A Calculus of Contracting Processes

Massimo Bartoletti
Dipartimento di Matematica e Informatica, Università degli Studi di Cagliari

Roberto Zunino
Dipartimento di Ingegneria e Scienza dell'Informazione, Università degli Studi di Trento

Abstract

We propose a formal theory of contract-based computing. We model contracts as formulae in an intuitionistic logic extended with a "contractual" form of implication. Decidability holds for our logic: this allows us to mechanically infer the rights and the duties deriving from any set of contracts. We embed our logic in a core calculus of contracting processes, which combines features from concurrent constraints and calculi for multiparty sessions, while subsuming several idioms for concurrency.

1 Introduction

In Web transactions, the typical dynamics is that a client chooses a service provider that she trusts, relying on the fact that the service implements the required features. Such features are typically written in a "service level agreement" (SLA). Although this document is legally binding, it is not a formal specification. Formalising it would be desirable, for two main reasons. First, a formal SLA could be exploited by the client to mechanize the search of a service meeting her requests. Second, in the case the provider does not honour its SLA, automatic means could be devised to resolve the dispute. This would be more practical than taking legal steps against the provider, especially for transactions dealing with small amounts of money.

The interaction among parties has then to be regulated by a suitable *contract*, which formally subordinates the duties of a client to the duties of a service, and *vice versa*. The crucial problems are how to model a contract, how to infer when a set of contracts gives rise to an agreement among the stipulating parties, and how to single out the responsible of a possible violation.

An example. To give the intuition about our contracts, suppose there are two kids: Alice, who has a toy airplane, and Bob, who has a bike. Before sharing their toys, the two kids stipulate the following "gentlemen's agreement":

Alice: I will lend my airplane to you, Bob, provided that I borrow your bike.

Bob: I will lend my bike to you, Alice, provided that I borrow your airplane.

Let us write a for the atomic proposition "Alice lends her airplane" and b for "Bob lends his bike". A (wrong) formalisation of the above contracts in classical logic could model Alice's contract A as $\mathsf{b} \to \mathsf{a}$, and Bob's B as $\mathsf{a} \to \mathsf{b}$. However, from this we cannot deduce the expected agreement, i.e. $A \wedge B \to \mathsf{a} \wedge \mathsf{b}$ does not hold. To solve this issue, we propose Propositional Contract Logic (PCL), that extends intuitionistic logic IPC with a *contractual implication* connective $\twoheadrightarrow$. In PCL we have the desired agreement:

$$(\mathsf{b} \twoheadrightarrow \mathsf{a}) \ \wedge \ (\mathsf{a} \twoheadrightarrow \mathsf{b}) \ \to \ \mathsf{a} \wedge \mathsf{b}$$

To put our contracts at work, we introduce a process calculus which embeds our logic. This calculus belongs to the family of concurrent constraints [35], using PCL formulae as constraints. A process can assert a constraint c (a PCL formula) through the primitive $\mathsf{tell}\, c$. For instance, the following process models Alice exposing her contract:

$$(x)\ \mathsf{tell}\, \mathsf{b}(x) \twoheadrightarrow \mathsf{a}(x)$$

Formally, this will add $\mathsf{b}(x) \twoheadrightarrow \mathsf{a}(x)$ to the set of constraints. The formal parameter x represents the identifier of the actual session to be established between Alice and Bob. As it happens for sessions centered calculi [38, 14], sessions are an important aspect also in our calculus, since they allow for distinguishing among different instantiations of the same contract. The outer (x) is a scope delimitation for the variable x, similarly to the Fusion calculus [32].

After having exposed her contract, Alice will wait until finding that she has actually to lend her airplane to Bob. This is modelled as $\mathsf{fuse}_x\, \mathsf{a}(x)$. The primitive $\mathsf{fuse}_x\, c$ implements a contract-based multiparty agreement. To do that, it checks the entailment of the constraint c, and binds the variable x to an actual session identifier, shared among all the

parties involved in the contract. So, we will model Alice as:

$$Alice = (x)\ \big(\mathsf{tell}\,\mathsf{b}(x) \twoheadrightarrow \mathsf{a}(x).\ \mathsf{fuse}_x\,\mathsf{a}(x).\ lendAirplane\big)$$

where the process $lendAirplane$ (no further specified) models Alice actually lending her airplane to Bob. The overall behaviour of Alice is then: *(i)* issue the contract; *(ii)* wait until discovering the duty of lending the airplane; *(iii)* finally, lend the airplane. Dually, we model Bob as follows:

$$Bob = (y)\ \big(\mathsf{tell}\,\mathsf{a}(y) \twoheadrightarrow \mathsf{b}(y).\ \mathsf{fuse}_y\,\mathsf{b}(y).\ lendBike\big)$$

A possible interaction between Alice and Bob will be the following, where n stands for a fresh session identifier:

$$Alice|Bob\ \rightarrow^*\ (n)\ \big(lendAirplane\{^n/_x\}\ |\ lendBike\{^n/_y\}\big)$$

As expected, the resulting process shows Alice and Bob actually sharing their toys, in the session identified by n.

The logic PCL also allows for a more precise model of the above scenario, by linking the contracts with the identity of the principals issuing them (see Ex. 3). This information can be exploited in our calculus to automatically detect the responsible of a violation (see e.g. Ex. 6).

Contributions. We propose the logic PCL, which extends IPC with contractual implication. We provide it with an Hilbert-style axiomatisation and a Gentzen-style sequent calculus, which we prove equivalent. We study the relations between PCL, IPC and classical logic. The main results about PCL are cut elimination and decidability. We implement a proof search algorithm for PCL [37], also including an extension with a lax modality, to explicitly link contracts and principals. We then exploit PCL as a basic building block for designing a calculus of contracting processes. Our calculus is expressive enough to encode several concurrency idioms, among which Linda, the π-calculus and graph rewriting. We show our logic and calculus applicable to model real-world scenarios through several examples.

Because of space constraints, we include all the proofs, the encodings, as well as further results and examples about our logic and calculus, in two Technical Reports [6, 5].

2 A Logic for Contracts

Desirable properties. We start by characterizing our logic through a set of properties that we would expect to be enjoyed by any logic for contracts.

As shown in Sect. 1, a basic property of contractual implication is that of allowing two dual contracting parties to "handshake", so to make their agreement effective. This is resumed by the following *handshaking* property:

$$\vdash (p \twoheadrightarrow q) \wedge (q \twoheadrightarrow p)\ \rightarrow\ p \wedge q \tag{1}$$

A generalisation of the above property to the case of n contracting parties is also desirable. It is a sort of "circular" handshaking, where the $(i+1)$-th party, in order to promise some duty p_{i+1}, relies on a promise p_i made by the i-th party (in a circular fashion, the first party relies on the promise of the last one). In the case of n parties, we expect:

$$\begin{array}{l}\vdash (p_1 \twoheadrightarrow p_2) \wedge \cdots \wedge (p_{n-1} \twoheadrightarrow p_n) \wedge (p_n \twoheadrightarrow p_1)\\ \quad\rightarrow\ \ p_1 \wedge \cdots \wedge p_n\end{array} \tag{2}$$

As a concrete example, consider an e-commerce scenario where a client C can buy items from a seller S, and pay them through a credit card. To mediate the interaction between C and S, there is a bank B which manages payments. The contracts issued by the three parties could be as follows:

Client: I will click "pay" provided that my item is shipped

Seller: I will ship your item provided that I get the money

Bank: I will transfer money to the seller provided that the client clicks "pay".

Let the atomic propositions ship, click, and pay denote, respectively, the facts "seller ships item", "client clicks pay", and "bank transfers money". The above contracts can then be modelled as:

$$C = \mathsf{ship} \twoheadrightarrow \mathsf{click} \quad B = \mathsf{click} \twoheadrightarrow \mathsf{pay} \quad S = \mathsf{pay} \twoheadrightarrow \mathsf{ship}$$

Then, by property (2) we deduce a successful transaction:

$$\vdash C \wedge B \wedge S\ \rightarrow\ \mathsf{pay} \wedge \mathsf{ship}$$

Note that, in the special case $n = 1$, the above "circular" handshaking property turns into a particularly simple form:

$$\vdash (p \twoheadrightarrow p)\ \rightarrow\ p \tag{3}$$

Intuitively, (3) models that promising p provided that p, implies p (actually, also the converse holds, so that promise is equivalent to p). It also follows from (1) when $p = q$.

Another generalisation of the toy-exchange scenario to the case of n kids is also desirable. It is a sort of "greedy" handshaking, because now a party promises p_i only provided that *all* the other parties promise their duties, i.e. $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$.

$$\begin{array}{l}\vdash \bigwedge_{i\in 1..n}\Big((p_1 \wedge \cdots \wedge p_{i-1} \wedge p_{i+1} \wedge \cdots \wedge p_n) \twoheadrightarrow p_i\Big)\\ \quad\rightarrow\ \ p_1 \wedge \cdots \wedge p_n\end{array} \tag{4}$$

We will now focus on further logical properties of contractual implication. As shown by (1), a contract $p \twoheadrightarrow q$ becomes effective, i.e. implies the promise q, when matched by a dual contract $q \twoheadrightarrow p$. Even more directly, $p \twoheadrightarrow q$ should be effective also when the premise p is already true:

$$\vdash p\ \wedge\ (p \twoheadrightarrow q)\ \rightarrow\ q \tag{5}$$

In other words, contractual implication should be *stronger* than standard implication, i.e.:

$$\vdash (p \twoheadrightarrow q) \rightarrow (p \rightarrow q) \tag{6}$$

On the other hand, we do not want that also the converse holds, since this would equate the two forms of implication: that is, $\nvdash (p \rightarrow q) \rightarrow (p \twoheadrightarrow q)$.

We want contractual implication to share with standard implication a number of properties. First, a contract that promises true (written $\top$) is always satisfied, regardless of the precondition. So, we expect the following tautology:

$$\vdash p \twoheadrightarrow \top \tag{7}$$

Differently from standard implication, we do not want a contract with a false precondition (written $\bot$) to always hold, i.e. $\nvdash \bot \twoheadrightarrow p$. To see why, assume $\bot \twoheadrightarrow p$ is a tautology, for all p. Then, it would also be the case for $p = \bot$, so (3) would deduce a contradiction: $(\bot \twoheadrightarrow \bot) \rightarrow \bot$.

We want $\twoheadrightarrow$ enjoy transitivity, similarly to $\rightarrow$:

$$\vdash (p \twoheadrightarrow q) \ \wedge \ (q \twoheadrightarrow r) \ \rightarrow \ (p \twoheadrightarrow r) \tag{8}$$

Back to our previous example, transitivity would allow the promise of the client $(\mathsf{ship} \twoheadrightarrow \mathsf{click})$ and that of the bank $(\mathsf{click} \twoheadrightarrow \mathsf{pay})$ to be combined in the promise $\mathsf{ship} \twoheadrightarrow \mathsf{pay}$.

Contractual implication should also enjoy a stronger form of transitivity. We illustrate it with the help of an example. Suppose an air-flight customer who wants to book a flight. The customer contract promises to pay the required amount, provided that she obtains a flight reservation. Suppose now that an airline company starts a special offer, in the form of a free drink for each customer.

$$\mathit{Customer} = \mathsf{bookFlight} \twoheadrightarrow \mathsf{pay}$$
$$\mathit{AirLine} = \mathsf{pay} \twoheadrightarrow \mathsf{bookFlight} \wedge \mathsf{freeDrink}$$

Of course, the two contracts should give rise to an agreement, because the airline company is promising a better service than the one required by the customer contract. We then expect to be able to "weaken" the $\mathit{AirLine}$ contract:

$$\vdash \mathit{AirLine} \ \rightarrow \ (\mathsf{pay} \twoheadrightarrow \mathsf{bookFlight})$$

Alternatively, one could make the two contracts match by making stronger the precondition required by the customer:

$$\vdash \mathit{Customer} \ \rightarrow \ (\mathsf{bookFlight} \ \wedge \ \mathsf{freeDrink} \twoheadrightarrow \mathsf{pay})$$

More in general, we want the following two properties hold for any logic for contracts. They say that the promise in a contract can be arbitrarily weakened (9), while the precondition can be arbitrarily strengthened (10).

$$\vdash (p \twoheadrightarrow q) \ \wedge \ (q \rightarrow q') \ \rightarrow \ (p \twoheadrightarrow q') \tag{9}$$
$$\vdash (p' \rightarrow p) \ \wedge \ (p \twoheadrightarrow q) \ \rightarrow \ (p' \twoheadrightarrow q) \tag{10}$$

Note that (8), (9), (10) cover three of the four possible cases of transitivity which mix standard and contractual implications. The fourth case would, instead, make the two forms of implication coincide, so it is *not* a desirable property.

Another desirable property is that, if a promise q is already true, then it is also true any contract which promises q:

$$\vdash q \ \rightarrow \ (p \twoheadrightarrow q) \tag{11}$$

Of course, we do not want the converse to hold: a contract not always implies its promise: $\nvdash (p \twoheadrightarrow q) \rightarrow q$.

Syntax. The syntax of PCL extends that of IPC. It includes the standard connectives $\neg, \wedge, \vee, \rightarrow$ and the contractual implication $\twoheadrightarrow$. We assume a denumerable set $\{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}, \ldots\}$ of prime (atomic) formulae. PCL formulae are denoted with the letters $p, q, r, s, \ldots$ (note that the font differs from that used for prime formulae). The precedence of IPC operators is, from highest to lowest: $\neg, \wedge, \vee, \rightarrow$. We stipulate that $\twoheadrightarrow$ has the same precedence as $\rightarrow$.

Definition 1 *The formulae of* PCL *are defined as:*

$$p \ ::= \ \bot \ \mid \ \top \ \mid \ \mathsf{p} \ \mid \ \neg p \ \mid \ p \vee p \ \mid \ p \wedge p \ \mid \ p \rightarrow p \ \mid \ p \twoheadrightarrow p$$

We let $p \leftrightarrow q$ be syntactic sugar for $(p \rightarrow q) \wedge (q \rightarrow p)$.

We now present an Hilbert-style axiomatization for PCL.

Definition 2 *The proof system of* PCL *comprises all the axioms of IPC, the Modus Ponens rule* CUT*, and the axioms:*

$\top \twoheadrightarrow \top$	[ZERO]
$(p \twoheadrightarrow p) \rightarrow p$	[FIX]
$(p' \rightarrow p) \rightarrow (p \twoheadrightarrow q) \rightarrow (q \rightarrow q') \rightarrow (p' \twoheadrightarrow q')$	[PREPOST]

The above axioms are a subset of the properties discussed above. The axiom ZERO is a subcase of (7), FIX is just (3), while the axiom PREPOST combines (9) and (10). As expected, this set of axioms is actually sound and complete w.r.t. all the properties marked above as desirable.

Lemma 1 *Properties (1-11) are theorems of* PCL*. Also:*

$$\vdash (p \twoheadrightarrow q) \wedge (q \twoheadrightarrow r) \rightarrow (p \twoheadrightarrow (q \wedge r))$$
$$\vdash (p \twoheadrightarrow (q \wedge r)) \rightarrow (p \twoheadrightarrow q) \wedge (p \twoheadrightarrow r)$$
$$\vdash (p \twoheadrightarrow q) \vee (p \twoheadrightarrow r) \rightarrow (p \twoheadrightarrow (q \vee r))$$
$$\vdash (p \twoheadrightarrow q) \rightarrow ((q \rightarrow p) \rightarrow q)$$

We present below some of the most significant results about our logic. For a more comprehensive account, including detailed proofs of all our results, see [6].

First, PCL is consistent. Also, negation-free formulae do not lead to inconsistencies.

Theorem 1 PCL *is consistent, i.e.* $\not\vdash \bot$. *Also, if* p *is free from* $\{\bot, \neg\}$, *then* $\not\vdash p \to \bot$.

As expected, the following are *not* tautologies of PCL :

$$\not\vdash (p \to q) \to (p \twoheadrightarrow q) \qquad \not\vdash (p \twoheadrightarrow q) \to q$$
$$\not\vdash \bot \twoheadrightarrow p \qquad \not\vdash ((q \to p) \to q) \to (p \twoheadrightarrow q)$$

Note that if we augment our logic with the axiom of excluded middle, then $(p \twoheadrightarrow q) \leftrightarrow q$ becomes a theorem. This would make contractual implication coincide with right projection, so losing most of the intuition behind mutual agreements. For this reason we use IPC, instead of classical logic, as the basis of PCL .

Another main result about PCL is its decidability. To prove that, we have devised a Gentzen-style sequent calculus, equivalent to the Hilbert-style axiomatisation. In particular, we have extended the sequent calculus for IPC presented in [33] with rules for the contractual implication $\twoheadrightarrow$. Note that PREPOST introduces $\twoheadrightarrow$ on the right, and eliminates it on the left (similarly e.g. to $\bigcirc L$ of lax logic [19]).

Definition 3 *The sequent calculus of* PCL *includes all the rules for IPC [6], and the following additional rules.*

$$\text{[ZERO]}\ \frac{\Gamma \vdash q}{\Gamma \vdash p \twoheadrightarrow q} \qquad \text{[FIX]}\ \frac{\Gamma, p \twoheadrightarrow q, r \vdash p \quad \Gamma, p \twoheadrightarrow q, q \vdash r}{\Gamma, p \twoheadrightarrow q \vdash r} \qquad \text{[PREPOST]}\ \frac{\Gamma, p \twoheadrightarrow q, a \vdash p \quad \Gamma, p \twoheadrightarrow q, q \vdash b}{\Gamma, p \twoheadrightarrow q \vdash a \twoheadrightarrow b}$$

We now establish the equivalence between the two logical systems of PCL. In the following theorem, we denote with $\vdash_H$ provability in the Hilbert-style system, while $\vdash_G$ is used for Gentzen-style provability.

Theorem 2 *For all* PCL *formulae* p: $\vdash_H p \iff \emptyset \vdash_G p$.

Our sequent calculus enjoys cut elimination. The proof is non-trivial, since the rules for $\twoheadrightarrow$ are not dual, unlike e.g. left/right rules for $\wedge$. Nevertheless, the structural approach of [33] can be adapted. A cut on a formula p is replaced by cuts on strict subformulae of p, and cuts on p having a shorter proof tree. Some insightful cases of our proof, as well as the induction metric used, are in Appendix A; the full details are in [6].

Theorem 3 (Cut Elimination) *If* p *is provable in* PCL, *then there exists a proof of* p *not using the* CUT *rule.*

The subformula property holds in PCL. Cut-free proofs only involve subformulae of the sequent at hand.

Theorem 4 (Subformula Property) *Let* D *be a cut-free proof of* $\Gamma \vdash p$. *Then, the formulae occurring in* D *are subformulae of those occurring in* Γ *and* p.

Theorems 3 and 4 allow for exhaustively searching the proof space, so implying decidability.

Theorem 5 *The logic* PCL *is decidable.*

As a further support to our logic, we have implemented a proof search algorithm [37], which decides whether any given formula is a tautology or not. Despite the problem being PSPACE complete [36], the performance of our tool is acceptable for the examples presented in this paper.

We now establish some expressiveness results, relating PCL and IPC. More in detail, we consider whether sound and complete homomorphic encodings exist, i.e. whether $\twoheadrightarrow$ can be regarded as syntactic sugar for some IPC context.

Definition 4 *A homomorphic encoding* m *is a function from PCL formulae to IPC formulae such that:* m *is the identity on prime formulas,* $\top$, *and* $\bot$; *it acts homomorphically on* $\wedge, \vee, \to, \neg$; *it satisfies* $m(p \twoheadrightarrow q) = \mathcal{C}[m(p), m(q)]$ *for some fixed IPC context* $\mathcal{C}(\bullet, \bullet)$.

Of course, each homomorphic encoding is uniquely determined by the context $\mathcal{C}$. Several *complete* encodings exist:

Lemma 2 *The following homomorphic encodings are complete, i.e. they satisfy* $\vdash p \implies \vdash_{IPC} m_i(p)$. *Also, they are pairwise non-equivalent in IPC (the prime* a *is arbitrary).*

$$m_0(p \twoheadrightarrow q) = m_0(q)$$
$$m_1(p \twoheadrightarrow q) = (m_1(q) \to m_1(p)) \to m_1(q)$$
$$m_2(p \twoheadrightarrow q) = \neg\neg(m_2(q) \to m_2(p)) \to m_2(q)$$
$$m_3(p \twoheadrightarrow q) = \neg(m_3(q) \to m_3(p)) \vee m_3(q)$$
$$m_4(p \twoheadrightarrow q) = ((m_4(q) \to m_4(p)) \vee \mathsf{a}) \to m_4(q)$$

However, there can be no *sound* encodings, so $\twoheadrightarrow$ is not just syntactic sugar. Indeed, a sound encoding would allow us to derive Peirce's axiom in PCL, violating the fact that PCL conservatively extends IPC [6].

Theorem 6 *If* m *is a homomorphic encoding of* PCL *into IPC, then* m *is not sound, i.e. there exists a* PCL *formula* p *such that* $\vdash_{IPC} m(p)$ *and* $\not\vdash p$.

In [6] we have proved further properties of PCL, including some relations between PCL and IPC, the modal logic S4, and propositional lax logic. Also, we have explored there further application scenarios for our logic.

Example 1 (Online sale) *We describe a possible online sale between two parties. To buy an item, the buyer contacts the bank, to reserve from his account a given amount of money for the transaction. When this happens (modelled with the prime formula* lock*), that amount is no longer available. Then, the buyer makes an offer to the seller (*offer*). When provided with a good offer, and the money has been reserved, the seller will send the item (*send*). Otherwise, she cancels the transaction (*abort*). When the transaction is aborted, the bank cancels the reservation (*unlock*), so the money can be reused.*

Formally, the buyer agrees to $\mathsf{lock} \wedge \mathsf{offer}$*, provided that either the item is sent, or the reservation is cancelled. The seller agrees to evaluate the offer. The bank agrees to cancel the reservation when the transaction is aborted.*

$$Buyer = (\mathsf{send} \vee \mathsf{unlock}) \twoheadrightarrow (\mathsf{lock} \wedge \mathsf{offer})$$
$$Seller = \mathsf{offer} \twoheadrightarrow ((\mathsf{lock} \rightarrow \mathsf{send}) \vee \mathsf{abort})$$
$$Bank = (\mathsf{lock} \wedge \mathsf{abort}) \twoheadrightarrow \mathsf{unlock}$$

Under these assumptions, either the item is sent, or the transaction is aborted and the reservation cancelled:

$$\vdash (Buyer \wedge Seller \wedge Bank) \rightarrow (\mathsf{send} \vee (\mathsf{abort} \wedge \mathsf{unlock}))$$

Example 2 (Dining retailers) *Around a table,* n *cutlery retailers are about to have dinner. At the center of the table, there is a large dish of food. Despite the food being delicious, the retailers cannot start eating right now. To do that, and follow the proper etiquette, each retailer needs a complete cutlery set, consisting of* n *pieces of different kinds. Each of the* n *retailers owns a distinct set of* n *piece of cutlery, all of the same kind. The retailers start discussing about trading their cutlery, so that they can finally eat.*

We formalize this scenario as follows. We number the retailers $r_1, \ldots, r_n$ *together with the kinds of pieces of cutlery, so that* r_i *initially owns* n *pieces of kind* i*. We then write* $\mathsf{g}_{i,j}$ *for "*r_i gives *a piece (of kind* i*) to* r_j*". Since retailers can use their own cutlery, we assume* $\mathsf{g}_{i,i}$ *to be true. Retailer* r_i *can start eating whenever* $e_i = \bigwedge_j \mathsf{g}_{j,i}$*. Instead, he provides the cutlery to others whenever* $p_i = \bigwedge_j \mathsf{g}_{i,j}$*.*

Suppose that r_1 *commits to a simple exchange with* r_2*: they commit to* $\mathsf{g}_{2,1} \twoheadrightarrow \mathsf{g}_{1,2}$ *and* $\mathsf{g}_{1,2} \twoheadrightarrow \mathsf{g}_{2,1}$*, and the exchange takes place since* $\mathsf{g}_{2,1} \wedge \mathsf{g}_{1,2}$ *can be derived. While this seems a fair deal, it actually exposes* r_1 *to a risk: if* $r_3, \ldots, r_n$ *perform a similar exchange with* r_2*, then* $\mathsf{g}_{2,i} \wedge \mathsf{g}_{i,2}$ *for all* i*. In particular,* $\mathsf{g}_{i,2}$ *holds for all* i*, so* r_2 *can start eating. This is however not necessarily the case for* r_1*, since* r_3 *has not committed to any exchange with* r_1*.*

A wise retailer would then never agree to a simple exchange $\mathsf{g}_{2,1} \twoheadrightarrow \mathsf{g}_{1,2}$*. Instead, the retailer* r_1 *could commit to the following safer contract:*

$$e_1 \twoheadrightarrow p_1 = \mathsf{g}_{1,1} \wedge \mathsf{g}_{2,1} \wedge \cdots \wedge \mathsf{g}_{n,1} \twoheadrightarrow \mathsf{g}_{1,1} \wedge \mathsf{g}_{1,2} \wedge \cdots \wedge \mathsf{g}_{1,n}$$

The idea is simple: r_1 *requires each piece of cutlery, i.e.* r_1 *requires to be able to start eating (*e_1*). When this happens,* r_1 *agrees to provide each other retailer with a piece of his cutlery (*p_1*). If each retailer* r_i *commits to the analogous contract, we have the desired agreement (proof in [6]).*

$$\vdash \textstyle\bigwedge_i (e_i \twoheadrightarrow p_i) \rightarrow \bigwedge_i e_i$$

Principals. As illustrated by the examples above, PCL allows for inferring whether some promise is implied by a set of contracts. In real-world scenarios, each contract will be issued by a given *principal*. It would then be useful to represent the binding between principals and contracts within the logic. This would allow, for instance, to single out the principal who is responsible for a violation, and possibly to take countermeasures against him. To this aim, we extend our logic with a $says$ modality, similarly to [21].

Definition 5 *The syntax of* PCL^{says} *extends that of* PCL *(Def. 1) with the construct* $a \ says \ p$*, where we assume a set of (atomic)* principals*, ranged over by* $a, b, \ldots$

The formula $a \ says \ p$ represents the fact that the principal a has issued a contract p. We now develop the proof theory of PCL^{says}. Essentially, we extend the PCL axioms with those of the logic ICL [21]. This is an indexed lax logic, where the lax modality corresponds to $says$. Remarkably, this extension preserves all the main results of PCL, in particular its decidability.

Definition 6 *The Hilbert-style axiomatisation of* PCL^{says} *extends that of* PCL *(Def. 2) with the following axioms:*

$$p \rightarrow (a \ says \ p) \qquad [\textsc{SaysR}]$$
$$(a \ says \ a \ says \ p) \rightarrow a \ says \ p \qquad [\textsc{SaysM}]$$
$$(p \rightarrow q) \rightarrow (a \ says \ p) \rightarrow (a \ says \ q) \qquad [\textsc{SaysF}]$$

Example 3 *We now make explicit the binding between the duties and the principals of the toy exchange example of Sect. 1. Alice commits herself to lend her airplane, provided that Bob commits himself to lend his bike (and* vice versa*).*

$$p_{toy} \ = \ Alice \ says \ ((Bob \ says \ \mathsf{b}) \twoheadrightarrow \mathsf{a}) \wedge Bob \ says \ ((Alice \ says \ \mathsf{a}) \twoheadrightarrow \mathsf{b})$$

Such contract implies the expected duties:

$$\vdash p_{toy} \ \rightarrow \ Alice \ says \ \mathsf{a} \ \wedge \ Bob \ says \ \mathsf{b}$$

The above duties can be exploited by a third party (a sort of "automated" judge) which has to investigate the responsibilities of the involved parties, in the unfortunate case that the contract is not respected. For instance, if our judge is given the evidence that Alice's airplane has never been lent to Bob, then he will infer that Alice has not respected her contract (and possibly punish her), that is:

$$p_{toy} \wedge \neg \mathsf{a} \ \rightarrow \ (Alice \ says \ \mathsf{a}) \wedge \neg \mathsf{a} \ \rightarrow \ Alice \ says \ \bot$$

In Ex. 6 below, we shall see how the notion of principals is exploited in our calculus of contracting processes.

Explicitly representing principals has some additional benefits, especially when putting our logic at work in insecure environments populated by attackers. Actually, an attacker could maliciously issue a "fake" contract, where he

makes a promise that he cannot actually implement, e.g. because the promised task can only be performed by another party. By binding each contract with its principal, it is easy to realize when someone has attempted such a fraud, because the principal who has signed the contract is different from who is due to implement the promised behaviour.

Back to our technical development, all the main results of PCL are also enjoyed by PCL^{says} (see [6] for details).

Theorem 7 *If p is provable in* PCL^{says}*, then there exists a proof of p without* CUT*. Also,* PCL^{says} *is decidable.*

3 A contract calculus

We now define our calculus. We use a denumerable set of *names* $n, m, \ldots$ and a denumerable set of *variables* $x, y, \ldots$ Metavariables a, b range over both names and variables. Intuitively, names play the same role as in the π-calculus, while variables roughly behave as names in the fusion calculus [32]. Distinct names represent distinct concrete objects, each one with its own identity. Instead, distinct variables can be *fused*, by instantiating them to the same name. Unlike [32], our calculus can fuse a variable only once.

Syntax. We extend prime formulae of PCL with names and variables as parameters: $\mathsf{p}(a, b)$. Note that we do not introduce quantifiers in PCL, which then remains a propositional logic. Indeed, the formula $\mathsf{p}(a, b)$ is still atomic from the point of view of the logic. Similarly, names and variables can appear as principals in $a \ says \ p$. Intuitively, here names model known principals, while variables still-unknown ones (see e.g. Ex. 6). We let letters c, d range over formulae, while letters u, v range over $\{\bot, \neg\}$-free formulae. The syntax of our contract calculus follows.

$$\pi ::= \tau \mid \mathsf{tell}\, u \mid \mathsf{check}\, c \mid \mathsf{ask}_{\vec{x}}\, c \mid \mathsf{fuse}_x\, c \qquad \textit{(prefixes)}$$
$$P ::= u \mid \textstyle\sum_{i\in I} \pi_i.P_i \mid P|P \mid (a)P \mid X(\vec{a}) \qquad \textit{(processes)}$$

Processes are mostly standard, and include active constraints u, sum of guarded processes $\sum_i \pi_i.P_i$, parallel composition $P|P$, scope delimitation $(a)P$. We use a set of definitions $\{X_i(\vec{x}) \doteq P_i\}_i$ with the provision that each occurrence of X_j in P_k is guarded, i.e. behind some prefix.

We write 0 for the empty sum, and we use $+$ to merge sums, that is: $\sum_{i\in I} \pi_i.P_i + \sum_{i\in J} \pi_i.P_i = \sum_{i\in I\cup J} \pi_i.P_i$ when $I \cap J = \emptyset$. Singleton sums are simply written $\pi.P$.

The prefix τ is the standard silent operation of CCS. The prefixes tell and check are those of Concurrent Constraints (CC) [35]. A $\mathsf{tell}\, u$ augments the context with the (negation-free) formula u. By Theorem 1, the context will always be consistent. A $\mathsf{check}\, c$ checks if c is consistent with the context. The prefix $\mathsf{ask}_{\vec{x}}\, c$ generalizes the prefix $\mathsf{ask}\, c$ from CC: actually, they are equivalent when $\vec{x} = \emptyset$. An $\mathsf{ask}_{\vec{x}}\, c$ stops a process until formula c can be deduced from the context. To make that happen, variables $\vec{x}$ are suitably instantiated to names. The prefix $\mathsf{fuse}_x\, c$ is peculiar of our contract calculus, and drives the fusion of variables. Like ask, it stops a process until instantiating some variables to names makes c deducible. Note that the set of such variables is not specified in the prefix (as it was for $\mathsf{ask}_{\vec{x}}$); instead, it is inferred from the context according to the *local minimal fusion* policy, to be introduced in Def. 7. The variable x in $\mathsf{fuse}_x\, c$ is instantiated to a fresh name, that intuitively represents a fresh session identifier. Indeed, the intended use of a $\mathsf{fuse}_x\, c$ is to initiate a new session, by accepting a contract which implies c. To do that, x is replaced by a fresh session ID, and some variables in the context are possibly instantiated to names (e.g. to bind unknown principals to actual ones). Instead, an ask can be used to join an already initiated session (no fresh ID is then generated).

Free variables and names, as usual, are those not under a delimitation. Alpha conversion and substitutions are defined accordingly. As a special case, when a variable x is instantiated to a name, the prefix $\mathsf{fuse}_x\, c$ behaves as a plain $\mathsf{ask}\, c$, i.e. $(\mathsf{fuse}_x\, c)\{^n/_x\} = \mathsf{ask}\,(c\{^n/_x\})$. Also, $(\mathsf{ask}_{x\vec{y}}\, c)\{^n/_x\} = \mathsf{ask}_{\vec{y}}\,(c\{^n/_x\})$. Henceforth, we consider processes up-to alpha conversion.

Semantics. We now define the semantics of processes through a two-layered transition system (Fig. 3). The bottom layer is an LTS $\xrightarrow{\alpha}$ between processes, which provides a compositional semantics. Actions α are as follows, where C denotes a set of PCL formulae.

$$\alpha ::= \tau \mid C \mid C \vdash^A_{\vec{x}} c \mid C \vdash^F_x c \mid C \nvdash \bot \mid (a)\alpha \qquad \textit{(actions)}$$

The action τ represents an internal move. The action C is an advertisement of a set of active constraints. The action $C \vdash^A_{\vec{x}} c$ is a *tentative* action, generated by a process attempting to fire an $\mathsf{ask}_{\vec{x}}\, c$ prefix. This action also carries a set C as the collection of active constraints discovered so far. Similarly for $C \vdash^F_x c$ and $\mathsf{fuse}_x\, c$, as well as for $C \nvdash \bot$ and $\mathsf{check}\, c$. In the last case, C also includes c. The delimitation in $(a)\alpha$ is for scope extrusion, as in the labelled semantics of the π-calculus. We write $(\vec{a})\alpha$ for a set of distinct delimitations, neglecting their order, e.g. $(ab) = (ba)$. We simply write $(\vec{a}\vec{b})$ for $(\vec{a} \cup \vec{b})$.

The first two lines of Fig. 3 handle the base cases of our semantics. The rule TAU allows a τ prefix to fire. The rules ASK, FUSE and CHECK simply generate the corresponding actions. The rule TELL adds a constraint to the environment, thus making it *active*. Active constraints can then signal their presence through the CONSTR rule: each constraint generates its own singleton. A sum of guarded processes $\sum \pi.P$ can instead signal that it is *not* a constraint, by generating the empty set of constraints through IDLESUM.

The rules SUM, DEF (in the third line) are quite standard: they handle external choice and (possibly recursive) defini-

$$\tau.P \xrightarrow{\tau} P \quad [\textsc{Tau}] \qquad \mathsf{ask}_{\vec{x}}\, c.P \xrightarrow{\emptyset \vdash^A_{\vec{x}} c} P \quad [\textsc{Ask}] \qquad \mathsf{fuse}_x\, c.P \xrightarrow{\emptyset \vdash^F_x c} P \quad [\textsc{Fuse}] \qquad \mathsf{check}\, c.P \xrightarrow{\{c\} \not\vdash \bot} P \quad [\textsc{Check}]$$

$$\mathsf{tell}\, u.P \xrightarrow{\tau} u|P \quad [\textsc{Tell}] \qquad u \xrightarrow{\{u\}} u \quad [\textsc{Constr}] \qquad \textstyle\sum_i \pi_i.P_i \xrightarrow{\emptyset} \sum_i \pi_i.P_i \quad [\textsc{IdleSum}]$$

$$\frac{\pi_j.P_j \xrightarrow{\alpha} P'}{\sum_i \pi_i.P_i \xrightarrow{\alpha} P'} \ [\textsc{Sum}] \qquad \frac{P\{\vec{a}/\vec{x}\} \xrightarrow{\alpha} P'}{X(\vec{a}) \xrightarrow{\alpha} P'} \text{ if } X(\vec{x}) \doteq P \ [\textsc{Def}] \qquad \frac{P \xrightarrow{\alpha} P'}{(a)P \xrightarrow{\alpha} (a)P'} \text{ if } a \notin \alpha \ [\textsc{Del}] \qquad \frac{P \xrightarrow{\alpha} P'}{(a)P \xrightarrow{(a)\alpha} P'} \ [\textsc{Open}]$$

$$\frac{P \xrightarrow{(\vec{a})C} P' \quad Q \xrightarrow{(\vec{b})C'} Q'}{P|Q \xrightarrow{(\vec{a}\vec{b})(C \cup C')} P'|Q'} \ \dagger \ [\textsc{ParConstr}] \qquad \frac{P \xrightarrow{(\vec{a})C} P' \quad Q \xrightarrow{(\vec{b})(C' \not\vdash \bot)} Q'}{P|Q \xrightarrow{(\vec{a}\vec{b})(C \cup C' \not\vdash \bot)} P'|Q'} \ \dagger \ [\textsc{ParCheck}] \qquad \frac{P \xrightarrow{\tau} P'}{P|Q' \xrightarrow{\tau} P'|Q'} \ \dagger \ [\textsc{ParTau}]$$

$$\frac{P \xrightarrow{(\vec{a})C} P' \quad Q \xrightarrow{(\vec{b})(C' \vdash^A_{\vec{x}} c)} Q'}{P|Q \xrightarrow{(\vec{a}\vec{b})(C \cup C' \vdash^A_{\vec{x}} c)} P'|Q'} \ \dagger \ [\textsc{ParAsk}] \qquad \frac{P \xrightarrow{(\vec{a})C} P' \quad Q \xrightarrow{(\vec{b})(C' \vdash^F_x c)} Q'}{P|Q \xrightarrow{(\vec{a}\vec{b})(C \cup C' \vdash^F_x c)} P'|Q'} \ \dagger \ [\textsc{ParFuse}] \qquad (\dagger) \ \begin{array}{l} \vec{a} \text{ fresh in } \vec{b}, C', c, x, \vec{x}, Q' \\ \vec{b} \text{ fresh in } C, P' \end{array}$$

$$\frac{P \xrightarrow{(\vec{x}\vec{n}\vec{a})(C \vdash^A_{\vec{x}} c)} P'}{P \xrightarrow{\tau} (\vec{n}\vec{a})P'\sigma} \text{ if } \begin{array}{l} C\sigma \vdash c\sigma \\ \sigma(\vec{x}) \subseteq \vec{n} \end{array} \ [\textsc{CloseAsk}] \qquad \frac{P \xrightarrow{(x\vec{y}\vec{m}\vec{a})(C \vdash^F_x c)} P'}{P \xrightarrow{\tau} (n\vec{m}\vec{a})P'\sigma} \text{ if } \begin{array}{l} C \vdash^\sigma c \\ \sigma(x) = n \text{ fresh} \\ \sigma(\vec{y}) \subseteq n\vec{m} \end{array} \ [\textsc{CloseFuse}]$$

$$\frac{P \xrightarrow{\tau} P'}{P \rightarrowtail P'} \ [\textsc{TopTau}] \qquad \frac{P \xrightarrow{(\vec{a})(C \not\vdash \bot)} P'}{P \rightarrowtail (\vec{a})P'} \text{ if } C \not\vdash \bot \ [\textsc{TopCheck}]$$

Figure 1. The transition system for the contract calculus (symmetric rules for $|$ are omitted).

tions, respectively. The rules DEL, OPEN handle delimitation. As usual, when a is not mentioned in an action, we can propagate the action across a delimitation (a) using DEL. The rule OPEN instead allows for scope extrusion. Note that OPEN has no side conditions: the checks needed for scope extrusion are handled by the PAR* rules.

The next two lines of rules handle parallel composition. The rule PARCONSTR merges the sets of constraints advertised by two parallel processes. The rule PARASK allows for augmenting the constraints C in the tentative action $C \vdash^A_{\vec{x}} c$ generated by an ASK, by also accounting for the set of constraints advertised by the parallel process. Similarly for the rules PARFUSE and PARCHECK. The rule PARTAU simply propagates τ actions of parallel processes. The PAR* rules also merge the set of delimitations; variable and name captures are avoided through the side condition $(\dagger)$.

The CLOSE* rules are the crucial ones, since they provide the mechanism to finalize the actions generated by ask and fuse. The rule CLOSEASK instantiates the variables $\vec{x}$ (originally in a prefix $\mathsf{ask}_{\vec{x}}\, c$) to a subset of the names $\vec{n}$ collected so far. This is done through a substitution σ. When $c\sigma$ is deducible under the constraint set $C\sigma$, a silent action τ is generated, the delimitation $(\vec{n}\vec{a})$ is brought back to the process level, and σ is applied to the residual process P'. Note that the restriction $(\vec{x})$ is no longer needed, as all the variables in $\vec{x}$ have been instantiated by σ. The rule CLOSEFUSE instantiates a subset $\vec{y}$ of the variables collected in the action to a set of names. These names comprise a subset $\vec{m}$ of the collected ones, plus a *fresh* name n. The variable x (originally in a prefix $\mathsf{fuse}_x\, c$) is instantiated to n. A transition is then possible when the substitution σ, defining the instantiation, makes the formula c deducible from the constraint set C under the *local minimal fusion* relation $\vdash^\sigma$, to be defined in a while (Def. 7). When this happens, a silent action τ is generated, the delimitation $(\vec{n}\vec{m}\vec{a})$ is brought back to the process level, and σ is applied to the residual process P'. Apart from generating a fresh name, another key difference w.r.t. CLOSEASK is that the set of variables to be instantiated is chosen rather arbitrarily by CLOSEFUSE, while it is hard-wired in a prefix in CLOSEASK. This is the reason why CLOSEFUSE checks local minimal fusion, and not just deducibility, to further restrict the choice of σ.

Summing up, the LTS given by $\xrightarrow{\alpha}$ allows for the generation of tentative actions for ask and fuse (rules ASK,FUSE), which can then be converted to $\xrightarrow{\tau}$ whenever enough constraints are discovered (rules CLOSE*). Then, the τ action can be propagated towards the top level (rule PARTAU). A prefix check c, instead, cannot be handled in the same fashion, since it requires to check the consistency of c with respect to *all* the active constraints. To this aim, we use the reduction relation $\rightarrowtail$, layered over the $\xrightarrow{\alpha}$ relation. The reduction $\rightarrowtail$ only includes internal moves $\xrightarrow{\tau}$ (rule TOPTAU) and successful check moves (TOPCHECK). This effectively discards tentative actions, filtering the unsuccessful ones.

Definition 7 (Local Minimal Fusion) *We write $C \vdash^\sigma c$ iff:*

$$\exists C' \subseteq C \ : \ (C'\sigma \vdash c\sigma \ \wedge \ \nexists \sigma' \subset \sigma \ : \ C'\sigma' \vdash c\sigma')$$

A local minimal fusion requires a subset C' of C (*locality* restriction) such that C' entails c whenever σ is applied. Also, we want σ to instantiate only those variables actually involved in the entailment of c – i.e. we require that no restriction σ' of σ suffices for the entailment.

To understand the motivations underlying the locality restriction, note that a substitution may be minimal w.r.t. a set of constraints C', yet not minimal for a superset

$C \supset C'$, as the following example shows. Let $c = \mathsf{p}(x)$, $C' = \{\mathsf{q}(y), \mathsf{q}(z) \vee \mathsf{s} \rightarrow \mathsf{p}(y)\}$ and $C = C' \cup \{\mathsf{s}\}$. To obtain $C' \vdash c$, all the variables x, y, z must be instantiated to the same name. So, this fusion (call it σ_1) is minimal. Instead, to obtain $C \vdash c$, we can use another substitution σ_2, which fuses x with y and neglects z, because the premise $\mathsf{q}(z) \vee \mathsf{s}$ can now be discharged through s. So, in this case σ_1 is *not* minimal, since $\sigma_2 \subset \sigma_1$. This phenomenon could, in principle, lead to unexpected behaviour. For instance, let:

$$P = (x)(y)(z)(\mathsf{fuse}_x\, c.R \mid R') \mid \mathsf{s}$$
$$Q = (x)(y)(z)(\mathsf{fuse}_x\, c.R \mid R' \mid \mathsf{s})$$

where R' is the parallel composition of the constraints C'. Let us drop for a while the locality restriction. Then, in P two minimal fusions would be applicable, depending on CLOSEFUSE being used at the top-level (σ_2) or not (σ_1). Instead, in Q only σ_2 would be possible. This clearly clashes with our intuition that P and Q should be equivalent, since Q is obtained from P through a scope extrusion. The locality restriction allows for recovering such equality. Indeed, σ_1 is minimal for $C' \subseteq C$, so it is also applicable for Q.

The consequence of locality is that inspecting any set of (locally) known contracts is enough to decide if a set of contracts leads to an agreement – it is not necessary to (globally) explore the whole system. To a local observer, fusing x, y, z may appear minimal. To a more informed observer, the same fusion may appear non-minimal (the minimal one fusing x, y). Both fusion are allowed by our semantics.

Example 4 (Handshaking) *Recall from Sect. 1:*

$$Alice = (x)\ \big(\mathsf{tell}\,\mathsf{b}(x) \twoheadrightarrow \mathsf{a}(x).\ \mathsf{fuse}_x\,\mathsf{a}(x).\ lendAirplane\big)$$
$$Bob = (y)\ \big(\mathsf{tell}\,\mathsf{a}(y) \twoheadrightarrow \mathsf{b}(y).\ \mathsf{fuse}_y\,\mathsf{b}(y).\ lendBike\big)$$

A possible trace of $P = Alice \mid Bob$ is the following:

$$\begin{aligned} P \xrightarrow{\tau}{}^{2}\ & (x)\ \big(\mathsf{b}(x) \twoheadrightarrow \mathsf{a}(x) \mid \mathsf{fuse}_x\,\mathsf{a}(x).\ lendAirplane\big) \mid \\ & (y)\ \big(\mathsf{a}(y) \twoheadrightarrow \mathsf{b}(y) \mid \mathsf{fuse}_y\,\mathsf{b}(y).\ lendBike\big) \\ \xrightarrow{\tau}\ & (n)\ \big(\mathsf{b}(n) \twoheadrightarrow \mathsf{a}(n) \mid lendAirplane\{^n/_x\} \mid \\ & \qquad \mathsf{a}(n) \twoheadrightarrow \mathsf{b}(n) \mid \mathsf{ask}\,\mathsf{b}(n).\ lendBike\{^n/_y\}\big) \\ \xrightarrow{\tau}\ & (n)\ \big(\mathsf{b}(n) \twoheadrightarrow \mathsf{a}(n) \mid lendAirplane\{^n/_x\} \mid \\ & \qquad \mathsf{a}(n) \twoheadrightarrow \mathsf{b}(n) \mid lendBike\{^n/_y\}\big) \end{aligned}$$

In the first two steps we fire the prefixes $\mathsf{tell}\,\mathsf{b}(x) \twoheadrightarrow \mathsf{a}(x)$ and $\mathsf{tell}\,\mathsf{a}(y) \twoheadrightarrow \mathsf{b}(y)$ through TELL, PARTAU, DEL. *The third step is the crucial one. The prefix $\mathsf{fuse}_x\,\mathsf{a}(x)$ is fired through* FUSE. *Through* CONSTR, PARFUSE, *we discover the active constraint $c_a = \mathsf{b}(x) \twoheadrightarrow \mathsf{a}(x)$. We then use* OPEN *to obtain the action $(x)\{c_a\} \vdash^F_x \mathsf{a}(x)$ for Alice. For Bob, we use* CONSTR *to discover $c_b = \mathsf{a}(y) \twoheadrightarrow \mathsf{b}(y)$, which we merge with the empty set of constraints obtained through* IDLESUM*; we then use* OPEN *to get $(y)\{c_b\}$. At the top level, we then apply* PARFUSE *to deduce $(x\,y)\{c_a, c_b\} \vdash^F_x \mathsf{a}(x)$. Finally,* CLOSEFUSE *fuses x and y to the fresh name n. Let $\sigma = \{^n/_x, {}^n/_y\}$ be such fusion. It is easy to check that σ is a local minimal fusion, so $\{c_a, c_b\} \vdash^\sigma \mathsf{a}(x)$. The instantiation transforms $\mathsf{fuse}_y\,\mathsf{b}(y)$ into $\mathsf{ask}\,\mathsf{b}(n)$, fired in the last step.*

Example 5 (Unfair handshaking) *To get further insights on the role played by contractual implication, consider an alternative handshaking, which makes NO use of $\twoheadrightarrow$.*

$$Alice' = (x)\ \big(\mathsf{tell}\,\mathsf{a}(x).\ \mathsf{fuse}_x\,\mathsf{b}(x).\ lendAirplane\big)$$
$$Bob' = (y)\ \big(\mathsf{tell}\,\mathsf{b}(y).\ \mathsf{fuse}_y\,\mathsf{a}(y).\ lendBike\big)$$

Since the handshaking is still performed, $P' = Alice'|Bob'$ behaves as $P = Alice|Bob$ in Ex. 4. Yet, P and P' behave quite differently in the presence of a third kid. Let:

$$Carl = (z)\ \big(\mathsf{tell}\,\mathsf{c}(z).\ \mathsf{fuse}_z\,\mathsf{a}(z).\ lendCar\big)$$

The system $P' \mid Carl$ could lead to the execution of $lendCar$, since Carl can receive (through fuse*) the promise $\mathsf{a}(x)$ from $Alice'$. However, $Alice'$ will be stuck waiting for a bike, so she will never respect her promise, and Carl will never obtain the expected airplane. In the system $P \mid Carl$, instead, Carl will not lend his car in vain. His* fuse *will be stuck, because the contract of Alice now requires a bike, and Carl does not provide it.*

Example 6 (Principals) *Consider an online market, where buyers and sellers trade items. The contract of a buyer n_B is to pay for an item, provided that some (still unknown) seller x_S promises to send it; dually, the contract of a seller n_S is to send an item, provided that some buyer y_B pays.*

$$c_B = n_B\ says\ \big((x_S\ says\ \mathsf{send}(x)) \twoheadrightarrow \mathsf{pay}(x)\big)$$
$$c_S = n_S\ says\ \big((y_B\ says\ \mathsf{pay}(y)) \twoheadrightarrow \mathsf{send}(y)\big)$$

A buyer first issues her contract c_B, then waits until discovering she has to pay, and eventually proceeds with the process B'. At this point, the buyer may either refuse to pay (process $NoPay$), or actually pay the item, by issuing a $\mathsf{paid}(x)$. After the item has been paid, the buyer may wait for the item to be sent or open a dispute with the seller.

$$B = (x)(x_S)(n_B)\ \big(\mathsf{tell}\,c_B.\ \mathsf{fuse}_x\,(n_B\ says\ \mathsf{pay}(x)).\ B'\big)$$
$$B' = \tau.NoPay\ +\ \tau.\,\mathsf{tell}\,(n_B\ says\ \mathsf{paid}(x)).\ B''$$
$$B'' = \mathsf{ask}\,(x_S\ says\ \mathsf{sent}(x)) + \tau.\,\mathsf{tell}\,(n_B\ says\ \mathsf{dispute}(x))$$

The behaviour of the seller n_S is dual.

$$S = (y)(y_B)(n_S)\ \big(\mathsf{tell}\,c_S.\ \mathsf{fuse}_y\,(n_S\ says\ \mathsf{send}(y)).\ S'\big)$$
$$S' = \tau.NoSend\ +\ \tau.\,\mathsf{tell}\,(n_S\ says\ \mathsf{sent}(y)).\ S'$$
$$S'' = \mathsf{ask}\,(y_B\ says\ \mathsf{pay}(y)) + \tau.\,\mathsf{tell}\,(n_S\ says\ \mathsf{dispute}(y))$$

An handshaking is reached through CLOSEFUSE. *The fusion is $\sigma = \{^m/_x, {}^m/_y, {}^{n_S}/_{x_S}, {}^{n_B}/_{y_B}\}$, where m is a fresh name.*

To automatically resolve disputes, a judge J can enter a session initiated between a buyer and a seller, provided that

a dispute has been opened, and either the obligations pay *or* send *have been inferred. This is done through the* $\mathsf{ask}_{\vec{x}}$ *primitive, where* $\vec{x} = \{z, x_S, y_B\}$*. This binds the variable* z *to the session identifier* m*,* x_S *to the actual name of the seller (*n_S*), and* y_B *to the actual name of the buyer (*n_B*).*

$$\begin{aligned} J = (\vec{x})\big(&\mathsf{ask}_{\vec{x}}\,(y_B\ says\ \mathsf{pay}(z)\ \wedge\ x_S\ says\ \mathsf{dispute}(z)).\\ &\quad \mathsf{check}\,\neg(y_B\ says\ \mathsf{paid}(z)).\,jail(y_B)\\ &\mid \mathsf{ask}_{\vec{x}}\,(x_S\ says\ \mathsf{send}(z)\ \wedge\ y_B\ says\ \mathsf{dispute}(z)).\\ &\quad \mathsf{check}\,\neg(x_S\ says\ \mathsf{sent}(z)).\,jail(x_S)\big) \end{aligned}$$

If the obligation $\mathsf{pay}(z)$ *is found, but the item has not been actually paid then the buyer is convicted (modelled by* $jail(y_B)$*, not further detailed). Similarly, if the obligation* $\mathsf{send}(z)$ *has not been supported by a corresponding* $\mathsf{sent}(z)$*, then the seller is convicted.*

Encodings. We have studied the expressive power of our contract calculus by encoding several concurrency idioms, among which semaphores, Linda [23], the π-calculus and graph rewriting. See [5] for all the details.

4 Related Work

The complexity of real-world scenarios, where several concepts like principals, contracts, authorizations, duties, delegation, mandates, regulations, *etc.* are inextricably intermingled, have led to a steady flourishing of new logics over the years. These take inspiration and extend e.g. classical [17], modal [16], deontic [34, 22], default [24] and defeasible logics [25]. We think none of these logics, including our PCL, captures *all* the facets of contracts. Each of these logics is designed to represent some particular aspect of contracts, e.g. obligations, permissions and prohibitions in deontic logics, violation of contracts in default and defeasible logics, and agreement in our contract logic. We argue that, since these aspects are orthogonal, it is possible to extend PCL with features from some of these logics.

The motivations underlying our logic seem related to those introduced in [3] to compose assume-guarantee specifications [2]. The idea is that a system will give some guarantee M_1 about its behaviour, provided that the environment it operates within will behave according to some assumption M_2, and *vice versa*. This is rendered in [3] as the judgment $(M_1 \to M_2) \wedge (M_2 \to M_1) \vdash M_1 \wedge M_2$. However, since $\to$ is the usual intuitionistic implication, this judgment (not valid in IPC) only holds in particular models, where e.g. M_1, M_2 must be interpreted as safety properties. Our approach is different: we make the above judgement valid by using $\twoheadrightarrow$ instead of $\to$. We then develop a decidable proof theory (exploited to design our calculus of contracting processes) while abstracting from the specific models. Actually, finding sound and complete models for contracts seems to be hard, e.g. [31] shows the impossibility of devising, in some lattice-based models, a sound and complete set of compositional rules for circular assume-guarantee.

Our research seems also related to foundational research on authorization logics for distributed systems [1, 21, 30]. A crucial difference is that, while authorizations logics are focussed on deciding, given a bunch of logical authorization assertions, *if* a principal is allowed to perform some action, in our contract logic we are also concerned with discovering *what* that principal has to promise in return.

In our model of contracts we have abstracted from most of the implementation issues. For instance, in insecure environments populated by attackers, the operation of exchanging contracts requires particular care. This is related to the problem of establishing *common knowledge* in distributed systems [26]. A trusted third party might then be in order to make the contract exchange fair. We expect to apply standard techniques for guaranteeing non-repudation [28, 39], fair exchange [4], and contract signing [15, 27]. Note that our logic works at a higher abstraction level than the protocol level, by focussing on how the exchanged messages lead to an handshaking between the parties.

Contracts are modelled as processes in [12, 13], to specify the interaction behaviour of clients and services. A client contract complies with a service contract if any possible interaction will always succeed. The crux is how to define (and decide) a subcontract relation, that allows for safely substituting services without affecting the compliance with their clients. Our contracts could be seen as a declarative underspecified description of which behavioural contracts are an implementation. Behavioural contracts seem more rigid than ours, as they precisely fix the order in which the actions must be performed. Even though in some cases this may be desirable, many real-world contracts allow for a more liberal way of constraining the involved parties (e.g., "I will pay before the deadline"). While the crucial notion in [12] is *compatibility* (which results in a yes/no output), we focus on the inferring the *obligations* that arise from a set of contracts. This provides a fine-grained quantification of the reached agreement, e.g. we may identify who is responsible of a contract violation.

Negotiation and service-level agreement are dealt with in cc-pi [10], a calculus combining features from concurrent constraints and name passing. As in the π-calculus, synchronization is channel-based: it only happens between two processes sharing a name. Synchronization fuses two names, similarly to the fusion calculus and ours. A main difference between cc-pi and our calculus is that in cc-pi only two parties may simultaneously reach an agreement, while our fuse allows for simultaneous multiparty agreements. Also, in our calculus the parties involved in an agreement do not have to share a pre-agreed name. This is useful for modelling scenarios where a contract can be accepted

by any party meeting the required terms (see e.g. Ex. 6).

Our contracts could be exploited to enhance the compensation mechanism of long-running transactions [7, 8, 11]. There, a long transaction is partitioned into a sequence of smaller ones, each one associated with a *compensation*, to be run upon failures of the standard execution [20]. While in long-running transactions clients have little or no control on the compensations (they are specified by the designer), in our approach clients can use contracts to select those services offering the desired compensation. In [9], cc-pi is extended with rules for handling transactions. This gives to the client more control on the choice of compensations. The differences w.r.t. our calculus noted above still apply.

The *reputation* of the principal issuing a contract could be influential in deciding whether accepting such contract or not. Reputation systems which evaluate the past behaviour of principals, e.g. [29] can then be used to make our contracts more expressive. Time is another feature that may arise while modelling contracts. Temporal extensions of our logics, e.g. like [18], will allow to check whether a promise is violated in a given trace (e.g. the deadline passed).

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM TOPLAS*, 4(15):706–734, 1993.

[2] M. Abadi and L. Lamport. Composing specifications. *ACM TOPLAS*, 15(1):73–132, 1993.

[3] M. Abadi and G. D. Plotkin. A logical view of composition. *Theoretical Computer Science*, 114(1):3–30, 1993.

[4] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *EUROCRYPT*, 1998.

[5] M. Bartoletti and R. Zunino. A calculus of contracting processes. Technical Report DISI-09-056, Univ. Trento, 2009.

[6] M. Bartoletti and R. Zunino. A logic for contracts. Technical Report DISI-09-034, Univ. Trento, 2009. (Nov 2009 revision).

[7] L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long running transactions. In *Proc. FMOODS*, 2003.

[8] R. Bruni, H. C. Melgratti, and U. Montanari. Theoretical foundations for compensations in flow composition languages. In *Proc. POPL*, 2005.

[9] M. G. Buscemi and H. C. Melgratti. Transactional service level agreement. In *Proc. TGC*, 2007.

[10] M. G. Buscemi and U. Montanari. CC-Pi: A constraint-based language for specifying service level agreements. In *Proc. ESOP*, 2007.

[11] M. J. Butler, C. A. R. Hoare, and C. Ferreira. A trace semantics for long-running transactions. In *25 Years Communicating Sequential Processes*, 2004.

[12] G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. *ACM TOPLAS*, 31(5), 2009.

[13] G. Castagna and L. Padovani. Contracts for mobile processes. In *Proc. CONCUR*, 2009.

[14] M. Coppo and M. Dezani-Ciancaglini. Structured communications with concurrent constraints. In *Proc. TGC*, 2008.

[15] V. Cortier, R. Küsters, and B. Warinschi. A cryptographic model for branching time security properties – the case of contract signing protocols. In *Proc. ESORICS*, 2007.

[16] A. Daskalopulu and T. Maibaum. Towards electronic contract performance. In *Proc. DEXA*, 2001.

[17] H. Davulcu, M. Kifer, and I. Ramakrishnan. CTR-S: A logic for specifying contracts in semantic web services. In *Proc. WWW*, 2004.

[18] H. DeYoung, D. Garg, and F. Pfenning. An authorization logic with explicit time. In *Proc. CSF*, 2008.

[19] M. Fairtlough and M. Mendler. Propositional lax logic. *Information and Computation*, 137(1):1–33, 1997.

[20] H. Garcia-Molina and K. Salem. Sagas. In *SIGMOD Conference*, 1987.

[21] D. Garg and M. Abadi. A modal deconstruction of access control logics. In *Proc. FoSSaCS*, 2008.

[22] J. Gelati, A. Rotolo, G. Sartor, and G. Governatori. Normative autonomy and normative co-ordination: Declarative power, representation, and mandate. *Artificial Intelligence and Law*, 12(1-2):53–81, 2004.

[23] D. Gelernter. Generative communication in Linda. *ACM TOPLAS*, 7(1), 1985.

[24] G. K. Giannikis and A. Daskalopulu. The representation of e-contracts as default theories. In *New Trends in Applied Artificial Intelligence*, 2007.

[25] G. Governatori. Representing business contracts in RuleML. *Int. J. of Cooperative Information Systems*, 14(2-3), 2005.

[26] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *J. ACM*, 37(3), 1990.

[27] D. Kähler and R. Küsters. Constraint solving for contract-signing protocols. In *Proc. CONCUR*, 2005.

[28] S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of fair non-repudiation protocols. *Computer Communications*, 25, 2002.

[29] K. Krukow, M. Nielsen, and V. Sassone. A logical framework for history-based access control and reputation systems. *Journal of Computer Security*, 16(1):63–101, 2008.

[30] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM TISSEC*, 6(1):128–171, 2003.

[31] P. Maier. Compositional circular assume-guarantee rules cannot be sound and complete. In *Proc. FoSSaCS*, 2003.

[32] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. LICS*, 1998.

[33] F. Pfenning. Structural cut elimination - intuitionistic and classical logic. *Information and Computation*, 157, 2000.

[34] C. Prisacariu and G. Schneider. A formal language for electronic contracts. In *Proc. FMOODS*, 2007.

[35] V. Saraswat, P. Panangaden, and M. Rinard. Semantic foundations of concurrent constraint programming. In *Proc. POPL*, 1991.

[36] R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9, 1979.

[37] The PCL web site. *http://www.disi.unitn.it/∼zunino/PCL*.

[38] M. Wirsing et al. SENSORIA process calculi for service-oriented computing. In *Proc. TGC*, 2006.

[39] J. Zhou. *Non-repudiation in Electronic Commerce*. Artech House, 2001.

A Selected Cases for Cut elimination

Case ZERO / PREPOST

$$\dfrac{\dfrac{D0}{\dfrac{\Gamma \vdash q}{\Gamma \vdash p \twoheadrightarrow q}\textsc{Zero}} \quad \dfrac{D1 \quad \dfrac{D2}{\Gamma, p \twoheadrightarrow q, q \vdash b}}{\Gamma, p \twoheadrightarrow q \vdash a \twoheadrightarrow b}\textsc{PrePost}}{\Gamma \vdash a \twoheadrightarrow b}\textsc{cut} \quad \Longrightarrow \quad \dfrac{\dfrac{D0}{\Gamma \vdash q} \quad \dfrac{\dfrac{}{\Gamma, q \vdash q}\textsc{id}}{\Gamma, q \vdash p \twoheadrightarrow q}\textsc{Zero} \quad \dfrac{D2}{\Gamma, q, p \twoheadrightarrow q \vdash b}}{\Gamma, q \vdash b}\textsc{cut}}{\Gamma \vdash b}\textsc{cut}}{\Gamma \vdash a \twoheadrightarrow b}\textsc{Zero}$$

Case ZERO / FIX

$$\dfrac{\dfrac{D0}{\dfrac{\Gamma \vdash q}{\Gamma \vdash p \twoheadrightarrow q}\textsc{Zero}} \quad \dfrac{D1 \quad \dfrac{D2}{\Gamma, p \twoheadrightarrow q, q \vdash a}}{\Gamma, p \twoheadrightarrow q \vdash a}\textsc{Fix}}{\Gamma \vdash a}\textsc{cut} \quad \Longrightarrow \quad \dfrac{\dfrac{D0}{\Gamma \vdash q} \quad \dfrac{\dfrac{}{\Gamma, q \vdash q}\textsc{id}}{\Gamma, q \vdash p \twoheadrightarrow q}\textsc{Zero} \quad \dfrac{D2}{\Gamma, q, p \twoheadrightarrow q \vdash a}}{\Gamma, q \vdash a}\textsc{cut}}{\Gamma \vdash a}\textsc{cut}$$

Case PREPOST / FIX, assuming $(p \twoheadrightarrow q) \in \Gamma$

$$\dfrac{\dfrac{D0}{\Gamma, a \vdash p} \quad \dfrac{D1}{\Gamma, q \vdash b}}{\Gamma \vdash a \twoheadrightarrow b}\textsc{PrePost} \quad \dfrac{\dfrac{D2}{\Gamma, a \twoheadrightarrow b, r \vdash a} \quad \dfrac{D3}{\Gamma, a \twoheadrightarrow b, b \vdash r}}{\Gamma, a \twoheadrightarrow b \vdash r}\textsc{Fix}}{\Gamma(p \twoheadrightarrow q) \vdash r}\textsc{cut} \quad \Longrightarrow \quad \dfrac{\hat{D}_0 \quad \hat{D}_1}{\Gamma, r \vdash p \quad \Gamma, q \vdash r}\textsc{Fix}}{\Gamma \vdash r}$$

where $\hat{D}_0 = \dfrac{\dfrac{\dfrac{D0+}{\Gamma, r, a \vdash p} \quad \dfrac{D1+}{\Gamma, r, q \vdash b}}{\Gamma, r \vdash a \twoheadrightarrow b}\textsc{PrePost} \quad \dfrac{D2}{\Gamma, r, a \twoheadrightarrow b \vdash a}\textsc{cut}}{\Gamma, r \vdash a} \quad \dfrac{D0+}{\Gamma, r, a \vdash p}\textsc{cut}}{\Gamma, r \vdash p}\textsc{cut}$

and $\hat{D}_1 = \dfrac{\dfrac{D1}{\Gamma, q \vdash b} \quad \dfrac{\dfrac{\dfrac{D0+}{\Gamma, q, b, a \vdash p} \quad \dfrac{D1+}{\Gamma, q, b, q \vdash b}}{\Gamma, q, b \vdash a \twoheadrightarrow b}\textsc{PrePost} \quad \dfrac{D3+}{\Gamma, q, b, a \twoheadrightarrow b \vdash r}\textsc{cut}}{\Gamma, q, b \vdash r}\textsc{cut}}{\Gamma, q \vdash r}\textsc{cut}$

Case FIX / any rule, assuming $(p \twoheadrightarrow q) \in \Gamma$

$$\dfrac{\dfrac{\dfrac{D0}{\Gamma, a \vdash p} \quad \dfrac{D1}{\Gamma, q \vdash a}}{\Gamma \vdash a}\textsc{Fix} \quad \dfrac{D2}{\Gamma, a \vdash b}*}{\Gamma(p \twoheadrightarrow q) \vdash b}\textsc{cut} \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{}{\Gamma, b, p \vdash p}\textsc{id} \quad \dfrac{\dfrac{D1+}{\Gamma, b, q \vdash a} \quad \dfrac{D0+}{\Gamma, b, q, a \vdash p}}{\Gamma, b, q \vdash p}\textsc{cut}}{\Gamma, b \vdash p}\textsc{Fix} \quad \dfrac{\dfrac{D1}{\Gamma, q \vdash a} \quad \dfrac{D2+}{\Gamma, q, a \vdash b}}{\Gamma, q \vdash b}\textsc{cut}}{\Gamma \vdash b}\textsc{Fix}$$

Our cut metric is the following. We associate to each cut the weight $(p, h(D_0), h(D_1))$ where p is the cut formula, and $h(D_i)$ denotes the height of the subderivation D_i. The stuctural ordering of formulae and the usual one on naturals induce the lexicographical ordering of weights. This is a well-founded ordering, and each reduction above replaces a cut with *lighter* ones, only.