculus, to model protocols with lists of unbounded length, give its formal semantics, and translate it automatically to Horn clauses.