AUTOMATA FOR SPECIFYING AND ORCHESTRATING SERVICE CONTRACTS

DAVIDE BASILE, PIERPAOLO DEGANO, AND GIAN-LUIGI FERRARI

e-mail address, {basile,degano,giangi}@di.unipi.it:,
Dipartimento di Informatica, Università di Pisa, Italy

ABSTRACT. An approach to the formal description of service contracts is presented in terms of automata. We focus on the basic property of guaranteeing that in the multi-party composition of principals each of them gets his requests satisfied, so that the overall composition reaches its goal. Depending on whether requests are satisfied synchronously or asynchronously, we construct an orchestrator that at static time either yields composed services enjoying the required properties or detects the principals responsible for possible violations. To do that in the asynchronous case we resort to Linear Programming techniques. We also relate our automata with two logically based methods for specifying contracts.

1. INTRODUCTION

Modern software applications are not *stand-alone* entities and are embedded in a dynamic distributed environment where new functionalities are added or deleted in a relatively short period of time. *Service Oriented Computing* [44] is a paradigm for designing distributed applications where applications are built by combining several *fine-grained* and *loosely-coupled* distributed components, called *services*. Services can be combined to accomplish a certain computational task or to form a more complex service. A service exposes both the functionalities it provides and the parameters it requires. Clients exploit service public information to discover and bind the services that better fit their requirements.

Service coordination is a fundamental mechanism of the service-oriented approach because it dictates how the involved services are compositionally assembled together. Service coordination policies differ on the interaction supports that are adopted to pass information among services. At design time, a main task of software engineers is therefore to express the assumptions that shape these policies and that will drive the construction of a correct service coordination. *Orchestration* and *choreography* are the standard solutions to coordinate distributed services. In an orchestrated approach, services coordinate with each other by interacting with a distinguished service, the *orchestrator*, which at run-time regulates how the computation evolves. In a choreographed approach,

1998 ACM Subject Classification: F.1.2 Modes of Computation, F.3.1 Specifying and Verifying and Reasoning about Programs, H.3.5 Online Information Services, H.5.3 Group and Organization Interfaces.

Key words and phrases: Web Services, Coordination and Orchestration, Contract Agreement, Control Theory, Linear Programming, Intuitionistic Logic.

This work has been partially supported by the projects *Security Horizons*, funded by MIUR, and PRA_2016_64 *Through the fog*, funded by the University of Pisa.

LOGICAL METHODS
IN COMPUTER SCIENCE

DOI:10.2168/LMCS-???

the distributed services autonomously execute and interact with each other without a central coordinator. Here, we concentrate on orchestration, whereas we injected some aspects of our proposal within the choreographed approach in [20, 16].

We argue that the design of correct service coordination policies is naturally supported by relying on the notion of *service contract* which specifies what a service is going to guarantee and offer (hereafter an *offer*) and what in turn it expects and requires (hereafter a *request*). The coordination policy has therefore to define the duties and responsibilities for each of the different services involved in the coordination through the *overall contract agreement*. Obviously, this arrangement is based on the contracts of the involved services, and ensures that all requests are properly served when all the duties are properly kept. The coordinator then organises the service coordination policy and proposes the resulting overall contract agreement to all the parties. This process is called *contract composition*.

The main contribution of this paper is twofold. First, we propose a rigorous formal technique for describing and composing contracts, suitable to be automated. Second, we develop techniques capable of determining when a contract composition is correct and leads to the design of a correct service orchestration. More in detail, we introduce an automata-based model for contracts called *contract automata*, that are a special kind of finite state automata, endowed with two operations for composing them. A contract automaton may represent a single service or a composition of several services, hereafter called *principals*. The traces accepted by a contract automaton show the possible interactions among the principals, by recording which offers and requests are performed, and by which principals in the composition. This provides the basis to define criteria that guarantee a composed service to well behave with respect to the overall service contract.

We equip our model with formal notions in language-theoretic terms aiming at characterising when contracts are honoured within a service composition. We first consider properties of a single trace. We say that a trace is in *agreement* when all the requests made are synchronously matched, i.e. satisfied by corresponding offers. The second property, *weak agreement*, is more liberal, in that requests can be asynchronously matched, and an offer can be delivered even before a corresponding request, and vice-versa. Then we say that a contract automaton is *safe* (*weakly safe*, respectively) when *all* its traces are in agreement (weak agreement, respectively).

The notions of safety presented above may appear too strict since they require that all the words belonging to the language recognised by a contract automaton must satisfy agreement or weak agreement. We thus introduce a more flexible notion that characterises when a service composition may be successful, i.e. at least one among all the possible traces enjoys one of the properties above. We say that a contract automaton *admits* (weak) agreement when such a trace exists.

When a contract automaton admits (weak) agreement, but it is not (weakly) safe, we define those principals in a contract that are (weakly) *liable*, i.e. those responsible for leading a contract composition into a failure. Note that the orchestration of contracts imposes further constraints on each principal: some of the interactions dictated by its service contract may break the overall composition and thus the orchestrator will ban them.

For checking when a contract automaton enjoys the properties sketched above, we propose two formal verification techniques that have been also implemented [15].[1] The first one amounts to build the so-called controllers in Control Theory [26]. We show that controllers are powerful enough to synthesise a correct orchestrator enforcing agreement and to detect the liable principals. In order to check weak agreement and detect weak liability we resort to Linear Programming techniques borrowed from Operational Research [34], namely optimisation of network flows. The intuitive

[1] Available at `https://github.com/davidebasile/workspace`

idea is that service coordination is rendered as an optimal flow itinerary of offers and requests in a network, automatically constructed from the contract automaton.

Finally, we establish correspondence results between (weak) agreement and provability of formulae in two fragments of different intuitionistic logics, that have been used for modelling contracts. The first one, Propositional Contract Logic [14], has a special connective to deal with circularity between offers and requests, arising when a principal requires, say a, before offering b to another principal who in turn first requires b and then offers a; note that weak agreement holds for this kind of circularity. The second fragment, Intuitionist Linear Logic with Mix [21] is a linear logic capable of modelling the exchange of resources with the possibility of recording debts, that arise when the request of a principal is satisfied and not yet paid back.

Plan of the paper. In Section 2 we introduce contract automata and two operators of composition. Section 3 discusses the properties of agreement and safety. The techniques for checking and enforcing them are also presented here, along with the notion of liability. Weak agreement and weak liability are defined in Section 4, along with a technique to check them. In Section 5 we present correspondence results with fragments of Propositional Contract Logic and Intuitionistic Linear Logic with Mix. A case study is proposed in Section 6. Finally, related work is in Section 7 and the concluding remarks are in Section 8. All the proofs of our results, and a few auxiliary definitions can be found in the appendix. Portions of Sections 2, 3, and 4 appeared in a preliminary form in [18].

2. The Model

This section formally introduces the notion of contract automata, that are finite state automata with a partitioned alphabet. A contract automaton represents the behaviour of a set of principals (possibly a singleton) capable of performing some *actions*; more precisely, the actions of contract automata allow them to "make" requests, "advertise" offers or "matching" a pair of "complementary" request/offer. The number of principals in a contract automaton is called *rank*, and we use a vectorial representation to record the action performed by each principal in a transition of a contract automaton, as well as its state as the vector of the states of its principals.

Let $\Sigma = \mathbb{R} \cup \mathbb{O} \cup \{\square\}$ be the alphabet of *basic actions*, made of *requests* $\mathbb{R} = \{a, b, c, \dots\}$ and *offers* $\mathbb{O} = \{\overline{a}, \overline{b}, \overline{c}, \dots\}$ where $\mathbb{R} \cap \mathbb{O} = \emptyset$, and $\square \notin \mathbb{R} \cup \mathbb{O}$ is a distinguished element representing the *idle* move. We define the involution $co(\bullet) : \Sigma \mapsto \Sigma$ such that $co(\mathbb{R}) = \mathbb{O}$, $co(\mathbb{O}) = \mathbb{R}$, $co(\square) = \square$.

Let $\vec{v} = (a_1, ..., a_n)$ be a vector of *rank* $n \geq 1$, in symbols r_v, and let $\vec{v}_{(i)}$ denote the i-th element with $1 \leq i \leq r_v$. We write $\vec{v}_1 \vec{v}_2 \dots \vec{v}_m$ for the concatenation of m vectors $\vec{v}_i$, while $|\vec{v}| = n$ is the rank (length) of $\vec{v}$ and $\vec{v}^n$ is the vector obtained by n concatenations of $\vec{v}$.

The alphabet of a contract automaton consists of vectors, each element of which intuitively records the activity, i.e. the occurrence of a basic action of a single principal in the contract. In a vector $\vec{v}$ there is either a single offer or a single request, or a single pair of request-offer that matches, i.e. there exists exactly i, j such that $\vec{v}_{(i)}$ is an offer and $\vec{v}_{(j)}$ is the complementary request or vice-versa; all the other elements of the vector contain the symbol $\square$, meaning that the corresponding principals stay idle. In the following let $\square^m$ denote a vector of rank m, all elements of which are $\square$. Formally:

Definition 2.1 (Actions). Given a vector $\vec{a} \in \Sigma^n$, if

- $\vec{a} = \square^{n_1} \alpha \square^{n_2}, n_1, n_2 \geq 0$, then $\vec{a}$ is a *request (action) on* α if $\alpha \in \mathbb{R}$, and is an *offer (action) on* α if $\alpha \in \mathbb{O}$
- $\vec{a} = \square^{n_1} \alpha \square^{n_2} co(\alpha) \square^{n_3}, n_1, n_2, n_3 \geq 0$, then $\vec{a}$ is a *match (action) on* α, where $\alpha \in \mathbb{R} \cup \mathbb{O}$.

Two actions $\vec{a}$ and $\vec{b}$ are *complementary*, in symbols $\vec{a} \bowtie \vec{b}$ if and only if the following conditions hold: (i) $\exists \alpha \in \mathbb{R} \cup \mathbb{O}$: $\vec{a}$ is either a request or an offer on α; (ii) $\vec{a}$ is an offer on $\alpha \implies \vec{b}$ is a request on $co(\alpha)$ and (iii) $\vec{a}$ is a request on $\alpha \implies \vec{b}$ is an offer on $co(\alpha)$.

We now extract from an action the request or offer made by a principal, and the matching of a request and an offer, and then we lift this procedure to a sequence of actions, i.e. to a trace of a contract automaton that intuitively corresponds to an execution of a service composition.

Definition 2.2 (Observable)**.** Let $w = \vec{a_1} \dots \vec{a_n}$ be a sequence of actions, and let ε be the empty one, then its *observable* is given by the partial function $Obs(w) \in (\mathbb{R} \cup \mathbb{O} \cup \{\tau\})^*$ where:

$$Obs(\varepsilon) = \varepsilon$$

$$Obs(\vec{a}\, w') = \begin{cases} \vec{a}_{(i)}\, Obs(w') & \text{if } \vec{a} \text{ is an offer/request and } \vec{a}_{(i)} \neq \Box \\ \tau Obs(w') & \text{if } \vec{a} \text{ is a match} \end{cases}$$

We now define contract automata, the actions and the states of which are actually vectors of basic actions and of states of principals, respectively.

Definition 2.3 (Contract Automata)**.** Assume as given a finite set of states $\mathfrak{Q} = \{q_1, q_2, \dots\}$. Then a *contract automaton* $\mathcal{A}$, CA for short, of rank n is a tuple $\langle Q, \vec{q_0}, A^r, A^o, T, F \rangle$, where

- $Q = Q_1 \times \dots \times Q_n \subseteq \mathfrak{Q}^n$
- $\vec{q_0} \in Q$ is the initial state
- $A^r \subseteq \mathbb{R}, A^o \subseteq \mathbb{O}$ are finite sets (of requests and offers, respectively)
- $F \subseteq Q$ is the set of final states
- $T \subseteq Q \times A \times Q$ is the set of transitions, where $A \subseteq (A^r \cup A^o \cup \{\Box\})^n$ and if $(\vec{q}, \vec{a}, \vec{q'}) \in T$ then both the following conditions hold:
  - $\vec{a}$ is either a request or an offer or a match
  - $\forall i \in 1 \dots n.$ if $\vec{a}_{(i)} = \Box$ then it must be $\vec{q}_{(i)} = \vec{q'}_{(i)}$

A *principal* contract automaton (or simply *principal*) has rank 1 and it is such that $A^r \cap co(A^o) = \emptyset$.
A step $(w, \vec{q}) \rightarrow (w', \vec{q'})$ occurs if and only if $w = \vec{a}w', w' \in A^*$ and $(\vec{q}, \vec{a}, \vec{q'}) \in T$.
The language of $\mathcal{A}$ is $\mathscr{L}(\mathcal{A}) = \{w \mid (w, \vec{q_0}) \rightarrow^* (\varepsilon, \vec{q}), \vec{q} \in F\}$ where $\rightarrow^*$ is the reflexive, transitive closure of the transition relation $\rightarrow$.

Note that for principals we have the restriction $A^r \cap co(A^o) = \emptyset$. Indeed, a principal who offers what he requires makes little sense.

Example 2.4. Figure 1 shows three contract automata. The automaton $\mathcal{A}_1$ may be understood as producing a certain number of resources through one or more offers $\overline{res}$ and it terminates with the request of receiving a signal *sig*. The contract $\mathcal{A}_2$ starts by sending the signal $\overline{sig}$ and then it collects the resources produced by $\mathcal{A}_1$. The contract $\mathcal{A}_3$ represents the contract automaton where $\mathcal{A}_1$ and $\mathcal{A}_2$ interact as discussed below. Both $\mathcal{A}_1$ and $\mathcal{A}_2$ have rank 1 while $\mathcal{A}_3$ has rank 2.

Contract automata can be composed, by making the cartesian product of their states and of the labels of the joined transitions, with the additional possibility of labels recording matching request-offer. This is the case for the action $(sig, \overline{sig})$ of the contract automaton $\mathcal{A}_3$ in Figure 1.

Below, we introduce two different operators for composing contract automata. Both products interleave all the transitions of their operands. We only force a synchronisation to happen when two contract automata are ready on their respective request/offer action. These operators represent two different policies of orchestration. The first operator is called simply *product* and it considers the case when a service S joins a group of services already clustered as a single orchestrated service S'. In the product of S and S', the first can only accept the still available offers (requests, respectively)

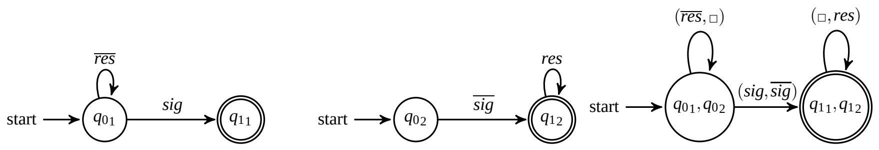


FIGURE 1. Three contract automata: from left $\mathcal{A}_1, \mathcal{A}_2$, and $\mathcal{A}_3$ (composition of $\mathcal{A}_1$ and $\mathcal{A}_2$)

of S' and vice-versa. In other words, S cannot interact with the principals of the orchestration S', but only with it as a whole component. This is obtained in Definition 2.5 through the relation $\bowtie$ (see Definition 2.1), which is only defined for actions that are not matches. This is not the case with the second operation of composition, called *a-product*: it puts instead all the principals of S at the same level of those of S'. Any matching request-offer of either contracts can be split, and the offers and requests, that become available again, can be re-combined with complementary actions of S, and vice-versa. The a-product turns out to satisfactorily model coordination policies in dynamically changing environments, because the a-product supports a form of *dynamic orchestration*, that adjusts the workflow of messages when new principals join the contract.

We now introduce our first operation of composition; recall that we implicitly assume the alphabet of a contract automaton of rank m to be $A \subseteq (A^r \cup A^o \cup \{\square\})^m$. Note that the first case of the definition of T below is for the matching of actions of two component automata, while the other considers the action of a single component.

Definition 2.5 (Product)**.** Let $\mathcal{A}_i = \langle Q_i, \vec{q_0}_i, A^r_i, A^o_i, T_i, F_i \rangle, i \in 1 \dots n$ be contract automata of rank r_i. The *product* $\bigotimes_{i \in 1 \dots n} \mathcal{A}_i$ is the contract automaton $\langle Q, \vec{q_0}, A^r, A^o, T, F \rangle$ of rank $m = \sum_{i \in 1 \dots n} r_i$, where:

- $Q = Q_1 \times ... \times Q_n, \quad$ where $\vec{q_0} = \vec{q_0}_1 \dots \vec{q_0}_n$
- $A^r = \bigcup_{i \in 1 \cdots n} A^r_i, \quad A^o = \bigcup_{i \in 1 \cdots n} A^o_i$
- $F = \{\vec{q}_1 \dots \vec{q}_n \mid \vec{q}_1 \dots \vec{q}_n \in Q, \vec{q}_i \in F_i, i \in 1 \dots n\}$
- T is the least subset of $Q \times A \times Q$ s.t. $(\vec{q}, \vec{c}, \vec{q}\,') \in T$ iff, when $\vec{q} = \vec{q}_1 \dots \vec{q}_n \in Q$,
  - either there are $1 \le i < j \le n$ s.t. $(\vec{q}_i, \vec{a}_i, \vec{q}\,'_i) \in T_i$, $(\vec{q}_j, \vec{a}_j, \vec{q}\,'_j) \in T_j$, $\vec{a}_i \bowtie \vec{a}_j$ and
  $$\left\{ \begin{array}{l} \vec{c} = \square^u \vec{a}_i \square^v \vec{a}_j \square^z \text{ with } u = r_1 + \ldots + r_{i-1},\ v = r_{i+1} + \ldots + r_{j-1}, |\vec{c}| = m \\ \textit{and} \\ \vec{q}\,' = \vec{q}_1 \dots \vec{q}_{i-1}\ \vec{q}\,'_i\ \vec{q}_{i+1} \dots\ \vec{q}_{j-1}\ \vec{q}\,'_j\ \vec{q}_{j+1} \dots \vec{q}_n \end{array} \right.$$
  - or there is $1 \le i \le n$ s.t. $(\vec{q}_i, \vec{a}_i, \vec{q}\,'_i) \in T_i$ and
  $\vec{c} = \square^u \vec{a}_i \square^v$ with $u = r_1 + \ldots + r_{i-1}$, $v = r_{i+1} + \ldots + r_n$, and
  $\vec{q}\,' = \vec{q}_1 \dots \vec{q}_{i-1}\ \vec{q}\,'_i\ \vec{q}_{i+1} \dots \vec{q}_n$ and
  $\forall j \neq i, 1 \le j \le n, (\vec{q}_j, \vec{a}_j, \vec{q}\,'_j) \in T_j$ it does not hold that $\vec{a}_i \bowtie \vec{a}_j$.

There is a simple way of retrieving the principals involved in a composition of contract automata obtained through the product introduced above: just introduce projections $\prod^i$ as done below. For example, for the contract automata in Figure 1, we have $\mathcal{A}_1 = \prod^1(\mathcal{A}_3)$ and $\mathcal{A}_2 = \prod^2(\mathcal{A}_3)$.

Definition 2.6 (Projection)**.** Let $\mathcal{A} = \langle Q, \vec{q_0}, A^r, A^o, T, F \rangle$ be a contract automaton of rank n, then the *projection* on the i-th principal is $\prod^i(\mathcal{A}) = \langle \prod^i(Q), \vec{q_0}_{(i)}, \prod^i(A^r), \prod^i(A^o), \prod^i(T), \prod^i(F) \rangle$ where $i \in 1 \dots n$ and:

$$\textstyle\prod^i(Q) = \{\vec{q}_{(i)} \mid \vec{q} \in Q\} \quad \prod^i(F) = \{\vec{q}_{(i)} \mid \vec{q} \in F\} \quad \prod^i(A^r) = \{a \mid a \in A^r, (q, a, q') \in \prod^i(T)\}$$

$$\textstyle\prod^i(T) = \{(\vec{q}_{(i)}, \vec{a}_{(i)}, \vec{q'}_{(i)}) \mid (\vec{q}, \vec{a}, \vec{q'}) \in T \wedge \vec{a}_{(i)} \neq \square\} \quad \prod^i(A^o) = \{\overline{a} \mid \overline{a} \in A^o, (q, \overline{a}, q') \in \prod^i(T)\}$$

The following proposition states that decomposition is the inverse of product, and its proof is immediate.

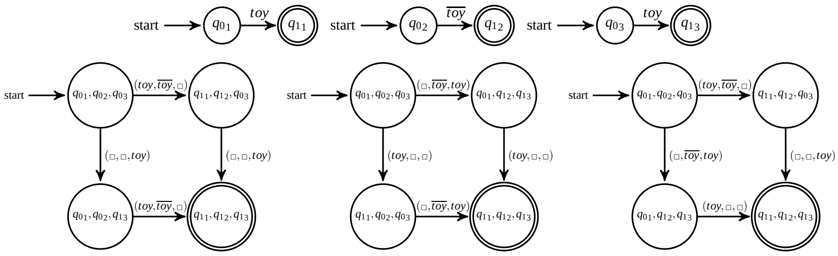

start
q_{0_1}
toy
q_{1_1}
start
q_{0_2}
$\overline{toy}$
q_{1_2}
start
q_{0_3}
toy
q_{1_3}
start
q_{0_1},q_{0_2},q_{0_3}
$(toy,\overline{toy},\square)$
q_{1_1},q_{1_2},q_{0_3}
$(\square,\square,toy)$
$(\square,\square,toy)$
q_{0_1},q_{0_2},q_{1_3}
$(toy,\overline{toy},\square)$
q_{1_1},q_{1_2},q_{1_3}
start
q_{0_1},q_{0_2},q_{0_3}
$(\square,\overline{toy},toy)$
q_{0_1},q_{1_2},q_{1_3}
$(toy,\square,\square)$
$(toy,\square,\square)$
q_{1_1},q_{0_2},q_{0_3}
$(\square,\overline{toy},toy)$
q_{1_1},q_{1_2},q_{1_3}
start
q_{0_1},q_{0_2},q_{0_3}
$(toy,\overline{toy},\square)$
q_{1_1},q_{1_2},q_{0_3}
$(\square,\overline{toy},toy)$
$(\square,\square,toy)$
q_{0_1},q_{1_2},q_{1_3}
$(toy,\square,\square)$
q_{1_1},q_{1_2},q_{1_3}

FIGURE 2. From left to right and top-down: the principal contract automata of Bill, Mary and John, the contract automata $(Bill \otimes Mary) \otimes John$, $Bill \otimes (Mary \otimes John)$ and $Bill \boxtimes Mary \boxtimes John$.

Proposition 2.7 (Product Decomposition). *Let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be a set of principal contract automata, then $\prod^i(\bigotimes_{j \in 1 \dots n} \mathcal{A}_j) = \mathcal{A}_i$.*

Our second operation of composition first extracts from its operands the principals they are composed of, and then reassembles them.

Definition 2.8 (a-Product). Let $\mathcal{A}_1, \mathcal{A}_2$ be two contract automata of rank n and m, respectively, and let $I = \{\prod^i(\mathcal{A}_1) \mid 0 < i \leq n\} \cup \{\prod^j(\mathcal{A}_2) \mid 0 < j \leq m\}$. Then the *a-product* of $\mathcal{A}_1$ and $\mathcal{A}_2$ is $\mathcal{A}_1 \boxtimes \mathcal{A}_2 = \bigotimes_{\mathcal{A}_i \in I} \mathcal{A}_i$.

Note that if $\mathcal{A}, \mathcal{A}'$ are principal contract automata, then $\mathcal{A} \otimes \mathcal{A}' = \mathcal{A} \boxtimes \mathcal{A}'$. From now onwards we assume that every contract automaton $\mathcal{A}$ of rank $r_{\mathcal{A}} > 1$ is composed by principal contract automata using the operations of product and a-product. E.g. in Figure 1, we have that $\mathcal{A}_3 = \mathcal{A}_1 \otimes \mathcal{A}_2 = \mathcal{A}_1 \boxtimes \mathcal{A}_2$. Finally, both compositions are commutative, up to the expected rearrangement of the vectors of actions, and $\boxtimes$ is also associative, while $\otimes$ is not, as shown by the following example.

Example 2.9. In Figure 2 Mary (the automaton in the central position) offers a toy that both Bill (at left) and John (at right) request. In the product $(Bill \otimes Mary) \otimes John$ the toy is assigned to Bill who first enters into the composition with Mary, no matter if John performs the same move. Instead, in the product $Bill \otimes (Mary \otimes John)$ the toy is assigned to John. In the last row we have the a-product of the three automata that represents a dynamic re-orchestration: no matter of who is first composed with Mary, the toy will be non-deterministically assigned to either principal.

Proposition 2.10. *The following properties hold:*

– $\exists \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 . (\mathcal{A}_1 \otimes \mathcal{A}_2) \otimes \mathcal{A}_3 \neq \mathcal{A}_1 \otimes (\mathcal{A}_2 \otimes \mathcal{A}_3)$
– $\forall \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 . (\mathcal{A}_1 \boxtimes \mathcal{A}_2) \boxtimes \mathcal{A}_3 = \mathcal{A}_1 \boxtimes (\mathcal{A}_2 \boxtimes \mathcal{A}_3)$

3. ENFORCING AGREEMENT

It is common to say that some contracts are in agreement when all the requests they make have been fulfilled by corresponding offers [27, 28, 24, 37, 2, 3, 41, 23, 6, 12]. In terms of contract automata, this is rendered in two different ways, the first of which is introduced below and resembles the notion of compliance introduced in [27, 28]. We say that two or more contract automata are in *agreement* when the final states of their product are reachable from the initial state by traces only made of matches and offer actions. Our goal is to enforce the behaviour of principals so that they only follow the traces of the automaton which lead to agreement. Additionally, it is easy to track every action performed by each principal, because we use vectors of actions as the elements of the alphabet of contract automata. It is equally easy finding who is liable in a bad interaction, i.e. the principals who perform a transition leaving a state from which agreement is possible, reaching a state where instead agreement is no longer possible.

We now introduce the notion of *agreement* as a property of the language recognised by a contract automaton.

Definition 3.1 (Agreement). A trace accepted by a contract automaton is in *agreement* if it belongs to the set

$$\mathfrak{A} = \{w \in (\Sigma^n)^* \mid Obs(w) \in (\mathbb{O} \cup \{\tau\})^*, n > 1\}$$

Note that, if an action observable in w is a request, i.e. it belongs to $\mathbb{R}$, then w is not in agreement. Intuitively, a trace is in agreement if it only contains offer and match actions, i.e. if no requests are left unsatisfied.

Example 3.2. The automaton $\mathcal{A}_3$ in Figure 1 has a trace in agreement: $Obs((\overline{res}, \square)(sig, \overline{sig})) = \overline{res}\,\tau \in \mathfrak{A}$, and one not in agreement: $Obs((sig, \overline{sig})(\square, res)) = \tau\, res \notin \mathfrak{A}$.

A contract automaton is safe when all the traces of its language are in agreement, and admits agreement when at least one of its traces is in agreement. Formally:

Definition 3.3 (Safety). A contract automaton $\mathcal{A}$ is *safe* if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{A}$, otherwise it is *unsafe*. Additionally, if $\mathscr{L}(\mathcal{A}) \cap \mathfrak{A} \neq \emptyset$ then $\mathcal{A}$ *admits agreement*.

Example 3.4. The contract automaton $\mathcal{A}_3$ of Figure 1 is unsafe, but it admits agreement since $\mathscr{L}(\mathcal{A}_3) \cap \mathfrak{A} = (\overline{res}, \square)^*(sig, \overline{sig})$. Consider now the contract automata *Bill* and *Mary* in Figure 2; their product $Bill \otimes Mary$ is safe because $\mathscr{L}(Bill \otimes Mary) = (toy, \overline{toy}) \subset \mathfrak{A}$.

Note that the set $\mathfrak{A}$ can be seen as a safety property in the default-accept approach [48], where the set of bad prefixes of $\mathfrak{A}$ contains those traces ending with a trailing request, i.e. $\{w\vec{a} \mid w \in \mathfrak{A}, Obs(\vec{a}) \in \mathbb{R}\}$. One could then consider a definition of product that disallows the occurrence of transitions labelled by requests only. However, this choice would not prevent a product of contracts to reach a deadlock. In addition, compositionality would have been compromised, as shown in the following example.

Example 3.5. In what follows, we feel free to present contract automata through a sort of extended regular expressions. Consider a simple selling scenario involving two parties *Ann* and *Bart*.

Bart starts by notifying Ann that he is ready to start the negotiation, and waits from Ann to select a pen or a book. In case Ann selects the pen, he may decide to withdraw and restart the negotiation again, or to accept the payment. As soon as Ann selects the book, then Bart cannot withdraw any longer, and waits for the payment. The contract of *Bart* is:

$$Bart = (\overline{init}.pen.\overline{cancel})^*.(\overline{init}.book.pay + \overline{init}.pen.pay)$$

The contract of Ann is dual to Bart's. Ann waits to receive a notification from Bart when ready to negotiate. Then Ann decides what to buy. If she chooses the pen, she may proceed with the payment unless a withdrawal from Bart is received. In this case, Ann can repeatedly try to get the pen, until she succeeds and pays for it, or buys the book but omits to pay it (violating the contract $Ann \otimes Bart$ resulting from the orchestration, see below).

The contract of *Ann* is:

$$Ann = (init.\overline{pen}.cancel)^*.(init.\overline{pen}.\overline{pay} + init.\overline{book})$$

The contract $\mathcal{A} = Ann \otimes Bart$ is in Figure 3 top. Assume now to change the product $\otimes$ so to disallow transitions labelled by requests. The composition of *Ann* and *Bart* is in Figure 3, bottom right part, and contains the malformed trace in which *Bart* does not reach a final state:

$$(init, \overline{init})(\overline{book}, book)$$

In addition, if a third principal $Carol = \overline{pay}$ were involved, willing to pay for everybody, the following trace in agreement would not be accepted

$$(init, \overline{init}, \square)(\overline{book}, book, \square)(\square, pay, \overline{pay})$$

because Bart's request was discarded by the wrongly amended composition operator. So, compositionality would be lost.

To avoid the two unpleasant situations of deadlock and lack of compositionality, we introduce below a technique for driving a safe composition of contracts, in the style of the Supervisory Control for Discrete Event Systems [26].

A discrete event system is a finite state automaton, where *accepting* states represent the successful termination of a task, while *forbidden* states should never be traversed in "good" computations. Generally, the purpose of supervisory control theory is to synthesise a controller that enforces good computations. To do so, this theory distinguishes between *controllable* events (those the controller can disable) and *uncontrollable* events (those always enabled), besides partitioning events into *observable* and *unobservable* (obviously uncontrollable). If all events are observable then a most permissive controller exists that never blocks a good computation [26].

The purpose of contracts is to declare all the activities of a principal in terms of requests and offers. Therefore all the actions of a (composed) contract are controllable and observable. Clearly, the behaviours that we want to enforce upon a given contract automaton $\mathcal{A}$ are exactly the traces in agreement, and so we assume that a request leads to a forbidden state. A most permissive controller exists for contract automata and is defined below.

Definition 3.6 (Controller)**.** Let $\mathcal{A}$ and $\mathcal{K}$ be contract automata, we call $\mathcal{K}$ *controller* of $\mathcal{A}$ if and only if $\mathscr{L}(\mathcal{K}) \subseteq \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$.
A controller $\mathcal{K}$ of $\mathcal{A}$ is the *most permissive controller (mpc)* if and only if for all $\mathcal{K}'$ controller of $\mathcal{A}$ it is $\mathscr{L}(\mathcal{K}') \subseteq \mathscr{L}(\mathcal{K})$.

Since the most permissive controller eliminates the traces not in agreement, the following holds.

Proposition 3.7. *Let $\mathcal{K}$ be the mpc of the contract automaton $\mathcal{A}$, then $\mathscr{L}(\mathcal{K}) = \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$.*

In order to effectively build the most permissive controller, we introduce below the notion of hanged state, i.e. a state from which no final state can be reached.

Definition 3.8 (Hanged state)**.** Let $\mathcal{A} = \langle Q, \vec{q_0}, A^r, A^o, T, F \rangle$ be a contract automaton, then $\vec{q} \in Q$ is *hanged*, and belongs to the set $Hanged(\mathcal{A})$, if for all $\vec{q}_f \in F, \nexists w.(w, \vec{q}) \rightarrow^* (\varepsilon, \vec{q}_f)$.

Definition 3.9 (Mpc construction)**.** Let $\mathcal{A} = \langle Q, \vec{q_0}, A^r, A^o, T, F \rangle$ be a contract automaton, $\mathcal{K}_1 = \langle Q, \vec{q_0}, A^r, A^o, T \setminus (\{t \in T \mid t \text{ is a request transition } \}, F \rangle$ and define

$$\mathcal{K}_{\mathcal{A}} = \langle Q \setminus Hanged(\mathcal{K}_1), \vec{q_0}, A^r, A^o, T_{\mathcal{K}_1} \setminus \{(\vec{q}, a, \vec{q'}) \mid \{\vec{q}, \vec{q'}\} \cap Hanged(\mathcal{K}_1) \neq \emptyset\}, F \rangle$$

Proposition 3.10 (Mpc)**.** *The controller $\mathcal{K}_{\mathrm{A}}$ of Definition 3.9 is the most permissive controller of the contract automaton $\mathcal{A}$.*

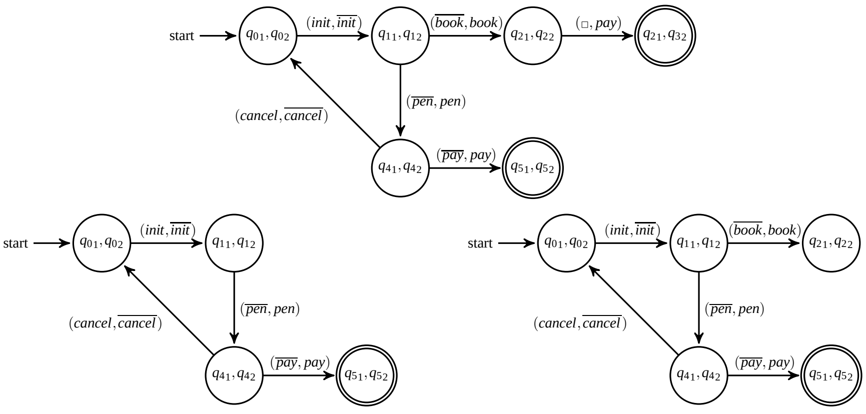


FIGURE 3. The contract automata of Example 3.5: top the contract automaton $\mathcal{A}$; bottom left its most permissive controller $\mathcal{K}_{\mathcal{A}}$, bottom right an automaton obtained with an inaccurate filtering composition.

Example 3.11. Consider again Example 3.5. For obtaining the most permissive controller we first compute the auxiliary set $\mathcal{K}_1$ that does not contain the transition $((q_{21},q_{22}),(\square,pay),(q_{21},q_{32}))$ because it represents a request from Bart which is not fulfilled by Ann. As a consequence, some states are hanged:

$$Hanged(\mathcal{K}_1) = \{(q_{21},q_{22})\}$$

By removing them, we eventually obtain $\mathcal{K}_{\mathcal{A}}$, the most permissive controller of $\mathcal{A}$ depicted in Figure 3, bottom left part.

The following proposition rephrases the notions of safe, unsafe and admits agreement on automata in terms of their most permissive controllers.

Proposition 3.12. *Let $\mathcal{A}$ be a contract automaton and let $\mathcal{K}_{\mathcal{A}}$ be its mpc, the following hold:*

- *if $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) = \mathscr{L}(\mathcal{A})$ then $\mathcal{A}$ is safe, otherwise if $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subset \mathscr{L}(\mathcal{A})$ then $\mathcal{A}$ is unsafe;*
- *if $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \neq \emptyset$, then $\mathcal{A}$ admits agreement.*

We introduce now an original notion of *liability*, that characterises those principals potentially responsible of the divergence from the behaviour in agreement. The liable principals are those who perform the first transition in a run, that is not possible in the most permissive controller. As noticed above, after this step is done, a successful state cannot be reached any longer, and so the principals who performed it will be blamed. Note in passing that hanged states play a crucial role here: just removing the request transitions from $\mathcal{A}$ would result in a contract automaton language equivalent to the mpc, but detecting liable principals would be much more intricate.

Definition 3.13 (Liability)**.** Let $\mathcal{A}$ be a contract automaton and $\mathcal{K}_{\mathcal{A}}$ be its mpc of Definition 3.9; let $(v\vec{a}w,\vec{q_0}) \rightarrow^* (\vec{a}w,\vec{q})$ be a run of both automata and let $\vec{q}$ be such that $(\vec{a}w,\vec{q}) \rightarrow (w,\vec{q'})$ is possible in $\mathcal{A}$ but not in $\mathcal{K}_{\mathcal{A}}$. The principals $\Pi^i(\mathcal{A})$ such that $\vec{a}_{(i)} \neq \square, i \in 1\ldots r_{\mathcal{A}}$ are *liable for* $\vec{a}$ and belong to $Liable(\mathcal{A},v\vec{a}w)$. Then, the set of *liable* principals in $\mathcal{A}$ is $Liable(\mathcal{A}) = \{i \mid \exists w. i \in Liable(\mathcal{A},w)\}$.

Example 3.14. In Figure 3, bottom left, we have $Liable(\mathcal{A}) = \{1,2\}$, hence both Ann and Bart are possibly liable, because the match transition with label $(\overline{book},book)$ can be performed, that leads to a violation of the agreement.

The following proposition is immediate.

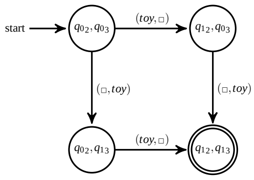


FIGURE 4. The contract automaton $Bill \otimes John$ of Example 3.16

.

Proposition 3.15. *A contract automaton $\mathcal{A}$ is safe if and only if $Liable(\mathcal{A}) = \emptyset$.*

Note that the set $Liable(\mathcal{A})$ can be rewritten as follows

$$\{i \mid (\vec{q},\vec{a},\vec{q}\,') \in T_{\mathcal{A}}, \vec{a}_{(i)} \neq \square, \vec{q} \in Q_{\mathcal{K}_{\mathcal{A}}}, \vec{q}\,' \notin Q_{\mathcal{K}_{\mathcal{A}}}\}$$

so making its calculation straightforward, as well as checking the safety of $\mathcal{A}$.

Some properties of $\otimes$ and $\boxtimes$ follow, that enable us to predict under which conditions a composition is safe without actually computing it.

We first introduce the notions of collaborative and competitive contracts. Intuitively, two contracts are *collaborative* if some requests of one meet the offers of the other, and are *competitive* if both can satisfy the same request. An example follows.

Example 3.16. Consider the contract automata $Bill, Mary, John$ in Figure 2. In Figure 4 the contract automaton $Bill \otimes John$ is displayed. The two contract automata $Mary$ and $Bill \otimes John$ are collaborative and not competitive, indeed the offer $\overline{toy}$ of $Mary$ is matched in $Bill \otimes John$, and no other principals interfere with this offer. Moreover, let $\mathcal{A}_1 = \overline{apple} + cake \otimes apple + \overline{cake}$ and $\mathcal{A}_2 = \overline{apple}$. The pair $\mathcal{A}_1, \mathcal{A}_2$ is competitive since $\mathcal{A}_2$ interferes with $\mathcal{A}_1$ on the $\overline{apple}$ offer.

Definition 3.17 (Competitive, Collaborative)**.** The pair of CA $\mathcal{A}_1 = \langle Q_1, \vec{q}_{01}, A_1^r, A_1^o, T_1, F_1\rangle$ and $\mathcal{A}_2 = \langle Q_2, \vec{q}_{02}, A_2^r, A_2^o, T_2, F_2\rangle$ are

- *competitive* if $A_1^o \cap A_2^o \cap co(A_1^r \cup A_2^r) \neq \emptyset$
- *collaborative* if $(A_1^o \cap co(A_2^r)) \cup (co(A_1^r) \cap A_2^o) \neq \emptyset$.

Note that *competitive* and *collaborative* are not mutually exclusive, as stated in the first and second item of Theorem 3.18 below. Moreover if two contract automata are *non-competitive* then all their match actions are preserved in their composition, indeed we have $\mathcal{A}_1 \boxtimes \mathcal{A}_2 = \mathcal{A}_1 \otimes \mathcal{A}_2$.

The next theorem says that the composition of safe and non-competitive contracts prevents all principals from harmful interactions, unlike the case of safe competitive contracts. In other words, when $\mathcal{A}_1$ and $\mathcal{A}_2$ are safe, no principals will be found liable in $\mathcal{A}_1 \otimes \mathcal{A}_2$ (i.e. $Liable(\mathcal{A}_1 \otimes \mathcal{A}_2) = \emptyset$), and the same happens for $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ if the two are also non-competitive (i.e. $Liable(\mathcal{A}_1 \boxtimes \mathcal{A}_2) = \emptyset$).

Theorem 3.18. *If two contract automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are*

(1) *competitive then they are collaborative,*
(2) *collaborative and safe, then they are competitive,*
(3) *safe then $\mathcal{A}_1 \otimes \mathcal{A}_2$ is safe, $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ admits agreement,*
(4) *non-collaborative, and one or both unsafe, then $\mathcal{A}_1 \otimes \mathcal{A}_2, \mathcal{A}_1 \boxtimes \mathcal{A}_2$ are unsafe,*
(5) *safe and non-competitive, then $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is safe.*

Note that in item 3 of Theorem 3.18 it can be that $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is not *safe*. Moreover consider the contract automata $\mathcal{A}_1$ and $\mathcal{A}_2$ of Example 3.16. We have that $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is unsafe because the trace $(\square, apple, \overline{apple})(cake, \square, \square)$ belongs to $\mathscr{L}(\mathcal{A}_1 \boxtimes \mathcal{A}_2)$.

4. Weak Agreement

As said in the introduction, we will now consider a more liberal notion of agreement, where an offer can be asynchronously fulfilled by a matching request, even though either of them occur beforehand. In other words, some actions can be taken on credit, assuming that in the future the obligations will be honoured. According to this notion, called here *weak agreement*, computations well behave when all the requests are matched by offers, in spite of lack of synchronous agreement, in the sense of Section 3. This may lead to a circularity, as shown by the example below, because, e.g. one principal first requires something from the other and then is willing to fulfil the request of the other principal, who in turn behaves in the same way. This is a common scenario in contract composition, and variants of weak agreement have been studied using many different formal techniques, among which Process Algebras, Petri Nets, non-classical Logics, Event Structures [13, 8, 5, 12].

Example 4.1. Suppose Alice and Bob want to share a bike and an airplane, but neither trusts the other. Before providing their offers they first ask for the complementary requests. As regular expressions: $Alice = bike.\overline{airplane}$ and $Bob = airplane.\overline{bike}$. The language of their composition is: $\mathscr{L}(Alice \otimes Bob) = \{(\square, airplane)(bike, \overline{bike})(\overline{airplane}, \square), (bike, \square)(\overline{airplane}, airplane)(\square, \overline{bike})\}$. In both possible traces the contracts fail in exchanging the bike or the airplane synchronously, hence $\mathscr{L}(Alice \otimes Bob) \cap \mathfrak{A} = \emptyset$ and the composition does not admit agreement.

The circularity in the requests/offers is solved by weakening the notion of agreement, allowing a request to be performed on credit and making sure that in the future a complementary offer will occur, giving rise to a trace in weak agreement. We now formally define weak agreement.

Definition 4.2 (Weak Agreement)**.** A trace accepted by a contract automaton of rank $n > 1$ is in *weak agreement* if it belongs to $\mathfrak{W} = \{w \in (\Sigma^n)^* \mid w = \vec{a}_1 \dots \vec{a}_m, \exists$ a function $f : [1..m] \to [1..m]$ total and injective on the (indexes of the) request actions of w, and such that $f(i) = j$ only if $\vec{a}_i \bowtie \vec{a}_j\}$.

Needless to say, a trace in agreement is also in weak agreement, so $\mathfrak{A}$ is a proper subset of $\mathfrak{W}$, as shown below.

Example 4.3. Consider $\mathcal{A}_3$ in Figure 1, whose trace $(\overline{res}, \square)(sig, \overline{sig})(\square, res)$ is in $\mathfrak{W}$ but not in $\mathfrak{A}$ (all f such that $f(3) = 1$ certify the membership) , while $(\overline{res}, \square)(sig, \overline{sig})(\square, res)(\square, res) \notin \mathfrak{W}$.

Definition 4.4 (Weak Safety)**.** Let $\mathcal{A}$ be a contract automaton. Then

- if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{W}$ then $\mathcal{A}$ is *weakly safe*, otherwise is *weakly unsafe*;
- if $\mathscr{L}(\mathcal{A}) \cap \mathfrak{W} \neq \emptyset$ then $\mathcal{A}$ *admits weak agreement*.

Example 4.5. In Example 4.1 we have $\mathscr{L}(Alice \otimes Bob) \subset \mathfrak{W}$, hence the composition of $Alice$ and Bob is weakly safe. Indeed every f such that $f(1) = 3$ certifies the membership for both traces.

The following theorem states the conditions under which weak agreement is preserved by our operations of contract composition.

Theorem 4.6. *Let $\mathcal{A}_1, \mathcal{A}_2$ be two contract automata, then if $\mathcal{A}_1, \mathcal{A}_2$ are*

1. *weakly safe then $\mathcal{A}_1 \otimes \mathcal{A}_2$ is weakly safe, $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ admits weak agreement*
2. *non-collaborative and one or both unsafe, then $\mathcal{A}_1 \otimes \mathcal{A}_2, \mathcal{A}_1 \boxtimes \mathcal{A}_2$ are weakly unsafe*
3. *safe and non-competitive, then $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is weakly safe.*

The example below shows that weak agreement is not a context-free notion, in language theoretical sense; rather we will prove it context-sensitive. Therefore, we cannot define a most permissive controller for weak agreement in terms of contract automata, because they are finite state automata.

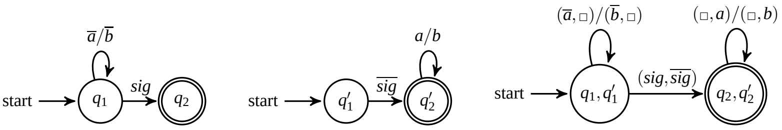


FIGURE 5. From left to right the contract automata of Example 4.7: $\mathcal{A}_4, \mathcal{A}_5$, and $\mathcal{A}_4 \otimes \mathcal{A}_5$.

Example 4.7. Let $\mathcal{A}_4$, $\mathcal{A}_5$ and $\mathcal{A}_4 \otimes \mathcal{A}_5$ be the contract automata in Figure 5, then we have that $L = \mathfrak{W} \cap \mathscr{L}(\mathcal{A}_4 \otimes \mathcal{A}_5) \neq \emptyset$ is not context-free. Consider the following regular language

$$L' = \{(\overline{a}, \square)^*(\overline{b}, \square)^*(sig, \overline{sig})(\square, a)^*(\square, b)^*\}$$

We have that

$$L \cap L' = \{(\overline{a}, \square)^{n_1}(\overline{b}, \square)^{m_1}(sig, \overline{sig})(\square, a)^{n_2}(\square, b)^{m_2} \mid n_1 \geq n_2 \geq 0, m_1 \geq m_2 \geq 0\}$$

is not context-free (by pumping lemma), and since L' is regular, L is not context-free.

Theorem 4.8. *$\mathfrak{W}$ is a context-sensitive language, but not context-free. Word decision can be done in $O(n^2)$ time and $O(n)$ space.*

In general, it is undecidable checking whether a regular language L is included in a context-sensitive one, as well as checking emptiness of the intersection of a regular language with a context-sensitive one. However in our case these two problems are decidable: we will introduce an effective procedure to check whether a contract automaton $\mathcal{A}$ is weakly safe, or whether it admits weak agreement. The technique we propose amounts to find optimal solutions to network flow problems [34], and will be used also for detecting weak liability.

As an additional comment, note that the membership problem is polynomial in time for mildly context-sensitive languages [35], but it is PSPACE-complete for arbitrary ones. In the first case, checking membership can be done in polynomial time through *two way deterministic pushdown automata* [33], that have a read-only input tape readable backwards and forwards. It turns out that $\mathfrak{W}$ is mildly context-sensitive, and checking whether $w \in \mathfrak{W}$ can be intuitively done by repeating what follows for all the actions occurring in w. Select an action α; scroll the input; and push all the requests on α on the stack; scroll again the input and pop a request, if any, when a corresponding offer is found. If at the end the stack is empty the trace w is in $\mathfrak{W}$.

Before presenting our decision procedure we fix some useful notation. Assume as given a contract automaton $\mathcal{A}$, with a single final state $\vec{q}_f \neq \vec{q}_0$. If this is not the case, one simply adds artificial dummy transitions from all the original final states to the new single final state. Clearly, if the modified contract automaton admits weak agreement, also the original one does — and the two will have the same liable principals. We assume that all states are reachable from $\vec{q}_0$ and so is $\vec{q}_f$ from each of them. In addition, we enumerate the requests of $\mathcal{A}$, i.e. $A^r = \{a^i \mid i \in I_l = \{1, 2, \dots, l\}\}$, as well as its transitions $T = \{t_1, \dots, t_n\}$. Also, let $FS(\vec{q}) = \{(\vec{q}, \vec{a}, \vec{q}') \mid (\vec{q}, \vec{a}, \vec{q}') \in T\}$ be the *forward star* of a state $\vec{q}$, and let $BS(\vec{q}) = \{(\vec{q}', \vec{a}, \vec{q}) \mid (\vec{q}', \vec{a}, \vec{q}) \in T\}$ be its *backward star*. For each transition t_i we introduce the *flow variables* $x_{t_i} \in \mathbb{N}$, and $z_{t_i}^{\vec{q}} \in \mathbb{R}$ where $\vec{q} \in Q, \vec{q} \neq \vec{q}_0$.

We are ready to define the set $F_{\vec{s},\vec{d}}$ of *flow constraints*, an element of which $\vec{x} = (x_{t_1}, \dots, x_{t_n}) \in F_{\vec{s},\vec{d}}$ defines traces from the source state $\vec{s}$ to the target state $\vec{d}$. The intuition is that each variable x_{t_i} represents how many times the transition t_i is traversed in the traces defined by $\vec{x}$. Hereafter, we will abbreviate $F_{\vec{q}_0,\vec{q}_f}$ as F_x, and we identify a transition through its source and target states.

An example follows.

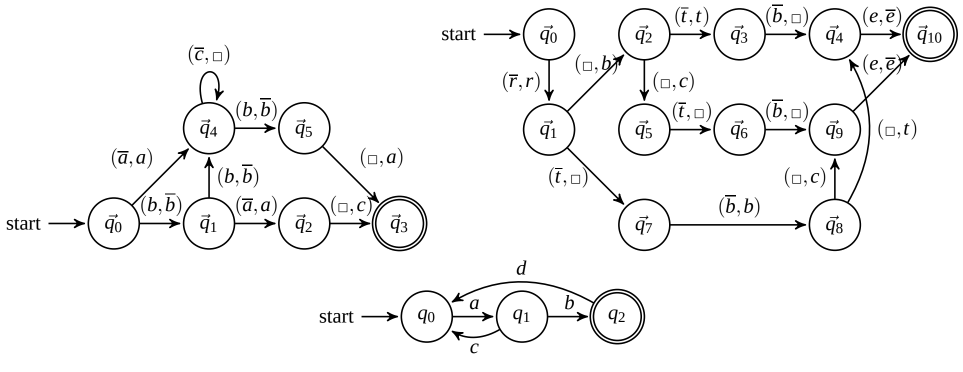


FIGURE 6. Top left: the product of two contract automata of Examples 4.9 and 4.11; top right the booking service of Example 4.9; bottom: the principal contract automaton whose flow constraints generate many traces, as discussed at the end of Example 4.9.

Example 4.9. Figure 6 (top right) shows a simple service of booking, which is the composition of a client and a hotel contracts.

The contract of the client requires to book a room (r), including breakfast (b) and a transport service, by car (c) or taxi (t); finally it sends a signal of termination ($\overline{e}$). The contract of the client is then:

$$C = r.b.(c+t).\overline{e}$$

The hotel offers a room, breakfast and taxi. Its contract is:

$$H = \overline{r}.\overline{t}.\overline{b}.e$$

Four traces accepted by the automaton $H \otimes C$ are:

$$w_1 = (\overline{r},r)(\square,b)(\overline{t},t)(\overline{b},\square)(e,\overline{e})$$
$$w_2 = (\overline{r},r)(\square,b)(\square,c)(\overline{t},\square)(\overline{b},\square)(e,\overline{e})$$
$$w_3 = (\overline{r},r)(\overline{t},\square)(\overline{b},b)(\square,t)(e,\overline{e})$$
$$w_4 = (\overline{r},r)(\overline{t},\square)(\overline{b},b)(\square,c)(e,\overline{e})$$

We now detail the flows associated with each trace giving the set of variables with value 1, all the others having value 0, because there are no loops. The associated flows are:

$$w_1 : \{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_2}, x_{\vec{q}_2,\vec{q}_3}, x_{\vec{q}_3,\vec{q}_4}, x_{\vec{q}_4,\vec{q}_{10}}\}$$
$$w_2 : \{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_2}, x_{\vec{q}_2,\vec{q}_5}, x_{\vec{q}_5,\vec{q}_6}, x_{\vec{q}_6,\vec{q}_9}, x_{\vec{q}_9,\vec{q}_{10}}\}$$
$$w_3 : \{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_7}, x_{\vec{q}_7,\vec{q}_8}, x_{\vec{q}_8,\vec{q}_4}, x_{\vec{q}_4,\vec{q}_{10}}\}$$
$$w_4 : \{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_7}, x_{\vec{q}_7,\vec{q}_8}, x_{\vec{q}_8,\vec{q}_9}, x_{\vec{q}_9,\vec{q}_{10}}\}$$

Note that a flow $\vec{x}$ may represent many traces that have the same balance of requests/offers for each action occurring therein. For example, in the contract automaton at the bottom of Figure 6, the same flow $x_{q_0,q_1} = 3, x_{q_1,q_2} = 2, x_{q_2,q_0} = x_{q_1,q_0} = 1$ represents both $w_1 = acabdab$ and $w_2 = abdacab$.

The following auxiliary definition introduces a notation for flow constraints. It is beneficial in the statements of Theorems 4.12, 4.14 and 4.17 below.

Definition 4.10. Given a source state $\vec{s}$ and a destination state $\vec{d}$, the set of *flow constraints* $F_{\vec{s},\vec{d}}$ from $\vec{s}$ to $\vec{d}$ is defined as:

$$
\begin{aligned}
F_{\vec{s},\vec{d}} = \{(x_{t_1},\dots,x_{t_n}) \mid \forall \vec{q} : (\sum_{t_i \in BS(\vec{q})} x_{t_i} - \sum_{t_i \in FS(\vec{q})} x_{t_i}) &= \begin{cases} -1 & \text{if } \vec{q} = \vec{s} \\ 0 & \text{if } \vec{q} \neq \vec{s},\vec{d} \\ 1 & \text{if } \vec{q} = \vec{d} \end{cases} \\
\forall \vec{q} \neq \vec{s}, t_i. \quad 0 \leq z^{\vec{q}}_{t_i} \leq x_{t_i}, & \\
\forall \vec{q} \neq \vec{s}, \forall \vec{q}\,' : \; (\sum_{t_i \in BS(\vec{q}\,')} z^{\vec{q}}_{t_i} - \sum_{t_i \in FS(\vec{q}\,')} z^{\vec{q}}_{t_i}\,) &= \begin{cases} -p^{\vec{q}} & \text{if } \vec{q}\,' = \vec{s} \\ 0 & \text{if } \vec{q}\,' \neq \vec{s},\vec{q} \\ p^{\vec{q}} & \text{if } \vec{q}\,' = \vec{q} \end{cases} \\
\text{where} \quad p^{\vec{q}} = \begin{cases} 1 & \text{if } \sum_{t_i \in FS(\vec{q})} x_{t_i} > 0 \\ 0 & otherwise \end{cases} \qquad \qquad & \}
\end{aligned}
$$

In the definition above, the variables $z^{\vec{q}}_{t_i}$ represent $|Q|-1$ auxiliary flows and make sure that a flow $\vec{x}$ represents valid runs only, i.e. they guarantee that there are no disconnected cycles with a positive flow. A more detailed discussion is in Example 4.11 below. Note that the values of $z^{\vec{q}}_{t_i}$ are *not* integers, and so we are defining Mixed Integer Linear Programming problems that have efficient solutions [34].

We eventually define a set of variables $a^i_{t_j}$ for each action and each transition, that take the value -1 for requests, 1 for offers, and 0 otherwise; they help counting the difference between offers and requests of an action in a flow (recall that I_l contains the indexes of the requests).

$$
\forall t_j = (\vec{q},\vec{a},\vec{q}\,') \in T, \forall i \in I_l : \quad a^i_{t_j} = \begin{cases} 1 & \text{if } Obs(\vec{a}) = \overline{a^i} \\ -1 & \text{if } Obs(\vec{a}) = a^i \\ 0 & otherwise \end{cases}
$$

Example 4.11. Figure 6 (top left) depicts the contract $A \otimes B$, where

$$
A = \overline{a}.\overline{c}^*.b + b.(b.\overline{c}^*.b + \overline{a}) \qquad\qquad B = a.\overline{b}.a + \overline{b}.(\overline{b}.\overline{b}.a + a.c)
$$

To check whether there exists a run recognising a trace w with less or equal requests than offers (for each action) we solve $\sum_{t_j} a^i_{t_j} x_{t_j} \geq 0$, for $\vec{x} \in F_x$.

We illustrate how the auxiliary variables $z^{\vec{q}}_{t_i}$ ensure that the considered solutions represent valid runs. Consider the following assignment to $\vec{x}$: $x_{\vec{q}_0,\vec{q}_1} = x_{\vec{q}_1,\vec{q}_2} = x_{\vec{q}_2,\vec{q}_3} = 1, x_{\vec{q}_4,\vec{q}_4} \geq 1$, and null everywhere else. It does not represent valid runs, because the transition $(\vec{q}_4,(\overline{c},\square),\vec{q}_4)$ cannot be fired in a run that only takes transitions with non-null values in $\vec{x}$. However, the constraints on the flow $\vec{x}$ are satisfied (e.g. we have $\sum_{t_j \in FS(\vec{q}_4)} x_{t_j} = \sum_{t_j \in BS(\vec{q}_4)} x_{t_j}$). Now the constraints on the auxiliary $z^{\vec{q}}_{t_i}$ play their role, checking if a node is reachable from the initial state on a run defined by $\vec{x}$. The assignment above is not valid since for $z^{\vec{q}_4}$ we have :

$$
0 \leq z^{\vec{q}_4}_{(\vec{q}_0,\vec{q}_4)} \leq x_{(\vec{q}_0,\vec{q}_4)} = 0
$$

$$
0 \leq z^{\vec{q}_4}_{(\vec{q}_1,\vec{q}_4)} \leq x_{(\vec{q}_1,\vec{q}_4)} = 0
$$

$$
0 \leq z^{\vec{q}_4}_{(\vec{q}_4,\vec{q}_5)} \leq x_{(\vec{q}_4,\vec{q}_5)} = 0
$$

Hence $\sum_{t_j \in BS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} = z^{\vec{q}_4}_{(\vec{q}_4,\vec{q}_4)}, \sum_{t_j \in FS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} = z^{\vec{q}_4}_{(\vec{q}_4,\vec{q}_4)}$ and we have:

$$
\sum_{t_j \in BS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} - \sum_{t_j \in FS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} = 0 \neq 1 = p^{\vec{q}_4}
$$

Finally, note in passing that there are no valid flows $\vec{x} \in F_x$ for this problem.

More importantly, note that the auxiliary variables $z_{t_i}^{\vec{q}}$ are not required to have integer values, which is immaterial for checking that those solutions represent valid runs, but makes finding them much easier.

The main results of this section follow.

Theorem 4.12. *Let $\vec{v}$ be a binary vector. Then a contract automaton $\mathcal{A}$ is* weakly safe *if and only if min $\gamma \geq 0$ where:*

$$\sum_{i \in I_l} v_i \sum_{t_j \in T} a_{t_j}^i x_{t_j} \leq \gamma \quad \sum_{i \in I_l} v_i = 1 \quad \forall i \in I_l.\, v_i \in \{0,1\} \quad (x_{t_1} \ldots x_{t_n}) \in F_x \quad \gamma \in \mathbb{R}$$

The minimum value of γ selects the trace and the action a for which the difference between the number of offers and requests is the minimal achievable from $\mathcal{A}$. If this difference is non-negative, there will always be enough offers matching the requests, and so $\mathcal{A}$ will never generate a trace not in $\mathfrak{W}$. In other words, $\mathcal{A}$ is *weakly safe*, otherwise it is not.

Example 4.13. Consider again Example 4.9 and let $a^1 = r$, $a^2 = b$, $a^3 = t$, $a^4 = c$, $a^5 = e$.
If $v_1 = 1$, for each flow $\vec{x} \in F_x$, we have that $\sum_{t_j} a_{t_j}^1 x_{t_j} = 0$ (for $i \neq 1$, we have $v_i = 0$). This means that the request of a room is always satisfied. Similarly for breakfast and the termination signal e. If $v_3 = 1$, for the flow representing the traces w_1, w_3 we have $\sum_{t_j} a_{t_j}^3 x_{t_j} = 0$, while for the flow representing the traces w_2, w_4 the result is 1. The requests are satisfied also in this case. Instead, when $v_4 = 1$, for the flow representing the traces w_1, w_4 we have $\sum_{t_j} a_{t_j}^4 x_{t_j} = 0$, but for the flow representing w_2, w_3, the result is -1. Hence $\min \gamma = -1$, and the contract automaton $H \otimes C$ is not *weakly safe*, indeed we have $w_2, w_3 \notin \mathfrak{W}$.

In a similar way, we can check if a contract automaton offers a trace in weak agreement.

Theorem 4.14. *The contract automaton $\mathcal{A}$ admits weak agreement if and only if max $\gamma \geq 0$ where*

$$\forall i \in I_l.\, \sum_{t_j \in T} a_{t_j}^i\, x_{t_j} \geq \gamma \quad (x_{t_1} \ldots x_{t_n}) \in F_x \quad \gamma \in \mathbb{R}$$

The maximum value of γ in Theorem 4.14 selects the trace w that maximises the least difference between offers and requests of an action in w. If this value is non-negative, then there exists a trace w such that for all the actions in it, the number of requests is less or equal than the number of offers. In this case, $\mathcal{A}$ admits weak agreement; otherwise it does not.

Example 4.15. In Example 4.9, $\max \gamma = -1$ for the flows representing the traces w_2, w_3 and $\max \gamma = 0$ for those of the traces w_1, w_4, that will be part of the solution and are indeed in weak agreement. Consequently, $H \otimes C$ admits weak agreement.

We now define the *weakly liable* principals: those who perform the first transition t of a run such that after t it is not possible any more to obtain a trace in $\mathfrak{W}$, i.e. leading to traces $w \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$ that cannot be extended to $ww' \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$.

Definition 4.16. Let $\mathcal{A}$ be a contract automaton and let $w = w_1 \vec{a} w_2$ such that $w \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$, $\forall w'.ww' \notin \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}, \forall w_3.w_1\vec{a}w_3 \notin \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$ and $\exists w_4.w_1w_4 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$.

The principals $\Pi^i(\mathcal{A})$ such that $\vec{a}_{(i)} \neq \square$ are *weakly liable* and form the set $WLiable(\mathcal{A}, w_1\vec{a})$.

Let $WLiable(\mathcal{A}) = \{i \mid \exists w$ such that $i \in WLiable(\mathcal{A}, w)\}$ be the set of all *potentially weakly liable* principals in $\mathcal{A}$.

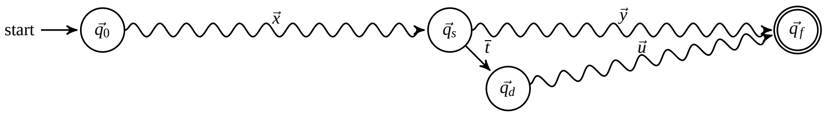


FIGURE 7. The three flows computed by Theorem 4.17

For computing the set $WLiable(\mathcal{A})$ we optimise a network flow problem for a transition $\bar{t}$ to check if there exists a trace w in which $\bar{t}$ reveals some weakly liable principals. By solving this problem for all transitions we obtain the set $WLiable(\mathcal{A})$.

Theorem 4.17. *The principal $\Pi^i(\mathcal{A})$ of a contract automaton $\mathcal{A}$ is* weakly liable *if and only if there exists a transition $\bar{t} = (\vec{q}_s, \vec{a}, \vec{q}_d) \in T_{\mathcal{A}}$, $\vec{a}_{(i)} \neq \Box$ such that $\gamma_{\bar{t}} < 0$, where*

$$\gamma_{\bar{t}} = min\ \{g(\vec{x}) \mid \vec{x} \in F_{\vec{q}_0,\vec{q}_s},\ \vec{y} \in F_{\vec{q}_s,\vec{q}_f},\ \forall i \in I_l.\ \sum_{t_j \in T} a^i_{t_j}(x_{t_j} + y_{t_j}) \geq 0\}$$

$$g(\vec{x}) = max\ \{\gamma \mid \vec{u} \in F_{\vec{q}_d,\vec{q}_f},\ \forall i \in I_l.\ \sum_{t_j \in T} a^i_{t_j}(x_{t_j} + u_{t_j}) + a^i_{\bar{t}} \geq \gamma, \gamma \in \mathbb{R}\}$$

Figure 7 might help to understand how the flows $\vec{x}$,$\vec{y}$ (and $\vec{u}$) and the transition $\bar{t}$ are composed to obtain a path from the initial to the final state. Intuitively, the flow defined above can be seen as split into three parts: the flow $\vec{x}$ from $\vec{q}_0$ to $\vec{q}_s$, the flow $\vec{y}$ from $\vec{q}_s$ to $\vec{q}_f$, and the flow $\vec{u}$ from $\vec{q}_d$ to $\vec{q}_f$, computed through the function g.

This function takes as input the flow $\vec{x}$ and selects a flow $\vec{u}$ such that, by concatenating $\vec{x}$ and $\vec{u}$ through $\bar{t}$, we obtain a trace w where the least difference between offers and requests is maximised for an action in w. Using the same argument of Theorem 4.14, if the value computed is negative, then there exists no flow $\vec{u}$ that composed with $\vec{x}$ selects traces in weak agreement.

Finally $\gamma_{\bar{t}}$ yields the minimal result of $g(\vec{x})$, provided that there exists a flow $\vec{y}$, that combined with $\vec{x}$ represents only traces in weak agreement. If $\gamma_{\bar{t}} < 0$ then the transition $\bar{t}$ identifies some *weakly liable* principals. Indeed the flow $\vec{x}$ represents the traces w such that (1) $\exists w_1$, represented by $\vec{y}$, with $ww_1 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$ and (2) $\forall w_2$, represented by $\vec{u}$, with $w\vec{a}w_2 \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$. Note that if a flow $\vec{x}$ reveals some weakly liable principals, the minimisation carried on by $\gamma_{\bar{t}}$ guarantees that the relevant transition $\bar{t}$ is found. Finding the weakly liable principals is a hard task, and belongs to the family of bilevel problems [4]. Basically, these problems contain two optimisation problems, one embedded in the other, and finding optimal solutions to them is still a hot research topic.

Example 4.18. In Figure 6 (top right), the transitions $(\vec{q}_2, (\Box, c), \vec{q}_5)$ and $(\vec{q}_8, (\Box, c), \vec{q}_9)$ reveal the second principal (i.e. C) *weakly liable*. Indeed the trace $(\bar{r}, r)(\Box, b)$ ending in $\vec{q}_2$ can be extended to one in weak agreement, while $(\bar{r}, r)(\Box, b)(\Box, c)$ cannot. Also the trace $(\bar{r}, r)(\bar{t}, \Box)(\bar{b}, b)$ can be extended to one in weak agreement while $(\bar{r}, r)(\bar{t}, \Box)(\bar{b}, b)(\Box, c)$ cannot.

For the transition $(\vec{q}_2, (\Box, c), \vec{q}_5)$ we have the trace $(\bar{r}, r)(\Box, b)$ for the flow $\vec{x}$ and $(\bar{t}, t)(\bar{b}, \Box)(e, \bar{e})$ for the flow $\vec{y}$, and we have $\forall i \in I_l. \sum_{t_j \in T} a^i_{t_j}(x_{t_j} + y_{t_j}) \geq 0$. Note that if we select as flow $\vec{y}$ the trace $(\Box, c)(\bar{t}, \Box)(\bar{b}, \Box)(e, \bar{e})$ then the constraints $\forall i \in I_l. \sum_{t_j \in T} a^i_{t_j}(x_{t_j} + y_{t_j}) \geq 0$ are not satisfied for the action $a^4 = c$ (recall Example 4.13). For the flow $\vec{u}$ the only possible trace is $(\bar{t}, \Box)(\bar{b}, \Box)(e, \bar{e})$, and max $\gamma = -1 = \gamma_{(\vec{q}_2, (\Box, c), \vec{q}_5)}$ since $\sum_{t_j \in T} a^4_{t_j}(x_{t_j} + u_{t_j}) + (-1) = -1$.

For the transition $(\vec{q}_8, (\Box, c), \vec{q}_9)$ the flow $\vec{x}$ selects the trace $(\bar{r}, r)(\bar{t}, \Box)(\bar{b}, b)$, the flow $\vec{y}$ selects the trace $(\Box, t)(e, \bar{e})$, since the other possible trace, that is $(\Box, c)(e, \bar{e})$, does not respect the constraints for the action a^4 (i.e. c). Finally, for the flow $\vec{u}$ we have the trace $(e, \bar{e})$, and as the previous case max $\gamma = -1 = \gamma_{(\vec{q}_8, (\Box, c), \vec{q}_9)}$.

5. Automata and Logics for Contracts

Recently, the problem of expressing contracts and of controlling that the principals in a composition fulfil their duties has been studied in Intuitionistic Logic, where a clause is interpreted as a principal in a contract, in turn rendered as the conjunction of several clauses. Actually, the literature only considers fragments of Horn logics because they have an immediate interpretation in terms of contracts. More in detail, these Horn fragments avoid contradiction clauses, as well as formulae with a single Horn clause. These two cases are not relevant because their interpretation as contracts makes little sense, e.g. a contract requires at least two parties. It turns out that these theories can be interpreted as contract automata, without much effort.

The first logic we consider is Propositional Contract Logic (PCL) [14] able to deal with circular obligations. Its distinguishing feature is a new implication, called *contractual implication*, that permits to assume as true the conclusions even before the premises have been proved, provided that they will be in the future. Roughly, a contract is rendered as a Horn clause, and a composition is a conjunction of contracts. When a composition is provable, then all the contracts are fulfilled, i.e. all the requests (represented as premises of implications) are entailed.

In the next sub-sections, we translate a fragment of the Horn formulae of Propositional Contract Logic into contract automata, and we prove that a formula is provable if and only if the corresponding contract automaton admits agreement.

We then study the connection between contract automata and the Intuitionistic Linear Logic with Mix (ILL^{mix})[21]. This logic is used for modelling exchange of resources between partners with the possibility of recording debts (requests satisfied by a principal offer but not yet paid back by honouring one of its requests), and has been recently given a model in terms of Petri Nets [11]. In this logic one can represent the depletion of resources, in our case of offers, that also here can be put forward before a request occurs. Again, we translate a fragment of Horn formulae as contract automata, and we prove that a theorem there corresponds to an automaton that admits agreement.

Our constructions have been inspired by analogous ones [11]; ours however offer a more flexible form of compositionality. Indeed, for checking if two separate formulas are provable, it suffices to check if the composition of the two corresponding automata is still in agreement. If the two automata are separately shown to be safe, then their composition is in agreement due to Theorem 3.18. With Debit Petri Nets [11] instead, one needs to recompute the whole translation for the composed formulas, while here we propose a modular approach.

5.1. **Propositional Contract Logic.** The usual example for showing the need of circular obligations is Example 4.1. In the Horn fragment of PCL we use, called H-PCL, the contracts of Alice and Bob make use of the new contractual implication $F \twoheadrightarrow F'$, whose intuition is that the formula F' is deducible, provided that later on in the proof also F will be deduced.

According to this intuition and elaborating over Example 4.1, Alice's contract (*I offer you my aeroplane provided that in the future you will lend me your bike*) and Bob's (*I offer you my bike provided that in the future you will lend me your aeroplane*) are rendered as $bike \twoheadrightarrow airplane$, $airplane \twoheadrightarrow bike$, respectively. Their composition is obtained by joining the two, and one represents that both Alice and Bob are proved to obtain the toy they request by

$$((bike \twoheadrightarrow airplane) \wedge (airplane \twoheadrightarrow bike)) \vdash (bike \wedge airplane)$$

In words, the composition of the two contracts entails *all* the requests (*bike* by Alice and *airplane* by Bob). We now formally introduce the fragment of H-PCL [5, 7] that has a neat interpretation in contract automata, under the assumption that a principal cannot offer and require the same.

$$\frac{\Gamma \vdash q}{\Gamma \vdash p \twoheadrightarrow q}\,\mathit{Zero} \qquad \frac{\Gamma, p \twoheadrightarrow q, r \vdash p \quad \Gamma, p \twoheadrightarrow q, p \vdash q}{\Gamma, p \twoheadrightarrow q \vdash r}\,\mathit{Fix}$$

$$\frac{\Gamma, p \twoheadrightarrow q, a \vdash p \quad \Gamma, p \twoheadrightarrow q, q \vdash b}{\Gamma, p \twoheadrightarrow q \vdash a \twoheadrightarrow b}\,\mathit{PrePost}$$

FIGURE 8. The three rules of PCL for the contractual implication.

Definition 5.1 (H-PCL)**.** Assume a denumerable set of atomic formulae $Atoms = \{a, b, c, \dots\}$ indexed by $i \in I, j \in J$ where I and J are finite set of indexes; then the *H-PCL formulae* $p, p', \dots$ and the clauses $\alpha, \alpha_i, \dots$ are generated by the following BNF grammar

$$p ::= \bigwedge_{i \in I} \alpha_i \qquad \alpha ::= \bigwedge_{j \in J} a_j \mid (\bigwedge_{j \in J} a_j) \to b \mid (\bigwedge_{j \in J} a_j) \twoheadrightarrow b$$
$$\text{where } |I| \geq 2, |J| \geq 1, i \neq j \text{ implies } a_i \neq a_j, \text{ and } \forall j \in J. a_j \neq b$$

Also, let $\lambda(p)$ be the conjunction of all atoms in p.

In Figure 8 we recall the three rules of the sequent calculus for the contractual implication [14, 13]; the others are the standard ones of the Intuitionistic Logic and are in the appendix, Figure 13.

As anticipated, in H-PCL all requests of principals are satisfied if and only if the conjunction p of the contracts of all principals entails all the atoms mentioned.

Definition 5.2. The formula p represents a composition whose principals respect all their obligations if and only if $p \vdash \lambda(p)$.

Below, we define the translation from H-PCL formulae to contract automata. A simple inspection of the rules below suffices to verify that the obtained automata are deterministic.

Definition 5.3 (From H-PCL to CA)**.** A H-PCL formula, with sets of indexes I and J as in Definition 5.1, is translated into a contract automaton by the following rules, where $\mathcal{P} = \{q \cup \{*\} \mid q \in 2^J\}$:

$$[\![\bigwedge_{i \in I} \alpha_i]\!] = \boxtimes_{i \in I} [\![\alpha_i]\!]$$

$$[\![\bigwedge_{j \in J} a_j]\!] = \langle \{\{*\}\}, \{*\}, \emptyset, \{\overline{a_j} \mid j \in J\}, \{(\{*\}, \overline{a_j}, \{*\}) \mid \overline{a_j} \in A^o\}, \{\{*\}\} \rangle$$

$$\begin{aligned}[\![(\bigwedge_{j \in J} a_j) \to b]\!] =& \langle \mathcal{P}, J \cup \{*\}, \{a_j \mid j \in J\}, \{\overline{b}\}, \\ & \{(J' \cup \{j\}, a_j, J') \mid J' \cup \{j\} \in \mathcal{P}, j \in J\} \cup \{(\{*\}, \overline{b}, \{*\})\}, \{\{*\}\} \rangle\end{aligned}$$

$$\begin{aligned}[\![(\bigwedge_{j \in J} a_j) \twoheadrightarrow b]\!] =& \langle \mathcal{P}, J \cup \{*\}, \{a_j \mid j \in J\}, \{\overline{b}\}, \\ & \{(J' \cup \{j\}, a_j, J') \mid J' \cup \{j\} \in \mathcal{P}, j \in J\} \cup \{(q, \overline{b}, q) \mid q \in \mathcal{P}\}, \{\{*\}\} \rangle\end{aligned}$$

As expected, a Horn formula is translated as the product of the automata raising from its components α_i. In turn, a conjunction of atoms yields an automaton with a single state and loops driven by offers in bijection with the atoms. Each state stores the number of requests that are waiting to fire, and $\{*\}$ stands for no requests. A (standard) implication shuffles all the requests corresponding to the premises a_j and then has the single offer corresponding to the conclusion b. A contractual implication is similar, except that the offer ($\overline{b}$ in the definition) can occur at *any* position in the shuffle, and from there onwards it will be always available. Note that there is no control on the number of times an offer can be taken, as H-PCL is not a linear logic.

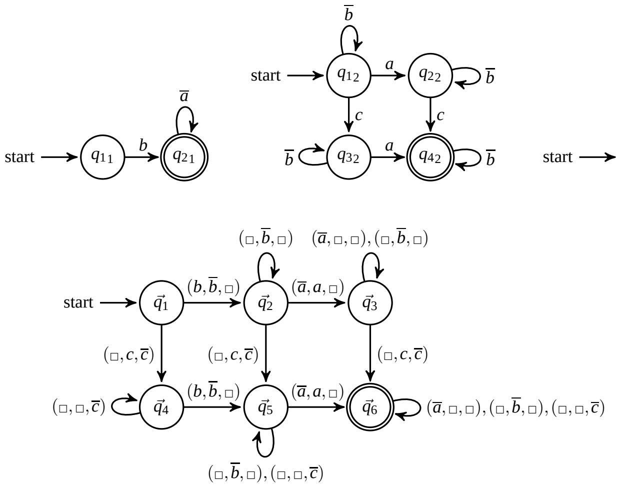

start
q_{11}
b
q_{21}
$\overline{a}$
start
q_{12}
a
q_{22}
$\overline{b}$
c
c
q_{32}
a
q_{42}
$\overline{b}$
$\overline{b}$
$\overline{b}$
start
q_{13}
$\overline{c}$
start
$\vec{q}_1$
$(b,\overline{b},\square)$
$\vec{q}_2$
$(\overline{a},a,\square)$
$\vec{q}_3$
$(\square,\overline{b},\square)$
$(\overline{a},\square,\square),(\square,\overline{b},\square)$
$(\square,c,\overline{c})$
$(\square,c,\overline{c})$
$(\square,c,\overline{c})$
$(\square,\square,\overline{c})$
$\vec{q}_4$
$(b,\overline{b},\square)$
$\vec{q}_5$
$(\overline{a},a,\square)$
$\vec{q}_6$
$(\overline{a},\square,\square),(\square,\overline{b},\square),(\square,\square,\overline{c})$
$(\square,\overline{b},\square),(\square,\square,\overline{c})$

FIGURE 9. The contract automata of Examples 5.4 and 9.11, top from left to right: $[\![Alice]\!], [\![Bob]\!], [\![Charlie]\!]$; bottom: $\mathcal{K}_{[\![Alice]\!]\otimes[\![Bob]\!]\otimes[\![Charlie]\!]}$.

Example 5.4. Consider again Example 4.1, and let us modify it to better illustrate some peculiarities of H-PCL. Assume then that there are three kids: Alice, Bob and Charlie, who want to share some toys of theirs: a bike b, an aeroplane a and a car c. The contract of Alice says "I will lend you my aeroplane provided that you lend me your bike". The contract of Bob says "I will lend you my bike provided that in the future you will lend me your aeroplane and your car". The contract of Charlie says "I will lend you my car". The contract of Alice is expressed by the classical implication $b \to a$. The contract of Bob is $(a \wedge c) \twoheadrightarrow b$, while the contract of Charlie is simply c. The three contracts reach an agreement: the conjunction of the formulae representing the contracts entails all its atoms, that is $(b \to a) \wedge ((a \wedge c) \twoheadrightarrow b) \wedge c \vdash a \wedge c \wedge b$.

Figure 9 shows the translation of $Alice \wedge Bob \wedge Charlie$, according to Definition 5.3. It is immediate verifying that the automaton is safe, since all its traces are in agreement.

The following proposition helps to understand the main result of this section.

Proposition 5.5. *Given a H-PCL formula p and the automaton $[\![p]\!] = \langle Q, q_0, A^r, A^o, T, F\rangle$:*

(1) *$F = \{\vec{q} = \langle\{*\},\dots,\{*\}\rangle\}$, and all $(\vec{q},\vec{a},\vec{q'})$ are such that $\vec{q'} = \vec{q}$ and $\vec{a}$ is an offer;*
(2) *every state $\vec{q} = \langle J_1,\dots,J_n\rangle$ has as many request or match outgoing transitions as the request actions prescribed by $\bigcup_{i\in 1\dots n} J_i$;*
(3) *$[\![p]\!]$ is deterministic.*

As said above, when seen in terms of composed contracts, the formula $p \vdash \lambda(p)$ expresses that all the requests made by principals in p must be fulfilled sooner or later. We now show that the contract automaton $[\![p]\!]$ admits agreement if and only if $p \vdash \lambda(p)$ is provable.

Theorem 5.6. *Given a H-PCL formula p we have $p \vdash \lambda(p)$ if and only if $[\![p]\!]$ admits agreement.*

We have constructively proved that a formula p fulfils all its obligations if and only if the corresponding automaton $[\![p]\!]$ admits agreement. Interestingly, a contractual implication $a \twoheadrightarrow b$ corresponds to a contract automaton that is enabled to fire the conclusion b at each state; while for the standard implication $c \to d$ the conclusion is available only after the premise c has been satisfied.

Example 5.7. Consider Example 5.4. The conjunction of all the formulas entails its atoms, indeed the corresponding translation into contract automata displayed in Figure 9 admits agreement.

Needless to say, the provability of $p \vdash \lambda(p)$ implies that $[\![p]\!]$ admits weak agreement. However, the implication is in one direction only, as shown by the following example.

Example 5.8. Consider the H-PCL formula $p = (b \twoheadrightarrow a) \wedge (a \rightarrow b)$. We have that $[\![p]\!]$ does not admit agreement and $p \not\vdash \lambda(p)$. Nevertheless $[\![p]\!]$ admits weak agreement. For example, $(b,-)(\overline{a},a)(-,\overline{b}) \in \mathscr{L}([\![p]\!])$ is a trace in weak agreement.

As a matter of fact, weak agreement implies provability when a formula p contains no (standard) implications, as stated below.

Theorem 5.9. *Let p be a H-PCL formula with no occurrence of standard implications $\rightarrow$, then $p \vdash \lambda(p)$ if and only if $[\![p]\!]$ admits weak agreement.*

This result helps to gain insights on the relation between the contractual implication $\twoheadrightarrow$ and the property of weak agreement. Indeed, checking weak agreement on a contract automaton $[\![p]\!]$ is equivalent to prove that the formula p fulfils all its obligations (i.e. $p \vdash \lambda(p)$) *only if* p contains no standard implication $\rightarrow$.

5.2. **Intuitional Linear Logic with Mix.** In this sub-section, we will interpret a fragment of the Intuitionistic Linear Logic with Mix (*ILL*^{mix}) [21] in terms of contract automata. Originally, this logic has been used for modelling exchange of resources between partners with the possibility of recording debts, through the so-called *negative atoms*. Below, we slightly modify Example 5.4 to better illustrate some features of *ILL*^{mix}.

Example 5.10. Alice, Bob and Charlie want to share their bike, aeroplane and car, according to the same contracts declared in Example 5.4. In *ILL*^{mix} the contract of Alice is expressed by the linear implication $b \multimap a$; the contract of Bob is $a^{\perp} \otimes c^{\perp} \otimes b$ ($\otimes$ is the tensor product of Linear Logic); the contract of Charlie is the offer c. The intuition is that a positive atom, e.g. c in the contract of Charlie, represents a resource that can be used; similarly for the b of Bob. Instead, the negative atoms ($a^{\perp}$ and $c^{\perp}$ of Bob) represent missing resources that however can be taken on credit to be honoured later on. The implication of Alice says that the resource a is produced by consuming b, provided b is available. (There are some restrictions on the occurrences of negative atoms made precise below). The composition (via tensor product) of the three contracts is successful, in that all resources are exchanged and all debts honoured. Indeed, it is possible to prove that all the negative atoms, i.e. all the requests, will be eventually satisfied. In this case we have that all the resources are consumed, and that the following sequent is provable: $Alice \otimes Bob \otimes Charlie \vdash$.

We now recall the basics of *ILL*^{mix}. Let $\mathbf{A}, \mathbf{A}^{\perp}$ be respectively the set of *positive* and *negative atoms*, ranged over by $a,b,c,\ldots \in \mathbf{A}$ and by $a^{\perp}, b^{\perp}, c^{\perp}, \cdots \in \mathbf{A}^{\perp}$. Let $\mathbf{L} = \mathbf{A} \cup \mathbf{A}^{\perp}$ be the set of *literals*, and assume $Y \subseteq \mathbf{A}, X \subseteq \mathbf{L}$, where X does not contain any atom a and its negation $a^{\perp}$, according to Definition 2.3 (recall that a principal automaton is such that $A^r \cap co(A^o) = \emptyset$). A *positive* tensor product is a tensor product of positive atoms.

As said, we only consider a fragment of Horn *ILL*^{mix} called H-*ILL*^{mix}, defined below. It only has tensor products and *Horn implications*: $\bigotimes_{b \in Y} b \multimap \bigotimes_{a \in X} a$. Note that the premises of the Horn implications are always positive tensor products, and the conclusions are tensor products of literals, possibly negative.

Since the treatment for non-linear implications of *ILL*^{mix} is similar to that presented in Section 5.1, we feel free to only deal below with linear implications and tensor products of literals.

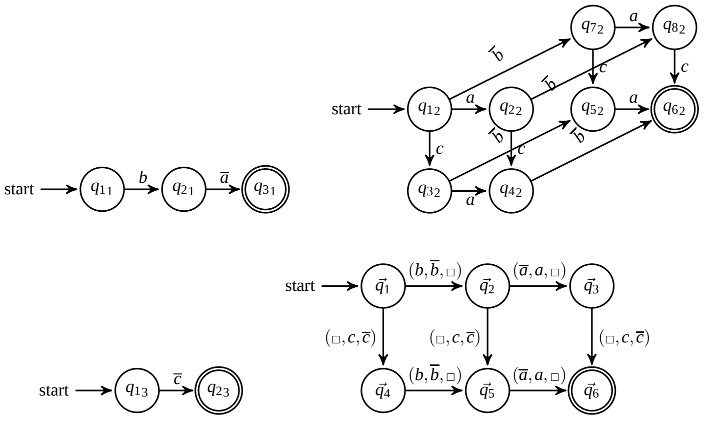


FIGURE 10. The contract automata of Example 5.10. Top from left to right: $[\![Alice]\!], [\![Bob]\!]$. Bottom from left to right: $[\![Charlie]\!], \mathcal{K}_{[\![Alice]\!]\boxtimes[\![Bob]\!]\boxtimes[\![Charlie]\!]}$.

$$\frac{}{A \vdash A}\ Ax \qquad \frac{\Gamma \vdash \quad \Gamma' \vdash \gamma}{\Gamma,\Gamma' \vdash \gamma}\ Mix \qquad \frac{\Gamma \vdash A}{\Gamma, A^{\perp} \vdash}\ NegL$$

$$\frac{\Gamma, A, B \vdash \gamma}{\Gamma, A \otimes B \vdash \gamma}\ \otimes L \qquad \frac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma,\Gamma' \vdash A \otimes B}\ \otimes R$$

$$\frac{\Gamma \vdash A \quad \Gamma', B \vdash \gamma}{\Gamma,\Gamma', A \multimap B \vdash \gamma}\ \multimap L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ \multimap R$$

FIGURE 11. A subset of the rules of the sequent calculus of ILL^{mix}.

Definition 5.11 (H-ILL^{mix})**.** The Horn formulae $p, p_i, \ldots$ and the clauses $\alpha, \alpha_i, \ldots$ of $H\text{-}ILL^{mix}$ are defined by

$$p ::= \bigotimes_{i \in I} \alpha_i \qquad \alpha ::= \bigotimes_{a \in X} a \mid \bigotimes_{b \in Y} b \multimap \bigotimes_{a \in X} a$$

where $|I| \geq 2$; $|X|, |Y| \geq 1$; $\{a, a^{\perp}\} \not\subseteq X$; and $b \in Y$ implies $b \notin X$.

The subset of the rules of the sequent calculus of ILL^{mix} relevant to our treatment is in Figure 11, where A, B stand for a Horn formula p or clause α, while γ may also be empty (note that in rule $(NegL)$, $A = a$ and so $A^{\perp} = a^{\perp}$); Γ and Γ' stand for multi-sets containing Horn formulae or clauses; and Γ, Γ' is the multi-set union of Γ and Γ', assuming $\Gamma, \emptyset = \Gamma$. The complete set of rules for ILL^{mix} is in [21], and can be found in the appendix.

The following auxiliary definition of the concatenation of two automata helps to translate a H-ILL^{mix} formula.

Definition 5.12 (Concatenation of CA)**.** Given two principal contract automata $\mathcal{A}^1 = \langle Q^1, q_0^1, A^{r^1}, A^{o^1}, T^1, F^1 \rangle$ and $\mathcal{A}^2 = \langle Q^2, q_0^2, A^{r^2}, A^{o^2}, T^2, F^2 \rangle$, their *concatenation* is

$$\begin{aligned} \mathcal{A}^1 \cdot \mathcal{A}^2 = \ & \langle Q^1 \cup Q^2, q_0^1, A^{r1} \cup A^{r2}, A^{o1} \cup A^{o2}, \\ & (T^1 \setminus \{(q,a,q') \in T^1 \mid q' \in F^1\}) \cup T^2 \\ & \qquad \cup \{(q,a,q_0^2) \mid (q,a,q') \in T^1, q' \in F^1\}, F^2 \rangle \end{aligned}$$

Concatenation is almost standard, with the proviso that we replace every transition of $\mathcal{A}^1$ leading to a final state with a transition with the same label leading to the initial state of $\mathcal{A}^2$. Note also that loops can be ignored, because the automata obtained by the translation in Definition 5.13 below have no cycles.

Similarly to what has been done in the previous sub-section, a tensor product is rendered as all the possible orders in which the automaton can fire (the actions corresponding to) its literals. If the literal is a positive atom, then it becomes an offer, while it originates a request if the atom is negative. A linear implication is rendered as the concatenation of the automaton coming from the premise, and that of the conclusion, with the following proviso. In the premise all the atoms are positive, but they are *all* rendered as *requests* (i.e. as negative atoms), and shuffled. The states are in correspondence with the atoms still to be fired and $\{*\}$ stands for the (final) state where all atoms have been fired.

Definition 5.13 (Translation of H-*ILL*^{mix})**.** Given a set of atoms X, let $P = \{q \cup \{*\} \mid q \in 2^X\}$ with typical element Z. The translation of a H-*ILL*^{mix} formula p into a contract automata $[\![p]\!]$ is inductively defined by the following rules:

$$[\![\bigotimes_{i\in I} \alpha_i]\!] = \boxtimes_{i\in I}[\![\alpha_i]\!]$$

$$\begin{aligned}[\![\bigotimes_{a\in X} a]\!] = & \langle P, X\cup\{*\}, \{a \mid a^{\perp} \in X \cap \mathbf{A}^{\perp}\}, \{\overline{a} \mid a \in X\cap \mathbf{A}\}, \\ & \{(Z\cup\{a^{\perp}\}, a, Z) \mid Z\cup\{a^{\perp}\} \in P, a^{\perp} \in X\}\cup \\ & \{(Z\cup\{a\}, \overline{a}, Z) \mid Z\cup\{a\} \in P, a \in X\}, \\ & \{\{*\}\}\rangle\end{aligned}$$

$$[\![\bigotimes_{b\in Y} b \multimap \bigotimes_{a\in X} a]\!] = [\![\bigotimes_{b\in Y} b^{\perp}]\!] \cdot [\![\bigotimes_{a\in X} a]\!]$$

Moreover, we homomorphically translate multi-sets of Horn formulae and clauses as follows:

$$[\![p,\Gamma]\!] = [\![p]\!] \boxtimes [\![\Gamma]\!] \qquad\qquad [\![\alpha,\Gamma]\!] = [\![\alpha]\!] \boxtimes [\![\Gamma]\!]$$

The automata obtained by translating the formulae representing the contracts of Alice, Bob and Charlie in Example 5.10 are in Figure 10.

Definition 5.14. A sequent $\Gamma \vdash Z$ is *honoured* if and only if it is provable and Z is a positive tensor product or empty.

Intuitively, honoured sequents can be proved and additionally they have no negative atoms, i.e. no debts. The main result of this section is that a sequent $\Gamma \vdash Z$ is honoured if and only if the corresponding contract automaton $[\![\Gamma]\!]$ admits agreement. An important outcome is the possibility of expressing each H-*ILL*^{mix} formula as a contract automaton $\mathcal{A}$, so to use our verification techniques. It is then possible to compose several H-*ILL*^{mix} formulae through the composition operators of contract automata, exploiting compositionality and the related results (for example Theorem 3.18) for efficiently checking the provability of formulae in H-*ILL*^{mix}. In the statement below and in the proofs in the appendix, we say that $[\![\Gamma]\!]$ admits agreement on Z whenever there exists a trace in $\mathscr{L}([\![\Gamma]\!])$ only made of match actions and offers in correspondence with the literals in Z.

Theorem 5.15. *Given a multi-set of Horn formulae* Γ*, we have that*

$$\Gamma \vdash Z \text{ is an honoured sequent if and only if } [\![\Gamma]\!] \text{ admits agreement on } Z$$

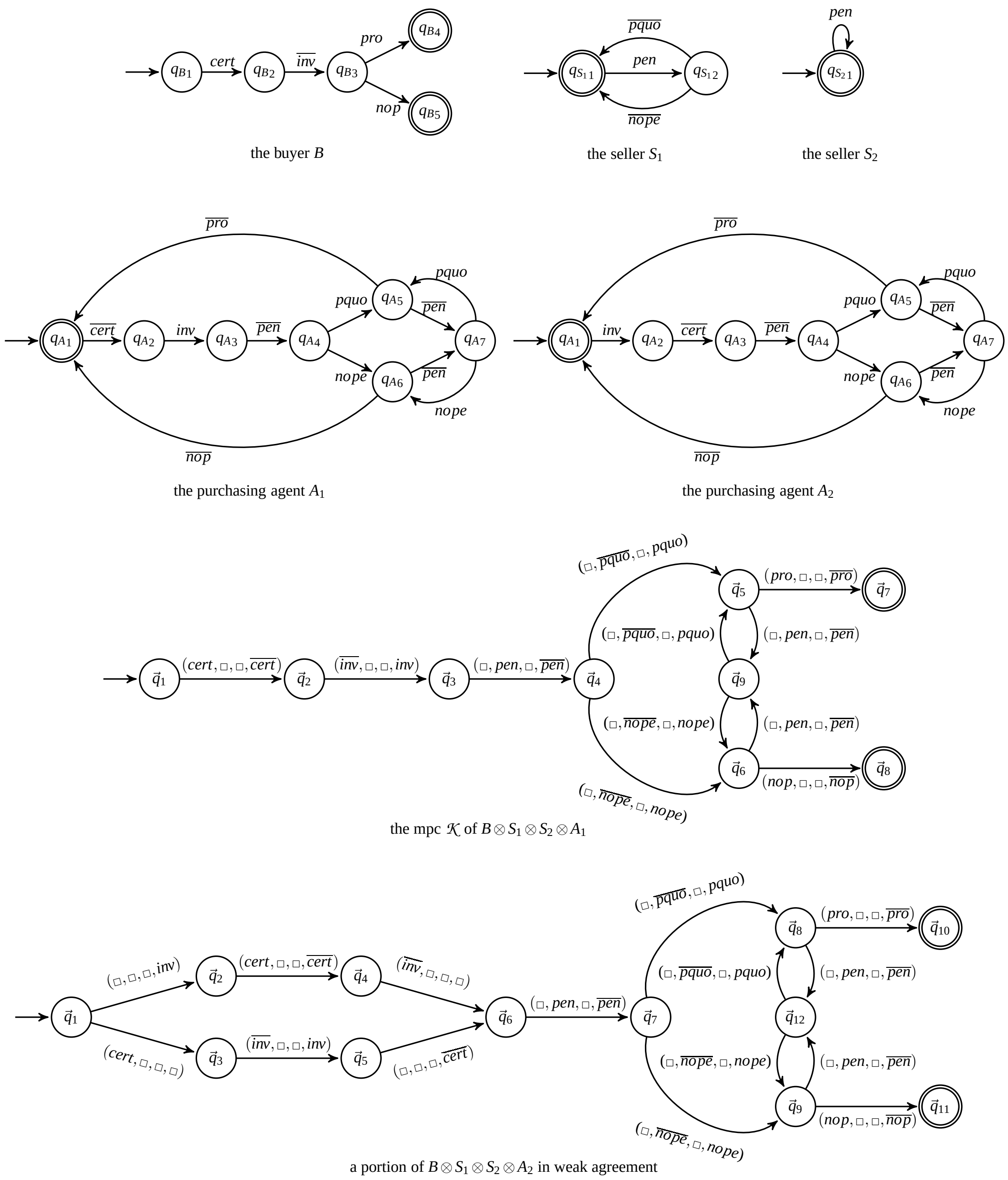
Through this result we have linked the problem of verifying the correctness of a composition of services to the generation of a deduction tree that proves a H-ILL^{mix} formula. Moreover, we have shown that the possibility of recording debts in H-ILL^{mix} solves circularity issues arising from a composition of services.

6. An example

In this section we consider a well-known case study taken from [45]. This is a purchasing system scenario, where a manufacturer (the buyer) wants to build a product. To configure it, the buyer lists in an inventory the needed components and contacts a purchasing agent. The agent looks for suppliers of these components, and eventually sends back to the buyer its proposal, if any. A supplier is assumed to signal whether it can fulfil a request or not; if neither may happen, the interactions between it and the purchasing agent are rolled back, so as to guarantee the transactional integrity of the overall process. A description of the WSDL of the services, as well as the BPEL process from the purchasing agent's perspective are in [45], where the transactional integrity is maintained using the tags `<faultHandlers>` and `<scope>` of BPEL.

We slightly modify the original protocol, where the purchasing agent guarantees its identity to the buyer through a public-key certificate. For brevity, here we assume to have two sellers S_1 and S_2, and two purchasing agents A_1 and A_2, that behave differently. A service instance involves the buyer, an agent and both sellers. The buyer B requires the certificate of an agent (action $cert$), then it offers the inventory requirements ($\overline{inv}$). Finally, it terminates by receiving either a proposal (pro) or a negative message (nop), if no proposal can be formulated. The seller S_1 waits for a request (pen) of a component from an agent. It then replies by offering a quote for that part ($\overline{pquo}$), or a negative message ($\overline{nope}$) if it is unavailable, and restarts. The second seller S_2 always accepts a request, but never replies. The first agent A_1 offers its certificate ($\overline{cert}$), then requires the inventory list (inv). It then sends a request to and waits for a reply from the sellers. The agent must communicate at least with one supplier before replying to the buyer, and it can span over all the available suppliers in the network, unknown a priori, before compiling its proposal. Finally, it sends to the buyer a proposal ($\overline{prop}$), or the negative message ($\overline{nop}$). The second agent A_2 behaves similarly to A_1, except the first two actions are exchanged: before sending its certificate to B, it first requires the inventory list.

In Figure 12 from top to bottom, we display, from left to right, the automata B, S_1 and S_2; the automata A_1 and A_2; then the most permissive controller $\mathcal{K}$ of $B \otimes S_1 \otimes S_2 \otimes A_1$ (the whole composition is omitted to save space); finally a portion of $B \otimes S_1 \otimes S_2 \otimes A_2$ in weak agreement. This example shows that through contract automata one can identify which traces reach success, and which a failure, together with those principals responsible for diverging from the behaviour in agreement, as well as to single out which failures depend on the order of actions, and which not. Indeed, by inspecting $\mathcal{K}$, that of course is safe, one can notice that A_1 never interacts with S_2 because it never replies and so it is recognised liable. As a matter of fact, the composed automaton $B \otimes S_1 \otimes S_2 \otimes A_1$ admits agreement, but it is not safe. Note that $\mathcal{K}$ blocks every communication with S_2, so enforcing transactional integrity, because $\mathcal{K}$ removes all possibilities of rollbacks from a trace not in agreement. The composed automaton $B \otimes S_1 \otimes S_2 \otimes A_2$ admits weak agreement but not agreement (and its most permissive controller is empty), because B and A_2 fail in exchanging the certificate and the inventory requirements, as both are stuck waiting for the fulfilment of their



the buyer B the seller S_1 the seller S_2

the purchasing agent A_1 the purchasing agent A_2

the mpc $\mathcal{K}$ of $B \otimes S_1 \otimes S_2 \otimes A_1$

a portion of $B \otimes S_1 \otimes S_2 \otimes A_2$ in weak agreement

FIGURE 12. The contract automata for the example

requests. However, by abstracting away the order in which actions are performed, circularity is no longer a problem, and these requests satisfied. Note that S_2 is detected to be also weakly liable.

7. RELATED WORK

Contract automata are similar to I/O [40] and Interface Automata [1], introduced in the field of Component Based Software Engineering. A first difference is that principal contract automata have no internal transitions, and that our operators of composition track each principal, to find the possible liable ones. Also we do not allow input enabled operations and non-linear behaviour (i.e. broadcasting offers to every possible request), and our notion of agreement is dual to that of compatibility in [1], that requires all the *offers* to be matched.

We now relate our approach to the growing body of work in the literature introduced to describe and analyse service contracts.

Behavioural contracts. In [24] the behaviour of web-services is described through automata, equivalent to our principal contract automata. However, only bi-party interactions are considered, i.e. interactions between a single client and a single server, while our model deals with multi-party interactions through orchestration. Different notions of compliance are introduced, and one of them is close to our notion of agreement. In [27] behavioural contracts are expressed in CCS and the interactions between services are modelled via I/O actions. The main focus of this work is on formalising the notion of progress of interactions. Two different choice operators, namely internal and external, describe how two services interact. The internal choice requires the other party to be able to synchronise with all the possible branches of the first, while for the external choice it suffices to synchronise with at least one branch. A client and a server are compliant if their interactions never get stuck. This approach is extended to a multi-party version by extending the π-calculus in [28] with the above notions of non-deterministic choice. Our model represents internal/external choice as a branching of requests/offers, and it is intrinsically multi-party. Also, we consider stronger properties than theirs: progress guarantees that a subset of contracts meets their requests, while (weak) agreement requires that all of them do, i.e. that each principal reaches a successful state. We also consider (weak) liability of principals, and conditions under which (weak) safety is preserved by composition (collaborative and competitive). A CCS-like process calculus, called BPEL *abstract activities* is used in [37] to represent BPEL activities [42], for connecting BPEL processes with contracts in [27]. The calculus is endowed with a notion of compliance and sub-contract relation (see below). Contract automata and this formalism are very close, e.g. both are finite state, so it would not be difficult to formally relate them.

In [43] the approach of [27] is extended by exploiting an orchestrator for managing the *sub-contract* relation. A contract σ_1 is sub-contract of σ_2 if σ_1 is more deterministic or allows more interactions or is a permutation of the same channels of σ_2. However, it is not always the case that a contract σ, compliant with σ_1, is also compliant with σ_2. A technique for synthesising an orchestrator is presented to enforce compliance of contracts under the sub-contract relation. This approach is further extended in [2], where an orchestrator is synthesised from *session contracts*, where actions in a branching can only be all inputs or outputs. Only bi-party contracts are considered, and synthesis is decidable even in the presence of messages never delivered to the receiver (orphan messages). Two notions of compliance are studied: respectful and disrespectful. In the first, orphan messages and circularities are ruled out by the orchestrator, while in the second they are allowed. Our notion of weak agreement is close to the orchestrator of [43, 2] in the case of *disrespectful compliance*.

In [3] the contracts of [27] are enriched with a mechanism for recovering from a stuck computation. The external choices are called *retractable*, and a client contract $\overline{a}+\overline{b}$ is compliant with a server a since, in case the client decides to send b, it can retract the choice and perform the correct operation $\overline{a}$. In our work, the controller for the case of agreement cuts all the paths which may lead one principal to perform a retract. Hence, a controlled interaction of services needs not to roll back, as the orchestrator *prevents* firing of liable transitions. This means that, if a composition of contracts is safe then the contracts are compliant according to [3]. The converse does not hold. Indeed, our notion of agreement is stronger, as we force an interaction of services to reach a successful state.

The compliance relations studied in [27, 28, 37, 43, 2, 3] are mainly inspired by testing equivalence [41]: a CCS process (in our case the service) is tested against an observer (the client), in two different ways. A service *may-satisfy* a client if there exists a computation that ends in a successful state, and a service *must-satisfy* a client if in every maximal trace (an infinite trace or a trace that can not be prolonged) the client can terminate successfully. We conjecture that may-test corresponds to the notion of *strong agreement* of [20, 16] (there exists a trace only composed of matches), while must-test implies *strong safety* (all traces are in strong agreement), but not vice-versa. For example the service $\overline{a}^*.\overline{b}$ does not must-satisfy the client $a^*.b$, but their product is strongly safe (if unfair, the service may never offer $\overline{b}$ to its client). Actually, strong safety is alike *should testing* of [47], where the divergent computations are ruled out.

Session types and choreographies. Session types have been introduced to reason over the behaviour of communicating processes, and are used for typing channel names by structured sequences of types [31]. Session types can be global or local. A *global type* represents a formal specification of a choreography of services in terms of their interactions. The projection of a safe global type to its components yields a safe *local type*, which is a term of a process algebra similar to those of [27]. Conversely, from safe local types it is possible to synthesise a choreography as a safe global type [38, 39]. In [22] the contracts of [27] are shown to be a model of first-order session types [32]. This approach is then extended in [23] by introducing a notion of higher-order contracts and relating them to higher-order session types, that also handle session delegation.

Although the above approaches and ours seem unrelated, one can compare them by resorting to communicating finite states machines [25], that are finite state automata similar to ours, to which local types are proved to correspond [30]. These automata interact through FIFO buffers, hence a principal can receive an input only if it was previously enqueued, and in this they differ from contract automata, where offers and requests can match or even fire unmatched in any order. However, under mild conditions, the two classes of automata are equivalent [20, 16], so establishing a first bridge between the choreography model based on session types and our automata model of orchestration.

Many properties of communicating finite state machines, as compliance in the asynchronous case, are not decidable in general [25], but some become such by using FIFO queues and bags [29]. Moreover in [39] compliance between communicating finite state machines is guaranteed whenever it is possible to synthesise a global choreography from them. It would be interesting to describe compliance of [25] in terms of flow control, as done for weak agreement, and to study a relaxation of the linear problem which makes the problem decidable.

In [37] the compliance and sub-contract relations are extended to deal with choreographies. Compliance is obtained by seeing a choreography as a compound service, similarly to our composed contract automata. Since a client cannot interact with the choreography on actions already used while synchronising by other services, in order to obtain compliance the client must be *non-competitive* with the other services.

λ-calculus, logics, event-structures. Services are represented in [10, 9] by λ-expressions, and safety policies are imposed over their interactions. A type and effect system is used to compute the types of the services and their abstract behaviours, that are then model checked at static time to guarantee that the required policies are always satisfied. A main result shows how to construct a plan that associates requests with offers so to guarantee that no executions will violate the security requirements. In [17, 19] these techniques have been applied to an automata based representation of the contracts of [27], recovering the same notion of progress.

Propositional Contract Logic [14] and Intuitionistic Linear Logic with Mix [21] have been already discussed in Section 5.

Processes and contracts are two separate entities in [12], unlike ours. In this formalism contracts are represented as formulae or as process algebras. A process can fulfil its duty by obeying its contract or it behaves dishonestly and becomes *culpable* — and redeems by performing later on the prescribed actions. Also our principals can be at fault, but our notion of liability slightly differs from culpability, mainly because we do not admit the possibility of redeeming.

Contracts are represented in [6] through Event Structures endowed with certain notions from Game Theory. An agreement property is proposed, ensuring safe interactions among participants, that is similar to ours under an eager strategy. A principal is culpable if it has not yet fired an enabled event, it is otherwise innocent. In particular a principal agrees to a contract if it has a positive payoff in case all the principals are innocent, or if someone else is found culpable. Additionally the authors study protection: a protected principal has a non-losing strategy in every possible context, but this is not always possible. Finally two encodings from session types to Event Structures are proposed, and compliance between bi-party session types is shown to correspond to agreement of the corresponding event structures via an eager strategy.

8. Concluding Remarks

We have studied contract composition for services, focussing on orchestration. Services are formally represented by a novel class of finite state automata, called contract automata. They have two operators that compose services according to two different notions of orchestrations: one when a principal joins an existing orchestration with no need of a global reconfiguration, and the other when a global adaptive re-orchestration is required. We have defined notions that illustrate when a composition of contracts behaves well, roughly when all the requests are fulfilled. These properties have been formalised as agreement and safety, and have been studied both in the case when requests are satisfied synchronously and asynchronously. Furthermore, a notion of liability has been put forward. A liable principal is a service leading the contract composition into a fail state. Key results of the paper are ways to enforce good behaviour of services. For the synchronous versions of agreement and safety, we have applied techniques from Control Theory, while for the asynchronous versions we have taken advantage of Linear Programming techniques borrowed from Operational Research. Using them, we efficiently find the optimal solutions of the flow in the network automatically derived from contract automata.

We have also investigated the relationships between our contract automata and two intuitionistic logics, particularly relevant for their ability in describing the potential, but harmless and often essential circularity occurring in services. We have considered a fragment of the Propositional Contract Logic [14, 13] particularly suited to describe contracts, and we relate it through a translation of its formulas into contract automata. Similarly, we have examined certain sequents of the Intuitionistic Linear Logic with Mix that naturally represent contracts in which all requests are satisfied. Then we have proved that these sequents are provable if and only if a suitable translation of them as contract automata admits agreement.

A main advantage of our framework is that it supports the development of automatic verification tools for checking and verifying properties of contract composition. In particular, the formal treatment of contract composition in terms of optimal solutions of network flows paves the way of exploiting efficient optimisation algorithms. We have developed a prototypical verification tool [15], available at `https://github.com/davidebasile/workspace`.

ACKNOWLEDGEMENTS

We are deeply indebted with Giancarlo Bigi for many discussions and suggestions on the usage of optimisation techniques, with Massimo Bartoletti for many insights on the logical aspects of our proposal, and with the anonymous referees for their valuable comments and remarks.

REFERENCES

[1] de Alfaro, L., Henzinger, T.A.: Interface automata. In: ESEC / SIGSOFT FSE. pp. 109–120. ACM (2001)

[2] van Bakel, S., Barbanera, F., de'Liguoro, U.: Orchestrated compliance for session-based client/server interactions. In: Proceedings 8th Interaction and Concurrency Experience. EPTCS, vol. 189, pp. 21–36 (2015)

[3] Barbanera, F., Dezani-Ciancaglini, M., Lanese, I., de'Liguoro, U.: Retractable contracts. In: Proceedings Eighth International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software. EPTCS, vol. 203 (2016)

[4] Bard, J.F.: Practical Bilevel Optimization: Algorithms and Applications (Nonconvex Optimization and Its Applications). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)

[5] Bartoletti, M., Cimoli, T., Pinna, G.M.: Lending Petri nets and contracts. In: Arbab, F., Sirjani, M. (eds.) FSEN. LNCS, vol. 8161, pp. 66–82. Springer (2013)

[6] Bartoletti, M., Cimoli, T., Pinna, G.M., Zunino, R.: Contracts as games on event structures. Journal of Logical and Algebraic Methods in Programming 85(3), 399–424 (2016), http://www.sciencedirect.com/science/article/pii/S235222081500036X

[7] Bartoletti, M., Cimoli, T., Pinna, G., Zunino, R.: Models of circular causality. In: Natarajan, R., Barua, G., Patra, M. (eds.) Distributed Computing and Internet Technology, Lecture Notes in Computer Science, vol. 8956, pp. 1–20. Springer International Publishing (2015), http://dx.doi.org/10.1007/978-3-319-14977-6_1

[8] Bartoletti, M., Cimoli, T., Zunino, R.: A theory of agreements and protection. In: Basin, D.A., Mitchell, J.C. (eds.) POST. LNCS, vol. 7796, pp. 186–205. Springer (2013)

[9] Bartoletti, M., Degano, P., Ferrari, G.L.: Planning and verifying service composition. Journal of Computer Security 17(5), 799–837 (2009)

[10] Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Call-by-contract for service discovery, orchestration and recovery. In: Wirsing, M., Hölzl, M.M. (eds.) Results of the SENSORIA Project, LNCS, vol. 6582, pp. 232–261. Springer (2011)

[11] Bartoletti, M., Degano, P., Di Giamberardino, P., Zunino, R.: Debits and credits in Petri Nets and Linear Logic. In: Logic, Rewriting, and Concurrency. LNCS, vol. 9200, pp. 135–159. Springer (2015)

[12] Bartoletti, M., Tuosto, E., Zunino, R.: Contract-oriented computing in co2. Sci. Ann. Comp. Sci. 22(1), 5–60 (2012)

[13] Bartoletti, M., Zunino, R.: A logic for contracts. In: Cherubini, A., Coppo, M., Persiano, G. (eds.) ICTCS. pp. 34–37 (2009)

[14] Bartoletti, M., Zunino, R.: A calculus of contracting processes. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom. pp. 332–341. IEEE Computer Society (2010), http://dx.doi.org/10.1109/LICS.2010.25

[15] Basile, D., Degano, P., Ferrari, G., Tuosto, E.: Playing with our cat and communication-centric applications. In: Prooceedings of the 36th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE). LNCS, vol. 9688. Springer (2016), to appear

[16] Basile, D., Degano, P., Ferrari, G., Tuosto, E.: Relating two automata-based models of orchestration and choreography. Journal of Logical and Algebraic Methods in Programming 85(3), 425–446 (2016), http://www.sciencedirect.com/science/article/pii/S2352220815000930

[17] Basile, D., Degano, P., Ferrari, G.L.: Secure and unfailing services. In: Malyshkin, V. (ed.) PaCT. LNCS, vol. 7979, pp. 167–181. Springer (2013)

[18] Basile, D., Degano, P., Ferrari, G.L.: Automata for analysing service contracts. In: Trustworthy Global Computing - 9th International Symposium, TGC 2014, Rome, Italy, September 5-6, 2014. Revised Selected Papers, Lecture Notes in Computer Science, vol. 8902, pp. 34–50. Springer (2014)

[19] Basile, D., Degano, P., Ferrari, G.L.: A formal framework for secure and complying services. The Journal of Supercomputing 69(1), 43–52 (2014)

[20] Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: From orchestration to choreography through contract automata. In: Lanese, I., Lluch-Lafuente, A., Sokolova, A., Vieira, H.T. (eds.) Proceedings 7th Interaction and

Concurrency Experience, ICE 2014, Berlin, Germany, 6th June 2014. EPTCS, vol. 166, pp. 67–85 (2014), http://dx.doi.org/10.4204/EPTCS.166.8

[21] Benton, P.N.: A mixed linear and non-linear logic: Proofs, terms and models. In: Computer Science Logic. pp. 121–135. Springer (1995)

[22] Bernardi, G., Hennessy, M.: Modelling session types using contracts. In: SAC'12. pp. 1941–1946 (2012)

[23] Bernardi, G., Hennessy, M.: Using higher-order contracts to model session types (extended abstract). In: Baldan, P., Gorla, D. (eds.) CONCUR 2014 - Concurrency Theory, Lecture Notes in Computer Science, vol. 8704, pp. 387–401. Springer Berlin Heidelberg (2014), http://dx.doi.org/10.1007/978-3-662-44584-6_27

[24] Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are two web services compatible? In: Shan, M.C., Dayal, U., Hsu, M. (eds.) Technologies for E-Services, Lecture Notes in Computer Science, vol. 3324, pp. 15–28. Springer Berlin Heidelberg (2005)

[25] Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)

[26] Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)

[27] Castagna, G., Gesbert, N., Padovani, L.: A theory of contracts for web services. ACM Trans. Program. Lang. Syst. 31(5) (2009)

[28] Castagna, G., Padovani, L.: Contracts for mobile processes. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR. LNCS, vol. 5710, pp. 211–228. Springer (2009)

[29] Clemente, L., Herbreteau, F., Sutre, G.: Decidable topologies for communicating automata with FIFO and bag channels. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8704, pp. 281–296. Springer (2014)

[30] Deniélou, P.M., Yoshida, N.: Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) ICALP (2). LNCS, vol. 7966, pp. 174–186. Springer (2013)

[31] Dezani-Ciancaglini, M., De'Liguoro, U.: Sessions and session types: An overview. In: Proceedings of the 6th International Conference on Web Services and Formal Methods. pp. 1–28. WS-FM'09, Springer-Verlag, Berlin, Heidelberg (2010), http://dl.acm.org/citation.cfm?id=1880906.1880907

[32] Gay, S., Hole, M.: Subtyping for session types in the pi calculus. Acta Informatica 42(2-3), 191–225 (2005), http://dx.doi.org/10.1007/s00236-005-0177-z

[33] Gray, J., Harrison, M.A., Ibarra, O.H.: Two-way pushdown automata. Information and Control 11(1/2), 30–70 (1967)

[34] Hemmecke, R., Koppe, M., Lee, J., Weismantel, R.: Nonlinear integer programming. In: Junger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.) 50 Years of Integer Programming 1958-2008, pp. 561–618. Springer Berlin Heidelberg (2010)

[35] Joshi, A.K., Shanker, K.V., Weir, D.: The convergence of mildly context-sensitive grammar formalisms (1990)

[36] Kuroda, S.Y.: Classes of languages and linear-bounded automata. Information and Control 7(2), 207–223 (1964)

[37] Laneve, C., Padovani, L.: An algebraic theory for web service contracts. Formal Aspects of Computing pp. 1–28 (2015)

[38] Lange, J., Tuosto, E.: Synthesising choreographies from local session types. In: Koutny, M., Ulidowski, I. (eds.) CONCUR. LNCS, vol. 7454, pp. 225–239. Springer (2012)

[39] Lange, J., Tuosto, E., Yoshida, N.: From communicating machines to graphical choreographies. In: Rajamani, S.K., Walker, D. (eds.) Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015. pp. 221–232. ACM (2015)

[40] Lynch, N.A., Tuttle, M.R.: An introduction to input/output automata. CWI Quarterly 2, 219–246 (1989)

[41] de Nicola, R., Hennessy, M.: Testing equivalences for processes. In: Diaz, J. (ed.) Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 154, pp. 548–560. Springer Berlin Heidelberg (1983)

[42] OASIS-Technical-Committee: OASIS WSBPEL TC, Web services business process execution language version 2.0 (2007), technical Report, OASIS, available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[43] Padovani, L.: Contract-based discovery of web services modulo simple orchestrators. Theor. Comput. Sci. 411(37), 3328–3347 (2010), http://dx.doi.org/10.1016/j.tcs.2010.05.002

[44] Papazoglou, M.P., Georgakopoulos, D.: Introduction: Service-oriented computing. Commun. ACM 46(10), 24–28 (Oct 2003), http://doi.acm.org/10.1145/944217.944233

[45] Peltz, C.: Web services orchestration and choreography. IEEE Computer 36(10), 46–52 (2003)

[46] Pfenning, F.: Structural cut elimination: Intuitionistic and classical logic. Information and Computation 157(1-2), 84 – 141 (2000), `"http://www.sciencedirect.com/science/article/pii/S0890540199928328"`

[47] Rensink, A., Vogler, W.: Fair testing. Information and Computation 205(2), 125 – 198 (2007), `http://www.sciencedirect.com/science/article/pii/S0890540106001106`

[48] Schneider, F.B.: Enforceable security policies. ACM Transactions on Information and System Security (TISSEC) 3(1), 30–50 (2000)

9. APPENDIX

9.1. **The Model.**

Proposition 9.1. *The following properties hold:*

– $\exists \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 . (\mathcal{A}_1 \otimes \mathcal{A}_2) \otimes \mathcal{A}_3 \neq \mathcal{A}_1 \otimes (\mathcal{A}_2 \otimes \mathcal{A}_3)$
– $\forall \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 . (\mathcal{A}_1 \boxtimes \mathcal{A}_2) \boxtimes \mathcal{A}_3 = \mathcal{A}_1 \boxtimes (\mathcal{A}_2 \boxtimes \mathcal{A}_3)$

Proof. Example 2.9 suffices to prove the first statement. For the second statement one has $\mathcal{A} = (\mathcal{A}_1 \boxtimes \mathcal{A}_2) \boxtimes \mathcal{A}_3 = \bigotimes_{\mathcal{A}_i \in I} \mathcal{A}_i = \mathcal{A}_1 \boxtimes (\mathcal{A}_2 \boxtimes \mathcal{A}_3)$ where $I = \{\Pi^i(\mathcal{A}) \mid i \in 1,2,3\}$. □

9.2. **Agreement.**

Proposition 9.2. *Let $\mathcal{K}$ be the mpc of the contract automaton $\mathcal{A}$, then $\mathscr{L}(\mathcal{K}) = \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$.*

Proof. The existence of $\mathcal{K}$ is guaranteed since all actions are controllable and observable and $\mathscr{L}(\mathcal{A})$ is regular, as well as $\mathfrak{A}$ [26]. By contradiction assume $\mathscr{L}(\mathcal{K}) \subset \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$, then there exists another controller $\mathcal{K}'_{\mathcal{A}}$ such that $\mathscr{L}(\mathcal{K}) \subset \mathscr{L}(\mathcal{K}') = \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$. □

Proposition 9.3 (Mpc)**.** *The controller $\mathcal{K}_A$ of Definition 3.9 is the most permissive controller of the contract automaton $\mathcal{A}$.*

Proof. In $\mathcal{K}_{\mathcal{A}}$ every request transition is removed in the first step, so it must be $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subseteq \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$. We will prove that $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) = \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$, from this follows that $\mathcal{K}_{\mathcal{A}}$ is the most permissive controller. By contradiction assume that exists a trace $w \in \mathfrak{A} \cap \mathscr{L}(\mathcal{A}), w \notin \mathscr{L}(\mathcal{K}_{\mathcal{A}})$. Then there exists a transition $t = (\vec{q}, \vec{a}, \vec{q'}) \notin T_{\mathcal{K}_{\mathcal{A}}}$ in the accepting path of w (i.e. the sequence of transitions used to recognise w). The transition t is not a request since $w \in \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$, and $\vec{q}, \vec{q'} \notin Hanged(\mathcal{K}_{\mathcal{A}})$ because the transition belongs to an accepting path. Since the only transitions removed to obtain $\mathcal{K}_{\mathcal{A}}$ are requests and those involving hanged states, it follows that $t \in T_{\mathcal{K}_{\mathcal{A}}}$. □

Theorem 3.18. *If two contract automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are*

(1) *competitive then they are collaborative,*
(2) *collaborative and safe, then they are competitive,*
(3) *safe then $\mathcal{A}_1 \otimes \mathcal{A}_2$ is safe, $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ admits agreement,*
(4) *non-collaborative, and one or both unsafe, then $\mathcal{A}_1 \otimes \mathcal{A}_2, \mathcal{A}_1 \boxtimes \mathcal{A}_2$ are unsafe,*
(5) *safe and non-competitive, then $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is safe.*

Proof. 1) Assume by contradiction that $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-collaborative, that is

$$(A_1^o \cap co(A_2^r)) \cup (co(A_1^r) \cap A_2^o) = \emptyset$$

Since the two automata are competitive, we have

$$A_1^o \cap A_2^o \cap (co(A_1^r) \cup co(A_2^r)) \neq \emptyset$$

By the distributive law

$$(A_1^o \cap (co(A_1^r) \cup co(A_2^r))) \cap (A_2^o \cap (co(A_1^r) \cup co(A_2^r))) \neq \emptyset$$

By hypothesis the two automata are non-collaborative, hence the above term can be rewritten as

$$(A_1^o \cap co(A_1^r)) \cap (co(A_2^r) \cap A_2^o) \neq \emptyset$$

By associative and commutative laws

$$(A_1^o \cap co(A_2^r)) \cap (co(A_1^r) \cap A_2^o) \neq \emptyset$$

Which implies

$$(A_1^o \cap co(A_2^r)) \cup (co(A_1^r) \cap A_2^o) \neq \emptyset$$

obtaining a contradiction.

2) By hypothesis the automata are collaborative:

$$(A_1^o \cap co(A_2^r)) \cup (A_2^o \cap co(A_1^r)) \neq \emptyset$$

By hypothesis $\mathcal{A}_1$ and $\mathcal{A}_2$ are safe, hence for each request there is a corresponding action, that is $co(A_i^r) \subseteq A_i^o$ where $i = 1,2$. Then the following holds

$$A_i^o \cap co(A_i^r) = co(A_i^r) \qquad i = 1,2$$

By substitution in the previous term we obtain

$$(A_1^o \cap A_2^o \cap co(A_2^r)) \cup (A_2^o \cap A_1^o \cap co(A_1^r)) \neq \emptyset$$

Which implies

$$(A_1^o \cap A_2^o \cap (co(A_1^r) \cup co(A_2^r))) \cup (A_2^o \cap A_1^o \cap (co(A_1^r) \cup co(A_2^r))) \neq \emptyset$$

By simplification we have

$$(A_1^o \cap A_2^o \cap (co(A_1^r) \cup co(A_2^r))) \neq \emptyset$$

Hence $\mathcal{A}_1$ and $\mathcal{A}_2$ are competitive.

3) Note that the labels of $\mathcal{A}_1 \otimes \mathcal{A}_2$ are the union of the labels of $\mathcal{A}_1$ and $\mathcal{A}_2$ (extended with idle actions for fitting the rank), hence no request transitions are added, and $\mathcal{A}_1 \otimes \mathcal{A}_2$ is *safe*. Since the traces of $\mathcal{A}_1 \otimes \mathcal{A}_2$ are a subset of $\mathcal{A} = \mathcal{A}_1 \boxtimes \mathcal{A}_2$, $\mathcal{A}$ has at least a trace in agreement. Example 3.16 shows that not all the traces of $\mathcal{A}$ admit agreement.

4) Without loss of generality assume that $\mathcal{A}_1$ is unsafe, hence there exists a request $\vec{a}$, and traces w, v such that $w\vec{a}v \in \mathscr{L}(\mathcal{A}_1)$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-collaborative there will be no match between the actions of $\mathcal{A}_1$ and $\mathcal{A}_2$, hence we have $w_1\vec{a}'v_1 \in \mathscr{L}(\mathcal{A}_1 \otimes \mathcal{A}_2), w_2\vec{a}'v_2 \in \mathscr{L}(\mathcal{A}_1 \boxtimes \mathcal{A}_2)$ for some w_1, w_2, v_1, v_2, where $\vec{a}'$ is obtained from $\vec{a}$ by adding the idle actions to principals from $r_{\mathcal{A}_1} + 1$ to $r_{\mathcal{A}_1} + r_{\mathcal{A}_2}$.

5) The proof is similar to that of item 3, indeed it suffices to prove that no new matches between principals in $\mathcal{A}_1$ and $\mathcal{A}_2$ are introduced in $\mathcal{A}_1 \boxtimes \mathcal{A}_2$. By item 2 it follows that $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-collaborative:

$$(A_1^o \cap co(A_2^r)) \cup (A_2^o \cap co(A_1^r)) \neq \emptyset$$

This suffices to prove that no matches will be introduced in their composition. □

9.3. Weak Agreement.

Theorem 4.6. *Let $\mathcal{A}_1, \mathcal{A}_2$ be two contract automata, then if $\mathcal{A}_1, \mathcal{A}_2$ are*

1. *weakly safe then $\mathcal{A}_1 \otimes \mathcal{A}_2$ is weakly safe, $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ admits weak agreement*
2. *non-collaborative and one or both unsafe, then $\mathcal{A}_1 \otimes \mathcal{A}_2, \mathcal{A}_1 \boxtimes \mathcal{A}_2$ are weakly unsafe*
3. *safe and non-competitive, then $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is weakly safe.*

Proof. Let req_a^w, of_a^w be the number of requests and offers of an action $a \in \mathbb{R} \cup \mathbb{O}$ in a trace w.

(1) For $\otimes$: we will prove that in every trace of $\mathcal{A}_1 \otimes \mathcal{A}_2$, for each action the number of requests are less than or equal to the number of offers, and the thesis follows. By contradiction, assume that there exists a trace w in $\mathcal{A}_1 \otimes \mathcal{A}_2$ and an action a with $req_a^w > of_a^w$. Assume that w is obtained combining two traces w_1, w_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$, that is each principal in each automaton performs the moves prescribed by its trace. Since both automata are weakly safe, we have $req_a^{w_1} \leq of_a^{w_1}$ and $req_a^{w_2} \leq of_a^{w_2}$ for all actions a.

Independently of how many matches occur, in w we still have more requests than offers: $req_a^{w1} + req_a^{w2} - k \leq of_a^{w1} + of_a^{w2} - k$ where k are the new matches.

For $\boxtimes$ it suffices to take a trace w in $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ obtained by combining two traces w_1, w_2 of respectively $\mathcal{A}_1$ and $\mathcal{A}_2$, where the match actions of both automata are maintained in w (the matches are performed by the same principals). In this case, the trace w will be present also in $\mathcal{A}_1 \otimes \mathcal{A}_2$, hence $w \in \mathfrak{W}$.

(2) Without loss of generality assume that $\mathcal{A}_1$ is weakly unsafe, hence there exists an action a and a trace w_1 in $\mathcal{A}_1$ such that $req_a^{w_1} > of_a^{w_1}$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are non-collaborative, in every trace w of $\mathcal{A}_1 \otimes \mathcal{A}_2$ or $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ obtained by shuffling w_1 with an arbitrary w_2 in $\mathcal{A}_2$ we will have $req_a^w > of_a^w$.

(3) from Theorem 3.18 item 5, $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is safe and since $\mathfrak{A} \subset \mathfrak{W}$ the thesis follows.

□

The following proposition helps the proof of Theorem 4.8.

Proposition 9.4. *Let* $WA(\mathfrak{W}) = \{w \in (\mathbb{R} \cup \mathbb{O} \cup \{\tau\})^* \mid \exists f : [1 \dots |w|] \to [1 \dots |w|]$ *injective and such that* $f(i) = j$ *only if* $w_{(i)} = co(w_{(j)})$, *total on the requests of* $w\}$.
Then, $Obs(w) \in WA(\mathfrak{W})$ *implies* $w \in \mathfrak{W}$.

Proof. Let $\sigma = Obs(w) \in WA(\mathfrak{W})$, and let f be a function that certifies that $\sigma \in WA(\mathfrak{W})$, i.e. that all the requests in w are fulfilled. Then f certifies $w \in \mathfrak{W}$. □

Theorem 4.8. $\mathfrak{W}$ *is a context-sensitive language, but not context-free. Word decision can be done in* $O(n^2)$ *time and* $O(n)$ *space.*

Proof. Example 4.7 shows that the property is not context-free. For proving that $\mathfrak{W}$ is context-sensitive we now outline a Linear Bounded Automata (LBA) [36] that decides whether a trace w belongs to $\mathfrak{W}$, giving us time and space complexity for the membership problem. Roughly, a LBA is a Turing machine with a tape, linearly bounded by the size of the input. Since we have an infinite alphabet due to the (unbounded) rank of vector $\vec{a}$, we compute $Obs(w)$ and decide if $Obs(w) \in WA(\mathfrak{W})$. By Proposition 9.4 we obtain the thesis. Below is the scheme of the algorithm:

```
for $i = 0; i < length(w); i++$ do
    if $w_i \in \mathbb{R}$ then
        for $j = 0; j < length(w); j++$ do
            if $w_j = co(w_i)$ then
                $w_j \leftarrow \#$
                break
            else
                if $j = length(w) - 1$ then return false
return true
```

The length of the tape equals the length of w, so the algorithm is $O(n)$ space, while it is $O(n^2)$ time, because of the two nested **for** cycles. □

The following is an auxiliary result to the theorems below.

Lemma 9.5. *Let $\mathcal{A}$ be a contract automaton such that $\vec{x} \in F_x$, then there exists a run $(w, \vec{q_0}) \to^* (\varepsilon, \vec{q_f})$ that passes through each $t_j \in T$ exactly x_{t_j} times.*

Proof. We outline an algorithm that visits all the transitions t_j with $x_{t_j} > 0$, starting from $\vec{q_f}$ and proceeding backwards to $\vec{q_0}$.

We use auxiliary variables $\overline{x}_{t_j}, t_j \in T$, initialised to zero, for storing how many times we have passed through a transition t_j. At each iteration the algorithm selects non deterministically a transition $\hat{t}$ in the backward star of the selected node such that $x_{\hat{t}} - \overline{x}_{\hat{t}} > 0$, and increases by one unit the variable $\overline{x}_{\hat{t}}$ for the selected $\hat{t}$. The next node will be the starting state of $\hat{t}$. The algorithm terminates when for all the transitions t_j in the backward star we have $x_{t_j} - \overline{x}_{t_j} = 0$.

We prove that the algorithm terminates and constructs a trace that passes through each t_j exactly x_{t_j} times, and the last transition considered leaves the initial state. For the first step we have $\sum_{t_j \in BS(\vec{q_f})} x_{t_j} - \sum_{t_j \in FS(\vec{q_f})} x_{t_j} = 1$ hence there exists at least one $t_i \in BS(\vec{q_f})$ such that $x_{t_i} > 0$ (and $\overline{x}_{t_i} = 0$).

Pick up one of these transitions, say t_i, and assign it to the iteration variable $\hat{t}$. Two cases may arise, depending on the source of $\hat{t}$:

(1) the source of $\hat{t}$ is $\vec{q} \neq \vec{q_0}$: we have $\sum_{t_j \in BS(\vec{q})} x_{t_j} - \sum_{t_j \in FS(\vec{q})} x_{t_j} \geq 0$ and we know that $\sum_{t_j \in FS(\vec{q})} x_{t_j} > 0$, because $\hat{t} \in FS(\vec{q})$ and $x_{\hat{t}} > 0$, hence $\sum_{t_j \in BS(\vec{q})} x_{t_j} > 0$.

   We now show that there is at least one $t \in BS(\vec{q})$ such that $(x_t - \overline{x}_t) > 0$. By contradiction, assume $\sum_{t_j \in BS(\vec{q})} x_{t_j} - \sum_{t_j \in BS(\vec{q})} \overline{x}_{t_j} = 0$. We distinguish two cases:
   - $\vec{q} = \vec{q_f}$: we have $\sum_{t_j \in FS(\vec{q})} \overline{x}_{t_j} = \sum_{t_j \in BS(\vec{q})} \overline{x}_{t_j}$, since at every iteration we increase of one unit the value of $\overline{x}_{t_i}$ for $\hat{t}$ and we are proceeding backwards starting from $\vec{q_f}$ (the flow variable of a loop belongs to both backward and forward star). Since $\sum_{t_j \in BS(\vec{q})} x_{t_j} > \sum_{t_j \in FS(\vec{q})} x_{t_j}$, we have $\sum_{t_j \in FS(\vec{q})} x_{t_j} - \sum_{t_j \in FS(\vec{q})} \overline{x}_{t_j} < 0$. Contradiction, since by definition the value $\overline{x}_{t_j}$ for a transition t_j will never be greater then the corresponding value x_{t_j}.
   - $\vec{q} \neq \vec{q_f}$: we have $\sum_{t_j \in FS(\vec{q})} \overline{x}_{t_j} > \sum_{t_j \in BS(\vec{q})} \overline{x}_{t_j}$. Since $\sum_{t_j \in BS(\vec{q})} x_{t_j} = \sum_{t_j \in FS(\vec{q})} x_{t_j}$, we have $\sum_{t_j \in FS(\vec{q})} x_{t_j} - \sum_{t_j \in FS(\vec{q})} \overline{x}_{t_j} < 0$ obtaining a contradiction as above.

   Then, we iterate the algorithm taking the above t as $\hat{t}$.

(2) the source of t_i is $\vec{q_0}$: we have $\sum_{t_j \in BS(\vec{q_0})} x_{t_j} - \sum_{t_j \in FS(\vec{q_0})} x_{t_j} = -1$.
   Let $k_1 = \sum_{t_j \in FS(\vec{q_0})} x_{t_j} - \sum_{t_j \in FS(\vec{q_0})} \overline{x}_{t_j}$, $k_2 = \sum_{t_j \in BS(\vec{q_0})} x_{t_j} - \sum_{t_j \in BS(\vec{q_0})} \overline{x}_{t_j}$, and note that since we are proceeding backwards starting from $\vec{q_f}$ it must be that $\sum_{t_j \in FS(\vec{q_0})} \overline{x}_{t_j} = 1 + \sum_{t_j \in BS(\vec{q_0})} \overline{x}_{t_j}$. Hence, from the previous equations it must be that $k_2 - k_1 = 0$. We have that:
   - if $k_1 = 0$, we have $k_2 = 0$ and the algorithm terminates;
   - if $k_1 > 0$, we have $k_2 > 0$ and the algorithm continues by selecting a transition $\hat{t} \in BS(\vec{q_0})$ such that $x_{\hat{t}} - \overline{x}_{\hat{t}} = 0$.

Since at every iteration we increase the value $\overline{x}_{\hat{t}}$, the constraints on F_x guarantee that the algorithm will eventually terminate. Moreover there exists an execution of the algorithm that traverses all the possible cycles of the trace induced by $\vec{x}$. Hence we have a trace from $\vec{q_0}$ to $\vec{q_f}$ that passes through each transition t_j visited by the algorithm exactly x_{t_j} times.

It remains to prove that for all the transitions t_j not visited by the algorithm we have $x_{t_j} = 0$. By contradiction assume that there exists a transition $t_i = (\vec{q_s}, \vec{a}, \vec{q_d})$ with $x_{t_i} - \overline{x}_{t_i} > 0$ for all the possible executions of the algorithm.

This is possible only if $\vec{q_d}$ it is not connected to $\vec{q_f}$ by the flow $\vec{x}$. Moreover in this case by the flow constraints on $\vec{x}$ it follows that $\vec{q_s}$ is not reachable from $\vec{q_0}$ by the flow $\vec{x}$, i.e. t_i is not part of the trace induced by $\vec{x}$. Then there must exist a cycle $C = \{t_{c1}, \dots, t_{cm}\}$ with $t_i \in C$ and disconnected from $\vec{q}_0$ and $\vec{q}_f$ with positive flow. Let Q_C be the set of nodes having ingoing or outgoing transitions in C. The constraints $\sum_{t \in BS(\vec{q})} x_t - \sum_{t \in FS(\vec{q})} x_t = 0$ are satisfied for all $\vec{q} \in C$.

We show that C will eventually violate the constraints defined by the variables $z_{t_j}^{\vec{q}_s}$. We have:

$$\forall \vec{q'} \in Q: \sum_{t_j \in BS(\vec{q'})} z_{t_j}^{\vec{q}_s} - \sum_{t_j \in FS(\vec{q'})} z_{t_j}^{\vec{q}_s} = \begin{cases} -p^{\vec{q}_s} & \text{if } \vec{q'} = \vec{q}_0 \\ 0 & \text{if } \vec{q'} \neq \vec{q}_0, \vec{q}_s \\ p^{\vec{q}_s} & \text{if } \vec{q'} = \vec{q}_s \end{cases}$$

$$\forall t_j \in T.\, z_{t_j}^{\vec{q}_s} \in \mathbb{R}, \quad 0 \leq z_{t_j}^{\vec{q}_s} \leq x_{t_j}$$

We have $\sum_{t_j \in FS(\vec{q}_s)} x_{t_j} > 0$ and $p^{\vec{q}_s} = 1$, hence $\sum_{t_j \in BS(\vec{q}_s)} z_{t_j}^{\vec{q}_s} - \sum_{t_j \in FS(\vec{q}_s)} z_{t_j}^{\vec{q}_s} = 1$ and for all $\vec{q} \in Q_C, \vec{q} \neq \vec{q}_s : \sum_{t_j \in BS(\vec{q})} z_{t_j}^{\vec{q}_s} - \sum_{t_j \in FS(\vec{q})} z_{t_j}^{\vec{q}_s} = 0$. Note that is not possible to satisfy these constraints since for all $t \in C$, x_t are all equal and positive and $0 \leq z_t^{\vec{q}_s} \leq x_t$. □

Theorem 4.12. *Let $\vec{v}$ be a binary vector. Then a contract automaton $\mathcal{A}$ is* weakly safe *if and only if min $\gamma \geq 0$ where:*

$$\sum_{i \in I_l} v_i \sum_{t_j \in T} a_{t_j}^i x_{t_j} \leq \gamma \quad \sum_{i \in I_l} v_i = 1 \quad \forall i \in I_l.\, v_i \in \{0,1\} \quad (x_{t_1} \dots x_{t_n}) \in F_x \quad \gamma \in \mathbb{R}$$

Proof. ($\Rightarrow$) By contradiction assume that min $\gamma < 0$. Hence there exists an action a^j such that $v_j = 1, \forall i \in I_l, i \neq j. v_i = 0$ and $\gamma = \sum_{t_j \in T} a_{t_j}^j x_{t_j} < 0$. By Lemma 9.5 we know that $\vec{x}$ builds a trace recognising $w \in \mathscr{L}(\mathcal{A})$, and the number of offers for a^j in w are less than the corresponding number of requests since $\sum_{t_j \in T} a_{t_j}^j x_{t_j} < 0$, hence $w \notin \mathfrak{W}$.

($\Leftarrow$) By contradiction there exists $w \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$. Hence there exists an action a^j that occurs in w fewer times as an offer than as a request. Let $\vec{x}$ be the flow induced in the obvious way by the trace w, counting the number of times each transition occurs in the path accepting w. We have $\sum_{t_j \in T} a_{t_j}^j x_{t_j} < 0$, hence it must be min $\gamma < 0$. □

Theorem 4.14. *The contract automaton $\mathcal{A}$ admits weak agreement if and only if max $\gamma \geq 0$ where*

$$\forall i \in I_l.\, \sum_{t_j \in T} a_{t_j}^i\, x_{t_j} \geq \gamma \quad (x_{t_1} \dots x_{t_n}) \in F_x \quad \gamma \in \mathbb{R}$$

Proof. ($\Rightarrow$) Let w be a trace in weak agreement, and let $\vec{x}$ be the flow induced by w. Then by construction $\forall i \in I_l. \sum_{t_j \in T} a_{t_j}^i x_{t_j} \geq 0$, hence max $\gamma \geq 0$.

($\Leftarrow$) Follows from Lemma 9.5 and the hypothesis. □

Theorem 4.17. *The principal $\Pi^i(\mathcal{A})$ of a contract automaton $\mathcal{A}$ is* weakly liable *if and only if there exists a transition $\bar{t} = (\vec{q}_s, \vec{a}, \vec{q}_d) \in T_{\mathcal{A}}$, $\vec{a}_{(i)} \neq \square$ such that $\gamma_{\bar{t}} < 0$, where*

$$\gamma_{\bar{t}} = \mathit{min}\ \{g(\vec{x}) \mid \vec{x} \in F_{\vec{q}_0, \vec{q}_s},\ \vec{y} \in F_{\vec{q}_s, \vec{q}_f},\ \forall i \in I_l.\, \sum_{t_j \in T} a_{t_j}^i (x_{t_j} + y_{t_j}) \geq 0\}$$

$$g(\vec{x}) = \mathit{max}\ \{\gamma \mid \vec{u} \in F_{\vec{q}_d, \vec{q}_f},\ \forall i \in I_l.\, \sum_{t_j \in T} a_{t_j}^i (x_{t_j} + u_{t_j}) + a_{\bar{t}}^i \geq \gamma, \gamma \in \mathbb{R}\}$$

Proof. ($\Rightarrow$) By hypothesis $\exists w_1$ such that $\forall w_3. w_1 \vec{a} w_3 \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$ and $\exists w_2. w_1 w_2 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$. Let $\bar{t} = (\vec{q}_s, \vec{a}, \vec{q}_d)$ be the transition such that $(w_1 \vec{a}, \vec{q}_0) \rightarrow^* (\vec{a}, \vec{q}_s) \rightarrow (\varepsilon, \vec{q}_d)$, i.e. the principal i in $\vec{a}$ is weakly liable. We show that $\gamma_{\bar{t}} < 0$.

Let w_1 from $\vec{q}_0$ to $\vec{q}_s$ induce the flow $\vec{x}$, while w_2 from $\vec{q}_s$ to $\vec{q}_f$ induce $\vec{y}$. Since $w_1 w_2$ is in weak agreement, $\forall i \in I_l. \sum_{t_j \in T} a_{t_j}^i (x_{t_j} + y_{t_j}) \geq 0$.

Since by hypothesis the i-th principal is liable, the flow $\vec{x}$ corresponding to the trace w_1 is such that $g(\vec{x}) < 0$. Otherwise if $g(\vec{x}) \geq 0$ we can choose a trace, say, w_3 such that $w_1\vec{a}w_3 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$, obtaining a contradiction. Therefore, $\gamma_{\bar{t}} \leq g(\vec{x}) < 0$.

$(\Leftarrow)$ by hypothesis $\gamma_{\bar{t}} < 0$ and by Lemma 9.5 $\vec{x}$ corresponds to a run w from the initial state to $\vec{q}_s$ such that (by hypothesis again) $\forall w_3.w_1\vec{a}w_3 \notin \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$ and $\exists w_4.w_1w_4 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$, that is $\bar{t}$ is a weakly liable transition. □

9.4. **Automata and Horn Propositional Contract Logic.** For completeness, we first define the grammar for the full PCL, while the rules for its sequent calculus are in Figure 13. Unless stated differently, in what follows we only consider proofs without the rules $(weakR)$ and (cut), which are proved to be redundant in [13].

Definition 9.6 (PCL). The formulae of PCL are inductively defined by the following grammar.

$$
\begin{array}{lll}
p & ::= & \bot \qquad\qquad\qquad\qquad \text{false} \\
 & & \top \qquad\qquad\qquad\qquad \text{true} \\
 & & a \qquad\qquad\qquad\qquad \text{prime} \\
 & & \neg p \qquad\qquad\qquad\quad\;\; \text{negation} \\
 & & p \vee p \qquad\qquad\qquad\; \text{disjunction} \\
 & & p \wedge p \qquad\qquad\qquad\; \text{conjunction} \\
 & & p \rightarrow p \qquad\qquad\qquad \text{implication} \\
 & & p \twoheadrightarrow p \qquad\qquad\qquad \text{contractual implication}
\end{array}
$$

The following proposition will be helpful later on.

Proposition 9.7. *Given a H-PCL formula p and the automaton* $[\![p]\!] = \langle Q, q_0, A^r, A^o, T, F\rangle$*:*

(1) $F = \{\vec{q} = \langle\{*\}, \dots, \{*\}\rangle\}$*, and all* $(\vec{q}, \vec{a}, \vec{q'})$ *are such that* $\vec{q'} = \vec{q}$ *and* $\vec{a}$ *is an offer;*
(2) *every state* $\vec{q} = \langle J_1, \dots, J_n\rangle$ *has as many request or match outgoing transitions as the request actions prescribed by* $\bigcup_{i \in 1\dots n} J_i$*;*
(3) $[\![p]\!]$ *is deterministic.*

Proof. The first item follows immediately from Definition 5.3.
For the second item, we first consider the translation of the clauses in the formula. By construction, for each of them two cases are possible when considering request actions: either $[\![(\bigwedge_{j \in J_i} a_j) \rightarrow b]\!]$ or $[\![(\bigwedge_{j \in J_i} a_j) \twoheadrightarrow b]\!]$. In both cases we have outgoing request transitions of the form $\{(J' \cup \{j\}, a_j, J') \mid J' \cup \{j\} \in 2^{J_i}, j \in J\}$. Finally by applying the associative composition $\boxtimes$ (Definition 2.8), some requests may be matched with corresponding offers, but no new request can be originated.
The third item follows immediately by the translation and by the condition in Definition 5.1, that all the atoms are different. □

The following lemma shows that if an atom a is entailed by a formula p then there is a trace recognised by the contract automaton $[\![p]\!]$ where the request corresponding to the atom a, if any, is always matched.

Lemma 9.8. *Given a H-PCL formula p and an atom a in p we have:*

$$p \vdash a \text{ is provable implies } \exists w \in \mathscr{L}([\![p]\!]) \text{ such that no } \vec{a} \text{ request on } a \text{ occurs in } w$$

$$\dfrac{}{\Gamma, p \vdash p}\,\mathit{id} \qquad \dfrac{\Gamma, p \wedge q, p \vdash r}{\Gamma, p \wedge q \vdash r}\,\wedge L1 \qquad \dfrac{\Gamma, p \wedge q, q \vdash r}{\Gamma, p \wedge q \vdash r}\,\wedge L2 \qquad \dfrac{\Gamma \vdash p \quad \Gamma \vdash q}{\Gamma \vdash p \wedge q}\,\wedge R$$

$$\dfrac{\Gamma, p \vee q, p \vdash r \quad \Gamma, p \vee q, q \vdash r}{\Gamma, p \vee q \vdash r}\,\vee L \qquad \dfrac{\Gamma \vdash p}{\Gamma \vdash p \vee q}\,\vee R1 \qquad \dfrac{\Gamma \vdash q}{\Gamma \vdash p \vee q}\,\vee R2$$

$$\dfrac{\Gamma \vdash p \quad \Gamma, p \vdash q}{\Gamma \vdash q}\;\mathit{cut} \qquad \dfrac{\Gamma, p \rightarrow q \vdash p \quad \Gamma, p \rightarrow q, q \vdash r}{\Gamma, p \rightarrow q \vdash r}\,\rightarrow L \qquad \dfrac{\Gamma, p \vdash q}{\Gamma \vdash p \rightarrow q}\,\rightarrow R$$

$$\dfrac{\Gamma, \neg p \vdash p}{\Gamma, \neg p \vdash r}\,\neg L \qquad \dfrac{\Gamma, p \vdash \bot}{\Gamma \vdash \neg p}\,\neg R \qquad \dfrac{}{\Gamma, \bot \vdash p}\,\bot L$$

$$\dfrac{}{\Gamma \vdash \top}\,\top R \qquad \dfrac{\Gamma \vdash \bot}{\Gamma \vdash p}\;\mathit{weakR} \qquad \dfrac{\Gamma \vdash q}{\Gamma \vdash p \twoheadrightarrow q}\;\mathit{Zero}$$

$$\dfrac{\Gamma, p \twoheadrightarrow q, r \vdash p \quad \Gamma, p \twoheadrightarrow q, q \vdash r}{\Gamma, p \twoheadrightarrow q \vdash r}\;\mathit{Fix} \qquad \dfrac{\Gamma, p \twoheadrightarrow q, a \vdash p \quad \Gamma, p \twoheadrightarrow q, q \vdash b}{\Gamma, p \twoheadrightarrow q \vdash a \twoheadrightarrow b}\;\mathit{PrePost}$$

FIGURE 13. The rules of the sequent calculus for PCL. The contractual implication rules are *Zero*,*Fix* and *Prepost* while the others are the standards for Intuitionistic logic.

Proof. Consider each of the conjuncts α of p. If a does not appear in α as the premise of an implication/contractual implication, then the statement follows trivially by Definition 5.3 and by hypothesis, since the translation of a is an offer action. Otherwise a also occurs in α within:

1. a conjunction, or
2. the conclusion of a contractual implication, or
3. the conclusion of an implication.

For the first two cases, by Definition 5.3, a transition labelled by the relevant offer $\overline{a}$ is available in all states, so preventing a request a to appear in $[\![p]\!]$, i.e. after the product of the principals (Definition 2.8).

For proving case 3, $\alpha = \bigwedge_{j \in J} a_j \rightarrow a$ and we proceed by induction on the depth of the proof of $p \vdash a$. It must be the case then that $\forall j$ it holds $p \vdash a_j$. We can now either re-use the proof for cases 1 and 2 (that act as base cases), or the induction hypothesis if a_j occurs in the conclusion of an implication. By Definition 5.3 after all a_j are matched, the offer a will be always available, preventing a request a to appear. □

In order to keep the following definition compact, we use $\circ$ for either $\rightarrow$ or $\twoheadrightarrow$. In addition, by abuse of the notation we also use $\wedge$ to operate between formulas, we write p' for an empty formula or with a single clause, and we allow the indexing sets J and K in clauses to be empty. Finally, we let $(\bigwedge_{j \in \emptyset} a_j) \circ b$ stand for b.

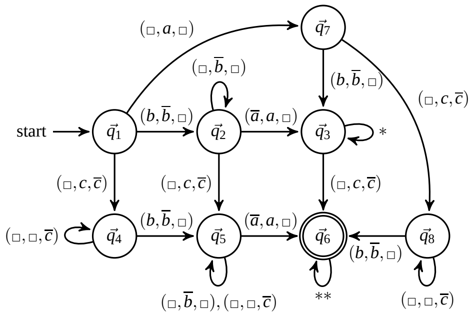


FIGURE 14. The contract automaton $[\![Alice \wedge Bob \wedge Charlie]\!]$discussed in Example 9.11 is displayed here, where the principals are those of Figure 9, and $* = (\overline{a},\Box,\Box),(\Box,\overline{b},\Box)$, $** = (\overline{a},\Box,\Box),(\Box,\overline{b},\Box),(\Box,\Box,\overline{c})$.

Definition 9.9. Given a formula p, if from the initial state of $[\![p]\!]$ there is an outgoing offer or an outgoing match transition with label $\vec{a}$, we define

$$p/\vec{a} = \begin{cases} p & \text{if } \vec{a} \text{ is an offer} \\ p' \wedge (\bigwedge_{z \in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j \in J} a_j) \circ b' & \text{if } \vec{a} \text{ is a match with } \vec{a}_{(i)} = b \text{ and} \\ & p = p' \wedge (\bigwedge_{z \in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j \in J} a_j \wedge b) \circ b' \\ p' \wedge (\bigwedge_{k \in K} a_k \wedge b) \wedge (\bigwedge_{j \in J} a_j) \circ b' & \text{if } \vec{a} \text{ is a match with } \vec{a}_{(i)} = b \text{ and} \\ & p' \wedge (\bigwedge_{k \in K} a_k \wedge b) \wedge (\bigwedge_{j \in J} a_j \wedge b) \circ b' \end{cases}$$

We now establish a relation between $[\![p/\vec{a}]\!]$, and the contract automaton obtained by changing the initial state $\vec{q}_0$ of $[\![p]\!]$ to $\vec{q}$, for the transition $(\vec{q}_0,\vec{a},\vec{q})$ of $[\![p]\!]$. The main idea is to relate the formula $p/\vec{a}$ to the residual of the automaton $[\![p]\!]$ after the execution of an initial transition labelled by $\vec{a}$, that is $[\![p/\vec{a}]\!]$. Recall that the translation given in Definition 5.3 yields deterministic automata.

Lemma 9.10. *Given a H-PCL formula p and the contract automaton $[\![p]\!] = \langle Q,\vec{q}_0,A^r,A^o,T,F\rangle$, if $t = (\vec{q}_0,\vec{a},\vec{q}) \in T$ is an offer or a match transition, then $\mathscr{L}(\mathcal{A}) = \mathscr{L}([\![p/\vec{a}]\!])$ where $\mathcal{A} = \langle Q,\vec{q},A^r,A^o,T,F\rangle$.*

Proof. The proof is by cases of $\vec{a}$. If $\vec{a}$ is an offer, then by Definition 5.3 it must be $\vec{q} = \vec{q}_0$ and trivially $\mathcal{A} = [\![p/\vec{a}]\!]$.

Otherwise, since $\vec{a}$ is a match action, say on atom b, it contains a request from, say, the i-th principal and a corresponding offer from another. Therefore, $p = \bigwedge_{k \in K} \alpha_k$ contains within a clause α_j the atom b, originating the offer, as a conjunction or as a conclusion of a contractual implication (note that it cannot be an implication because we are in the initial state), and α_i also contains b originating this time the request. We now prove that the automata $\mathcal{A}$ and $[\![p/\vec{a}]\!]$ have the same initial state. Let $\vec{q}_0 = \langle J_1,\dots,J_n\rangle$, then, since $\vec{a}_{(i)} = b$, the states $\vec{q}_0$ and $\vec{q}$ only differ in the i-th element, where in $\vec{a}_{(i)}$ the request action b is not available anymore; formally, $\forall j \neq i$ it must be $\vec{q}_{(j)} = \vec{q}_{0(j)} = J_j$, and $\vec{q}_{(i)} = \vec{q}_{0(i)} \setminus \{i\}$. By Definition 9.9 p and $p/\vec{a}$ differ because of the single atom b has been removed from α_i. By these facts and by item 2 of Proposition 5.5 the language equivalence follows. Indeed, $[\![p/\vec{a}]\!]$ is the product of the same $[\![\alpha_k]\!], k \neq i$ used for $[\![p]\!]$, and the match on b of $\mathcal{A}$ leaves $\vec{q}_0$, that is not reachable from $\vec{q}$. □

Example 9.11. Let $[\![p]\!]$ be the automaton shown in Figure 14, where $p = Alice \wedge Bob \wedge Charlie$ and the principals are those of Figure 9. Consider now $p' = p/(b,\overline{b},\Box) = (a \wedge ((a \wedge c) \twoheadrightarrow b) \wedge c)$ and build $[\![p']\!] = \{\langle\{\vec{q}_2,\vec{q}_3,\vec{q}_5,\vec{q}_6\},\vec{q}_2,A^r,A^o,T,\vec{q}_6\rangle\}$ (transitions, alphabets and states are taken from

$[\![p]\!]$). It is immediate to verify that the language of $[\![p']\!]$ is the same of $[\![p]\!]$, when the initial state is $\vec{q}_2$ instead of $\vec{q}_1$.

The following lemma is auxiliary for proving the next theorem. Its second item is similar to Lemma 1 in [46].

Lemma 9.12. *Let a, b be atoms, p, q be conjunction of atoms, with q possibly empty, $p_1, \dots, p_n$ be formulae, and $\circ \in \{\rightarrow, \twoheadrightarrow\}$, then*

$$\begin{array}{rlll}
\text{(i)} & \textit{if} \quad \dfrac{\Delta}{\Gamma, q \circ b \vdash p} & \textit{then} & \exists \Delta' : \dfrac{\Delta'}{\Gamma, (q \wedge a) \circ b, a \vdash p} \\
\text{(ii)} & \textit{if} \quad \Gamma \vdash p & \textit{then} & \forall \Gamma'.\, \Gamma, \Gamma' \vdash p \\
\text{(iii)} & \textit{if} \quad \bigwedge_{i \in 1 \dots n} p_i \vdash q & \textit{then} & p_1, \dots, p_n \vdash q \\
\text{(iv)} & \textit{if} \quad \Gamma \vdash \bigwedge_{i \in 1 \dots n} p_i & \textit{then} & \forall i.\, \Gamma \vdash p_i
\end{array}$$

Proof. To prove the first item, we proceed by induction on the depth of Δ and by case analysis on the last rule applied. In the base case Δ is empty, we have two cases

(1) q non-empty or $p \neq b$: then it must be that $\Gamma = p, \Gamma'$ for some Γ' and the last rule applied is id. Trivially, Δ' will be empty and we have

$$\dfrac{}{\Gamma', p, (q \wedge a) \circ b, a \vdash p}\,id$$

(2) q empty and $p = b$: our hypothesis reads as $\dfrac{}{\Gamma, b \vdash b}\,id$, and we build the following deduction

$$\dfrac{\dfrac{}{\Gamma', a \circ b, a \vdash a}\,id \qquad \dfrac{}{\Gamma, a \circ b, a, b \vdash b}\,id}{\Gamma, a \circ b, a \vdash b}\,\Diamond$$

where if $\circ = \rightarrow$ then $\Gamma' = \Gamma$ and $\Diamond = \rightarrow L$, otherwise if $\circ = \twoheadrightarrow$ then $\Gamma' = \Gamma, b$ and $\Diamond = Fix$.

For the inductive step, we distinguish two cases:

(1) the last rule applied to deduce the hypothesis does not involve $q \circ b$. Hence the rule must be applied on p or on a formula in Γ. We can apply the same rule to $\Gamma, (q \wedge a) \circ b, a \vdash p$ and use the inductive hypothesis.

(2) the last rule applied to deduce the hypothesis involves $q \circ b$. There are two exhaustive cases

(a) $\circ = \rightarrow$, then the last rule applied is $\rightarrow L$ and the deduction tree has the following form:

$$\dfrac{\dfrac{\Delta_1}{\Gamma, q \rightarrow b \vdash q} \qquad \dfrac{\Delta_2}{\Gamma, q \rightarrow b, b \vdash p}}{\Gamma, q \rightarrow b \vdash p}\,\rightarrow L$$

Then by induction hypothesis we have

$$\dfrac{\Delta'_1}{\Gamma, (q \wedge a) \rightarrow b, a \vdash q} \qquad\qquad \dfrac{\Delta'_2}{\Gamma, (q \wedge a) \rightarrow b, a, b \vdash p}$$

From the right one and a derivation tree Δ_3 detailed below, we build

$$\dfrac{\Delta_3 \qquad \dfrac{\Delta'_2}{\Gamma, (q \wedge a) \rightarrow b, a, b \vdash p}}{\Gamma, (q \wedge a) \rightarrow b, a \vdash p}\,\rightarrow L$$

Δ_3 is the derivation tree:

$$\dfrac{\dfrac{\Delta'_1}{\Gamma, (q \wedge a) \rightarrow b, a \vdash q} \qquad \dfrac{}{\Gamma, (q \wedge a) \rightarrow b, a \vdash a}\,id}{\Gamma, (q \wedge a) \rightarrow b, a \vdash q \wedge a}\,\wedge R$$

(b) $\circ = \twoheadrightarrow$, then the last rule applied is Fix and the deduction tree has the following form:

$$\dfrac{\dfrac{\Delta_1}{\Gamma, q \twoheadrightarrow b, p \vdash q} \qquad \dfrac{\Delta_2}{\Gamma, q \twoheadrightarrow b, b \vdash p}}{\Gamma, q \twoheadrightarrow b \vdash p}\,Fix$$

Then by the induction hypothesis we have

$$\dfrac{\Delta_1'}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, p \vdash q} \qquad\qquad \dfrac{\Delta_2'}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, b \vdash p}$$

From the above, we build the following

$$\dfrac{\dfrac{\dfrac{\Delta_1'}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, p \vdash q} \quad \dfrac{}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, p \vdash a}\,id}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, p \vdash q \wedge a}\,\wedge R \qquad \dfrac{\Delta_2'}{\Gamma, (q \wedge a) \twoheadrightarrow b, a, b \vdash p}}{\Gamma, (q \wedge a) \twoheadrightarrow b, a \vdash p}\,Fix$$

For the second item, we prove a stronger fact: the last rule used to deduce $\Gamma, \Gamma' \vdash p$ is the same used for proving $\Gamma \vdash p$. We proceed by induction on the depth of the derivation for $\Gamma \vdash p$ and then by case analysis on the last rule applied.

The base case is when the axiom id is applied, and the proof is immediate.

For the inductive case, we assume that for some rule $\diamondsuit$

$$\dfrac{\Delta}{\Gamma \vdash p}\,\diamondsuit \qquad \text{implies} \qquad \dfrac{\overline{\Delta}}{\Gamma, \Gamma' \vdash p}\,\diamondsuit$$

Rather than considering each rule at a time, we group them in two classes: those with two premises, and those with one premise. Below, we discuss the first case, and the second follows simply erasing one premise in what follows. The deduction tree in the premise above has the following form

$$\dfrac{\dfrac{\Delta'}{\overline{\Gamma} \vdash q} \qquad \dfrac{\Delta''}{\overline{\Gamma'} \vdash q'}}{\Gamma \vdash p}\,\diamondsuit$$

and by applying the induction hypothesis to both the premises we conclude

$$\dfrac{\dfrac{\overline{\Delta'}}{\overline{\Gamma}, \Gamma' \vdash q} \qquad \dfrac{\overline{\Delta''}}{\overline{\Gamma'}, \Gamma' \vdash q'}}{\Gamma, \Gamma' \vdash p}\,\diamondsuit$$

Moreover note that in this fragment no contradictions can be introduced.

For the third item, we have a derivation tree Δ for the sequent $\bigwedge_{i \in 1 \ldots n} p_i \vdash q$. To build a derivation tree Δ' for $p_1, \ldots, p_n \vdash q$ apply the following two steps. The first step removes from Δ all the rules $\wedge L_i$ applied to (each sub-term of) $\bigwedge_{i \in 1 \ldots n} p_i$, obtaining Δ''. Then, replace all applications of the axiom (id) in Δ'' of the form

$$\dfrac{}{\Gamma, \bigwedge_{j \in J} p_j \vdash \bigwedge_{j \in J} p_j}\,id$$

with a derivation tree with $k = |J|$ leaves of the form

$$\dfrac{}{\Gamma, p_1, p_2, \ldots, p_k \vdash p_j}\,id$$

and by repeatedly applying the rule $(\wedge R)$ until we obtain the relevant judgement

$\Gamma, p_1, p_2, \ldots, p_k \vdash \bigwedge_{i \in 1 \ldots n} p_i$.

For the fourth item, we have a derivation tree Δ for the sequent $\Gamma \vdash \bigwedge_{i \in \{1 \ldots n\}} p_i$.

For each sequent $\Gamma \vdash p_j$, $j \in \{1 \dots n\}$, the derivation tree is then:

$$\dfrac{\dfrac{\Delta}{\Gamma \vdash p_j \wedge \bigwedge_{i\in\{1\dots n\}\setminus\{j\}} p_i} \qquad \dfrac{\dfrac{}{p_j, \bigwedge_{i\in\{1\dots n\}\setminus\{j\}} p_i \vdash p_j}\,id}{p_j \wedge \bigwedge_{i\in\{1\dots n\}\setminus\{j\}} p_i \vdash p_j}\,\wedge L1}{\Gamma \vdash p_j}\,cut$$

□

Theorem 5.6. *Given a H-PCL formula p we have $p \vdash \lambda(p)$ if and only if $[\![p]\!]$ admits agreement.*

Proof. ($\Rightarrow$) Since $p \vdash \lambda(p)$ by Lemma 9.12(iv) (where $\Gamma = p$) we have $p \vdash a$ for all atoms a in p. It suffices to apply Lemma 9.8 to each of these atoms, and by Definition 5.3 the offers are never consumed, there must be a trace $w \in \mathscr{L}([\![p]\!])$ where all the requests are matched.

($\Leftarrow$) Let $\vec{q}_0$ be the initial state of $[\![p]\!]$ and $\vec{f}$ be the final state. We proceed by induction on the length of w.

In the base case w is empty, hence the initial state of $[\![p]\!]$ is also final. This situation only arises when the second rule of Definition 5.3 has been applied for all conjuncts α_i corresponding to principals. Therefore it must be that p is a conjunction of atoms, so $p = \lambda(p)$ and the thesis holds immediately.

For the inductive step we have $w = \vec{a}w_2$, and $(\vec{a}w_2, \vec{q}_0) \rightarrow (w_2, \vec{q}) \rightarrow^{+} (\varepsilon, \vec{f})$. By inductive hypothesis and Lemma 9.10 we have $p/\vec{a} \vdash \lambda(p/\vec{a})$. If $\vec{a}$ is an offer by Definition 9.9 we have $p = p/\vec{a}$ and the thesis holds directly. Note that $\lambda(p) = \lambda(p/\vec{a})$ because $\vec{a}$ labels a match or an offer transition outgoing from $\vec{q}_0$ and the offer comes from the conclusion of a contractual implication or a conjunction of atoms, that is unmodified in $p/\vec{a}$. Hence since by inductive hypothesis $p/\vec{a} \vdash \lambda(p/\vec{a})$ and since $\lambda(p) = \lambda(p/\vec{a})$, proving $p \vdash p/\vec{a}$ entails $p \vdash \lambda(p)$. This is because of the following proof (note that there exists a longer one, cut-free) and Lemma 9.12 (ii)

$$\dfrac{p \vdash p/\vec{a} \qquad p, p/\vec{a} \vdash \lambda(p)}{p \vdash \lambda(p)}\,cut$$

To prove $p \vdash p/\vec{a}$ we proceed by cases according to the structure of p, (omitting the cases for $J = \emptyset$ for which the proof is trivial)

- if $p = p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \wedge b \rightarrow b')$ we have to prove the sequent $p \vdash p/\vec{a}$ that reads as

$$(\,p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \wedge b \rightarrow b')\,) \vdash (\,p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \rightarrow b')\,)$$

For readability, we first determine the sequent $\Gamma \vdash (\bigwedge_{j\in J} a_j) \rightarrow b'$ where $\Gamma = p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \wedge b) \rightarrow b'$ from p, by applying the rule $\wedge R$, and Lemma 9.12(iii). Then we build the following derivation, where * is detailed below:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \bigwedge_{j\in J} a_j \vdash \bigwedge_{j\in J} a_j}\,id \qquad \dfrac{*}{\Gamma, \bigwedge_{j\in J} a_j \vdash b}\,\diamondsuit}{\Gamma, \bigwedge_{j\in J} a_j \vdash \bigwedge_{j\in J} a_j \wedge b}\,\wedge R \qquad \dfrac{}{\Gamma, \bigwedge_{j\in J} a_j, b' \vdash b'}\,id}{\Gamma, \bigwedge_{j\in J} a_j \vdash b'}\,\rightarrow L}{\Gamma \vdash (\bigwedge_{j\in J} a_j) \rightarrow b'}\,\rightarrow R$$

The fragment * of the proof can have two different forms, depending on the set Z:

– if $Z = \emptyset$, then * is empty and the rule $\diamondsuit$ is *id*

- otherwise the fragment * consists of the two sub-derivations below, and the rule $\Diamond$ applied to them is Fix

$$\begin{gathered} \frac{\Delta_3}{\Gamma, \bigwedge_{j\in J} a_j, b \vdash \bigwedge_{z\in Z} c_z} \\ \frac{}{\Gamma, \bigwedge_{j\in J} a_j, b \vdash b}\,id \end{gathered} \tag{9.1}$$

We now show how to obtain Δ_3. Let Δ be the derivation tree for $p/\vec{a} \vdash \lambda(p/\vec{a})$, that exists by the inductive hypothesis. Note that since $\lambda(p/\vec{a})$ is a conjunction where $\bigwedge_{z\in Z} c_z$ occurs, the following proof can be obtained by applying Lemma 9.12(iv) for all c_z and by combining them with rule $\wedge R$:

$$\frac{\Delta_2}{(p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \rightarrow b')) \vdash \bigwedge_{z\in Z} c_z} \tag{9.2}$$

Now, in order to obtain the following from the proof (9.2), i.e.

$$\frac{\Delta_3}{(p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \wedge b) \rightarrow b', \bigwedge_{j\in J} a_j, b) \vdash \bigwedge_{z\in Z} c_z} \tag{9.3}$$

we apply Lemma 9.12(ii): the left hand-side of the sequent

$$(p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \rightarrow b')) \vdash \bigwedge_{z\in Z} c_z$$

is augmented with $\bigwedge_{j\in J} a_j$. Finally by applying Lemma 9.12(i), the formula $\bigwedge_{j\in J} a_j \rightarrow b'$ above becomes $(\bigwedge_{j\in J} a_j \wedge b) \rightarrow b', b$, obtaining (9.3).

- if $p = p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge ((\bigwedge_{j\in J} a_j \wedge b) \rightarrow b')$ we have to prove the sequent $p \vdash p/\vec{a}$ that reads as

$$(p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \wedge b) \rightarrow b') \vdash (p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \rightarrow b'))$$

For readability, we first determine the sequent $\Gamma \vdash (\bigwedge_{j\in J} a_j) \rightarrow b'$ where $\Gamma = p', (\bigwedge_{k\in K} a_k \wedge b), ((\bigwedge_{j\in J} a_j \wedge b) \rightarrow b')$ from p, by applying the rule $\wedge R$ and Lemma 9.12(iii). Then we build the following derivation, where * is detailed below:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \bigwedge_{j\in J} a_j \vdash \bigwedge_{j\in J} a_j}\,id \qquad \dfrac{*}{\Gamma, \bigwedge_{j\in J} a_j \vdash b}\,\Diamond}{\Gamma, \bigwedge_{j\in J} a_j \vdash \bigwedge_{j\in J} a_j \wedge b}\,\wedge R \qquad \dfrac{}{\Gamma, \bigwedge_{j\in J} a_j, b' \vdash b'}\,id}{\Gamma, \bigwedge_{j\in J} a_j \vdash b'}\,\rightarrow L}{\Gamma \vdash (\bigwedge_{j\in J} a_j) \rightarrow b'}\,\rightarrow R$$

The fragment * of the proof can have two different forms, depending on the set K:

- if $K = \emptyset$, then * is empty and the rule $\Diamond$ is id
- otherwise the rule $\Diamond$ is $\wedge L2$ applied to the fragment * below

$$\frac{}{p', (\bigwedge_{k\in K} a_k \wedge b), b, ((\bigwedge_{j\in J} a_j \wedge b) \rightarrow b'), \bigwedge_{j\in J} a_j \vdash b}\,id$$

- if $p = p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge ((\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b')$ we have to prove the sequent $p \vdash p/\vec{a}$ that reads as

$$(\, p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b' \,) \vdash (\, p' \wedge (\bigwedge_{z\in Z} c_z \twoheadrightarrow b) \wedge (\bigwedge_{j\in J} a_j \twoheadrightarrow b') \,)$$

For readability, we first determine the sequent $\Gamma \vdash \bigwedge_{j\in J} a_j \twoheadrightarrow b'$ where $\Gamma = p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \wedge b \twoheadrightarrow b')$, by applying the rule $\wedge R$ and Lemma 9.12(iii). Then we build the following derivation, where * is detailed afterwards:

$$\dfrac{\dfrac{\dfrac{*}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,Fix \qquad \dfrac{}{\Gamma, b' \vdash b'}\,id}{\Gamma \vdash b'}\,Fix}{\Gamma \vdash \bigwedge_{j\in J} a_j \twoheadrightarrow b'}\,Zero$$

The fragment * of the proof can have two different forms, depending on the set Z:

– if $Z = \emptyset$, we have that $\Gamma = p', b, (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b'$ and

$$\dfrac{\dfrac{}{\Gamma, b', \bigwedge_{j\in J} a_j \wedge b \vdash \bigwedge_{j\in J} a_j \wedge b}\,id \qquad \dfrac{\dfrac{\Delta_3'}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j} \qquad \dfrac{}{\Gamma, b' \vdash b}\,id}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,\wedge R}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,Fix$$

Since the inductive hypothesis guarantees that $p/\vec{a} \vdash \lambda(p/\vec{a})$ holds and $\lambda(p/\vec{a})$ is a conjunction where $\bigwedge_{j\in J} a_j$ occurs, by applying the reasoning of the previous case we have a derivation tree Δ_2' for the sequent

$$(\, p', b, (\bigwedge_{j\in J} a_j \twoheadrightarrow b') \,) \vdash \bigwedge_{j\in J} a_j$$

As done above, by applying Lemma 9.12 we obtain the derivation tree Δ_3' for

$$(\, \Gamma'', p', b, (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b', b' \,) \vdash \bigwedge_{j\in J} a_j$$

– if $Z \neq \emptyset$ we obtain:

$$\dfrac{\dfrac{(**)}{\Gamma, b', \bigwedge_{j\in J} a_j \wedge b \vdash \bigwedge_{z\in Z} c_z} \qquad \dfrac{\dfrac{(***)}{\Gamma, b', b \vdash \bigwedge_{j\in J} a_j} \qquad \dfrac{}{\Gamma, b', b \vdash b}\,id}{\Gamma, b', b \vdash \bigwedge_{j\in J} a_j \wedge b}\,\wedge R}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,Fix$$

From the induction hypothesis, with the argument used in the previous cases, we prove the following sequent

$$(p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \twoheadrightarrow b')) \vdash \bigwedge_{z\in Z} c_z$$

Now, we apply Lemma 9.12 to it, we determine the deduction $(**)$ and a proof for the leftmost sequent above

$$(p', b', \bigwedge_{j\in J} a_j \wedge b, (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \wedge b \twoheadrightarrow b')) \vdash \bigwedge_{z\in Z} c_z$$

Just as done above, from the induction hypothesis we prove the sequent

$$(p', (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j) \twoheadrightarrow b') \vdash \bigwedge_{j\in J} a_j$$

from which we obtain the right most sequent above (***), by applying Lemma 9.12

$$(p', b', b, (\bigwedge_{z\in Z} c_z \twoheadrightarrow b), (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b') \vdash \bigwedge_{j\in J} a_j$$

- if $p = p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \wedge b \twoheadrightarrow b')$ we have to prove the sequent $p \vdash p/\vec{a}$ that reads as

$$(\, p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \wedge b \twoheadrightarrow b')\,) \vdash (\, p' \wedge (\bigwedge_{k\in K} a_k \wedge b) \wedge (\bigwedge_{j\in J} a_j \twoheadrightarrow b')\,)$$

For readability, we first determine the sequent $\Gamma \vdash \bigwedge_{j\in J} a_j \twoheadrightarrow b'$ where $\Gamma = p', (\bigwedge_{k\in K} a_k \wedge b), (\bigwedge_{j\in J} a_j \wedge b \twoheadrightarrow b')$, by applying the rule $\wedge R$ and Lemma 9.12(iii). Then we build the following derivation, where * is detailed afterwards:

$$\dfrac{\dfrac{\dfrac{*}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,Fix \qquad \dfrac{}{\Gamma, b' \vdash b'}\,id}{\Gamma \vdash b'}\,Fix}{\Gamma \vdash \bigwedge_{j\in J} a_j \twoheadrightarrow b'}\,Zero$$

The fragment * of the proof can have two different forms, depending on the set K:

– if $K = \emptyset$, we have $\Gamma = p', b, (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b'$ and

$$\dfrac{\dfrac{}{\Gamma, b', \bigwedge_{j\in J} a_j \wedge b \vdash \bigwedge_{j\in J} a_j \wedge b}\,id \qquad \dfrac{\dfrac{\Delta_3'}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j} \qquad \dfrac{}{\Gamma, b' \vdash b}\,id}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,\wedge R}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,Fix$$

Since the inductive hypothesis guarantees that $p/\vec{a} \vdash \lambda(p/\vec{a})$ holds and $\lambda(p/\vec{a})$ is a conjunction where $\bigwedge_{j\in J} a_j$ occurs, by applying the reasoning of the previous case we have a derivation tree Δ_2' for the sequent

$$(\, p', b, (\bigwedge_{j\in J} a_j \twoheadrightarrow b')\,) \vdash \bigwedge_{j\in J} a_j$$

As done above, by applying Lemma 9.12 we obtain the derivation tree Δ_3' for

$$(\, p', b, (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b', b'\,) \vdash \bigwedge_{j\in J} a_j$$

– if $K \neq \emptyset$ we have that $\Gamma = p', (\bigwedge_{k\in K} a_k \wedge b), (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b'$ and

$$\dfrac{\dfrac{}{\Gamma, b', \bigwedge_{j\in J} a_j \wedge b \vdash \bigwedge_{j\in J} a_j \wedge b}\,id \qquad \dfrac{\dfrac{\Delta_3'}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j} \qquad \dfrac{\dfrac{}{\Gamma, b', b \vdash b}\,id}{\Gamma, b' \vdash b}\,\wedge L2}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,\wedge R}{\Gamma, b' \vdash \bigwedge_{j\in J} a_j \wedge b}\,Fix$$

Since the inductive hypothesis guarantees that $p/\vec{a} \vdash \lambda(p/\vec{a})$ holds and $\lambda(p/\vec{a})$ is a conjunction where $\bigwedge_{j\in J} a_j$ occurs, by applying the reasoning of the previous case we have a derivation tree Δ_2' for the sequent

$$(\,p', (\bigwedge_{k\in K} a_k \wedge b), (\bigwedge_{j\in J} a_j \twoheadrightarrow b')\,) \vdash \bigwedge_{j\in J} a_j$$

As done above, by applying Lemma 9.12 we obtain the derivation tree Δ_3' for

$$(\,p', (\bigwedge_{k\in K} a_k \wedge b), (\bigwedge_{j\in J} a_j \wedge b) \twoheadrightarrow b', b'\,) \vdash \bigwedge_{j\in J} a_j$$

□

Theorem 5.9. *Let p be a H-PCL formula with no occurrence of standard implications $\rightarrow$, then $p \vdash \lambda(p)$ if and only if $[\![p]\!]$ admits weak agreement.*

Proof. ($\Rightarrow$) Straightforward from Theorem 5.6 and from $\mathfrak{A} \subset \mathfrak{W}$

($\Leftarrow$) Since $[\![p]\!]$ admits weak agreement there exists a trace $w \in \mathscr{L}([\![p]\!])$ where each request is combined with a corresponding offer. For proving $p \vdash \lambda(p)$ we will prove $p \vdash a$ for all the atoms a in $\lambda(p)$ and the thesis follows by repeatedly applying the rule $\wedge R$. If a occurs within:

(1) $\bigwedge_{j\in J} a_j$: it suffices to apply the rules $\wedge L_1, \wedge L_2, id$;

(2) $\bigwedge_{j\in J} a_j \twoheadrightarrow a$: $p \vdash a$ holds if we prove the sequent

$$\Gamma, (\bigwedge_{j\in J} a_j \twoheadrightarrow a) \vdash a$$

that is obtained from $p \vdash a$ by repeatedly applying the rules $\wedge L_i$, for some Γ containing p and sub-formulas of p. The proof of this sequent has the following form:

$$\dfrac{\dfrac{*}{\Gamma, (\bigwedge_{j\in J} a_j \twoheadrightarrow a), a \vdash \bigwedge_{j\in J} a_j} \qquad \dfrac{}{\Gamma, (\bigwedge_{j\in J} a_j \twoheadrightarrow a), a \vdash a}\,id}{\Gamma, (\bigwedge_{j\in J} a_j \twoheadrightarrow a) \vdash a}\,Fix$$

We prove the sequent in the left premise, it suffices to establish the sequents $\Gamma, (\bigwedge_{j\in J} a_j \twoheadrightarrow a), a \vdash a_j$, for all the atoms a_j of $\bigwedge_{j\in J} a_j$. Then, the derivation proceeds by repeatedly applying the rule $\wedge R$. We are left to prove $\Gamma, (\bigwedge_{j\in J} a_j \twoheadrightarrow a), a \vdash a_j$, which is done by recursively applying the construction of cases (1) and (2). This procedure will eventually terminate. Indeed, at each iteration a_j is either a conjunct in $\bigwedge_{k\in K} a_k$ (case 1) and the proof is closed by rule (id), or a_j is the conclusion of the contractual implication $\bigwedge_{k\in K} a_k \twoheadrightarrow a_j$ and the proof proceeds as in case (2) by applying the rule (Fix). In the last case, the premise in the left hand-side becomes $\Gamma', (\bigwedge_{k\in K} a_k \twoheadrightarrow a_j), a, a_j \vdash \bigwedge_{k\in K} a_k$, so adding a_j in the left part of the sequent. The number of iterations is therefore bound by the number of atoms in p.

(3) $\bigwedge_{j\in J} a_j \twoheadrightarrow b$ where $a \neq b$. This case reduces to one of the above two, because if $\exists j \in J$ such that $a_j = a$, then a must also appear in another conjunct $\bigwedge_{z\in Z} a_z$ or in another contractual implication $\bigwedge_{z\in Z} a_z \twoheadrightarrow a$, otherwise all the traces of $[\![p]\!]$ would have an unmatched request on a, against the hypothesis that it admits weak agreement.

□

$$\frac{}{A\vdash A}\,Ax \qquad \frac{\Gamma\vdash \quad \Gamma'\vdash\gamma}{\Gamma,\Gamma'\vdash\gamma}\,Mix \qquad \frac{\Gamma\vdash A}{\Gamma,A^{\perp}\vdash}\,NegL \qquad \frac{\Gamma,A,B\vdash\gamma}{\Gamma,A\otimes B\vdash\gamma}\,\otimes L$$

$$\frac{\Gamma\vdash A \quad \Gamma'\vdash B}{\Gamma,\Gamma'\vdash A\otimes B}\,\otimes R \qquad \frac{\Gamma\vdash A \quad \Gamma',B\vdash\gamma}{\Gamma,\Gamma',A\multimap B\vdash\gamma}\,\multimap L \qquad \frac{\Gamma,A\vdash B}{\Gamma\vdash A\multimap B}\,\multimap R$$

$$\frac{\Gamma\vdash A \quad \Gamma',A\vdash\gamma}{\Gamma,\Gamma'\vdash\gamma}Cut \qquad \frac{\Gamma,A\vdash}{\Gamma\vdash A^{\perp}}NegR \qquad \frac{\Gamma\vdash}{\Gamma\vdash\perp}\perp R \qquad \frac{}{\perp\vdash}\perp L$$

$$\frac{}{\vdash 1}1R \qquad \frac{\Gamma\vdash\gamma}{\Gamma,1\vdash\gamma}1L \qquad \frac{}{\Gamma\vdash\top}\top \qquad \frac{}{\Gamma,0\vdash A}0L$$

$$\frac{\Gamma,A\vdash\gamma \quad \Gamma,B\vdash\gamma}{\Gamma,A\oplus B\vdash\gamma}\oplus L \qquad \frac{\Gamma\vdash A}{\Gamma\vdash A\oplus B}\oplus R1 \qquad \frac{\Gamma\vdash B}{\Gamma\vdash A\oplus B}\oplus R2$$

$$\frac{\Gamma\vdash A \quad \Gamma\vdash B}{\Gamma\vdash A\& B}\& R \qquad \frac{\Gamma,A\vdash\gamma}{\Gamma,A\& B\vdash\gamma}\& L1 \qquad \frac{\Gamma,B\vdash\gamma}{\Gamma,A\& B\vdash\gamma}\& L2$$

$$\frac{\Gamma,A\vdash\gamma}{\Gamma,!A\vdash\gamma}!L \qquad \frac{!\Gamma\vdash A}{!\Gamma\vdash !A}!R \qquad \frac{\Gamma\vdash\gamma}{\Gamma,!A\vdash\gamma}weakL \qquad \frac{\Gamma,!A,!A\vdash\gamma}{\Gamma,!A\vdash\gamma}coL$$

FIGURE 15. The sequent calculus for ILL^{mix}

9.5. **Automata and Intuitionistic Linear Logic with Mix.** We recall for completeness the full grammar of ILL^{mix}.

Definition 9.13. The formulas $A,B,\dots$ of $ILL^{mix}$are defined as follows:

$$A ::= a \mid A^{\perp} \mid A\otimes A \mid A\multimap A \mid A\& A \mid A\oplus A \mid !A \mid 1 \mid 0 \mid \top \mid \perp$$

The full sequent calculus for $ILL^{mix}$is displayed in Figure 15. We will only consider proofs without the rule *Cut*, which is redundant by [21], Theorem 24.

The following definition and lemmata are auxiliary.

Lemma 9.14. *If $\Gamma\vdash Z$ is an honoured sequent, there exists a derivation tree for $\Gamma\vdash Z$ such that:*

- *it only uses the rules $Ax, Mix, NegL, \otimes L, \otimes R$ and $\multimap L$ of Figure 15;*
- *it is only made of honoured sequents.*

Proof. Recall that we are in the Horn fragment and we only consider cut-free proofs. Since Z is a positive tensor product (or empty), a simple inspection on the rules in Figure 15 suffices to prove the first statement. The second statement is proved because $Ax, Mix, NegL, \otimes L, \otimes R$ and $\multimap L$ introduce no sequents with negative literals on their right hand-side. □

Lemma 9.15. *Let $\Gamma\vdash Z$ be an honoured sequent, then:*

$$\Gamma\vdash Z \textit{ implies } [\![\Gamma]\!] \textit{ admits agreement on } Z.$$

Proof. We will prove that there exists a trace $w\in\mathscr{L}([\![\Gamma]\!])$ made of matches and as many offers as the literals in $Z=\bigotimes_{a\in Y} a$ (recall that they all are positive), or it is made by only matches if Z is empty. Also, note that the sequents in the proof of $\Gamma\vdash Z$ are all honoured, by hypothesis and Lemma 9.14. We proceed by induction on the depth of the proof of $\Gamma\vdash Z$.

In the base case, the proof consists of a single application of the rule *Ax*. By Definition 5.13 one first has an offer transition for each a in Z, and then interleaves them in any possible order. Hence the thesis holds trivially.

For the inductive case we proceed by cases on the last rule applied. We assume that all clauses (i.e. principals) in Γ are divided by commas, which can be easily obtained by repeatedly applying the rule $\otimes L$. In the following, let $\overline{a}$ be offers in correspondence with the literals a in Z, we will consider only the relevant rules as stated by Lemma 9.14.

- $\dfrac{\Gamma \vdash \quad \Gamma' \vdash Z}{\Gamma, \Gamma' \vdash Z}$ *Mix* By induction hypothesis there exists $w \in \mathscr{L}(\llbracket \Gamma \rrbracket)$ with match actions, only, and $w_1 \in \mathscr{L}(\llbracket \Gamma' \rrbracket)$ with match actions and offers in correspondence with the literals in Z (if non-empty). By Definition 2.8, there exists $w_2 \in \mathscr{L}(\llbracket \Gamma \rrbracket \boxtimes \llbracket \Gamma' \rrbracket)$ in agreement.

- $\dfrac{\Gamma \vdash A}{\Gamma, A^{\perp} \vdash}$ *NegL* By induction hypothesis there exists $w \in \mathscr{L}(\llbracket \Gamma \rrbracket)$ with match actions, and with offers in correspondence with the literals in A. By Definition 5.13 the traces of the automaton $\llbracket A^{\perp} \rrbracket$ are all the possible permutations of the requests in correspondence with the literals in $A^{\perp}$. The thesis follows, because there is an offer for each request, and by Definition 2.8.

- $\dfrac{\Gamma, A, B \vdash Z}{\Gamma, A \otimes B \vdash Z}$ $\otimes L$ By the induction hypothesis there exists $w \in \mathscr{L}(\llbracket \Gamma, A, B \rrbracket) = \mathscr{L}(\llbracket \Gamma \rrbracket \boxtimes \llbracket A \rrbracket \boxtimes \llbracket B \rrbracket)$ with offers in correspondence with the literals in Z (if non-empty). No atom and its negation can occur in $A \otimes B$ by Definition 5.11, because it is a principal. Hence $\llbracket A \otimes B \rrbracket$ and $\llbracket A, B \rrbracket$ are the same automaton (with a different rank), and the statement follows immediately.

- $\dfrac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \otimes B}$ $\otimes R$ By the induction hypothesis there exist $w \in \mathscr{L}(\llbracket \Gamma \rrbracket)$ and $w' \in \mathscr{L}(\llbracket \Gamma' \rrbracket)$ with only match actions and offers in correspondence with the literals in A and in B, respectively. Now Definition 2.8 guarantees that there exists a trace in $\mathscr{L}(\llbracket \Gamma \rrbracket \boxtimes \llbracket \Gamma' \rrbracket)$ in agreement.

- $\dfrac{\Gamma \vdash A \quad \Gamma', B \vdash Z}{\Gamma, \Gamma', A \multimap B \vdash Z} \multimap L$ By the induction hypothesis there exists $w \in \mathscr{L}(\llbracket \Gamma \rrbracket)$ and $w' \in \mathscr{L}(\llbracket \Gamma', B \rrbracket)$ with only match actions and offers in correspondence with the literals in A and in Z (if non-empty), respectively. By Definition 5.13 the literals occurring in A become requests in $\llbracket A \multimap B \rrbracket$, in all possible ordering. The trace w contains exactly the needed matching offers. We conclude by noting that no other request is possible in $\mathscr{L}(\llbracket \Gamma, \Gamma', A \multimap B \rrbracket)$.

□

In order to keep the following definition compact, with a slight abuse of the notation we use $\otimes$ to operate between formulas; we remove the constraints of Definition 5.11 on the indexing sets I in formulas and X_1, X_2 and Y in clauses; and we let $\bigotimes_{b \in \emptyset} b \multimap \bigotimes_{a \in X_2} a$ to stand for $\bigotimes_{a \in X_2} a$.

Definition 9.16. Given a Horn formula p and an offer or match transition leaving the initial state of $\llbracket p \rrbracket$ with label $\vec{a}$, then define the formula $p/\vec{a}$ as:

$$p/\vec{a} = \begin{cases} p' \otimes \bigotimes_{a_1 \in X_1} a_1 & \text{if } \vec{a} \text{ is an offer on } c \text{ and} \\ & p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \\ p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2 & \text{if } \vec{a} \text{ is a match on } c \text{ and} \\ & p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes \\ & \quad \bigotimes_{a_2 \in X_2 \cup \{c^\perp\}} a_2 \\ p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{b \in Y} b \multimap \bigotimes_{a_2 \in X_2} a_2 & \text{if } \vec{a} \text{ is a match on } c \text{ and} \\ & p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes \\ & \quad \bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2 \end{cases}$$

We now establish a relation between $\llbracket p/\vec{a} \rrbracket$, and the contract automaton obtained by changing the initial state $\vec{q}_0$ of $\llbracket p \rrbracket$ to $\vec{q}$, for the transition $(\vec{q}_0, \vec{a}, \vec{q})$ of $\llbracket p \rrbracket$. Without loss of generality we assume that the automaton obtained from Definition 5.13 is deterministic. If not, we first transform the non deterministic automaton to a deterministic one.

Lemma 9.17. *Given a Horn formula p and the contract automaton $\llbracket p \rrbracket = \langle Q, \vec{q}_0, A^r, A^o, T, F \rangle$, if $t = (\vec{q}_0, \vec{a}, \vec{q}) \in T$ is an offer or a match transition, then $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\llbracket p/\vec{a} \rrbracket)$, where $\mathcal{A} = \langle Q, \vec{q}, A^r, A^o, T, F \rangle$.*

Proof. The proof is similar to the one of Lemma 9.10. The statement follows by noting that in Definition 5.13 a tensor product is translated in all the possible permutations of actions corresponding to the literals, and noting that in $p/\vec{a}$ we remove exactly the actions fired in $\vec{a}$, that are therefore not available any more in the state $\vec{q}$. □

The following lemma suggests that we can safely substitute a multi-set of Horn formulae and clauses Γ with a single Horn formula, without affecting the corresponding automaton.

Lemma 9.18. *Let Γ be a non-empty multi-set of Horn formulae, then there exists a Horn formula p such that:*

$$\llbracket \Gamma \rrbracket = \llbracket p \rrbracket$$

Proof. Immediate from Definition 5.13 (recall that we abuse the notation). □

We now prove the following lemma.

Lemma 9.19. *Let $\Gamma \neq \emptyset$ be a multi-set of Horn formulae and Z be a positive tensor product or empty. Then*

$$\llbracket \Gamma \rrbracket \textit{ admits agreement on } Z \textit{ implies } \Gamma \vdash Z \textit{ is an honoured sequent}$$

Proof. By hypothesis $w \in \mathscr{L}(\llbracket \Gamma \rrbracket)$ is a trace only composed of match and offer actions on Z. We proceed by induction on the length of w. In the base case w has length one. Note that it is not possible to have $w = \varepsilon$ by the hypothesis $\Gamma \neq \emptyset$ and Definition 5.11. Moreover by Definition 5.11 it must be that $w = \vec{a}$ where $\vec{a}$ is a match on action a (a Horn formula must contain at least two principals). Hence by Definition 5.13 it must be that $Z = \emptyset$ and $\Gamma = \{\alpha \otimes \alpha'\}$ where $\alpha = a$ and $\alpha' = a^\perp$ for some literal a. Then we have:

$$\dfrac{\dfrac{\dfrac{}{a \vdash a}\,Ax}{a, a^\perp \vdash}\,Neg}{a \otimes a^\perp \vdash}\,\otimes L$$

For the inductive step, let $w = \vec{a} w_2$, let $\vec{q}_0$ and $\vec{f}$ be the initial and the final states of $\llbracket \Gamma \rrbracket$, then $(\vec{a} w_2, \vec{q}_0) \rightarrow (w_2, \vec{q}) \rightarrow^+ (\varepsilon, \vec{f})$. Let p be a Horn formula such that $\llbracket \Gamma \rrbracket = \llbracket p \rrbracket$ (Lemma 9.18), so it suffices to prove $p \vdash Z$. By the induction hypothesis and Lemma 9.17 we have that $[\![p/\vec{a}]\!]$ admits agreement on some Z' implies $p/\vec{a} \vdash Z'$ is honoured. To build Z from Z', we proceed by cases on $\vec{a}$:

- if $\vec{a}$ is an offer action on c we prove that $p \vdash Z$ where $Z = Z' \otimes c$. We have the following

$$\dfrac{\dfrac{\Delta'}{p \vdash (p/\vec{a}) \otimes c} \qquad \dfrac{\dfrac{\Delta}{(p/\vec{a}) \vdash Z'} \quad \dfrac{}{c \vdash c}Ax}{p/\vec{a} \otimes c \vdash Z' \otimes c}\otimes R}{p \vdash Z}cut$$

  where Δ is obtained by the inductive hypothesis and for Δ' we have two cases depending on p:

  - $p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1$ then the derivation $\dfrac{\Delta'}{p \vdash (p/\vec{a}) \otimes c}$ becomes $\dfrac{}{p' \otimes c \vdash p' \otimes c}Ax$ if $X_1 = \emptyset$ and the following otherwise

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p' \vdash p'}Ax \quad \dfrac{}{\bigotimes_{a_1 \in X_1} a \vdash \bigotimes_{a_1 \in X_1} a}Ax}{p', \bigotimes_{a_1 \in X_1} a \vdash p' \otimes \bigotimes_{a_1 \in X_1} a}\otimes R \quad \dfrac{}{c \vdash c}Ax}{p', \bigotimes_{a_1 \in X_1} a, c \vdash p' \otimes \bigotimes_{a_1 \in X_1} a \otimes c}\otimes R}{p', \bigotimes_{a_1 \in X_1 \cup \{c\}} a \vdash p' \otimes \bigotimes_{a_1 \in X_1} a \otimes c}\otimes L}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a \vdash p' \otimes \bigotimes_{a_1 \in X_1} a \otimes c}\otimes L$$

- if $\vec{a}$ is a match action we prove that $p \vdash Z$. We have the following

$$\dfrac{\dfrac{\Delta'}{p \vdash p/\vec{a}} \qquad \dfrac{\Delta}{p/\vec{a} \vdash Z'}}{p \vdash Z'}cut$$

  where Δ is obtained by the inductive hypothesis, $Z = Z'$ because $\vec{a}$ is a match, and for Δ' we have eight cases depending on p:

  - $p = p' \otimes c \otimes c^{\perp}$; then the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes: $\dfrac{\dfrac{\Delta_{mix}}{p', c \otimes c^{\perp} \vdash p'}\otimes L}{p' \otimes c \otimes c^{\perp} \vdash p'}\otimes L$

    Since the deduction tree Δ_{mix} will be also used later on, we keep it more general, by writing q for p':

$$\Delta_{mix} \quad = \quad \dfrac{\dfrac{\dfrac{}{c \vdash c}Ax}{c, c^{\perp} \vdash}NegL \quad \dfrac{}{q \vdash q}id}{q, c, c^{\perp} \vdash q}Mix$$

  - $p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes c^{\perp}$

    then, writing in Δ_{mix} $p' \otimes \bigotimes_{a_1 \in X_1} a_1$ for q the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{mix}}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1, c^{\perp} \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1}\otimes L}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes c^{\perp} \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1}\otimes L$$

  - $p = p' \otimes \bigotimes_{a_2 \in X_2 \cup \{c^{\perp}\}} a_2 \otimes c$

then, writing in Δ_{mix} $p' \otimes \bigotimes_{a_2 \in X_2} a_2$ for q the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{mix}}{p' \otimes \bigotimes_{a_2 \in X_2 \cup \{c^\perp\}} a_2, c \vdash p' \otimes \bigotimes_{a_2 \in X_2} a_2}\otimes L}{p' \otimes \bigotimes_{a_2 \in X_2 \cup \{c^\perp\}} a_2 \otimes c \vdash p' \otimes \bigotimes_{a_2 \in X_2} a_2}\otimes L$$

- $p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes \bigotimes_{a_2 \in X_2 \cup \{c^\perp\}} a_2$

  then, writing in Δ_{mix} $p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2$ for q the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{mix}}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2, c^\perp \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2}\otimes L}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes \bigotimes_{a_2 \in X_2 \cup \{c^\perp\}} a_2 \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2}\otimes L$$

- $p = p' \otimes c \otimes (c \multimap \bigotimes_{a_2 \in X_2} a_2)$

  then the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{ax} \quad \Delta_{\multimap}}{p', c, (c \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash p' \otimes \bigotimes_{a_2 \in X_2} a_2}\otimes R}{p' \otimes c \otimes (c \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash p' \otimes \bigotimes_{a_2 \in X_2} a_2}\otimes L(x2)$$

  where letting $q = p'$

$$\Delta_{ax} \quad = \quad \dfrac{}{q \vdash q}Ax$$

  and $\Delta_{\multimap}$ is the following proof:

$$\dfrac{\dfrac{}{c \vdash c}Ax \qquad \dfrac{}{\bigotimes_{a_2 \in X_2} a_2 \vdash \bigotimes_{a_2 \in X_2} a_2}Ax}{c, (c \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash \bigotimes_{a_2 \in X_2} a_2}\multimap L$$

- $p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes (c \multimap \bigotimes_{a_2 \in X_2} a_2)$

  then, letting in Δ_{ax} $q = p' \otimes \bigotimes_{a_1 \in X_1} a_1$, the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{ax} \quad \Delta_{\multimap}}{p' \otimes \bigotimes_{a_1 \in X_1} a_1, c, (c \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2}\otimes R}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes (c \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes \bigotimes_{a_2 \in X_2} a_2}\otimes L \text{ — twice}$$

- $p = p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2)$

  then writing $\hat{q}$ for $p' \otimes \bigotimes_{a_1 \in X_1} a_1 \otimes (\bigotimes_{b \in Y} b \multimap \bigotimes_{a_2 \in X_2} a_2)$ below, the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{ax} \quad \Delta_{\multimap 2}}{p' \otimes \bigotimes_{a_1 \in X_1} a_1, c, (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash \hat{q}}\otimes R}{p' \otimes \bigotimes_{a_1 \in X_1 \cup \{c\}} a_1 \otimes (\bigotimes_{b \in Y \cup \{c\}} b \multimap \bigotimes_{a_2 \in X_2} a_2) \vdash \hat{q}}\otimes L(x2)$$

where $q = p' \otimes \bigotimes_{a_1 \in X_1} a_1$ in Δ_{ax}, and $\Delta_{\multimap 2}$ is the deduction tree below:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{c \vdash c}Ax \quad \dfrac{}{\bigotimes_{b\in Y} b \vdash \bigotimes_{b\in Y} b}Ax}{c, \bigotimes_{b\in Y} b \vdash \bigotimes_{b\in Y\cup\{c\}} b}\otimes R \qquad \dfrac{}{\bigotimes_{a_2\in X_2} a_2 \vdash \bigotimes_{a_2\in X_2} a_2}Ax}{c, (\bigotimes_{b\in Y\cup\{c\}} b \multimap \bigotimes_{a_2\in X_2} a_2), \bigotimes_{b\in Y} b \vdash \bigotimes_{a_2\in X_2} a_2}\multimap L}{c, (\bigotimes_{b\in Y\cup\{c\}} b \multimap \bigotimes_{a_2\in X_2} a_2) \vdash \bigotimes_{b\in Y} b \multimap \bigotimes_{a_2\in X_2} a_2}\multimap R$$

- $p = p' \otimes c \otimes (\bigotimes_{b\in Y\cup\{c\}} b \multimap \bigotimes_{a_2\in X_2} a_2)$

then, letting $q = p'$ in Δ_{ax}, the derivation $\dfrac{\Delta'}{p \vdash p/\vec{a}}$ becomes:

$$\dfrac{\dfrac{\Delta_{ax} \quad \Delta_{\multimap 2}}{p', c \otimes (\bigotimes_{b\in Y\cup\{c\}} b \multimap \bigotimes_{a_2\in X_2} a_2) \vdash p' \otimes (\bigotimes_{b\in Y} b \multimap \bigotimes_{a_2\in X_2} a_2)}\otimes R}{p' \otimes c \otimes (\bigotimes_{b\in Y\cup\{c\}} b \multimap \bigotimes_{a_2\in X_2} a_2) \vdash p' \otimes (\bigotimes_{b\in Y} b \multimap \bigotimes_{a_2\in X_2} a_2)}\otimes L$$

□

The main theorem of this sub-section has now an immediate proof.

Theorem 5.15. *Given a multi-set of Horn formulae Γ, we have that*

$$\Gamma \vdash Z \textit{ is an honoured sequent if and only if } [\![\Gamma]\!] \textit{ admits agreement on } Z$$

Proof. By Lemmata 9.15 and 9.19. □