Int J Softw Tools Technol Transfer
DOI 10.1007/s10009-006-0010-1

Compositional specification of commercial contracts

Jesper Andersen · Ebbe Elsborg · Fritz Henglein · Jakob Grue Simonsen · Christian Stefansen

Abstract We present a declarative language for compositional specification of contracts governing the exchange of resources. It extends Eber and Peyton Jones's declarative language for specifying financial contracts (Jones et al. in The Fun of Programming. 2003) to the exchange of money, goods and services amongst multiple parties and complements McCarthy's Resources, Events and Agents (REA) accounting model (McCarthy in Account Rev. **LVII**(3), 554–578, 1982) with a view-independent formal contract model that supports definition of user-defined contracts, automatic monitoring under execution and user-definable analysis of their state before, during and after execution. We provide several realistic examples of commercial contracts and their analyses. A variety of (real) contracts can be expressed in such a fashion as to support their integration, management and analysis in an operational environment that registers events. The language design is driven by both domain considerations and semantic language design methods: a contract denotes a set of traces of events, each of which is an alternative way of concluding the contract successfully, which gives rise to a CSP-style (Brooker et al. in J.ACM **31**(3), 560–599, 1984; Hoare in Communicating Sequential Processes, 1985) denotational semantics. The denotational semantics drives the development of a sound and complete small-step operational semantics, where a partially executed contract is represented as a (full) contract that represents the remaining contractual commitments. This operational semantics is then systematically refined in two stages to an instrumented operational semantics that reflects the bookkeeping practice of identifying the specific contractual commitment a particular event matches at the time the event occurs, as opposed to delaying this matching until the contract is concluded.

J. Andersen · F. Henglein (✉) · J. G. Simonsen · C. Stefansen
Department of Computer Science,
University of Copenhagen (DIKU),
Universitetsparken 1,
2100 Copenhagen, Denmark
e-mail: henglein@diku.dk

J. Andersen
e-mail: jespera@diku.dk

J. G. Simonsen
e-mail: simonsen@diku.dk

C. Stefansen
e-mail: cstef@diku.dk

E. Elsborg
Institute of Theoretical Computer Science,
IT University of Copenhagen (ITU),
Rued Langgaards Vej 7,
2300 Copenhagen S, Denmark
e-mail: elsborg@itu.dk

1 Introduction

When entrepreneurs enter contractual relationships with a large number of other parties, each with possible variations on standard contracts, they are confronted with the interconnected problems of *specifying* contracts, *monitoring* their execution for performance,[1] *analyzing* their ramifications for planning, pricing and other purposes prior to and during execution and *integrating* this information with accounting, workflow management, supply chain management, production planning, tax reporting, decision support, etc.

[1] *Performance* in contract lingo refers to *compliance* with the *promises* (contractual commitments) stipulated in a contract; non-performance is also termed *breach of contract*.

1.1 Contract management and information systems

Judging by publically available information, support for contracts in most present-day enterprise resource planning (ERP) systems is delegated to *functional silos*, specialized (sub)systems supporting a fixed catalogue of predefined contracts for specific application domains, e.g., creditor/debitor modules in ERP systems such as Microsoft Business Solutions' Navision 3.60 and Axapta (http://www.navision.dk) for simple commercial contracts, SAP's specialized contract management subsystems for particular industries such as the beverage industry (http://www.sap.com) or independent systems for managing portfolios of financial contracts such as Simcorp's IT/2 system for managing treasuries (http://www.simcorp.com). Common to these systems seems to be that they support a fixed and limited set of contract templates specialized to a particular application domain and lack flexible integration with other (parts of) enterprise systems. A notable exception is LexiFi (http://www.lexifi.com) whose products for complex financial derivatives incorporate some of the ideas pioneered in Peyton Jones, Eber, and Seward's research in financial engineering [11,12].

In the absence of support for user-definable (custom) contracts users are forced to adhere to stringent business processes or end up engaging in "off-book" activities, which are not easily tracked or integrated, e.g., oral or written contracts in natural language. Furthermore, development of new specialized contract modules incurs considerable development costs with little possibility for supporting efficient division of labor in a multi-stage development model where a software *vendor* produces a solution *framework*, *partners* with domain expertise *specialize (instantiate)* the framework to particular industries and *customers* (individual companies) *configure* and *deploy* specialized systems for their *end users*.

1.2 Problems with informal contract management

Typical problems that can arise in connection with informal modeling and representation of contracts and their execution include the following:

1. Disagreement on what a contract actually requires. Many contract disputes involve a disagreement between the parties about what the contract requires, and many rules of contract law pertain to interpretation of terms of a contract that are vague or ambiguous.
2. Agreement on contract, but disagreement on what events have actually happened (event history); e.g., buyer of goods claims that payment has been made, but seller claims not to have received it ("check is in the mail" phenomenon).
3. Agreement on contract and event history, but disagreement on remaining contractual obligations; e.g., seller applies payment by buyer to one of several commitments the buyer has, but buyer intends it for another commitment.
4. Breach or malexecution of contract: a party overlooks a deadline on a commitment and is in breach of contract (missed payment deadline) or incurs losses (deadline on lucrative put or call option overlooked).
5. Entering bad or undesirable contracts/missed opportunities; e.g., a company enters a contract or refrains from doing so because it cannot quickly analyze its value and risk.
6. Coordination of contractual obligations with production planning and supply chain management; e.g., company enters into an otherwise lucrative contract, but overlooks that it does not have the requisite production capacity due to other, preexisting contractual obligations.
7. Impossibility, slowness or costliness in evaluating the state of company affairs; e.g., bad business developments are detected late, or high due diligence costs affect chances and price of selling company.

Anecdotal evidence suggests that costs associated with these problems can be considerable. Eber estimates that a major French investment bank has costs of about 50 mio. Euro per year attributable to 1 and 4 above, with about half due to legal costs in connection with contract disputes and the other half due to malexecution of financial contracts [7].

In summary, capturing contractual obligations precisely and managing them conscientiously is important for a company's planning, evaluation and reporting to management, shareholders, tax authorities, regulatory bodies, potential buyers and others.

1.3 A domain-specific language for contracts

The ERP systems used today capture the activities of an enterprise based on the principles of double-entry bookkeeping. As the integration of this with subsystems for handling contract execution is characterized by ad hoc, makeshift solutions, it is interesting to consider if a specification language can be designed and integrated with the data model in which historical activities of the enterprise are collected. We argue that a declarative *domain-specific (specification) language (DSL)* for compositional specification of commercial contracts (defining contracts by combining subcontracts in various, well-defined ways) with an associated precise *operational semantics* is ideally suited to alleviating the above problems.[2]

Note that contracts are not put to a single use as programs are, whose sole use usually consists of *execution*. They are subjected to monitoring, which can be considered to be the standard semantics for contracts, plus various user-defined *analyses*.

In this sense contract specifications are more like *intelligent data* that are subjected to various uses. This is in contrast to *programs* that are exclusively executed.

As a consequence, both the syntactic structure of contract specifications and the ability of limiting their expressive (programming) power are of particular significance in their design.

We believe the DSL facilitates multi-stage development as the central interface between framework developer and partner:

1. The *framework developer* provides the DSL, which allows specification of an infinity of contracts in a domain-oriented fashion, but without (too much) prejudice towards specific industries; delivers a run-time environment for managing execution of all definable contracts; and provides a number of useful general-purpose standard contracts. Furthermore, the framework developer provides a language (or library) and run-time system for defining contract analyses and defines a number of standard analyses applicable to all definable contracts; e.g., next-point-of-interest computation for alerting users—human or computer—to commitments that require action (sending payment, making deliveries) or computation of accounts receivable and accounts payable for financial reporting.
2. The *partner* defines a collection of contract templates using the DSL for use in a particular industry and adds relevant industry-specific analyses using the vendor's analysis language. No general-purpose low-level programming expertise is required, but primarily domain knowledge and the ability to formalize it in the DSL and to express specialized analysis functions in the vendor's analysis language. The partner may leave some aspects (parameters) of the specialized system open for final configuration at the end user company.
3. The *customer organization* receives its system from the partner and configures and deploys it for use by its end users.

[2] Our language is rendered in ordinary linear syntax, but we do not intend to limit the scope of the term "language" to specify linear sequences of characters only, but to include graphical objects and the like.

Note that the DSL provides encapsulation and division of labor in this pipeline: discussions between end users and partners are performed in terms of domain concepts close to the DSL, but the end user does not need to know the DSL itself. Discussions between partners and the framework provider on design, functionality, limitations are in terms of the design and semantics of the DSL, not in terms of its underlying (general-purpose) implementation language; in particular, specific implementation choices by the framework developer are unobservable by the partners. The DSL encapsulates its implementation and thus facilitates upgrading of software throughout the pipeline.

1.4 Contributions

We make the following contributions in this article:

– We define a contract language for multi-party commercial contracts with iteration and first-order recursion. They involve explicit agents and transfers of arbitrary resources (money, goods and services, or even pieces of information), not only currencies. Our contract language is stratified into a pluggable base language for atomic contracts (commitments) and a combinator language for composing commitments into structured contracts.
– We provide a natural contract semantics based on an inductive definition for when a trace—a finite sequence of events—constitutes a successful ("performing") completion of a contract. This induces a trace-based denotational semantics, which compositionally maps contracts to trace sets.
– We systematically develop three operational semantics in a stepwise fashion, starting from the denotational semantics:
    1. A (sound and complete) reduction semantics for monitoring contract execution during arrival of events. It represents the residual obligations of a contract after an event as a bona fide (full) contract specification and defers matching of events to specific commitments until the whole contract has completed. It can be implemented by backtracking where events are tentatively matched to the first suitable commitment and

backtracking is performed if that choice turns out to be wrong later on.
2. A nondeterministic reduction semantics using *eager matching*, where matching decisions are made as events arrive and cannot be backtracked. Eager matching corresponds to bookkeeping practice, but leads to nondeterminacy in the case multiple commitments in a contract can be matched by the same event; in particular, the parties to a contract may perform different matches and may end up disagreeing on the contract's residual obligations.
3. An instrumentation of the eager matching semantics that equips events with explicit control information that *routes* the event unambiguously to the particular commitment it is to be matched with. This yields an eager matching semantics with a deterministic reduction semantics and thus ensures that all parties to a contract agree on the residual contract if they agree on the prior contract state and on which event (including its routing information) has happened.

– We validate applicability of our language by encoding a variety of existing contracts in it and illustrate analyzability of contracts by providing examples of compositional analysis.

The denotational semantics has been an instrumental methodological tool in deriving a small-step semantics.

Our work builds on a previous language design by Andersen and Elsborg [1] and is inspired by:

– Peyton-Jones and Eber's language for compositional specification of financial contracts [12], which has been the original impetus for the language design approach we have taken
– McCarthy's [15] Resources–Events–Agents (REA) accounting model, which has provided the ontological justification for modeling commercial contracts as being built from atomic commitments stipulating transfers (economic *events*) of scarce *resources* between *agents* (and nothing else)
– Hoare's Calculus of Sequential Processes (CSP), specifically its view-independent event synchronization model, and its associated trace theoretic semantics [5,10].

See Sect. 7 for a more detailed comparison with this and other related work.

2 Modeling commercial contracts

A *contract* is an agreement between two or more parties which creates obligations to do or not do the specific things that are the subject of that agreement. A *commercial contract* is a contract whose subject is the exchange of scarce *resources* (money, goods and services). Examples of commercial contracts are sales orders, service agreements and rental agreements. Adopting terminology from the REA accounting model [15] we shall also call obligations *commitments* and parties *agents*.

It is worth noticing that contracts may be *express* or *implied*. When two parties decide to exchange goods, more often than not there is no express contract. There is, however, an implied contract of the form of "Party *A* expects to pay *X* in exchange for party *B*'s provision of goods *Y*". Usually when no express contract is present, the contractual obligations are taken from common practice, general terms of trade or legislation. Thus the term *contract* should be understood in a broader sense as a structure that governs any trade or production even if it is not verbal.

2.1 Contract patterns

In its simplest form a contract commits two contract parties to an exchange of resources such as goods for money or services for money, i.e., to a pair of *transfers* of resources from one party to the other, where one transfer is in *consideration* of the other.

The sales order *template* in Fig. 1 commits the two parties (`seller`, `buyer`) to a pair of transfers, of `goods` from `seller` to `buyer` and of `money` from `buyer` to `seller`. Note that both commitments are predicated on when they must be satisfied: `seller` *may* deliver *any time*, but *must* do so by a given `date`, and `buyer` *must* pay at the time delivery happens. We can think of the sales order as being *composed sequentially* of two *atomic contracts*: the `seller`'s commitment to deliver goods, followed by the `buyer`'s commitment to pay for them. If goods are not delivered there is no commitment by `buyer` to pay anything, and only `seller` is in breach of contract. In a barter (goods for goods or goods for services) the commitments on each party may be *composed concurrently*; that is, both commitments are unconditional and must be satisfied independently of each other. If no party delivers on time and no explicit provision for this is made in the contract, *both* parties may be in breach of contract. Many commercial contracts are of this simple quid-pro-quo kind, but far from all. Consider the legal services agreement template in Fig. 2. Here commitments for rendering of a monthly legal service are *repeated*, and each monthly service con-

Section 1. (Sale of goods) Seller shall sell and deliver to buyer (description of goods) no later than (date).
Section 2. (Consideration) In consideration hereof, buyer shall pay (amount in dollars) in cash on delivery at the place where the goods are received by buyer.
Section 3. (Right of inspection) Buyer shall have the right to inspect the goods on arrival and, within (days) business days after delivery, buyer must give notice (detailed-claim) to seller of any claim for damages on goods.

Fig. 1 Agreement to Sell Goods

sists of a standard service part and an *optional* service part. More generally, a contract may allow for *alternative* executions, any one of which satisfies the given contract.

We can discern the following basic *contract patterns* for composing commercial contracts from subcontracts (a subcontract is a contract used as part of another contract):

- A *commitment* stipulates the transfer of a resource or set of resources between two parties; it constitutes an *atomic contract*
- A contract may require *sequential* execution of subcontracts
- A contract may require *concurrent* execution of subcontracts, i.e., execution of all subcontracts, where individual commitments may be interleaved in arbitrary order
- A contract may require execution of one of a number of *alternative* subcontracts
- A contract may require *repeated* execution of a subcontract.

Furthermore, commitments and, more generally, contracts usually carry *temporal constraints*, which stipulate when the actual resource transfers must happen.
In the remainder of this report we shall explore a declarative contract specification language based on these contract patterns.

3 Compositional contract language

In this section we present a core contract specification language and its properties. All proofs are relegated to Appendix 7.4.

The language should satisfy the following design criteria:

- Contracts should be specifiable compositionally, reflecting the contract composition patterns of Sect. 2.1.
- The language should separate contract composition (contract language) from definition of the atomic commitments (base language), including their temporal constraints; this is to make sure that the design can accommodate changes and extensions to the base language without simultaneously forcing substantial changes in the contract language.
- The language should obey good language design principles such as naming and parameterization, orthogonality and compositional semantics.
- The language should be expressive enough to represent partially executed contracts as (full) contracts and have a reduction semantics that reduces a contract under arrival of an event to a contract that represents the residual obligations. By representing partially executed contracts as contracts any contract analysis will also be applicable to partially executed contracts.
- The reduction semantics should be a good basis for "control" of execution, in particular, for *matching* of events against the specific (intended) commitment in a contract that it satisfies.

3.1 Syntax

Our contract language $\mathcal{C}^{\mathcal{P}}$ is defined inductively by the inference system for deriving judgments of the forms $\Gamma; \Delta \vdash c : \text{Contract}$ and $\Delta \vdash D : \Gamma$. Here Γ and Δ range over maps from identifiers to *contract template types* and to *base types*, respectively. The *map extension operator* on maps is defined as follows:

$$(m \oplus m')(x) = \begin{cases} m'(x) & \text{if } x \in \text{domain}(m') \\ m(x) & \text{otherwise.} \end{cases}$$

The language is built on top of a *base structure* of domains $(\mathcal{A}, \mathcal{R}, \mathcal{T})$ of *agents*, *resources*, *time* where $(\mathcal{T}, \leq_{\mathcal{T}})$ is totally ordered. It consists of a typed *base language* of expressions $\mathcal{P}$, for which we assume the existence of a set of valid typing judgments $\Delta \vdash a : \tau$ for expressions a, which include variables X and constants for each element in the base structure. Types τ include Agent, Resource, Time, which denote $(\mathcal{A}, \mathcal{R}, \mathcal{T})$, respectively, as well as Boolean for predicates (Boolean expressions). The expression language has a notion of substitution $b[\mathbf{a}/\mathbf{X}]$[3] and a denotation function $\mathcal{Q}[\![\Delta \vdash a : \tau]\!]$ that maps valid typing judgments to elements of domains given by $\text{Dom}[\![\Delta \to \tau]\!]$. (See Fig. 6

[3] We use the general convention that metavariables in boldface denote vectors (sequences) of what the metavariable denotes.

Section 1. The attorney shall provide, on a non-exclusive basis, legal services up to (n) hours per month, and furthermore provide services in excess of (n) hours upon agreement.
Section 2. In consideration hereof, the company shall pay a monthly fee of (amount in dollars) before the 8th day of the following month and (rate) per hour for any services in excess of (n) hours 40 days after the receival of an invoice.
Section 3. This contract is valid 1/1-12/31, 2004.

Fig. 2 Agreement to provide legal services

for a brief description of the thus denoted domains.) The only properties we shall assume are that substitution is compatible with judgments: if $\Delta \oplus \mathbf{X} : \boldsymbol{\tau} \vdash b : \tau_b$ and $\Delta \vdash \mathbf{a} : \boldsymbol{\tau}$ then $\Delta \vdash b[\mathbf{a}/\mathbf{X}] : \tau_b$ where $\mathbf{a} = a_1, \ldots, a_n$ and $\mathbf{X} = X_1, \ldots, X_n$ for some $n \geq 0$; and that the denotation function is compositional; i.e.

$$\mathcal{Q}[\![\Delta \vdash b[\mathbf{a}/\mathbf{X}] : \tau]\!]^{\delta} = \mathcal{Q}[\![\Delta \vdash b : \tau]\!]^{\delta \oplus \{X_i \mapsto \mathcal{Q}[\![\Delta \vdash a_i : \tau_i]\!]^{\delta}\}_i}.$$

We use metavariable P for Boolean expressions and abbreviate $\Delta \vdash P : \text{Bool}$ to $\Delta \vdash P$. For brevity and readability, we also abbreviate $\mathcal{Q}[\![\Delta \vdash a : \tau]\!]$ to $\mathcal{Q}[\![a]\!]$, leaving Δ and τ to be understood from the context. Finally, we write $\delta \models P$ for $\mathcal{Q}[\![P]\!]^{\delta} = \text{true}$.

The language $\mathcal{P}$ provides the possibility of referring to *observables* [11,12]. We shall introduce suitable base language expressions on an ad hoc basis in our examples for illustrative purposes.

The context-free structure of contracts directly reflects the contract patterns we discussed in Sect. 2.1:

$$\begin{aligned} c ::= {} & \text{Success} \mid \text{Failure} \mid f(\mathbf{a}) \mid \\ & \text{transmit}(A_1, A_2, R, T \mid P).\, c \mid \\ & c_1 + c_2 \mid c_1 \parallel c_2 \mid c_1; c_2. \end{aligned}$$

Success denotes the *trivial* or *(successfully) completed* contract: it carries no obligations on anybody. Failure denotes the *inconsistent* or *failed* contract; it signifies breach of contract or a contract that is impossible to fulfill. The environment $D = \{f_i[\mathbf{X_i}] = c_i\}_{i=1}^{m}$ contains named *contract templates* where $\mathbf{X}_i$ is a vector of formal parameters for use in the embedded contract c_i. A contract template needs to be instantiated with actual arguments from the base language. (The n_i on the τ indicates that different contracts may have a different number of formal parameters.) For a Boolean predicate P the contract expression $\text{transmit}(A_1, A_2, R, T \mid P).\, c$ represents a contract where the *commitment*

$$\text{transmit}(A_1, A_2, R, T \mid P)$$

must be satisfied first. Note that A_1, A_2, R, T are binding variable occurrences whose scope is P and c. The commitment must be *matched* by a *(transfer) event*

$$e = \text{transmit}(v_1, v_2, r, t)$$

of resource r from agent v_1 to agent v_2 at time t where the predicate $P(v_1, v_2, r, t)$ holds. After matching, the residual contract is c in which A_1, A_2, R, T are bound to v_1, v_2, r, t, respectively. In this fashion the subsequent contractual obligations expressed by c may depend on the actual values in event e. The *contract combinators* $\cdot + \cdot, \cdot \parallel \cdot$ and $\cdot ; \cdot$ compose subcontracts according to the contract patterns we have discerned: by alternation, concurrently, and sequentially, respectively. A (contract) context is a finite set of named contract template declarations of the form $f(\mathbf{X}) = c$. By using the *contract instantiation* (or *contract application*) construct $f(\mathbf{a})$ contract templates may be (mutually) recursive, which, in particular, lets us capture repetition of subcontracts. Contract template definitions occur only at top level.

As the contract language $\mathcal{C}^{\mathcal{P}}$ is statically typed its syntax is formally defined by the inference system in Fig. 3. If top-level judgment $\Delta \vdash \text{letrec}\, D\, \text{in}\, c : \text{Contract}$ is derivable we shall say that c is well formed in context D. Henceforth we shall assume that all contracts are well formed, where D may be implicitly understood.

What we call contracts should justly be called *precontracts* as they do not necessarily satisfy the legal requirement for validity. In particular the contracts Success, Failure and any expression that obligates only one agent are not judicially valid contracts. Following [1,2], we shall freely use the term contract, however. Note that consideration (*reciprocity* in REA terms) is not built into our language as a syntactic construct. This allows flexible definitions of contracts where commitments are not in a simple, syntactically evident one-to-one relation, and it allows different, user-defined notions of consideration to be applied as *analyses* to the same language.

In the following we shall adopt the convention that variables A_1, A_2, R, T must not be bound in environment Δ. If a variable from Δ or any expression a only involving variables bound in Δ occurs as an argument of a transmit, we interpret this as an abbreviation; for example the contract, $\text{transmit}(a, A_2, R, T \mid P).\, c$ abbreviates the contract $\text{transmit}(A_1, A_2, R, T \mid P \wedge A_1 = a).\, c$ where A_1 is a new (agent-typed) variable not bound in Δ and different from A_2, R and T. We abbreviate the contract

$$\text{transmit}(A_1, A_2, R, T \mid P).\, \text{Success}$$

$$\Gamma;\Delta \vdash \text{Success} : \text{Contract} \qquad \Gamma;\Delta \vdash \text{Failure} : \text{Contract}$$

$$\frac{\Gamma(f) = \boldsymbol{\tau} \to \text{Contract} \quad \Delta \vdash \mathbf{a} : \boldsymbol{\tau}}{\Gamma;\Delta \vdash f(\mathbf{a}) : \text{Contract}} \qquad \frac{\begin{array}{c}\Delta' = \Delta \oplus \{A_1 : \text{Agent}, A_2 : \text{Agent}, R : \text{Resource}, T : \text{Time}\}\\ \Gamma;\Delta' \vdash c : \text{Contract}\\ \Delta' \vdash P : \text{Boolean}\end{array}}{\Gamma;\Delta \vdash \text{transmit}(A_1, A_2, R, T \mid P).\, c : \text{Contract}}$$

$$\frac{\Gamma;\Delta \vdash c_1 : \text{Contract} \quad \Gamma;\Delta \vdash c_2 : \text{Contract}}{\Gamma;\Delta \vdash c_1 + c_2 : \text{Contract}} \qquad \frac{\Gamma;\Delta \vdash c_1 : \text{Contract} \quad \Gamma;\Delta \vdash c_2 : \text{Contract}}{\Gamma;\Delta \vdash c_1 \parallel c_2 : \text{Contract}}$$

$$\frac{\Gamma;\Delta \vdash c_1 : \text{Contract} \quad \Gamma;\Delta \vdash c_2 : \text{Contract}}{\Gamma;\Delta \vdash c_1; c_2 : \text{Contract}} \qquad \frac{\begin{array}{c}\Gamma = \{f_i \mapsto \tau_{i1} \times \ldots \times \tau_{in_i} \to \text{Contract}\}_{i=1}^m\\ \Gamma;\Delta \oplus \{X_{i1} : \tau_{i1}, \ldots, X_{in_i} : \tau_{in_i}\} \vdash c_i : \text{Contract}\end{array}}{\Delta \vdash \{f_i[\mathbf{X_i}] = c_i\}_{i=1}^m : \Gamma}$$

$$\frac{\Delta \vdash \{f_i[\mathbf{X_i}] = c_i\}_{i=1}^m : \Gamma \quad \Gamma;\Delta \vdash c : \text{Contract}}{\Delta \vdash \text{letrec}\, \{f_i[\mathbf{X_i}] = c_i\}_{i=1}^m \text{ in } c : \text{Contract}}$$

Fig. 3 Syntax for contract specifications

to

transmit$(A_1, A_2, R, T \mid P)$.

The contract from Fig. 1 is encoded in Fig. 4, and the contract in Fig. 2 is treated in depth in Sects. 4 and 5.

3.2 Event traces and contract satisfaction

A contract specifies a set of alternative performing event sequences (contract executions), each of which satisfies the obligations expressed in the contract and concludes it. In this section we make these notions precise for our language.

Recall that our *base structure* is a tuple of sets of resources $\mathcal{R}$, agents $\mathcal{A}$ and a totally ordered set $(\mathcal{T}, \leq_{\mathcal{T}})$ of *dates* (or *time points*) $(\mathcal{R}, \mathcal{T}, \mathcal{A})$. Whenever convenient, we will extend base structures with other sets for other types, as needed. A *(transfer) event* e is a term transmit(v_1, v_2, r, t), where $v_1, v_2 \in \mathcal{A}, r \in \mathcal{R}$ and $t \in \mathcal{T}$. An *(event) trace* s is a finite sequence of events that is chronologically ordered; i.e., for $s = e_1, \ldots, e_n$ the time points in $e_1, \ldots, e_n$ occur in nondescending order. We adopt the following notation: $\langle\rangle$ denotes the empty sequence; a trace consisting of a single event e is denoted by e itself; concatenation of traces s_1 and s_2 is denoted by juxtaposition: $s_1 s_2$; we write $(s_1, s_2) \rightsquigarrow s$ if s is an interleaving of the events in traces s_1 and s_2; we write $\mathbf{X}$ for the vector $X_1, \ldots, X_k$ with $k \geq 0$ and where k can be deduced from the context; we write $c[\mathbf{v}/\mathbf{X}]$, where $\mathbf{v} = v_1, \ldots, v_n$ and $\mathbf{X} = X_1, \ldots, X_n$ for some $n \geq 0$, for the result of simultaneously *substituting* elements v_i for the all free occurrences of the corresponding X_i in c. (Free and bound variables are defined as can be expected.)

We are now ready to specify when a trace *satisfies* a contract, i.e., gives rise to a performing execution of the contract. This is done inductively by the inference system for judgments $\delta' \vdash^{\delta}_{\mathrm{D}} s : c$ in Fig. 5, where $\mathrm{D} = \{f_i[\mathbf{X_i}] = c_i\}_{i=1}^m$ is a finite set of named *contract templates* and δ is a finite set of bindings of variables to elements (values of a domain) of the given base structure. A derivable judgment $\delta' \vdash^{\delta}_{\mathrm{D}} s : c$ expresses that event sequence s satisfies—successfully executes and concludes—contract c in an environment where contract templates are defined as in D, δ is the top-level environment for both D and c, and δ' is a local environment for additional free variables in c. Conversely, if $\delta' \vdash^{\delta}_{\mathrm{D}} s : c$ is not derivable then s does not satisfy c for given $\mathrm{D}, \delta, \delta'$. The condition $\delta \oplus \delta'' \models P$ in the third rule stipulates that P, with free variables bound as in $\delta \oplus \delta'$, must be true in the base language for an event to match the corresponding commitment.

3.3 Denotational semantics

A denotational semantics maps contract specifications compositionally into a domain of mathematical objects, i.e., by induction on the syntax (inference tree) of contract expressions as given by the inference rules of Fig. 3. A denotational semantics supports reasoning by structural induction on the syntax. In particular, any subcontract of a contract can be replaced by any other subcontract with the same denotation without changing the behavior of the whole contract.

The satisfaction relation relates each contract to a set of traces. We can use that to define the *extension* of a contract c to be the set of its performing executions: $\mathcal{E}[\![\text{letrec}\, \mathrm{D} \text{ in } c]\!]^{\delta} = \{s : \emptyset \vdash^{\delta}_{\mathrm{D}} s : c\}$. This, however, is not a denotational semantics as it is not compositional. Turning it into a compositional definition we arrive at the semantics given in Fig. 7. Note that each contract denotes a trace set, and the meaning of a compound

```
letrec
  nonconforming [seller, buyer, goods, payment, days, t1, notice] =
          transmit (buyer, seller, notice, T |
                   T < t1 + days d and #(goods,broken,t1) = 1).
          transmit (seller, buyer, payment/2, T' | T' < T + days d).

  sale [seller, buyer, goods, payment, t1, days, notice] =
          transmit (seller, buyer, goods, T | T < t1).
          transmit (buyer, seller, payment, T' | T' < t1).
          (Success + nonconforming (seller, buyer, goods, days, T', notice))
in
  sale ("Furniture maker", "Me", "Chair", 40, 2004.7.1, 8, "Chair broken")
```

Fig. 4 Specification of agreement to sell goods

Fig. 5 Contract satisfaction

$$\delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : \mathrm{Success} \qquad \frac{\mathbf{X} \mapsto \mathbf{v} \vdash^{\delta}_{\mathrm{D}} s : c \quad (f(\mathbf{X}) = c) \in \mathrm{D}, \mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta \oplus \delta'}}{\delta' \vdash^{\delta}_{\mathrm{D}} s : f(\mathbf{a})}$$

$$\frac{\delta \oplus \delta'' \models P \quad \delta'' \vdash^{\delta}_{\mathrm{D}} s : c \qquad (\delta'' = \delta' \oplus \{\mathbf{X} \mapsto \mathbf{v}\})}{\delta' \vdash^{\delta}_{\mathrm{D}} \mathrm{transmit}(\mathbf{v})\, s : \mathrm{transmit}(\mathbf{X}|P).\, c}$$

$$\frac{\delta' \vdash^{\delta}_{\mathrm{D}} s_1 : c_1 \quad \delta' \vdash^{\delta}_{\mathrm{D}} s_2 : c_2 \quad (s_1, s_2) \rightsquigarrow s}{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_1 \parallel c_2} \qquad \frac{\delta' \vdash^{\delta}_{\mathrm{D}} s_1 : c_1 \quad \delta' \vdash^{\delta}_{\mathrm{D}} s_2 : c_2}{\delta' \vdash^{\delta}_{\mathrm{D}} s_1 s_2 : c_1; c_2}$$

$$\frac{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_1}{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_1 + c_2} \qquad \frac{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_2}{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_1 + c_2}$$

$$\begin{aligned}
Dom[\![\mathrm{Boolean}]\!] &= (\{\mathrm{true}, \mathrm{false}\}, =) \\
Dom[\![\mathrm{Agent}]\!] &= (\mathcal{A}, =) \\
Dom[\![\mathrm{Resource}]\!] &= (\mathcal{R}, =) \\
Dom[\![\mathrm{Time}]\!] &= (\mathcal{T}, =) \\
\mathcal{E} &= \mathcal{A} \times \mathcal{A} \times \mathcal{R} \times \mathcal{T} \\
Tr &= (\mathcal{E}^*, =) \\
Dom[\![\mathrm{Contract}]\!] &= (2^{Tr}, \subseteq) \\
Dom[\![\tau_1 \times \ldots \times \tau_n \to \mathrm{Contract}]\!] &= Dom[\![\tau_1]\!] \times \ldots \times Dom[\![\tau_n]\!] \to Dom[\![\mathrm{Contract}]\!] \\
Dom[\![\Gamma]\!] &= \{\{f_i \mapsto v_i\}_{i=1}^{m} \mid v_i \in Dom[\![\tau_{i1}]\!] \times \ldots \times Dom[\![\tau_{in_i}]\!] \to Dom[\![\mathrm{Contract}]\!]\} \\
&\quad \text{where } \Gamma = \{f_i \mapsto \tau_{i1} \times \ldots \times \tau_{in_i} \to \mathrm{Contract}\}_{i=1}^{m} \\
Dom[\![\Delta]\!] &= \{\{X_i \mapsto v_i\}_{i=1}^{m} \mid v_i \in Dom[\![\tau_i]\!]\} \\
&\quad \text{where } \Delta = \{X_i : \tau_i\}_{i=1}^{m} \\
Dom[\![\Gamma; \Delta \vdash c : \mathrm{Contract}]\!] &= Dom[\![\Gamma]\!] \times Dom[\![\Delta]\!] \to Dom[\![\mathrm{Contract}]\!]
\end{aligned}$$

Fig. 6 Domains for $\mathcal{C}^{\mathcal{P}}$

contract can be explained in terms of a mathematical operation on the trace sets denoted by its constituent subcontracts without any reference to the actual syntax of the latter.

The presence of recursive contract definitions requires domain theory; see, e.g., Winskel [22]. Briefly, each type in our language is mapped to a *complete partial order (*cpo*)*, i.e., a set equipped with a partial order where each directed subset has a least upper bound (in the set). A *pointed complete partial order (*pcpo*)* is a cpo that has a least element. All our domains in Fig. 6 are cpos as we can choose equality for the base domains $\mathcal{A}, \mathcal{R}, \mathcal{T}$. Furthermore, 2^{Tr}, the powerset of all finite event sequences, is a pcpo under $\subseteq$, and the function space $D \to D'$ is a pcpo under pointwise ordering if D' is a pcpo. A function between cpos is *continuous* if the result of applying it to the least upper bound of a directed set is the same as the least upper bound of applying it to each element of the directed set individually. It is well known that each continuous function from a pcpo to the same pcpo has a least (unique minimal) fixed point. It is a routine matter to check that $\mathcal{C}[\![.]\!]^{\cdot}$, $\mathcal{E}[\![.]\!]^{\cdot}$ and $\mathcal{D}[\![.]\!]^{\cdot}$ map contracts under function environments, contract specifications and contract function environments, respectively, to continuous functions. Consequently the least fixed point in line 9 of Fig. 7 always exists.

Fig. 7 Denotational semantics

$$
\begin{aligned}
\mathcal{C}[\![\text{Success}]\!]^{\gamma;\delta} &= \{\langle\rangle\} & (1)\\
\mathcal{C}[\![\text{Failure}]\!]^{\gamma;\delta} &= \emptyset & (2)\\
\mathcal{C}[\![f(\mathbf{a})]\!]^{\gamma;\delta} &= \gamma(f)(\mathcal{Q}[\![\mathbf{a}]\!]^{\delta}) & (3)\\
\mathcal{C}[\![\text{transmit}(\mathbf{X} \mid P).\, c]\!]^{\gamma;\delta} &= \{\text{transmit}(\mathbf{v})\, s : \mathbf{v} \in \mathcal{E}, s \in \mathit{Tr} \mid & (4)\\
&\qquad \mathcal{Q}[\![P]\!]^{\delta \oplus \mathbf{X} \mapsto \mathbf{v}} = \text{true} \wedge s \in \mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}}\} & (5)\\
\mathcal{C}[\![c_1 + c_2]\!]^{\gamma;\delta} &= \mathcal{C}[\![c_1]\!]^{\gamma;\delta} \cup \mathcal{C}[\![c_2]\!]^{\gamma;\delta} & (6)\\
\mathcal{C}[\![c_1 \parallel c_2]\!]^{\gamma;\delta} &= \left\{s : s \in \mathit{Tr} \mid \exists s_1 \in \mathcal{C}[\![c_1]\!]^{\gamma;\delta}, s_2 \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta}.\, (s_1, s_2) \rightsquigarrow s\right\} & (7)\\
\mathcal{C}[\![c_1; c_2]\!]^{\gamma;\delta} &= \{s_1 s_2 : s_1, s_2 \in \mathit{Tr} \mid s_1 \in \mathcal{C}[\![c_1]\!]^{\gamma;\delta} \wedge s_2 \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta}\} & (8)\\
\mathcal{D}[\![\{f_i[\mathbf{X_i}] = c_i\}_{i=1}^m]\!]^{\delta} &= \mathit{least}\, \gamma : \gamma = \{f_i \mapsto \lambda \mathbf{v}_i.\mathcal{C}[\![c_i]\!]^{\gamma;\delta\oplus\mathbf{X}_i\mapsto\mathbf{v}_i}\}_{i=1}^m & (9)\\
\mathcal{E}[\![\text{letrec}\,\{f_i[\mathbf{X}_i] = c_i\}_{i=1}^m \text{ in } c]\!]^{\delta} &= \mathcal{C}[\![c]\!]^{\mathcal{D}[\![\{f_i[\mathbf{X}_i]=c_i\}_{i=1}^m]\!]^{\delta};\delta} & (10)
\end{aligned}
$$

We say c *denotes* a trace set S in context D, δ, if $\mathcal{C}[\![c]\!]^{\text{D};\delta} = S$. The following theorem states that the denotational semantics characterizes the satisfaction relation.

Theorem 1 (Denotational characterization of contract satisfaction)

$\mathcal{C}[\![c]\!]^{\mathcal{D}[\![\text{D}]\!]^{\delta};\delta\oplus\delta'} = \{s \mid \delta' \vdash^{\delta}_{\text{D}} s : c\}$.

3.4 Contract monitoring by residuation

Extensionally, contracts classify traces (event sequences) into performing and nonperforming ones. We are not only interested in classifying complete event sequences once they have happened, but also in *monitoring* contract execution as it unfolds in time under the arrival of events. We say a trace is *consistent* with a trace set S if it is a prefix of an element of S; it is *inconsistent* otherwise.

Given a trace set S denoted by a contract c and an event e, the *residuation function* $\cdot\backslash\cdot$ captures how c can be satisfied if the first event is e. It is defined as follows[4]:

$e\backslash S = \{s' \mid \exists s \in S : es' = s\}$

Conceptually, we can map contracts to trace sets and use the residuation function to monitor contract execution as follows:

1. Map a given contract c_0 to the trace set S_0 that it denotes. If $S_0 = \emptyset$, stop and output "inconsistent".
2. For $i = 0, 1, \ldots$ do:
   Receive message e_i.
   (a) If e_i is a transfer event, compute $S_{i+1} = e_i\backslash S_i$. If $S_{i+1} = \emptyset$, stop and output "breach of contract"; otherwise continue.
   (b) If e_i is a "conclude contract" message, check whether $\langle\rangle \in S_i$. If so, all obligations have been fulfilled and the contract can be terminated. Stop and output "successfully completed". If $\langle\rangle \notin S_i$, output "cannot be concluded now", let $S_{i+1} = S_i$ and continue to receive messages.

[4] Conway [6] calls $e\backslash S$ the *e-derivative* for a *language S* and *alphabet symbol e*. We use the term *residuation* instead to emphasize that $e\backslash S$ represents the *residual* obligations of a contract after execution of event e.

To make the conceptual algorithm for contract life cycle monitoring from Sect. 3.4 *operational*, we need to represent the residual trace sets and provide methods for deciding tests for emptiness and failure. In particular, we would like to use contracts as representations for trace sets. Not all trace sets are denotable by contracts, however. In particular, given a contract c that denotes a trace set S_c it is not a priori clear whether $e\backslash S_c$ is denotable by a contract c'. If it is, we call c' the *residual contract of c after e*.

Let us momentarily extend contract specifications with a *residuation operator*, which is the syntactic analog of residuation, but for contracts instead of trace sets:

$\mathcal{C}[\![e\backslash c]\!]^{\gamma;\delta} = \{s' \mid \exists s \in \mathcal{C}[\![c]\!]^{\gamma;\delta} : es' = s\}$.

Let us write $\text{D}, \delta \models c = c'$ if $\mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\delta'} = \mathcal{C}[\![c']\!]^{\gamma;\delta\oplus\delta'}$ for all δ', where $\gamma = \mathcal{D}[\![\text{D}]\!]^{\delta}$; analogously for $\text{D}, \delta \models c \subseteq c'$. To elide parentheses we use the following operator precedence order in contract expressions (highest precedence first): residuation $\cdot\backslash\cdot$, concurrent composition $\cdot \parallel \cdot$, alternation $\cdot + \cdot$, sequential composition $\cdot;\cdot$.

Lemma 1 (Correctness of residuation) *The residuation equalities in Fig. 8 are true.*

For the proof of this lemma we need an auxiliary lemma that extends the compositionality of the base language to the contract language:

Lemma 2 (Agreement of substitution and environment) *For all c, γ and δ:*

$\mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}} = \mathcal{C}[\![c[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}$.

Executing the residuation equations as left-to-right rewrite rules eliminates the residuation operator in $e\backslash c$, assuming c is residuation operator free to start with.

Fig. 8 Residuation equalities

$$
\begin{aligned}
\mathrm{D},\delta &\models e\backslash\mathrm{Success} = \mathrm{Failure}\\
\mathrm{D},\delta &\models e\backslash\mathrm{Failure} = \mathrm{Failure}\\
\mathrm{D},\delta &\models e\backslash f(\mathbf{a}) = e\backslash c[\mathbf{v}/\mathbf{X}] \text{ if } (f(\mathbf{X}) = c) \in \mathrm{D}, v = \mathcal{Q}[\![a]\!]^{\delta}\\
\mathrm{D},\delta &\models \mathrm{transmit}(\mathbf{v})\backslash(\mathrm{transmit}(\mathbf{X} \mid P).\, c) = \begin{cases} c[\mathbf{v}/\mathbf{X}] \text{ if } \delta \oplus \{\mathbf{X} \mapsto \mathbf{v}\} \models P\\ \text{Failure otherwise}\end{cases}\\
\mathrm{D},\delta &\models e\backslash(c_1 + c_2) = e\backslash c_1 + e\backslash c_2\\
\mathrm{D},\delta &\models e\backslash(c_1 \parallel c_2) = e\backslash c_1 \parallel c_2 + c_1 \parallel e\backslash c_2\\
\mathrm{D},\delta &\models e\backslash(c_1; c_2) = \begin{cases} (e\backslash c_1; c_2) + e\backslash c_2 & \text{if } \mathrm{D},\delta \models \mathrm{Success} \subseteq c_1\\ e\backslash c_1; c_2 & \text{otherwise}\end{cases}
\end{aligned}
$$

$$
\frac{\mathrm{D} \vdash c \text{ nullable} \quad (f(\mathbf{X}) = c) \in \mathrm{D}}{\mathrm{D} \vdash f(\mathbf{a}) \text{ nullable}}
\qquad
\frac{\mathrm{D} \vdash c \text{ nullable}}{\mathrm{D} \vdash c + c' \text{ nullable}}
\qquad
\frac{\mathrm{D} \vdash c' \text{ nullable}}{\mathrm{D} \vdash c + c' \text{ nullable}}
$$

$$
\mathrm{D} \vdash \mathrm{Success} \text{ nullable}
\qquad
\frac{\mathrm{D} \vdash c \text{ nullable} \quad \mathrm{D} \vdash c' \text{ nullable}}{\mathrm{D} \vdash c \parallel c' \text{ nullable}}
\qquad
\frac{\mathrm{D} \vdash c \text{ nullable} \quad \mathrm{D} \vdash c' \text{ nullable}}{\mathrm{D} \vdash c; c' \text{ nullable}}
$$

Fig. 9 Nullable contracts

That computation does not always terminate, however. Consider, e.g.,

letrec
  $f(N) = ($
    $\mathrm{transmit}(a_1, a_2, r, T \mid T \leq N) \parallel$
    $f(N+1)$
  $)$
in $f(0)$

and event $\mathrm{transmit}(a_1, a_2, r, 0)$. Applying the rewrite rules will not terminate. Intuitively, this is because the contract $\mathrm{transmit}(a_1, a_2, r, 0)$ can be matched against any one of the infinitely many commitments

$$
\begin{aligned}
&\mathrm{transmit}(a_1, a_2, r, T_0 | T_0 \leq 0) \parallel \cdots\\
&\quad \mathrm{transmit}(a_1, a_2, r, T_i | T_i \leq i) \parallel \cdots
\end{aligned}
$$

as $\mathrm{transmit}(a_1, a_2, r, 0)$ obviously satisfies the match condition of each one of them. Note that, semantically,

$$
\begin{aligned}
&f(N) = (\mathrm{transmit}(a_1, a_2, r, T \mid T \leq N) \parallel f(N+1)\\
&\quad \models f(0) = \mathrm{Failure}
\end{aligned}
$$

but left-to-right rewriting according to Fig. 8 does not rewrite $f(0)$ to Failure.

3.5 Nullable and guarded contracts

In this section we characterize *nullability* of a contract and introduce *guarding*, which is a sufficient condition on contracts for ensuring that residuation can be performed by reduction on contracts.

Definition 1 (Nullability)

1. We write $\mathrm{D} \models c$ nullable if $\mathrm{D}, \delta \models \mathrm{Success} \subseteq c$ for some δ; that is, $\langle\rangle \in \mathcal{C}[\![c]\!]^{\mathrm{D};\delta}$.
2. We say c is *nullable* (or *terminable*) in context D if $\mathrm{D} \vdash c$ nullable is derivable by the inference system in Fig. 9.

A nullable contract can be concluded successfully, but may possibly also be continued, e.g., the contract $\mathrm{Success} + \mathrm{transmit}(a_1, a_2, r, t|P)$ is nullable, as it may be concluded successfully (left choice). Note, however, that it may also be continued (right choice). It is easy to see that nullability is independent of δ and δ': $\langle\rangle \in \mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\delta'}$ if and only if $\langle\rangle \in \mathcal{C}[\![c]\!]^{\gamma;\hat{\delta}\oplus\hat{\delta}'}$ for any other $\hat{\delta}$ and $\hat{\delta}'$, where $\gamma = \mathcal{D}[\![\mathrm{D}]\!]^{\delta}$. Deciding nullability is required to implement Step 2b in contract monitoring. The following proposition expresses that nullability characterizes semantic nullability.

Proposition 1 (Syntactic characterization of nullability)

$$\mathrm{D} \models c \text{ nullable} \iff \mathrm{D} \vdash c \text{ nullable}.$$

Definition 2 (Guarded contract, guarded declarations) Let $\mathrm{D} = \{f_i[\mathbf{X}_i] = c_i\}_{i=1}^{m}$ be contract template declarations.

A contract c is *guarded* in context D if $\mathrm{D} \vdash c$ guarded is derivable from Fig. 10. We say D is guarded if c_i is guarded in context D for all i with $1 \leq i \leq m$.

Intuitively, guardedness ensures that we do not have (mutual) recursions such as $\{f(\mathbf{X}) = g(\mathbf{X}), g(\mathbf{X}) = f(\mathbf{X})\}$ that cause the residuation algorithm to loop infinitely. Guarded declarations ensure that all contracts built from them are also guarded:

Lemma 3 (Guardedness of contracts using guarded declarations) *For all* D, c, *if* D *is guarded then* $\mathrm{D} \vdash c$ guarded.

As we shall see, guardedness is key to ensuring termination of contract residuation and thus that every (guarded) contract has a residual contract under any event in the reduction semantics of Fig. 11.

$$\mathrm{D} \vdash \text{Success guarded} \qquad \mathrm{D} \vdash \text{Failure guarded}$$

$$\mathrm{D} \vdash \text{transmit}(\mathbf{X} \mid P).\, c \text{ guarded} \qquad \frac{\mathrm{D} \vdash c \text{ guarded} \quad (f(\mathbf{X}) = c) \in \mathrm{D}}{\mathrm{D} \vdash f(\mathbf{a}) \text{ guarded}}$$

$$\frac{\mathrm{D} \vdash c \text{ guarded} \quad \mathrm{D} \vdash c' \text{ guarded}}{\mathrm{D} \vdash c + c' \text{ guarded}} \qquad \frac{\mathrm{D} \vdash c \text{ guarded} \quad \mathrm{D} \vdash c' \text{ guarded}}{\mathrm{D} \vdash c \parallel c' \text{ guarded}}$$

$$\frac{\mathrm{D} \vdash c \text{ nullable} \quad \mathrm{D} \vdash c \text{ guarded} \quad \mathrm{D} \vdash c' \text{ guarded}}{\mathrm{D} \vdash c; c' \text{ guarded}} \qquad \frac{\mathrm{D} \nvdash c \text{ nullable} \quad \mathrm{D} \vdash c \text{ guarded}}{\mathrm{D} \vdash c; c' \text{ guarded}}$$

Fig. 10 Guarded contracts

$$\mathrm{D}, \delta \vdash_D \text{Success} \xrightarrow{e} \text{Failure} \qquad \mathrm{D}, \delta \vdash_D \text{Failure} \xrightarrow{e} \text{Failure}$$

$$\frac{\delta \oplus \{\mathbf{X} \mapsto \mathbf{v}\} \models P \quad (\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta})}{\mathrm{D}, \delta \vdash_D \text{transmit}(\mathbf{X}|P).\, c \xrightarrow{\text{transmit}(\mathbf{v})} c[\mathbf{v}/\mathbf{X}]} \qquad \frac{\delta \oplus \{\mathbf{X} \mapsto \mathbf{v}\} \not\models P \quad (\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta})}{\mathrm{D}, \delta \vdash_D \text{transmit}(\mathbf{X}|P).\, c \xrightarrow{\text{transmit}(\mathbf{v})} \text{Failure}}$$

$$\frac{\mathrm{D}, \delta \vdash_D c[\mathbf{v}/\mathbf{X}] \xrightarrow{e} c' \quad (f(\mathbf{X}) = c) \in \mathrm{D}, \mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta}}{\mathrm{D}, \delta \vdash_D f(\mathbf{a}) \xrightarrow{e} c'} \qquad \frac{\mathrm{D}, \delta \vdash_D c \xrightarrow{e} d \quad \mathrm{D}, \delta \vdash_D c' \xrightarrow{e} d'}{\mathrm{D}, \delta \vdash_D c + c' \xrightarrow{e} d + d'}$$

$$\frac{\mathrm{D}, \delta \vdash_D c \xrightarrow{e} d \quad \mathrm{D}, \delta \vdash_D c' \xrightarrow{e} d'}{\mathrm{D}, \delta \vdash_D c \parallel c' \xrightarrow{e} c \parallel d' + d \parallel c'} \qquad \frac{\mathrm{D} \vdash c \text{ nullable} \quad \mathrm{D}, \delta \vdash_D c \xrightarrow{e} d \quad \mathrm{D}, \delta \vdash_D c' \xrightarrow{e} d'}{\mathrm{D}, \delta \vdash_D c; c' \xrightarrow{e} (d; c') + d'}$$

$$\frac{\mathrm{D} \nvdash c \text{ nullable} \quad \mathrm{D}, \delta \vdash_D c \xrightarrow{e} d}{\mathrm{D}, \delta \vdash_D c; c' \xrightarrow{e} d; c'}$$

Fig. 11 Deterministic reduction (delayed matching)

3.6 Operational semantics I: deferred matching

The denotational semantics tells us what trace set is denoted by a contract, and residuation on trace sets tells us how to turn the denotational semantics conceptually into a *monitoring* semantics. In this section we present a *reduction semantics* for contracts, which lifts residuation on trace sets to contracts and is derived systematically from the residuation equalities of Fig. 8.

The ability of representing residual contract obligations of a partially executed contract and thus any state of a contract as a bona fide contract carries the advantage that any analysis that is performed on "original" contracts automatically extends to partially executed contracts as well; e.g., an investment bank that applies valuations to financial contracts before offering them to customers can apply their valuations to their portfolio of contracts under execution, e.g., to analyze its risk exposure under current market conditions. Likewise, a company that analyzes production and capacity requirements of a contract before offering it to a customer can apply the same analysis to the contracts it has under execution, e.g., to adjust planning based on present capacity requirements.

The reduction semantics is presented in Fig. 11. The basic *matching rule* is

$$\frac{\delta \oplus \{\mathbf{X} \mapsto \mathbf{v}\} \models P \quad (\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta})}{\mathrm{D}, \delta \vdash_D \text{transmit}(\mathbf{X}|P).\, c \xrightarrow{\text{transmit}(\mathbf{v})} c[\mathbf{v}/\mathbf{X}]}.$$

It *matches* an event with a specific commitment in a contract. There may be multiple commitments in a contract that match the same event. The semantics captures the possibilities of matching an event against multiple commitments by applying all possible reductions in alternatives and concurrent contract forms and forming the sum of their possible outcomes (some of which may actually be Failure).

The rule

$$\frac{\mathrm{D}, \delta \vdash_D c \xrightarrow{e} d \quad \mathrm{D}, \delta \vdash_D c' \xrightarrow{e} d'}{\mathrm{D}, \delta \vdash_D c + c' \xrightarrow{e} d + d'}$$

thus reduces both alternatives c and c' and then forms the sum of their respective results d, d'.

Likewise, the rule

$$\frac{\mathrm{D}, \delta \vdash_D c \xrightarrow{e} d \quad \mathrm{D}, \delta \vdash_D c' \xrightarrow{e} d'}{\mathrm{D}, \delta \vdash_D c \parallel c' \xrightarrow{e} c \parallel d' + d \parallel c'}$$

for concurrent subcontracts is expressing that the match could be in either one of c or c' and represents the result as the sum of those two possibilities.

Finally, the rule

$$\frac{\mathrm{D} \vdash c \text{ nullable} \quad \mathrm{D}, \delta \vdash_D c \xrightarrow{e} d \quad \mathrm{D}, \delta \vdash_D c' \xrightarrow{e} d'}{\mathrm{D}, \delta \vdash_D c; c' \xrightarrow{e} (d; c') + d'}$$

captures that e can be matched in c or, if c is nullable, in c'. Note that, if c is not nullable, e can only be matched in c, not c', as expressed by the rule

$$\frac{\mathrm{D} \not\vdash c \text{ nullable} \quad \mathrm{D}, \delta \vdash_D c \xrightarrow{e} d}{\mathrm{D}, \delta \vdash_D c; c' \xrightarrow{e} d; c'}.$$

In this fashion the semantics keeps track of the results of all possible matches in a reduction sequence as explicit *alternatives* (summands) and *defers* the decision as to *which specific* commitment is matched by a particular event during contract execution until the very end: by selecting a particular summand in a residual contract after a number of reduction steps that represents Success (and the contract is thus terminable) a particular set of matching decisions is chosen ex post. As presented, the reduction semantics gives rise to an implementation in which the multiple reducts of previous reduction steps are reduced in parallel, as they are represented as summands in a single contract, and the rule for reduction of sums reduces both summands. It is relatively straightforward to turn this into a backtracking semantics by an asymmetric reduction rule for sums, which delays reduction of the right summand.

The operational semantics fully and faithfully implements residuation:

Theorem 2 (Residuation by deferred matching)

1. *For any* c, c', δ, e *and* D*: if* $\mathrm{D}, \delta \vdash_D c \xrightarrow{e} c'$ *then* $\mathrm{D}, \delta \models e \backslash c = c'$.
2. *For all* c, δ *and guarded* D*, there exists a unique* c' *such that* $\mathrm{D}, \delta \vdash_D c \xrightarrow{e} c'$*; furthermore,* $\mathrm{D} \vdash c'$ guarded.

Using Theorem 2 we can turn our conceptual contract monitoring algorithm into a real algorithm:

1. Let contract c_0 be given. If c_0 is inconsistent, stop and output "inconsistent".
2. For $i = 0, 1, \ldots$, do:
   Receive message e_i.
   (a) If e_i is a transfer event, let c_{i+1} be such that $\vdash_D c_i \xrightarrow{e_i} c_{i+1}$. If c_{i+1} is inconsistent, stop and output "breach of contract"; otherwise continue.
   (b) If e is a "terminate contract" message, then check whether c_i is nullable. If so, all obligations have been fulfilled and the contract can be terminated. Stop and output "successfully completed". If c_i is not nullable, output "cannot be terminated now", let $c_{i+1} = c_i$ and continue to receive messages.

Proposition 1 provides a syntactic characterization of nullability, which can easily be turned into an algorithm. Deciding $\mathrm{D}, \delta \models c = \text{Failure}$,i.e. whether a contract has actually failed, is a much harder problem. See Fig. 22 for a sketch for a conservative approximation (some failed contracts may not be identified as such) to this.

The deferred matching semantics of Fig. 11 is flexible and faithful to the natural notion of contract satisfaction as defined in Fig. 5. But from an accounting practice view it is weird because matching decisions are deferred. In bookkeeping standard *modus operandi* is that events are matched against specific commitments *eagerly*, that is online, as events arrive.[5]

We shall turn the deferred matching semantics of Fig. 11 into an eager matching semantics (Fig. 12). The idea is simple: represent here-and-now choices as alternative *rules* (meta-level) as opposed to alternative contracts (object level). Specifically, we split the rules for reducing alternatives and concurrent subcontracts into multiple rules, and we capture the possibility of reducing in the second component of a sequential contract by adding τ-transitions, which "spontaneously" (without a driving external event) reduce a contract of the form Success;c to c. For this to be sufficient we have to make sure that a nullable contract indeed can be reduced to Success, not just a contract that is *equivalent* to Success, such as Success $\|$ Success. This is done by ensuring that τ-transitions are strong enough to guarantee reduction to Success as required.

Based on these considerations we arrive at the reduction semantics in Fig. 12, where meta-variable λ ranges over events e and the internal event τ. Note that it is nondeterministic and not even confluent: a contract c can be reduced to two different contracts by the same event. Consider, e.g., $c = a; b + a; b'$ where a, b, b' are commitments, no two of which match the same event. For

[5] There are standard accounting practices for changing such decisions, but both default and standard conceptual model are that matching decisions are made as early as possible. In general, it seems representing and deferring choices and applying *hypothetical* reasoning to them appears to be a rather unusual phenomenon in accounting.

$$\mathrm{D}, \delta \vdash_N \text{Success} \xrightarrow{e} \text{Failure} \qquad \mathrm{D}, \delta \vdash_N \text{Failure} \xrightarrow{e} \text{Failure}$$

$$\frac{\delta \oplus \{\mathbf{X} \mapsto \mathbf{v}\} \models P, \mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^\delta}{\mathrm{D}, \delta \vdash_N \text{transmit}(\mathbf{X} \mid P).\, c \xrightarrow{\text{transmit}(\mathbf{v})} c[\mathbf{v}/\mathbf{X}]} \qquad \frac{\delta \oplus \{\mathbf{X} \mapsto \mathbf{v}\} \not\models P, \mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^\delta}{\mathrm{D}, \delta \vdash_N \text{transmit}(\mathbf{X}|P).\, c \xrightarrow{\text{transmit}(\mathbf{v})} \text{Failure}}$$

$$\frac{(f(\mathbf{X}) = c) \in \mathrm{D}, \mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^\delta}{\mathrm{D}, \delta \vdash_N f(\mathbf{a}) \xrightarrow{\tau} c[\mathbf{v}/\mathbf{X}]} \qquad \mathrm{D}, \delta \vdash_N c + c' \xrightarrow{\tau} c \qquad \mathrm{D}, \delta \vdash_N c + c' \xrightarrow{\tau} c'$$

$$\frac{\mathrm{D}, \delta \vdash_N c \xrightarrow{\lambda} d}{\mathrm{D}, \delta \vdash_N c \parallel c' \xrightarrow{\lambda} d \parallel c'} \qquad \frac{\mathrm{D}, \delta \vdash_N c' \xrightarrow{\lambda} d'}{\mathrm{D}, \delta \vdash_N c \parallel c' \xrightarrow{\lambda} c \parallel d'}$$

$$\mathrm{D}, \delta \vdash_N \text{Success} \parallel c \xrightarrow{\tau} c \qquad \mathrm{D}, \delta \vdash_N c \parallel \text{Success} \xrightarrow{\tau} c \qquad \mathrm{D}, \delta \vdash_N \text{Success}; c' \xrightarrow{\tau} c'$$

$$\frac{\mathrm{D}, \delta \vdash_N c \xrightarrow{\lambda} d}{\mathrm{D}, \delta \vdash_N c; c' \xrightarrow{\lambda} d; c'} \qquad \frac{\mathrm{D}, \delta \vdash_N c \xrightarrow{\tau} c' \quad \mathrm{D}, \delta \vdash_N c' \xrightarrow{e} c''}{\mathrm{D}, \delta \vdash_N c \xrightarrow{e} c''}$$

$$\frac{\mathrm{D}, \delta \vdash_N c \xrightarrow{e} c'}{\delta \vdash_N \text{letrec}\,\mathrm{D}\text{ in } c \xrightarrow{e} \text{letrec}\,\mathrm{D}\text{ in } c'}$$

Fig. 12 Nondeterministic reduction (eager matching)

event e matching a we have $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} b$ and $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} b'$, but neither b nor b' can be reduced to Success or any other contract by the same event sequence. In reducing c we have not only resolved it against e, but also made a *decision*: whether to apply it to the first alternative of c or to the second. Technically, the reduction semantics is not closed under residuation: given c and e it is not always possible to find c' such that $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} c'$ and $\mathrm{D}; \delta \models e\backslash c = c'$. It is sound, however, in the sense that the reduct always denotes a subset of the residual trace set. It is furthermore complete in the sense that the set of all reductions do preserve residuation.

Theorem 3 (Soundness of eager matching)

1. *If* $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} c'$ *then* $\mathrm{D}, \delta \models c' \subseteq e\backslash c$.
2. *If* $\mathrm{D}, \delta \vdash_N c \xrightarrow{\tau} c'$ *then* $\mathrm{D}, \delta \models c' \subseteq c$.

Even though individual eager reductions do not preserve residuation, the set of all reductions does so:

Theorem 4 (Completeness of eager matching) *If* $\mathrm{D}, \delta \vdash_D c \xrightarrow{e} c'$ *then there exist contracts* $c_1, \ldots, c_n$ *for some* $n \geq 1$ *such that* $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} c_i$ *for all* $i = 1 \ldots n$ *and* $\mathrm{D}, \delta \models c' \subseteq \sum_{i=1}^n c_i$.

As a corollary, Theorems 3 and 4 combined yield that the object-level nondeterminism (expressed as contract alternatives) in the deferred matching semantics is faithfully reflected in the meta-level nondeterminism (expressed as multiple applicable rules) of the eager matching semantics.

3.7 Operational semantics III: eager matching with explicit routing

Consider the following execution model for contracts: two or more parties each have a copy of the contract they have previously agreed upon and monitor its execution under the arrival of events. Even if they agree on prior contract state and the next event, the parties may arrive at different residual contracts and thus different expectations as to the future events allowed under the contract. This is because of nondeterminacy in contract execution with eager matching; e.g., a payment of \$ 50 may match multiple payment commitments, and the parties may make different matches. We can remedy this by making *control* of contract reduction with eager matching explicit in order to make reduction deterministic: events are accompanied by control information that unambiguously prescribes how a contract is to be reduced. In this fashion parties that agree on what events have happened and on their associated control information will reduce their contract identically.[6]

The basic idea is that all nondeterminism in our reduction semantics (see Fig. 12) can be reduced to a series of choices and routing decisions to identify the particular commitment the event is to be matched with; in particular, we can express such a series as an element of

[6] The question of which party has the right of generating control information is very important, of course. It will be discussed only briefly later, as it is beyond the scope of this paper. We only require that a consensus on the events and their associated control information has been achieved, whether dictated by one party or the other having the (contractual) right to do so or by an actual consensus process.

Fig. 13 Eager matching with explicit reduction control

$$\mathrm{D}, \delta \vdash_C \text{Success} \xrightarrow{e} \text{Failure}$$

$$\mathrm{D}, \delta \vdash_C \text{Failure} \xrightarrow{e} \text{Failure}$$

$$\frac{\delta \oplus \{\mathbf{X} \mapsto \mathbf{v}\} \models P \quad (\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta})}{\mathrm{D}, \delta \vdash_C \text{transmit}(\mathbf{X} \mid P).\, c \xrightarrow{\text{transmit}(\mathbf{v})} c[\mathbf{v}/\mathbf{X}]}$$

$$\frac{\delta \oplus \{\mathbf{X} \mapsto \mathbf{v}\} \not\models P \quad (\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta})}{\mathrm{D}, \delta \vdash_C \text{transmit}(\mathbf{X} | P).\, c \xrightarrow{\text{transmit}(\mathbf{v})} \text{Failure}}$$

$$\frac{(f(\mathbf{X}) = c) \in \mathrm{D} \quad (\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta})}{\mathrm{D}, \delta \vdash_C f(\mathbf{a}) \xrightarrow{\tau} c[\mathbf{v}/\mathbf{X}]}$$

$$\frac{\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} d}{\mathrm{D}, \delta \vdash_C c + c' \xrightarrow{\tau} d + c'} \qquad \frac{\mathrm{D}, \delta \vdash_C c' \xrightarrow{\tau} d'}{\mathrm{D}, \delta \vdash_C c + c' \xrightarrow{\tau} c + d'}$$

$$\mathrm{D}, \delta \vdash_C \text{Success} + \text{Success} \xrightarrow{\tau} \text{Success}$$

$$\frac{\mathrm{D}, \delta \vdash_C c \xrightarrow{\mathbf{d}e} c'}{\mathrm{D}, \delta \vdash_C c + d \xrightarrow{\mathbf{fd}e} c'} \qquad \frac{\mathrm{D}, \delta \vdash_C d \xrightarrow{\mathbf{d}e} d'}{\mathrm{D}, \delta \vdash_C c + d \xrightarrow{\mathbf{sd}e} d'}$$

$$\frac{\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} d}{\mathrm{D}, \delta \vdash_C c \parallel c' \xrightarrow{\tau} d \parallel c'} \qquad \frac{\mathrm{D}, \delta \vdash_C c' \xrightarrow{\tau} d'}{\mathrm{D}, \delta \vdash_C c \parallel c' \xrightarrow{\tau} c \parallel d'}$$

$$\frac{\mathrm{D}, \delta \vdash_C c \xrightarrow{\mathbf{d}e} d}{\mathrm{D}, \delta \vdash_C c \parallel c' \xrightarrow{\mathbf{ld}e} d \parallel c'} \qquad \frac{\mathrm{D}, \delta \vdash_C c' \xrightarrow{\mathbf{d}e} d'}{\mathrm{D}, \delta \vdash_C c \parallel c' \xrightarrow{\mathbf{rd}e} c \parallel d'}$$

$$\mathrm{D}, \delta \vdash_C \text{Success} \parallel c \xrightarrow{\tau} c \qquad \mathrm{D}, \delta \vdash_C c \parallel \text{Success} \xrightarrow{\tau} c$$

$$\frac{\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} d}{\mathrm{D}, \delta \vdash_C c; c' \xrightarrow{\tau} d; c'} \qquad \frac{\mathrm{D}, \delta \vdash_C c \xrightarrow{e} d}{\mathrm{D}, \delta \vdash_C c; c' \xrightarrow{e} d; c'} \qquad \mathrm{D}, \delta \vdash_C \text{Success}; c' \xrightarrow{\tau} c'$$

I^* where $I = \{\mathrm{f}, \mathrm{s}, \mathrm{l}, \mathrm{r}\}$; see below. A control-annotated event then is an element of $I^*\mathcal{E}$. (Recall that $\mathcal{E}$ denotes the set of transfer events.) In Fig. 13 we note that $\mathbf{d} \in I^*$.

The τ-reductions in Fig. 13 rewrite a contract into a simplified form while preserving its semantics faithfully:

Proposition 2 (Soundness of τ-reduction) *For all* $\mathrm{D}, \delta, c, c'$, *if* $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} c'$ *then* $\mathrm{D}, \delta \models c = c'$.

Furthermore, they are strong enough to guarantee that any contract equivalent to Success actually reduces to Success.

Proposition 3 (Completeness of τ-reduction for concluded contracts) *For all* $\mathrm{D}, \delta, c, c'$: $\mathrm{D}, \delta \models c = \text{Success}$ *if and only if* $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau^*} \text{Success}$.

Finally, τ-rewriting is strongly normalizing and confluent, which means that each contract has a unique τ-normal form, which can be computed by applying the τ-rewriting rules exhaustively in arbitrary order.

Lemma 4 (Unique normalization of τ-reduction) *For all δ and guarded* D *there is a unique c' such that*

1. $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau^*} c'$.
2. *For no c'' do we have* $\mathrm{D}, \delta \vdash_C c' \xrightarrow{\tau} c''$.

We say c' in Lemma 4 is τ-*normalized* or simply *normalized* and we call it the τ-*normalized* form of c. We can observe that a contract is nullable if and only if its τ-normalized form has the form $\cdots + \text{Success} + \cdots$; that is, has a Success-summand.

The following theorem expresses that sequences of labels f, s, l, r preceding an economic event unambiguously determine how a contract should be reduced.

Theorem 5 (Correctness of eager matching with routing) *For each δ,* D*, normalized c and event e we have that* $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} c'$ *if and only if there exists* $\mathbf{d} \in \{\mathrm{f}, \mathrm{s}, \mathrm{l}, \mathrm{r}\}^*$ *such that* $\mathrm{D}, \delta \vdash_C c \xrightarrow{\mathbf{d}e} c'$. *Furthermore, for all c'' such that* $\mathrm{D}, \delta \vdash_C c \xrightarrow{\mathbf{d}e} c''$ *we have* $c' = c''$; *that is, given c and control-annotated event* $\mathbf{d}e$ *the residual contract c'' is uniquely determined.*

Intuitively, a control-annotated event $\mathbf{d}e$ conveys an event e and information $\mathbf{d}$ that unambiguously *routes* the event to the particular commitment it is to be matched with: f, s determine which branch of a . + .-contract is to be chosen, and l, r identify in which subcontract of a . ∥ .-contract the economic event is to be matched. This routing information ensures that all trading partners in a contract, each maintaining their own state of the contract, match events to the same atomic commitment and thus can be assured that they will also be in agreement on the residual contract. Other methods for controlling reduction in an eager matching semantics are discussed by Andersen and Elsborg [1].

Some of these left/right choices may be further eliminated in practice (i.e., inferred automatically) where they are "forced" (no other choice allows successful completion of contract).

4 Example contracts

We previously saw an encoding of the Agreement to Sell Goods (Fig. 4). In this section, two additional real-life example contracts are considered.

First, the previously presented abbreviated version of the natural language Legal Services Agreement (Fig. 2) is encoded in our contract specification language. Second, we present a natural language contract for software development (Fig. 15) and provide its encoding in our language (Fig. 16).

Before it is possible to express real-life contracts, however, the predicate language and the arithmetic language must be defined. For the purpose of demonstration we will afford ourselves a fairly advanced language that has multiple data types (e.g., integers and dates), common arithmetic operators, logical connectives, lists and a number of built-in functions. The syntax is common and straightforward, and hence we shall not delve into the technical details here. Later, in Sect. 5, we will define the language and consider possible restrictions that ameliorate contract analysis.

Writing the formal specification of the Legal Services Agreement (Fig. 2) is fairly straightforward, bar two points: Consider the validity period specified in Sect. 3 of the contract. Taken literally, it would imply that the attorney shall render services in the month of December, but receive no fee in consideration as January 2005 is outside the validity period. Surely, this is not the intention; in fact, consideration will defeat most deadlines as is clearly the intent here, and this is avoided in the encoding of the contract (Fig. 14). This weakness in the informal contract is revealed, which is a good thing, when encoding it formally.

The Agreement to Provide Legal Services fails to specify who decides if legal services should be rendered. In the encoding it is simply assumed that the attorney is the initiator and that all services rendered over a month can be modeled as one event. Based on the hours of services rendered, the attorney has a choice to invoice extra hours at the hourly rate. Furthermore, the attorney is assumed to give the notice `end` to allow contract termination. This is introduced to make sure that the contract is not nullable between every recursion.

Now consider the more elaborate Software Development Agreement in Fig. 15. When coding the contract, one notices that the contract fails to specify the ramifications of the client's nonapproval of a deliverable. One also sees that the contract does not specify what to do if due to delay, some approval deadline comes before the postponed delivery date. In the current code (Fig. 16), this is taken to mean further delay on the client's part even if the client gave approval at the same time as the deliverable was transmitted. It seems that contract coding is a healthy process in the sense that it will often unveil underspecification and errors in the natural language contract being coded. The Change Order described in Sect. 5 of the contract and the intellectual rights described in Sect. 6 are not coded due to certain limitations in our language. We will postpone the discussion of this until Sect. 6.

4.1 Example reduction

We now demonstrate how the Legal Services Agreement behaves under our three reduction strategies: deferred matching, eager matching and eager matching with explicit control. All three derivations assume that we invoke the contract as

```
legal (att, com, fee,
       invoice, pay, 0, 30, 60),
```

i.e., we would like it to be so that the contract is run for 2 months. Of course, the parameters `att`, `com`, `fee`, `invoice` and `pay` should be bound to values, but we leave them as is for readability as none of them have an impact on the control flow of the contract. This yields the contract body shown in Fig. 17. The subcontract `extra` has been taken out to reduce the size of the reductions. To facilitate comparison we will use the same basic event trace for all three reduction strategies:

$\xrightarrow{(att,com,h1,20)}$ Services rendered first month,

$\xrightarrow{(att,com,h2,37)}$ Services rendered second month,

$\xrightarrow{(com,att,fee,38)}$ Fee for first month,

$\xrightarrow{(com,att,fee,62)}$ Fee for second month,

$\xrightarrow{(att,com,end,64)}$ Attorney signals end-of-contract.

```
letrec
extra (att, com, invoice, pay) =
   ( Success
   + transmit (att, com, invoice, T2).
     transmit (com, att, pay, T3 | T3 <= T2 + 45d))

legal (att, com, fee, invoice, pay, n, m, end) =
    transmit (att, com, H, T | n < T and T <= m).
        (   extra (att, com, invoice, pay)
        ||  transmit (com att, fee, T | T <= m + 8d)
        ||  ( legal (att, com, fee, invoice, pay, m, min(m + 30d,end), end)
            + transmit (att, com, end, T | end <= T)))
in
  legal ("Attorney","Company",10000,invoice,pay,0,30,360)
```

Fig. 14 Specification of Agreement to Provide Legal Services

Section 1. The Developer shall develop software as described in Exhibit A (Requirements Specification) according the schedule set forth in Exhibit B (Project Schedule and Deliverables). Specifically, the Developer shall be responsible for the timely completion of the deliverables identified in Exhibit B.
Section 2. The Client shall provide written approval upon the completion of each deliverable identified in Exhibit B.
Section 3. In the event of any delay by the Client, all the Developer's remaining deadlines shall be extended by the greater of the two following: (i) five working days, (ii) two times the delay induced by the Client. The Client's deadlines shall be unchanged.
Section 4. In consideration of services rendered the Client shall pay USD $100.000 due on 7/1.
Section 5. If the Client wishes to add to the order, or if upon written approval of a deliverable, the Client wishes to make modifications to the deliverable, the Client and the Developer shall enter into a Change Order. Upon mutual agreement the Change Order shall be attached to this contract.
Section 6. The Developer shall retain all intellectual rights associated with the software developed. The Client may not copy or transfer the software to any third party without the explicit, written consent of the Developer.
Exhibit A. (omitted)
Exhibit B. Deadlines for deliverables and approval: (i) 1/1, 1/15; (ii) 3/1, 3/15, (final deadline) 7/1, 7/15.

Fig. 15 Software Development Agreement

```
letrec
    deliverables (dev, client, payment, deliv1, deadline1, approv1,
                                        deliv2, deadline2, approv2,
                                        delivf, deadlinef, approvf) =
      transmit(dev, client, deliv1, T1 | T1 <= deadline1)).
      transmit(client, dev, "ok", T).
      transmit(dev, client, deliv2, T2 |
               T2 <= deadline2 + max(5d, (T - approv1) * 2)).
      transmit(client, dev, "ok", T).
      transmit(dev, client, delivf, Tf |
               Tf <= deadlinef + max(5d, (T - approv2) * 2)).
      transmit(client, dev, "ok", T).
      transmit(dev, client, "done", T).
      Success

    software (dev, client, payment, paymentdeadline, ds) =
      deliverables (dev, client, deliv1, deadline1, approv1,
                                 deliv2, deadline2, approv2,
                                 delivf, deadlinef, approvf) ||
      transmit(client, dev, payment, T | T <= paymentdeadline)
in
  software ("Me", "Client", 100000, 2004.7.1, d1, 2004.1.1, 2004.1.15,
            d2, 2004.3.1, 2004.3.15, final, 2004.7.1, 2004.7.15)
```

Fig. 16 Specification of Software Development Agreement — note that we assume (easily defined) abbreviations for `max(x,y)` and allow subtraction on the domain Time

Fig. 17 Contract body of Legal Services Agreement

```
transmit (att, com, H, T | 0 < T and T <= 30).
(   transmit (com att, fee, T | T <= 30 + 8d)
||  ( legal (att, com, fee, invoice, pay, 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

```
transmit (att, com, H, T | 0 < T and T <= 30).
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  ( legal (..., 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

Services rendered first month:

$$\xrightarrow{(att,com,h1,20)}$$

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  ( legal (..., 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

Take the first branch in $+$ and unfold 'legal':

$$\xrightarrow{\tau}$$

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  (transmit (att, com, H, T | 30 < T and T <= 60).
     (   transmit (com, att, fee, T | T <= 60 + 8d)
     ||  ( legal (..., 60, min(60 + 30d,60), 60)
         + transmit (att, com, end, T | 60 <= T)))))
```

Services rendered second month:

$$\xrightarrow{(att,com,h2,37)}$$

The non-determinism is not constrained to viable options, but will allow any obviously wrong reduction to go wrong at any point. Assuming the desired outcome:

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  (   transmit (com, att, fee, T | T <= 60 + 8d)
    ||  ( legal (..., 60, min(60 + 30d,60), 60)
        + transmit (att, com, end, T | 60 <= T))))
```

The next event matches a transmit in the first iteration and a transmit in the second iteration. The contract could reduce properly or fail. We demonstrate the latter. Fee for first month:

$$\xrightarrow{(com,att,fee,38)}$$

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  (   Success
    ||  ( legal (..., 60, min(60 + 30d,60), 60)
        + transmit (att, com, end, T | 60 <= T))))
```

At time 39 the whole contract can terminate, because the $30 + 8d$ condition becomes unsatisfiable Assume that this possibility is exploited. Fee for second month:

$$\xrightarrow{(com,att,fee,62)}$$

Now, there is a serious problem. The choice of matching the first fee was unwise, and the limits of the eager matching semantics shows. The contract can now only fail.

```
(   Failure
||  (   Success
    ||  ( legal (..., 60, min(60 + 30d,60), 60)
        + transmit (att, com, end, T | 60 <= T))))
```

$$\xrightarrow{\tau}$$

```
Failure
```

```
transmit (att, com, H, T | 0 < T and T <= 30).
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  ( legal (..., 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

Services rendered first month:

$$\xrightarrow{(att,com,h1,20)}$$

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  ( legal (..., 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

We now take the first branch in $+$ and unfold 'legal'

$$\xrightarrow{\tau}$$

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  (transmit (att, com, H, T | 30 < T and T <= 60).
     (   transmit (com, att, fee, T | T <= 60 + 8d)
     ||  ( legal (..., 60, min(60 + 30d,60), 60)
         + transmit (att, com, end, T | 60 <= T)))))
```

Services rendered second month:

$$\xrightarrow{r(att,com,h2,37)}$$

We use explicit directives to point out the transmit we wish to match. Probably, the runtime system already suggested the options available and we picked one leaving the details to the system.

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  (   transmit (com, att, fee, T | T <= 60 + 8d)
    ||  ( legal (..., 60, min(60 + 30d,60), 60)
        + transmit (att, com, end, T | 60 <= T))))
```

Fee for first month

$$\xrightarrow{l(com,att,fee,38)}$$

$$\xrightarrow{\tau}$$

This event matches two different transmits, but the decision is taken "by" the directives:

```
(   transmit (com, att, fee, T | T <= 60 + 8d)
||  ( legal (..., 60, min(60 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

Fee for second month:

$$\xrightarrow{l(com,att,fee,62)}$$

$$\xrightarrow{\tau}$$

```
( legal (..., 60, min(60 + 30d,60), 60)
+ transmit (att, com, end, T | 60 <= T))
```

Attorney signals end-of-contract:

$$\xrightarrow{s(att,com,end,64)}$$

```
Success
```

Fig. 18 Eager matching without and with explicit control on the Legal Services Agreement

The trace will be furnished with reduction controls and interspersed with τ when mandated by the concrete semantics in question. Consider Fig. 18 for a juxtaposition of the two eager matching strategies (with and without explicit control) on the Legal Services Agreement and Fig. 19 for a demonstration of the deferred matching strategy.

```
transmit (att, com, H, T | 0 < T and T <= 30).
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  ( legal (..., 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

Services rendered first month:

$$\xrightarrow{(att,com,h1,20)}$$

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  ( legal (..., 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

Services rendered second month:

$$\xrightarrow{(att,com,h2,37)}$$

```
  (   Failure
  ||  ( legal (..., 30, min(30 + 30d,60), 60)
      + transmit (att, com, end, T | 60 <= T)))
+
  (   transmit (com, att, fee, T | T <= 30 + 8d)
  ||  (   (   transmit (com, att, fee, T | T <= 60 + 8d)
          ||  ( legal (..., 60, min(60 + 30d,60), 60)
              + transmit (att, com, end, T | 60 <= T)))
      + Failure))
```

Let us remove the failed parts, i.e. $C +$ Failure $\rightarrow C$ and $C \parallel$ Failure $\rightarrow$ Failure:

$$\xrightarrow{\tau}$$

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  (   transmit (com, att, fee, T | T <= 60 + 8d)
    ||  ( legal (..., 60, min(60 + 30d,60), 60)
        + transmit (att, com, end, T | 60 <= T))))
```

Fee for first month:

$$\xrightarrow{(com,att,fee,38)}$$

```
  (   Success
  ||  (   transmit (com, att, fee, T | T <= 60 + 8d)
      ||  ( legal (..., 60, min(60 + 30d,60), 60)
          + transmit (att, com, end, T | 60 <= T))))
+
  (   transmit (com, att, fee, T | T <= 30 + 8d)
  ||  (   Success
      ||  ( legal (..., 60, min(60 + 30d,60), 60)
          + transmit (att, com, end, T | 60 <= T))))
+
  (   transmit (com, att, fee, T | T <= 30 + 8d)
  ||  (   transmit (com, att, fee, T | T <= 60 + 8d)
      ||  ( Failure
          + Failure)))
```

And some more housecleaning, also Success $\parallel C \rightarrow C$:

$$\xrightarrow{\tau}$$

```
  (   transmit (com, att, fee, T | T <= 60 + 8d)
  ||  ( legal (..., 60, min(60 + 30d,60), 60)
      + transmit (att, com, end, T | 60 <= T)))
+
  (   transmit (com, att, fee, T | T <= 30 + 8d)
  ||  ( legal (..., 60, min(60 + 30d,60), 60)
      + transmit (att, com, end, T | 60 <= T)))
```

Two continuations are valid at time $T \leq 38$. The first has matched the first month's fee with the first iteration. The second represents matching the first fee with the second iteration. At time 39 the second branch can be rewritten to failure if our algorithm is able to decide that the condition $30 - 8d$ becomes unsatisfiable. But let us leave both branches for now and see what happens. Fee for second month:

$$\xrightarrow{(com,att,fee,62)}$$

This time let us skip the step where all non-matching branches get their own continuation, which is then removed immediately afterwards. Assume that we only attempt a match on the two transmits mentioning the fee:

```
  (   Success
  ||  ( legal (..., 60, min(60 + 30d,60), 60)
      + transmit (att, com, end, T | 60 <= T)))
+
  (   Failure
  ||  ( legal (..., 60, min(60 + 30d,60), 60)
      + transmit (att, com, end, T | 60 <= T)))
```

$$\xrightarrow{\tau}$$

```
( legal (..., 60, min(60 + 30d,60), 60)
+ transmit (att, com, end, T | 60 <= T)))
```

At this time a good failure algorithm would detect that the invocation of legal can be reduced to failure. Unfolding 'legal' gives predicates of the form $60 < T$ and $T \leq 60$ on all transmits, hence no event can match in 'legal'.
Attorney signals end-of-contract:

$$\xrightarrow{(att,com,end,64)}$$

```
Success
```

Technically, this is `Success || extra || extra` because we left out 'extra' during reduction. Events can still match the invoices (i.e. the attorney retains the right to invoice any extra hours of service previously rendered). The contract is terminable (nullable) at this point.

Fig. 19 Deferred matching on the Legal Services Agreement

5 Contract analysis

The formal groundwork in order, we can begin to ask ourselves questions about contracts such as: What is my first order of business? When is the next deadline? How much of a particular resource will I gain from my portfolio and at what times? What is the monetary value of my portfolio? Is the contract I just wrote "safe" and "fair"? Will contract fulfillment require more than the *x* units I currently have in stock?

The attempt to answer such questions is broadly referred to as *contract analysis*. Some analyses, notably "safeness", will primarily be of interest during the development of a contract, whereas other analyses apply to running contracts. The residuation property allows a contract analysis to be applied at any time (i.e., to any residual contract), and we can thus continuously monitor the execution of the contracts in our portfolio.

Recall that our contract specification language is parameterized over the language of predicates and arithmetic. There is a clear trade-off in play here: a sophisticated language buys expressiveness, but renders most of the analyses undecidable.

There is another source of difficulties. Variables may be bound to components of an event that is unknown at the time of analysis. An expression like transmit$(a_1, a_2, R, T|\text{true})$ offers little insight into the nature of R unless furnished with a probability vector over all resources.

Here we will circumvent these problems by making do with a restricted predicate language and accepting that analyses may not give answers on all input (but will give correct answers).

$$\frac{\Delta(var) = \sigma}{\Delta \vdash var : \sigma} \qquad \frac{\text{type}(const) = \sigma}{\Delta \vdash const : \sigma} \qquad \frac{\Delta \vdash e_1 : \text{Int} \quad \Delta \vdash e_2 : \text{Int} \quad \text{op} \in \{+, -, *, /\}}{\Delta \vdash e_1 \text{ op } e_2 : \text{Int}}$$

$$\frac{\Delta \vdash t : \text{Time} \quad \Delta \vdash e : \text{Int} \quad \text{f} \in \{\mathtt{y}, \mathtt{m}, \mathtt{d}\} \quad \text{op} \in \{+, -\}}{\Delta \vdash t \text{ op } e \text{ f} : \text{Time}} \qquad \frac{\Delta \vdash e : \text{Time} \quad \text{f} \in \{\mathtt{y}, \mathtt{m}, \mathtt{d}\}}{\Delta \vdash e\#\text{f} : \text{Int}}$$

$$\frac{\Delta \vdash r : \text{Resource} \quad \Delta \vdash t : \text{Time} \quad f \in fields(r)}{\Delta \vdash \#(r, f, t) : \text{Int}} \qquad \frac{\Delta \vdash e : \text{Int}}{\Delta \vdash e : \text{Resource}}$$

$$\frac{\Delta \vdash e_1 : \rho \quad \Delta \vdash e_2 : \rho}{\Delta \vdash e_1 < e_2 : \text{Boolean}} \qquad \frac{\Delta \vdash e_1 : \sigma \quad \Delta \vdash e_2 : \sigma}{\Delta \vdash e_1 = e_2 : \text{Boolean}}$$

$$\frac{\Delta \vdash b_1 : \text{Boolean} \quad \Delta \vdash b_2 : \text{Boolean} \quad \text{op} \in \{\mathtt{and}, \mathtt{or}\}}{\Delta \vdash b_1 \text{ op } b_2 : \text{Boolean}} \qquad \frac{\Delta \vdash b : \text{Boolean}}{\Delta \vdash \mathtt{not}\ b : \text{Boolean}}$$

Fig. 20 Example syntax for predicate language

The predicate language is plugged in at two locations. In function application $f(\mathbf{a})$ where all components of the vector $\mathbf{a}$ must be checked according to the rules of the predicate language and in transmit$(a_1, a_2, r, t|P)$ where P must have the type Boolean. As previously we require that a_1, a_2, r and t are either variables (bound or unbound) or constants. If some components are bound variables or constants, they must be equal to the corresponding components of an incoming event (a_1', a_2', r', t') for a match to occur.

Consider the syntax provided in Fig. 20. In addition to the types Agent, Resource and Time, the language has the fundamental types Int and Boolean. Take ρ to range over {Int, Time}, take σ to range over $\rho \cup$ {Agent, Resource}, and assume that constants can be uniquely typed (e.g., time constants are in ISO format, and agent and resource constants are disjoint and known).

The language allows arithmetic on integers, simple propositional logic and manipulation of the two abstract types Resource and Time. Given a time (date) t we may add an integral number of years, months or days. For example the expression 2004.1.1+3d+1y yields 2005.1.4. Resources permit a projection on a named component (field) and all fields are of type Int, e.g., to extract the total amount from an information resource named *invoice* we write #(*invoice*, *total*, t) where t is some date.[7] The fields of resources may change over time, hence the third parameter of type Time.

Observables can now be understood simply as fields of a ubiquitous resource named `obs`. An Int may double for a Resource in which case the Int is understood to be a currency amount.

5.1 Operational semantics II: eager matching

For the denotational semantics of the predicate language (Fig. 21) we define the following functions mapping syntactic expressions to mathematical objects:

$$\begin{aligned}
\mathcal{E}[\![const]\!] &= \lambda\delta \in \nabla.const \\
\mathcal{E}[\![var]\!] &= \lambda\delta \in \nabla.\delta(var) \\
\mathcal{E}[\![e_1 + e_2]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e_1]\!]\delta + \mathcal{E}[\![e_2]\!]\delta \\
\mathcal{E}[\![e_1 - e_2]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e_1]\!]\delta - \mathcal{E}[\![e_2]\!]\delta \\
\mathcal{E}[\![e_1 * e_2]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e_1]\!]\delta \cdot \mathcal{E}[\![e_2]\!]\delta \\
\mathcal{E}[\![e_1 / e_2]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e_1]\!]\delta \div \mathcal{E}[\![e_2]\!]\delta
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}[\![e\#\mathtt{d}]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e]\!]\delta \bmod 30 \\
\mathcal{E}[\![e\#\mathtt{m}]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e]\!]\delta \div 30 \bmod 12 \\
\mathcal{E}[\![e\#\mathtt{y}]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e]\!]\delta \div 360 \\
\mathcal{E}[\![e + f\ \mathtt{d}]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e]\!]\delta + (\mathcal{E}[\![f]\!]\delta)_t \\
\mathcal{E}[\![e + f\ \mathtt{m}]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e]\!]\delta + (\mathcal{E}[\![f]\!]\delta \cdot 30)_t \\
\mathcal{E}[\![e + f\ \mathtt{y}]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e]\!]\delta + (\mathcal{E}[\![f]\!]\delta \cdot 360)_t \\
\mathcal{E}[\![e - f\ \mathtt{d}]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e]\!]\delta - (\mathcal{E}[\![f]\!]\delta)_t \\
\mathcal{E}[\![e - f\ \mathtt{m}]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e]\!]\delta - (\mathcal{E}[\![f]\!]\delta \cdot 30)_t \\
\mathcal{E}[\![e - f\ \mathtt{y}]\!] &= \lambda\delta \in \nabla.\mathcal{E}[\![e]\!]\delta - (\mathcal{E}[\![f]\!]\delta \cdot 360)_t
\end{aligned}$$

$$\mathcal{E}[\![\#(r, f, t)]\!] = \lambda\delta \in \nabla.\varphi(\mathcal{E}[\![r]\!]\delta, f, \mathcal{E}[\![t]\!]\delta)$$

$$\begin{aligned}
\mathcal{B}[\![e_1 < e_2]\!] &= \lambda\delta \in \nabla.\begin{cases} t & \text{if } \mathcal{E}[\![e_1]\!]\delta < \mathcal{E}[\![e_2]\!]\delta \\ f & \text{otherwise} \end{cases} \\
\mathcal{B}[\![e_1 = e_2]\!] &= \lambda\delta \in \nabla.\begin{cases} t & \text{if } \mathcal{E}[\![e_1]\!]\delta = \mathcal{E}[\![e_2]\!]\delta \\ f & \text{otherwise} \end{cases} \\
\mathcal{B}[\![b_1 \mathtt{and}\ b_2]\!] &= \lambda\delta \in \nabla.\mathcal{B}[\![b_1]\!]\delta \wedge \mathcal{B}[\![b_2]\!]\delta \\
\mathcal{B}[\![b_1 \mathtt{or}\ b_2]\!] &= \lambda\delta \in \nabla.\mathcal{B}[\![b_1]\!]\delta \vee \mathcal{B}[\![b_2]\!]\delta \\
\mathcal{B}[\![\mathtt{not}\ b]\!] &= \lambda\delta \in \nabla.\neg\mathcal{B}[\![b]\!]\delta
\end{aligned}$$

Fig. 21 Denotational semantics for predicate language

[7] When a resource is introduced into the system through a match, it must be dynamically checked that it possesses the required fields. The set of required fields can be statically determined by a routine type check annotating resources with field names à la {*date*, *total*, *paymentdeadline*} Resource. To keep things simple we omit this type extension here.

Fig. 22 Failed contracts

$$\frac{\forall \delta', \forall t' \geq t : (\delta \oplus \delta' \oplus T \mapsto t' \models \neg P)}{\mathrm{D}, \delta, t \vdash \mathrm{transmit}(\mathbf{X}T \mid P).\, c \text{ failed}} \qquad \frac{\mathrm{D}, \delta, t \vdash c \text{ failed}}{\mathrm{D}, \delta, t \vdash \mathrm{transmit}(\mathbf{X}T \mid P).\, c \text{ failed}}$$

$$\mathrm{D}, \delta, t \vdash \text{Failure failed} \qquad \frac{\mathrm{D}, \delta, t \vdash c \text{ failed} \quad \mathrm{D}, \delta, t \vdash c' \text{ failed}}{\mathrm{D}, \delta, t \vdash c + c' \text{ failed}}$$

$$\frac{\mathrm{D}, \delta, t \vdash c \text{ failed}}{\mathrm{D}, \delta, t \vdash c \parallel c' \text{ failed}} \qquad \frac{\mathrm{D}, \delta, t \vdash c' \text{ failed}}{\mathrm{D}, \delta, t \vdash c \parallel c' \text{ failed}}$$

$$\frac{\mathrm{D}, \delta, t \vdash c \text{ failed}}{\mathrm{D}, \delta, t \vdash c; c' \text{ failed}} \qquad \frac{\mathrm{D}, \delta, t \vdash c' \text{ failed}}{\mathrm{D}, \delta, t \vdash c; c' \text{ failed}}$$

$$\frac{\mathrm{D}, \delta, t \vdash c \text{ failed} \quad (f(\mathbf{X}) = c) \in \mathrm{D}}{\mathrm{D}, \delta, t \vdash f(\mathbf{a}) \text{ failed}}$$

$$\mathcal{E} : Exp \rightarrow \nabla \rightarrow (\text{Agent} \cup \text{Resource} \cup \text{Int} \cup \text{Time}),$$
$$\mathcal{B} : Bexp \rightarrow \nabla \rightarrow \{t, f\},$$

where we assume the following mathematical environment:

- ∇ is the set of all possible bindings δ of variables to values.
- *Exp* is the set of all possible expressions of type Int, Time, Resource or *Agent* in the language.
- *Bexp* is the set of all possible expressions of type Boolean in the language.
- Resource and Agent are the sets of resources and agents, respectively.
- Int $= \mathbb{Z}$
- Time $= \{\ldots, -2_t, -1_t, 0_t, 1_t, 2_t, \ldots\}$ where operators $+$ and $-$ have the obvious interpretations, and we have the map $(\cdot)_t : \mathbb{Z} -> \text{Time}$ defined by $(n)_t = n_t$.
- Int $\subseteq$ Resource
- Agent, Resource, and Time are pairwise disjoint.
- (Agent $\cup$ Resource $\cup$ Int $\cup$ Time) is equipped with a (nontotal) order $<$ that is the union of the orders of the participating sets. Assume that Int and Time have the usual orderings.
- $\wedge$, $\vee$ and $\neg$ serve as logical operators with the usual meaning over the set $\{t, f\}$.
- If a and b are integers, $a \div b$ gives the largest integer c such that $c \cdot b \leq a$. mod is the corresponding modulo function so that $c \cdot b + a \bmod b = a$.
- φ : Resource $\times$ Field $\times$ Time $\rightarrow$ Int is a projection function on resources, and Field is a set of static field identifiers.

A contract analysis is a map from a syntactic description of a contract and some auxiliary information to a domain of our choice. The auxiliary information is often an agent or a point in time that the analysis should be relative to or an estimate of the probabilities associated with an underlying process. Ideally, a contract analysis can be performed *compositionally*. This section contains simple analyses with this property. Space considerations prevent a walkthrough of more involved examples, but the basic idea should be clear. We will assume for simplicity that recursively defined contracts are *guarded*. The analyses are presented using inference systems defined by induction on syntax, emphasizing the declarative and compositional nature of the analyses.

5.2 Example: failed contracts

A contract may accept a sequence of one of more events that is not a prefix of a performing trace. Thus the residual contract is failed and its denotation is the empty set — the contract is in an inconsistent state. The inference rules provided in Fig. 22 sketch how one could go about detecting this. The focal point is being able to decide if a predicate P can not hold true for any future values of its parameters. In practice, this often amounts to a simple argument: A deadline has been passed.

We have referred to the *failed* analysis numerous times in the example reductions. In Sect. 4 we saw that eager matching made a bad choice, which was not detected until much later. The failure analysis seeks to alleviate such situations as early as possible. Consider the scenario in Fig. 23 for an example under the eager matching regime. The failure of the contract is detected as soon as there is no remedy, i.e., at $T = 39$.

5.3 Example: task list

Given a contract or a portfolio of contracts it is tremendously important for an agent to know when and how to act. To this end we demonstrate how a very simple *task list* can be compiled.

Consider the definition given in Fig. 24. The function gives returns a list of outstanding commitments that can

```
transmit (att, com, H, T | 0 < T and T <= 30).
(  transmit (com, att, fee, T | T <= 30 + 8d)
|| ( legal (..., 30, min(30 + 30d,60), 60)
   + transmit (att, com, end, T | 60 <= T)))
```

$$\xrightarrow{\begin{array}{c}(att, com, h1, 20),\\ (att, com, h2, 37),\\ (com, att, fee, 38)\end{array}}$$

```
(  transmit (com, att, fee, T | T <= 30 + 8d)
|| (  Success
   || ( legal (..., 60, min(60 + 30d,60), 60)
      + transmit (att, com, end, T | 60 <= T))))
```

We would rather not wait for the next event $\xrightarrow{(com, att, fee, 62)}$ before realizing that the situation is not working. As soon as $T = 39$, `transmit (com att, fee, T | T <= 30 + 8d)` can transition to `Failure`. The relevant part of the derivation looks like this:

$$\frac{\dfrac{D, d, 39 \vdash 39 \leq 30 + 8d}{D, d, 39 \vdash \texttt{transmit (com att, fee, T | T <= 30 + 8d)}\ failed}}{D, d, 39 \vdash \begin{array}{l}\texttt{(transmit (com att, fee, T | T <= 30 + 8d)}\\ \texttt{|| (Success}\\ \texttt{ || (legal (..., 60, min(60 + 30d,60), 60)}\\ \texttt{   + transmit (att, com, end, T | 60 <= T))))}\end{array}\ failed}$$

Fig. 23 Example: failed Legal Services Agreement under eager matching (nondeterministic)

$$\mathrm{D}, \delta, \mathrm{a}, t \vdash \text{Success} \ : \ [] \qquad \mathrm{D}, \delta, \mathrm{a}, t \vdash \text{Failure} \ : \ []$$

$$\frac{\models a \neq a_1 \quad \mathbf{X} = (a_1, a_2, R, T)}{\mathrm{D}, \delta, \mathrm{a}, t \vdash \text{transmit}(\mathbf{X} \mid x \leq T \textbf{ and } T \leq y).\, c \ : \ []} \qquad \frac{\models \neg(x \leq t \textbf{ and } t \leq y)}{\mathrm{D}, \delta, \mathrm{a}, t \vdash \text{transmit}(\mathbf{X} \mid x \leq T \textbf{ and } T \leq y).\, c \ : \ []}$$

$$\frac{\models a = a_1 \quad \mathbf{X} = (a_1, a_2, R, T) \quad \models x \leq t \textbf{ and } t \leq y}{\mathrm{D}, \delta, \mathrm{a}, t \vdash \text{transmit}(\mathbf{X} \mid x \leq T \textbf{ and } T \leq y).\, c \ : \ [\text{transmit}(\mathbf{X} \mid x \leq T \textbf{ and } T \leq y).\, c]}$$

$$\frac{\mathrm{D}, \delta, \mathrm{a}, t \vdash c_1 \ : \ l_1 \quad \mathrm{D}, \delta, \mathrm{a}, t \vdash c_2 \ : \ l_2}{\mathrm{D}, \delta, \mathrm{a}, t \vdash c_1 + c_2 \ : \ l_1 \mathbin{@} l_2}$$

$$\frac{\mathrm{D} \vdash c_1 \text{ nullable} \quad \mathrm{D}, \delta, \mathrm{a}, t \vdash c_1 \ : \ l_1 \quad \mathrm{D}, \delta, \mathrm{a}, t \vdash c_2 \ : \ l_2}{\mathrm{D}, \delta, \mathrm{a}, t \vdash c_1 ; c_2 \ : \ l_1 \mathbin{@} l_2}$$

$$\frac{\mathrm{D} \not\vdash c_1 \text{ nullable} \quad \mathrm{D}, \delta, \mathrm{a}, t \vdash c_1 \ : \ l_1}{\mathrm{D}, \delta, \mathrm{a}, t \vdash c_1 ; c_2 \ : \ l_1} \qquad \frac{\mathrm{D}, \delta, \mathrm{a}, t \vdash c_1 \ : \ l_1 \quad \mathrm{D}, \delta, \mathrm{a}, t \vdash c_2 \ : \ l_2}{\mathrm{D}, \delta, \mathrm{a}, t \vdash c_1 \parallel c_2 \ : \ l_1 \mathbin{@} l_2}$$

$$\frac{(f(\mathbf{X}) = c) \in \mathrm{D} \quad \mathrm{D}, \delta, \mathrm{a}, t \vdash c \ : \ l}{\mathrm{D}, \delta, \mathrm{a}, t \vdash f(\mathbf{a}) \ : \ l}$$

Fig. 24 Task list analysis

be carried out at time t. We only admit interval conditions of the form $a \leq T$ and $T \leq b$ with T being the time variable in the enclosing transmit, as in "real" contracts hardly anything else is used. It is important to notice that the result of the analysis may be incomplete. A task is only added if the agents agree (i.e., `a=a1`), but if `a1` is not bound at the time t of analysis, the task is simply skipped. A more elaborate dataflow analysis might reveal that in fact `a1` is always bound to `a`.

Also notice the case for application $f(\mathbf{a})$. We expand the body of the named contract `f` given arguments `a` but only once (assuming f is guarded). This measure ensures termination of the analysis, but reduces the function's look-ahead horizon. Hence, any task or point of interest more than one recursive unfolding away is not detected. This is unlikely to have practical significance for two reasons: (1) recursively defined contracts are guarded and so a `transmit` must be matched before a new unfold can occur. This `transmit` therefore is presumably more relevant than any other `transmits` further down the line; (2) it would be utterly unidiomatic if some `transmit` t_1 was required to be matched before another `transmit` t_2, but nevertheless had a later deadline than that of t_2.

For an example of the task list analysis, we return to the Legal Services Agreement. The task list works best with eager matching with explicit reduction control. Eager matching alone is too careless, and deferred matching represents many states, which are all assumed valid, but may confuse the user when he or she sees overlapping tasks for every hypothetical state of the contract. Consider Fig. 25 for an example of how the task list evolves under reduction of the Legal Services Agreement.

Further analyses that are possible to implement in this way include:

- Resource flow forecasting (supply requirements).
- Terminability by agent, latest termination, earliest termination.

```
transmit (att, com, H, T | 0 < T and T <= 30).
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  ( legal (..., 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

$T = 0$:
```
att: transmit (att, com, H, T | 0 < T and T <= 30)
```

Services rendered first month:

$$\xrightarrow{(att,com,h1,20)}$$

$$\xrightarrow{\tau}$$

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  (transmit (att, com, H, T | 30 < T and T <= 60).
     (   transmit (com, att, fee, T | T <= 60 + 8d)
     ||  ( legal (..., 60, min(60 + 30d,60), 60)
         + transmit (att, com, end, T | 60 <= T)))))
```

$T = 20$:
```
com: [transmit (com att, fee, T | T <= 30 + 8d)]
```

$T = 31$:
```
att: transmit (att, com, H, T | 30 < T and T <= 60)
com: transmit (com att, fee, T | T <= 30 + 8d)
```

Services rendered second month:

$$\xrightarrow{r(att,com,h2,37)}$$

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  (   transmit (com, att, fee, T | T <= 60 + 8d)
    ||  ( legal (..., 60, min(60 + 30d,60), 60)
        + transmit (att, com, end, T | 60 <= T))))
```

$T = 37$:
```
com: transmit (com att, fee, T | T <= 30 + 8d)
com: transmit (com att, fee, T | T <= 60 + 8d)
```

Assuming the system was unable to decide predicates, two additional tasks would have been shown for `att`:

```
att: transmit (att, com, H, T | 60 < T and T <= 60)
att: transmit (att, com, end, T | 60 <= T)
```

Fee for first month:

$$\xrightarrow{l(com,att,fee,38)}$$

$$\xrightarrow{\tau}$$

```
(   transmit (com, att, fee, T | T <= 60 + 8d)
||  ( legal (..., 60, min(60 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

$T = 38$:
```
com: transmit (com att, fee, T | T <= 60 + 8d)
```

$T = 60$:
```
att: transmit (att, com, end, T | 60 <= T)
com: transmit (com att, fee, T | T <= 60 + 8d)
```

Fee for second month:

$$\xrightarrow{l(com,att,fee,62)}$$

$$\xrightarrow{\tau}$$

```
( legal (..., 60, min(60 + 30d,60), 60)
+ transmit (att, com, end, T | 60 <= T))
```

$T = 62$:
```
att: transmit (att, com, end, T | 60 <= T)
```

Attorney signals end-of-contract:

$$\xrightarrow{s(att,com,end,64)}$$

```
Success
```

$T \geq 64$: `No tasks!`

Fig. 25 Task list for the Legal Services Agreements under eager matching with explicit control

– Valuation, or simply put: What is the value to an agent of a given contract? The analysis is fairly intricate and requires knowledge of financial models and stochastic processes. Interested readers are referred to Peyton Jones and Eber [11,12] who provide a very readable introduction targeted at computer scientists.
– General model checking for business rules: (a) static, (b) dynamic/runtime (Timed LTL checking), cf. [14].

6 Discussion and future work

Our definition of contracts focuses on contracts as classifiers of event traces into performing and nonperforming ones. This is coarse, and many real-world issues are left out here.

The basic idea is to develop these notions within a general framework that may require specifications of runtime environment and protocols for event transmission. The inclusion of explicit operators in the language to mimic many standard steps in the contract lifecycle—say checking a contract for potential problems with current law—would not facilitate easy contract coding without both static ("does this contract conform to standard practice?") and dynamic ("is this sequence of events and their handling proper?") checks appealing to some enclosing structures.

We decided to pursue compositionality—hierarchical specification—from the outset as a central notion and thus follow a process algebra approach, basically to evaluate how far that would take us in the given domain. This can be contrasted to a network-oriented approach supported by suitable diagramming to appeal to visual faculties, which appears to be the preferred modeling approach for workflow systems (Petri nets) [21] and in object-oriented analysis (UML diagramming). Note that hierarchical specification is also needed in a network-oriented approach to achieve modular description and reuse of specification components. Furthermore, powerful specification mechanisms such as functional abstraction and (nontail) recursion have no simple visual representations.

The Software Development Agreement (Fig. 15) provides a good setting to observe the limitations to our approach and the ramifications of the design choices made.

The Change Order is not coded. It might be cleverly coded in the current language, again using constraints on the events passed around, but a more natural way would be using higher-order contracts, i.e., contracts taking contracts as arguments. Thus, a Change Order would simply be the passing back and forth of a contract followed by an instantiation upon agreement.

The transmission of rights can easily be coded, but the prohibition to transmit a particular resource affects all other contracts. Currently, we have no construct available to handle this situation.

Contracts often specify certain things that are not to be done (e.g., not copying the software). Such restrictions should intersect all other outstanding contracts and limit them appropriately. A higher-order language or predicates that could guard all `transmit`s of an entire subcontract might ameliorate this in a natural way.

A fuller range of language constructions that programmers are familiar with is also desirable; in the present incarnation of the contract language, several standard constructions have been left out in order to emphasize the core event model. In practice, conditionals and various sorts of lambda abstractions would make the language easier to use, though not strictly more expressive, as they can be encoded through events. A conditional that is *not* driven by events (i.e., an if–then–else) seems to be needed for natural coding in many real-world contracts. Also, a catch–throw mechanism for unexpected events would make contracts more robust.

Conversely, certain features of the language appear to be almost too strong for the domain; the inclusion of full recursion means that contracts active for an unlimited period of time, say leases, are easy to code, but make contract analysis significantly harder. In practice, contracts running for "unlimited" time periods often have external constraints (usually local legislation) forcing the contract to be reassessed by its parties, and possibly government representatives, from time to time. Having only a restricted form of recursion that suffices for most practical applications should simplify contract analysis.

The expressivity of the contract language and indeed the feasibility of non–trivial contract analysis depends heavily on the predicate language used. Predicates restricted to the form $[a; b]$ are surely too limited, and further investigation into the required expressiveness of the predicate language is desirable.

While the language is parametrized over the predicate language used, almost all real-world applications will require some model of time and timed events to be incorporated *vis-à-vis* the examples using interval in Sect. 5 . The current event model allows for encoding through the predicate language, but an extended set of events, with companion semantics, would make for easier contract programming; timer (or "trigger") events appear to be ubiquitous when encoding contracts.

7 Related work

The impetus for this work comes from two directions: the REA accounting model pioneered by McCarthy [15] and Peyton Jones, Eber and Seward's seminal article on specification of financial contracts [12]. Furthermore, given that contracts specify protocols as to how parties bound by them are to interact with each other there are links to process and workflow models.

7.1 Composing contracts

Peyton Jones, Eber and Seward [12] present a compositional language for specifying financial contracts. It provides a decomposition of known standard contracts such as zero coupon bonds, options, swaps, straddles, etc., into individual payment commitments that are combined declaratively using a small set of contract combinators. All contracts are two-party contracts, and the parties are implicit. The combinators (taken from [11], revised from [12]) correspond to Success, $\cdot \parallel \cdot$, $\cdot + \cdot$, transmit$(\cdot)$ of our language $\mathcal{C}^{\mathcal{P}}$; it has no direct counterparts to Failure, $\cdot;\cdot$ nor, most importantly, recursion or iteration. On the other hand, it provides conditionals and predicates that are applicable to arbitrary contracts, not just commitments as in $\mathcal{C}^{\mathcal{P}}$, something we have found to be worthwhile also for specifying commercial contracts. Furthermore, their language provides an `until`-operator that allows a party to terminate a contract successfully at a particular time, even if not all commitments have been satisfied. Using `until` for contract specification seems difficult, however, since it may—legally—cut off contract execution before all reciprocal commitments have been satisfied, e.g., the requirement to pay for a service that has been rendered.

Our contract language generalizes financial payment commitments to arbitrary transfers of resources and information, provides explicit agents and thus provides the possibility of specifying multi-party contracts.

We have provided a denotational semantics for $\mathcal{C}^{\mathcal{P}}$ and developed operational semantics for contract monitoring from it, whereas Peyton Jones, Eber and Seward focus on *valuation*, a sophisticated contract analysis based on stochastic analysis for pricing contracts.

7.2 Resources/Events/Agents (REA)

McCarthy [15] pioneered REA, an accounting model that focuses on the basic transaction patterns of the enterprise, the exchange of scarce goods and the transformation of resources by production and separates it from phenomena that can be derived by aggregation or other means. Geerts and McCarthy [8] complement REA's entity-relationship model of basic ex post notions of *events*, in which *agents* transmit scarce *resources*, with ex ante notions: commitments and sets of commitments making up contracts.[8] Contracts, however, are only modeled as sets of commitments whose concrete terms and constraints are usually described in natural language and as such live outside the scope of the entity-relationship model. Our work provides a formalization for contracts and their (performing) executions and thus complements the REA's data-centered notions with a well-defined process perspective.

7.3 Process algebra and logic

Disregarding the structure of events and their temporal properties, $\mathcal{C}^{\mathcal{P}}$ is basically a process algebra. It corresponds to Algebra of Communicating Processes (ACP) with deadlock (Failure), free merge ($\cdot \parallel \cdot$) and recursion, but without encapsulation [4]. Note that contracts are to be thought of as exclusively *reactive* processes, however, they respond to externally generated events, but do not autonomously generate them. This leads naturally to contracts classifying event traces, making CSP [5,10] and its trace-theoretic semantics a natural conceptual framework for our view-independent approach to contract specification. This is in contrast to CCS-like process calculi [9,16,17], which take a rather operational process-as-machine view; they treat communication as dual pairs of send and receive messages and allow observation of branching decisions in processes. Note that $\mathcal{C}^{\mathcal{P}}$, as presented here, contains no synchronization between concurrently executing subcontracts. A previous version of $\mathcal{C}^{\mathcal{P}}$ contained the contract conjunction operator $c\ \&\ c'$, whose denotational semantics is

$$\mathcal{C}[\![c\ \&\ c']\!]^{\mathrm{D};\delta} = \mathcal{C}[\![c]\!]^{\mathrm{D};\delta} \cap \mathcal{C}[\![c']\!]^{\mathrm{D};\delta}.$$

This is the parallel composition operator of CSP with synchronization at each step. A trace satisfies $c\ \&\ c'$ if it satisfies both c and c'. This makes it possible to specify a contract by providing a *basic* specification, c (sales order) and refining it by conjoining it with an additional *policy*, c' (no alcohol must be sold to minors), that a correct contract execution must satisfy. Our language can be extended to include contract conjunction. We have not included it here to keep the theoretical treatment of $\mathcal{C}^{\mathcal{P}}$ simple. Furthermore, it is our impression that the above asymmetry of c specifying the fundamental protocol for contract execution and c' filtering illegal executions may be better captured by formulating policies logically, e.g., in Linear-Time Logic (LTL), possibly enforced by run-time verification [13].

There are numerous timed variants of process algebras and temporal logics; see, e.g., Baeten and Middelburg [3] for timed process algebras. It should be noted that our contract language is fundamentally deterministic to avoid misunderstanding between contract partners: by design, nondeterministic implicit control decisions as in CCS-based process calculi are avoided. Indeed the eager matching semantics presented can be considered a process language with implicit control decisions (a process may evolve nondeterministically and autonomously). As this is considered undesirable in our context (though realistic as it reflects the matching ambiguities common in bookkeeping), its events (actions in process terminology) are enhanced with control ("routing") information to control process/contract evolution deterministically.

Note that, in contrast to conventional process calculi, we have included both sequential composition and parameterized recursion to support a separation of data (the base language) and control (the contract language).

Also, our base language is not fixed, but a parameter of the contract language so as to accommodate expressing temporal (and other) constraints modularly and "naturally". Indeed, the basic structure of events can be entirely encapsulated in the base language, making the technical development of the contract language (the "control part") independent of REA or other data models for that matter.

Timed process calculi tend to build on rudimentary models of time. These appear to be insufficient for expressing contract constraints naturally, but may turn out to be viable as core languages. Clearly, studying timing more closely as well as other connections to process calculi constitutes requisite future work.

Finally, most of the extant process algebras apparently do not consider the approach of contract monitoring by residuation. In this paper, the need for considering (prefixes of) *event traces* leads to the problem of allowing

[8] This is a highly simplified description of key parts of REA.

only contracts that ensure that the arrival of any event leads to a well-defined residual contract. Calculi such as CCS do not have a notion of *event* traces and do not encounter the problem, as the (structural) operational semantics turns out to be sound and complete for the set of structural equivalences defining a "program" in CCS. The main difference seems to be the liberal recursion operator employed in our language which admits mutual recursion, unlike CCS where the constructs of equal strength only admit transitions that are *syntactically* guarded in the sense that if an operator has a transition to a new term, the root of that term contains an operator of "lower" strength (e.g., the "replication" operator is guarded by the "parallel" operator in CCS).

7.4 Work flow and business process languages

In [19] an *event algebra* is developed which is used to monitor a discrete event system. The terms of the algebra contain the equivalent of Success, Failure, $\cdot \parallel \cdot, \cdot + \cdot, \cdot ; \cdot$ while the atomic contract transmit$(\cdot)$. $\cdot$ is replaced by an enumerated set of unique atomic constructs with no free variables. Iteration is stated to be done by instantiating terms such that atomic terms are relabeled to ensure uniqueness of all atomic terms. A trace semantics is given for terms as well as *residuation equations*. The equations allow monitoring of terms by a syntactic method as in $\mathcal{C}^{\mathcal{P}}$. Guardedness (in the sense of $\mathcal{C}^{\mathcal{P}}$) is guaranteed by excluding recursion from the language. It is not entirely clear how iteration is included in the language as no formal description of it is given. The residuation equations given essentially implement the *eager* semantics of $\mathcal{C}^{\mathcal{P}}$.

Another branch of research has focused on the specification and modelling of business processes. In this vein, the *Business Process Modeling Language* (BPML) is an XML-inspired specification language defined by a consortium of agents from industry and reported in several white papers and technical reports [2,20]. A "program" in the language is, essentially, an XML schema containing process specifications, including temporal and conditional statements, as well as a restricted iteration construct ("repeat"). The scope of entities that can reasonably be modelled by BPML is conceptually larger than the one considered in this paper, as arbitrary (internal or external) processes and commitments can be modeled — hence also contractual obligations. However, while the language operates with an execution model loosely based on π-calculus [18], a proper (and formal) semantics for process execution, performance and monitoring is lacking. The semantics of the framework is currently described only in terms of natural language, and any kind of *safe* automated or formal analysis of execution of processes specified in the language thus cannot be performed at present.

Acknowledgments This work has been partially funded by the NEXT Project, which is a collaboration between Microsoft Business Solutions, The IT University of Copenhagen (ITU) and the Department of Computer Science at the University of Copenhagen (DIKU). See `http://www.itu.dk/next` for more information on NEXT. We would like to thank Simon Peyton Jones and Jean-Marc Eber for valuable discussions on modeling financial contracts. Kasper Østerbye, Jesper Kiehn and the members of the NEXT Working Group have provided helpful comments and feedback on extending the work of Peyton Jones and Eber to commercial contracts based on the REA accounting model. Indeed, Kasper has worked out similar ideas on representing contracts as ours, but in an object-oriented setting.

Appendix: Full proofs

Proof (Theorem 1) Let $\mathrm{D} = \{f_i[\mathbf{X_i}] = c_i\}_{i=1}^m$ and δ be given. We prove

$$\mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\delta'} = \{s : \delta' \vdash^{\delta}_{\mathrm{D}} s : c\},$$

where $\gamma = \mathcal{D}[\![\mathrm{D}]\!]^{\delta}$.

"$\supseteq$": Define $\delta' \models^{\delta}_{\mathrm{D}} s : c \iff s \in \mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\delta'}$. We prove by induction on the derivation of $\delta' \vdash^{\delta}_{\mathrm{D}} s : c$ that $\delta' \models^{\delta}_{\mathrm{D}} s : c$.

$\boxed{\delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : \text{Success}}$ We need to show that $\delta' \models^{\delta}_{\mathrm{D}} \langle\rangle : \text{Success}$. This follows immediately from $\mathcal{C}[\![\text{Success}]\!]^{\gamma;\delta\oplus\delta'} = \{\langle\rangle\}$.

$\boxed{\dfrac{\mathbf{X}\mapsto\mathbf{v}\vdash^{\delta}_{\mathrm{D}} s:c \quad (f(\mathbf{X})=c)\in\mathrm{D}, \mathbf{v}=\mathcal{Q}[\![\mathbf{a}]\!]^{\delta\oplus\delta'}}{\delta'\vdash^{\delta}_{\mathrm{D}} s:f(\mathbf{a})}}$ Assume $\mathbf{X} \mapsto \mathbf{v} \models^{\delta}_{\mathrm{D}} s : c$ (induction hypothesis) with $\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta\oplus\delta'}$ and $(f(\mathbf{X}) = c) \in \mathrm{D}$. We need to show that $\delta' \models^{\delta}_{\mathrm{D}} s : f(\mathbf{a})$.
By definition we have
$\mathcal{C}[\![f(\mathbf{a})]\!]^{\gamma;\delta\oplus\delta'} = \gamma(f)(\mathcal{Q}[\![\mathbf{a}]\!]^{\delta\oplus\delta'})$
(by def. of $\mathbf{v}$) $= \gamma(f)(\mathbf{v})$
(by def. of γ) $= \mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}}$
and thus, as $\mathbf{X} \mapsto \mathbf{v} \models^{\delta}_{\mathrm{D}} s : c$ by induction hypothesis, we can conclude that $\delta' \models^{\delta}_{\mathrm{D}} s : f(\mathbf{a})$.

$\boxed{\dfrac{\delta\oplus\delta''\models P \quad \delta''\vdash^{\delta}_{\mathrm{D}} s:c \qquad (\delta''=\delta'\oplus\{\mathbf{X}\mapsto\mathbf{v}\})}{\delta'\vdash^{\delta}_{\mathrm{D}} \text{transmit}(\mathbf{v})\, s:\text{transmit}(\mathbf{X}|P).\, c}}$ Assume $\delta \oplus \delta'' \models P$ and $\delta'' \models^{\delta}_{\mathrm{D}} s : c$ where $\delta'' = \delta' \oplus \{\mathbf{X} \mapsto \mathbf{v}\}$. We need to show that $\delta' \models^{\delta}_{\mathrm{D}} \text{transmit}(\mathbf{v})\, s : \text{transmit}(\mathbf{X}|P).\, c$.
As $\delta \oplus \delta'' \models P$ and $\delta'' \models^{\delta}_{\mathrm{D}} s : c$ it follows immediately from the definition of $\mathcal{C}[\![\text{transmit}(\mathbf{X} \mid P).\, c]\!]^{\gamma;\delta\oplus\delta'}$ that $\delta' \models^{\delta}_{\mathrm{D}} \text{transmit}(\mathbf{v})\, s : \text{transmit}(\mathbf{X}|P).\, c$.

$\boxed{\dfrac{\delta'\vdash^{\delta}_{\mathrm{D}} s_1:c_1 \quad \delta'\vdash^{\delta}_{\mathrm{D}} s_2:c_2 \quad (s_1,s_2)\rightsquigarrow s}{\delta'\vdash^{\delta}_{\mathrm{D}} s:c_1 \parallel c_2}}$ Assume $\delta' \models^{\delta}_{\mathrm{D}} s_1 : c_1$, $\delta' \models^{\delta}_{\mathrm{D}} s_2 : c_2$ and $(s_1, s_2) \rightsquigarrow s$. We need to show that $\delta' \models^{\delta}_{\mathrm{D}} s : c_1 \parallel c_2$.
From the assumptions and the definition of $\mathcal{C}[\![c_1 \parallel c_2]\!]^{\gamma;\delta\oplus\delta'}$ it follows immediately that $\delta' \models^{\delta}_{\mathrm{D}} s : c_1 \parallel c_2$.

$\boxed{\dfrac{\delta'\vdash^{\delta}_{\mathrm{D}} s_1:c_1 \quad \delta'\vdash^{\delta}_{\mathrm{D}} s_2:c_2}{\delta'\vdash^{\delta}_{\mathrm{D}} s_1 s_2:c_1;c_2}}$ Immediate from the definition of $\mathcal{C}[\![c_1; c_2]\!]^{\gamma;\delta\oplus\delta'}$.

$\boxed{\frac{\delta' \vdash^{\delta}_{\mathrm{D}} s{:}c_1}{\delta' \vdash^{\delta}_{\mathrm{D}} s{:}c_1+c_2}}$ Immediate from the definition of $\mathcal{C}[\![c_1 + c_2]\!]^{\gamma;\delta\oplus\delta'}$.

$\boxed{\frac{\delta' \vdash^{\delta}_{\mathrm{D}} s{:}c_2}{\delta' \vdash^{\delta}_{\mathrm{D}} s{:}c_1+c_2}}$ Immediate from the definition of $\mathcal{C}[\![c_1 + c_2]\!]^{\gamma;\delta\oplus\delta'}$.

"$\subseteq$": We prove $\mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\delta'} \subseteq \{s \mid \delta' \vdash^{\delta}_{\mathrm{D}} s : c\}$.

Define $\gamma'(f_i) = \lambda\mathbf{v}.\{s \mid \mathbf{X}_i \mapsto \mathbf{v} \vdash^{\delta}_{\mathrm{D}} s : c_i\}$ for $1 \le i \le m$. (Recall that δ is fixed.)

Claim: $\mathcal{C}[\![c]\!]^{\gamma';\delta\oplus\delta'} = \{s \mid \delta' \vdash^{\delta}_{\mathrm{D}} s : c\}$ for all δ'.

Proof of claim: The proof is by structural induction on c.

– Consider $f(\mathbf{a})$ for some $(f(\mathbf{X}) = c) \in \mathrm{D}$. Let $\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta\oplus\delta'}$. We need to show that $\mathcal{C}[\![f(\mathbf{a})]\!]^{\gamma';\delta\oplus\delta'} = \{s \mid \delta' \vdash^{\delta}_{\mathrm{D}} s : f(\mathbf{a})\}$. We have:

$$\begin{aligned}\mathcal{C}[\![f(\mathbf{a})]\!]^{\gamma';\delta\oplus\delta'} &= \gamma'(f)(\mathcal{Q}[\![\mathbf{a}]\!]^{\delta\oplus\delta'})\\ &= \gamma'(f)(\mathbf{v})\\ &= \{s \mid \mathbf{X} \mapsto \mathbf{v} \vdash^{\delta}_{\mathrm{D}} s : c\}\\ &= \{s \mid \delta' \vdash^{\delta}_{\mathrm{D}} s : f(\mathbf{a})\},\end{aligned}$$

which concludes this case.

– Consider $\mathrm{transmit}(\mathbf{X} \mid P).\,c$. We may assume $\mathcal{C}[\![c]\!]^{\gamma';\delta\oplus\delta'} = \{s \mid \delta' \vdash^{\delta}_{\mathrm{D}} s : c\}$ for all δ'. We need to show that $\mathcal{C}[\![\mathrm{transmit}(\mathbf{X} \mid P).\,c]\!]^{\gamma';\delta\oplus\delta'} = \{s \mid \delta' \vdash^{\delta}_{\mathrm{D}} s : \mathrm{transmit}(\mathbf{X} \mid P).c\}$. We have:

$$\begin{aligned}&\mathcal{C}[\![\mathrm{transmit}(\mathbf{X} \mid P).\,c]\!]^{\gamma';\delta\oplus\delta'}\\ &= \{\mathrm{transmit}(\mathbf{v})\, s \mid \mathcal{Q}[\![P]\!]^{\delta\oplus\delta'\oplus\mathbf{X}\mapsto\mathbf{v}}\\ &= \mathrm{true} \wedge s \in \mathcal{C}[\![c]\!]^{\gamma';\delta\oplus\delta'\oplus\mathbf{X}\mapsto\mathbf{v}}\}\\ &= \{\mathrm{transmit}(\mathbf{v})\, s \mid \delta\oplus\delta'\oplus\mathbf{X}\mapsto\mathbf{v} \models P \wedge \delta'\oplus\delta'' \vdash^{\delta}_{\mathrm{D}} s : c\}\\ &= \{s' \mid \delta' \vdash^{\delta}_{\mathrm{D}} s' : \mathrm{transmit}(\mathbf{X} \mid P).\,c\},\end{aligned}$$

which concludes this case.

– The remaining cases are straightforward.

From $\mathcal{C}[\![c]\!]^{\gamma';\delta\oplus\delta'} = \{s \mid \delta' \vdash^{\delta}_{\mathrm{D}} s : c\}$ for all δ' follows immediately that $\gamma'(f) = \lambda\mathbf{v}.\mathcal{C}[\![c]\!]^{\gamma';\delta\oplus\mathbf{X}\mapsto\mathbf{v}}$ for all $(f(\mathbf{X}) = c) \in \mathrm{D}$. Since $\gamma = \mathcal{D}[\![\mathrm{D}]\!]^{\delta}$ is the least function with this property, it follows that $\gamma \sqsubseteq \gamma'$ and thus $\mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\delta'} \subseteq \mathcal{C}[\![c]\!]^{\gamma';\delta\oplus\delta'} = \{s \mid \delta' \vdash^{\delta}_{\mathrm{D}} s : c\}$ and we are done.

□

Proof (Lemma 2) The proof proceeds by structural induction on c assuming (A) for our base language:

$$\mathcal{Q}[\![\Delta \vdash b[\mathbf{v}/\mathbf{X}] : \tau]\!]^{\delta} = \mathcal{Q}[\![\Delta \vdash b : \tau]\!]^{\delta\oplus\{\mathbf{X}\mapsto\mathbf{v}}$$

.

We use Fig. 7 and abbreviate $\mathcal{D}[\![\mathrm{D}]\!]^{\delta}$ by γ where appropriate.

$\boxed{c \equiv \mathrm{Success}}$ To show: $\mathcal{C}[\![\mathrm{Success}]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}} = \mathcal{C}[\![\mathrm{Success}[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}$. We have $\mathcal{C}[\![\mathrm{Success}]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}} = \{\langle\rangle\} = \mathcal{C}[\![\mathrm{Success}[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}$.

$\boxed{c \equiv \mathrm{Failure}}$ This case proceeds exactly as the previous except that both sides denote $\emptyset$.

$\boxed{c \equiv f(\mathbf{b})}$ To show: $\mathcal{C}[\![f(\mathbf{b})[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta} = \mathcal{C}[\![f(\mathbf{b})]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}}$.

We have:

$$\begin{aligned}\mathcal{C}[\![f(\mathbf{b})[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta} &= \mathcal{C}[\![f(\mathbf{b}[\mathbf{v}/\mathbf{X}])]\!]^{\gamma;\delta}\\ &= \gamma(f)(\mathcal{Q}[\![\mathbf{b}[\mathbf{v}/\mathbf{X}]]\!]^{\delta}\\ &= \gamma(f)(\mathcal{Q}[\![\mathbf{b}]\!]^{\delta\oplus\mathbf{X}\mapsto\mathbf{v}})(\text{by (A)})\\ &= \mathcal{C}[\![f(\mathbf{b})]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}}.\end{aligned}$$

$\boxed{c \equiv \mathrm{transmit}(\mathbf{X'} \mid P).\,c'}$ To show: $\mathcal{C}[\![\mathrm{transmit}(\mathbf{X'} \mid P).\,c'[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta} = \mathcal{C}[\![\mathrm{transmit}(\mathbf{X'} \mid P).\,c']\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}}$.

We allow α-conversion and may thus assume that $\mathbf{X'}$ is chosen such that $\mathbf{X} \cap \mathbf{X'} = \emptyset$. We have:

$$\begin{aligned}&\mathcal{C}[\![\mathrm{transmit}(\mathbf{X'} \mid P).\,c'[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta} =\\ &\mathcal{C}[\![\mathrm{transmit}(\mathbf{X'} \mid P[\mathbf{v}/\mathbf{X}]).\,c'[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta} =\\ &\quad\{\mathrm{transmit}(\mathbf{v'})\; s \mid \mathcal{Q}[\![P[\mathbf{v}/\mathbf{X}]]\!]^{\delta\oplus\mathbf{X'}\mapsto\mathbf{v'}}\\ &\quad= \mathrm{true} \wedge s \in \mathcal{C}[\![c'[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta\oplus\mathbf{X'}\mapsto\mathbf{v'}}\} =\\ &\{\mathrm{transmit}(\mathbf{v})\; s \mid \mathcal{Q}[\![P]\!]^{\delta\oplus\mathbf{X'}\mapsto\mathbf{v'}\oplus\mathbf{X}\mapsto\mathbf{v}}\\ &\quad= \mathrm{true} \wedge s \in \mathcal{C}[\![c']\!]^{\gamma;\delta\oplus\mathbf{X'}\mapsto\mathbf{v'}\oplus\mathbf{X}\mapsto\mathbf{v}} =\\ &\{\mathrm{transmit}(\mathbf{v})\; s \mid \mathcal{Q}[\![P]\!]^{\delta\oplus\mathbf{X}\mapsto\mathbf{v}\oplus\mathbf{X'}\mapsto\mathbf{v'}}\\ &\quad= \mathrm{true} \wedge s \in \mathcal{C}[\![c']\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}\oplus\mathbf{X'}\mapsto\mathbf{v'}} =\\ &\mathcal{C}[\![\mathrm{transmit}(\mathbf{X'} \mid P).\,c']\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}}.\end{aligned}$$

$\boxed{c \equiv c_1 + c_2}$ To show: $\mathcal{C}[\![c_1 + c_2[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta} = \mathcal{C}[\![c_1 + c_2]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}}$.

We have:

$$\begin{aligned}\mathcal{C}[\![c_1 + c_2[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta} &= \mathcal{C}[\![c_1[\mathbf{v}/\mathbf{X}] + c_2[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}\\ &= \mathcal{C}[\![c_1[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta} \cup \mathcal{C}[\![c_2[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}\\ &= \mathcal{C}[\![c_1]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}} \cup \mathcal{C}[\![c_2]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}}\\ &= \mathcal{C}[\![c_1 + c_2]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}}.\end{aligned}$$

$\boxed{c \equiv c_1 \parallel c_2}$ Similar to $\cdot + \cdot$ case.

$\boxed{c \equiv c_1 ; c_2}$ Similar to $\cdot + \cdot$ case.

□

Proof (Lemma 1) We verify that each equation in Fig. 8 holds. Note that $\gamma = \mathcal{D}[\![\mathrm{D}]\!]^{\delta}$ in the following:

$\boxed{\mathcal{C}[\![e\backslash\mathrm{Failure}]\!]^{\gamma;\delta} = \mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta}}$ By definition of the residuation operator we have $\mathcal{C}[\![e\backslash\mathrm{Failure}]\!]^{\gamma;\delta} = e\backslash\emptyset = \emptyset = \mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta}$.

$\boxed{\mathcal{C}[\![e\backslash\mathrm{Success}]\!]^{\gamma;\delta} = \mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta}}$ By definition of the residuation operator we have $\mathcal{C}[\![e\backslash\mathrm{Success}]\!]^{\gamma;\delta} = e\backslash\{\langle\rangle\} = \emptyset = \mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta}$.

$\boxed{\mathcal{C}[\![e\backslash f(\mathbf{a})]\!]^{\gamma;\delta} = \mathcal{C}[\![e\backslash c[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}}$ This follows from $\mathcal{C}[\![f(\mathbf{a})]\!]^{\gamma;\delta} = \mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto v} = \mathcal{C}[\![c[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}$.

We assume $(f(\mathbf{X}) = c) \in \mathrm{D}$ and $\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta}$.

We have $\mathcal{C}[\![f(\mathbf{a})]\!]^{\gamma;\delta} = \mathcal{C}[\![c]\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto v}$ by definition of γ and assumption for $\mathbf{v}$. By Lemma 2 this can be rewritten as $\mathcal{C}[\![c[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}$, and we are done.

$\boxed{\mathcal{C}[\![\mathrm{transmit}(\mathbf{v})\backslash(\mathrm{transmit}(\mathbf{X} \mid P).\,c')]\!]^{\gamma;\delta}}$ There are two cases to consider:

– $\delta \oplus \{\mathbf{X} \mapsto \mathbf{v}\} \models P$.
To show: $\mathcal{C}[\![\mathrm{transmit}(\mathbf{v})\backslash(\mathrm{transmit}(\mathbf{X} \mid P).\,c')]\!]^{\gamma;\delta} = \mathcal{C}[\![c'[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}$.
Again we unfold the left-hand side of the equation and the goal is then:
$$\{s' \mid \exists s \in \mathcal{C}[\![\mathrm{transmit}(\mathbf{X} \mid P).\,c']\!]^{\gamma;\delta} : \mathrm{transmit}\,(\mathbf{v})s' = s\}$$
$$= \mathcal{C}[\![c'[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}.$$
From the denotational semantics we see that $\mathcal{C}[\![\mathrm{transmit}(\mathbf{X}|P).\,c']\!]^{\gamma;\delta}{=}\{\mathrm{transmit}(\mathbf{v})\; s' | s' {\in} \mathcal{C}[\![c']\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}}$.
What we need to show is then that $\mathcal{C}[\![c']\!]^{\gamma;\delta\oplus\mathbf{X}\mapsto\mathbf{v}} = \mathcal{C}[\![c'[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}$, which follows immediately from Lemma 2.

– $\delta \oplus \{\mathbf{X} \mapsto \mathbf{a}\} \not\models P$.
To show: $\mathcal{C}[\![\mathrm{transmit}(\mathbf{a})\backslash(\mathrm{transmit}(\mathbf{X} \mid P).\,c')]\!]^{\gamma;\delta} = \mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta}$.
We unfold the left-hand side of the equation using the denotational semantics and the goal is now to show:
$\{s' \mid \exists s {\in} \mathcal{C}[\![\mathrm{transmit}(\mathbf{X} \mid P).\,c']\!]^{\gamma;\delta} : \mathrm{transmit}(\mathbf{v})s' = s\} {=} \emptyset$.
As $\delta \oplus \{\mathbf{X} \mapsto \mathbf{a}\} \not\models P$ we know that $\mathcal{C}[\![\mathrm{transmit}(\mathbf{X}|P).\,c']\!]^{\gamma;\delta} = \emptyset$, and we are done.

$\boxed{\mathcal{C}[\![e\backslash(c_1 + c_2)]\!]^{\gamma;\delta} = \mathcal{C}[\![e\backslash c_1 + e\backslash c_2]\!]^{\gamma;\delta}}$ Unfolding the left-hand side gives $\{s' \mid \exists s \in \mathcal{C}[\![c_1 + c_2]\!]^{\gamma;\delta} : es' = s\}$. The denotation of a choice contract is given by $\mathcal{C}[\![c_1 + c_2]\!]^{\gamma;\delta} = \mathcal{C}[\![c_1]\!]^{\gamma;\delta} \cup \mathcal{C}[\![c_2]\!]^{\gamma;\delta}$. Any s' will thus be a trace of c_1 or a trace of c_2 with a prefix of e removed. The denotation of the right-hand side is $\mathcal{C}[\![e\backslash c_1]\!]^{\gamma;\delta} \cup \mathcal{C}[\![e\backslash c_2]\!]^{\gamma;\delta}$ which unfolds to $\{s'_1 \mid \exists s \in \mathcal{C}[\![c_1]\!]^{\gamma;\delta} : es'_1 = s\} \cup \{s'_2 \mid \exists s \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta} : es'_2 = s\}$. Thus any s'_1 or s'_2 is a trace of c_1 or c_2 with the prefix e removed. We can now conclude that in any case $s' = s'_i$ for $0 < i \leq 2$ as required.

$\boxed{\mathcal{C}[\![e\backslash(c_1 \parallel c_2)]\!]^{\gamma;\delta} = \mathcal{C}[\![e\backslash c_1 \parallel c_2 + c_1 \parallel e\backslash c_2]\!]^{\gamma;\delta}}$ Rewriting the left-hand side of the equation by definition of the residuation operator we arrive at the following equation:

$\{s' \mid \exists s \in \mathcal{C}[\![c_1 \parallel c_2]\!]^{\gamma;\delta} : es' = s\} = \mathcal{C}[\![e\backslash c_1 \parallel c_2 + c_1 \parallel e\backslash c_2]\!]^{\gamma;\delta}$.

Using the definition of the denotational semantics to rewrite the right-hand side we arrive at:

$$\begin{aligned}&\{s' \mid \exists s \in \mathcal{C}[\![c_1 \parallel c_2]\!]^{\gamma;\delta} : es' = s\}\\ &\quad = \mathcal{C}[\![e\backslash c_1 \parallel c_2]\!]^{\gamma;\delta} \cup \mathcal{C}[\![c_1 \parallel e\backslash c_2]\!]^{\gamma;\delta}.\end{aligned}$$

From the denotational semantics, we note that the trace set of a parallel contract is an interleaving of the events from *both* subcontracts:

$$\{s' \mid \exists s \in \{s'' \mid s_1 \in \mathcal{C}[\![c_1]\!]^{\gamma;\delta}, s_2 \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta} : (s_1, s_2) \leadsto s''\} : es' = s\} = \ldots$$

If e is a prefix of s_1 we have the trace set $\mathcal{C}[\![e\backslash c_1 \parallel c_2]\!]^{\gamma;\delta}$ and if e is a prefix of s_2 we have the trace set $\mathcal{C}[\![c_1 \parallel e\backslash c_2]\!]^{\gamma;\delta}$. Combining these two sets we conclude what was required.

$\boxed{\mathcal{C}[\![e\backslash(c_1; c_2)]\!]^{\gamma;\delta} = \begin{cases}(e\backslash c_1; c_2) + e\backslash c_2 & \text{if } \mathrm{D}, \delta \models \text{Success} \subseteq c_1\\ e\backslash c_1; c_2 & \text{otherwise}\end{cases}}$

– $\mathrm{D}, \delta \models \text{Success} \subseteq c_1$.

We unfold the left-hand side and the goal becomes:

$$\begin{aligned}&\{s' \mid \exists s \in \{s_1 s_2 \mid \exists s_1 \in \mathcal{C}[\![c_1]\!]^{\gamma;\delta}, s_2 \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta}\} : es' = s\}\\ &\quad = \mathcal{C}[\![e\backslash c_1; c_2 + e\backslash c_2]\!]^{\gamma;\delta}.\end{aligned}$$

Unfold the right-hand side:

$$\begin{aligned}&\{s' \mid \exists s \in \{s_1 s_2 \mid \exists s_1 \in \mathcal{C}[\![c_1]\!]^{\gamma;\delta}, s_2 \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta}\} : es' = s\},\\ &\qquad =\\ &\{s'_1 s'_2 \mid \exists s'_1 \in \mathcal{C}[\![e\backslash c_1]\!]^{\gamma;\delta}, s'_2 \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta}\} \cup \mathcal{C}[\![e\backslash c_2]\!]^{\gamma;\delta}.\end{aligned}$$

  – In case $s_1 = \langle\rangle$, we get that $es' = \mathcal{C}[\![c_2]\!]^{\gamma;\delta}$ and $\mathcal{C}[\![e\backslash c_1]\!]^{\gamma;\delta} = \emptyset$. Thus we need to show that: $\{s' \mid \exists s \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta} : es' = s\} = \mathcal{C}[\![e\backslash c_2]\!]^{\gamma;\delta}$, which is immediate from the definition of residuation.
  – If $s_1 \neq \langle\rangle$ there is some s_1 in which e occurs as the first event. Thus $s = es'_1 s'_2$, which means $s' = s'_1 s'_2$ as required. The added $\mathcal{C}[\![e\backslash c_2]\!]^{\gamma;\delta}$ are accounted for by the previous case.

– $\mathrm{D}, \delta \models \text{Success} \not\subseteq c_1$.

We unfold the left-hand side and the goal becomes:

$$\begin{aligned}&\{s' \mid \exists s \in \{s_1 s_2 \mid \exists s_1 \in \mathcal{C}[\![c_1]\!]^{\gamma;\delta}, s_2 \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta}\} : es' = s\}\\ &\quad = \mathcal{C}[\![e\backslash c_1; c_2]\!]^{\gamma;\delta}.\end{aligned}$$

Unfold the right-hand side:

$$\begin{aligned}&\{s' \mid \exists s \in \{s_1 s_2 \mid \exists s_1 \in \mathcal{C}[\![c_1]\!]^{\gamma;\delta}, s_2 \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta}\} : es' = s\}\\ &\quad = \{s'_1 s'_2 \mid \exists s'_1 \in \mathcal{C}[\![e\backslash c_1]\!]^{\gamma;\delta}, s'_2 \in \mathcal{C}[\![c_2]\!]^{\gamma;\delta}\}.\end{aligned}$$

As $\langle\rangle \notin \mathcal{C}[\![c_1]\!]^{\gamma;\delta}$, we know that $e \in s_1$ from which we immediately see that $s_2 = s'_2$; thus we just need to show that

$$\begin{aligned}&\{s' \mid \exists s \in \{s_1 \mid \exists s_1 \in \mathcal{C}[\![c_1]\!]^{\gamma;\delta}\} : es' = s\}\\ &\quad = \{s'_1 \mid \exists s'_1 \in \mathcal{C}[\![e\backslash c_1]\!]^{\gamma;\delta}\}.\end{aligned}$$

That is,

$\{s' \mid \exists s \in \mathcal{C}[\![c_1]\!]^{\gamma;\delta}\} : es' = s\} = \mathcal{C}[\![e\backslash c_1]\!]^{\gamma;\delta}$,

which is exactly the definition of the residuation operator.

□

Proof (Proposition 1) We show $\forall \mathrm{D}, \delta, \delta', c : \delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : c \iff \mathrm{D} \vdash c$ nullable. From this the proposition follows by Theorem 1.

$\Longrightarrow$": To show $\forall \mathrm{D}, \delta, \delta', c : \delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : c \Longrightarrow \mathrm{D} \vdash c$ nullable we proceed by induction on derivations of $\delta' \vdash^{\delta}_{\mathrm{D}} s : c$.

$\boxed{\delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : \text{Success}}$ We need to show that $\mathrm{D} \vdash \text{Success}$ nullable. This follows immediately from the nullability axiom for Success.

$\boxed{\dfrac{\mathbf{X} \mapsto \mathbf{v} \vdash^{\delta}_{\mathrm{D}} s : c \quad (f(\mathbf{X}) = c) \in \mathrm{D}, \mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta \oplus \delta'}}{\delta' \vdash^{\delta}_{\mathrm{D}} s : f(\mathbf{a})}}$ Assume $\mathrm{D} \vdash c$ nullable (induction hypothesis). We need to show that $\mathrm{D} \vdash f(\mathbf{a})$ nullable, which follows from the nullability inference rule for $f(\mathbf{a})$.

$\boxed{\dfrac{\delta \oplus \delta'' \models P \quad \delta'' \vdash^{\delta}_{\mathrm{D}} s : c \qquad (\delta'' = \delta' \oplus \{\mathbf{X} \mapsto \mathbf{v}\})}{\delta' \vdash^{\delta}_{\mathrm{D}} \text{transmit}(\mathbf{v})\, s : \text{transmit}(\mathbf{X} | P).\, c}}$ We need to show $\mathrm{D} \vdash \text{transmit}(\mathbf{X}|P).\, c$ nullable if $\text{transmit}(\mathbf{v})\, s = \langle\rangle$. This implication is vacuously true as the assumption $\text{transmit}(\mathbf{v})\, s = \langle\rangle$ is false.

$\boxed{\dfrac{\delta' \vdash^{\delta}_{\mathrm{D}} s_1 : c_1 \quad \delta' \vdash^{\delta}_{\mathrm{D}} s_2 : c_2 \quad (s_1, s_2) \leadsto s}{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_1 \parallel c_2}}$ Assume $(\langle\rangle, \langle\rangle) \leadsto s$; that is, $s = \langle\rangle$. Assume furthermore $\mathrm{D} \vdash c_1$ nullable and $\mathrm{D} \vdash c_2$ nullable. We need to show that $\mathrm{D} \vdash c_1 \parallel c_2$ nullable, which follows from the nullability inference rule for $c_1 \parallel c_2$.

$\boxed{\dfrac{\delta' \vdash^{\delta}_{\mathrm{D}} s_1 : c_1 \quad \delta' \vdash^{\delta}_{\mathrm{D}} s_2 : c_2}{\delta' \vdash^{\delta}_{\mathrm{D}} s_1 s_2 : c_1; c_2}}$ Immediate from nullability inference rule for $c_1; c_2$.

$\boxed{\dfrac{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_1}{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_1 + c_2}}$ Immediate from first nullability inference rule for $c_1 + c_2$.

$\boxed{\dfrac{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_2}{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_1 + c_2}}$ Immediate from second nullability inference rule for $c_1 + c_2$.

$\Longleftarrow$": To show $\forall \mathrm{D}, c : \mathrm{D} \vdash c$ nullable $\Longrightarrow \forall \delta, \delta'. \delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : c$ we proceed by induction on derivations of $\mathrm{D} \vdash c$ nullable.

$\boxed{\dfrac{\mathrm{D} \vdash c \text{ nullable} \quad (f(\mathbf{X}) = c) \in \mathrm{D}}{\mathrm{D} \vdash f(\mathbf{a}) \text{ nullable}}}$ Assume $\forall \delta, \delta'. \delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : c$ (induction hypothesis). We need to show $\forall \delta, \delta'. \delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : f(\mathbf{a})$. Let δ, δ' be arbitrary environments for D and $f(\mathbf{a})$. From the induction hypothesis it follows that $\mathbf{X} \mapsto \mathbf{v} \vdash^{\delta}_{\mathrm{D}} \langle\rangle : c$ where $\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta \oplus \delta'}$. And, using the satisfaction inference rule for contract application, we arrive at $\delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : f(\mathbf{a})$.

$\boxed{\dfrac{\mathrm{D} \vdash c \text{ nullable}}{\mathrm{D} \vdash c + c' \text{ nullable}}}$ Immediate.

$\boxed{\dfrac{\mathrm{D} \vdash c' \text{ nullable}}{\mathrm{D} \vdash c + c' \text{ nullable}}}$ Immediate.

$\boxed{\mathrm{D} \vdash \text{Success nullable}}$ Let δ, δ' be arbitrary environments. Using the satisfaction rule for Success we obtain $\delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : \text{Success}$.

$\boxed{\dfrac{\mathrm{D} \vdash c \text{ nullable} \quad \mathrm{D} \vdash c' \text{ nullable}}{\mathrm{D} \vdash c \parallel c' \text{ nullable}}}$ Immediate.

$\boxed{\dfrac{\mathrm{D} \vdash c \text{ nullable} \quad \mathrm{D} \vdash c' \text{ nullable}}{\mathrm{D} \vdash c; c' \text{ nullable}}}$ Immediate.

□

Proof (Lemma 3) This is proved by straightforward structural induction on the definition of contracts.

The only interesting cases are the cases of a contract application $f(\mathbf{a})$, where $(f(\mathbf{X}) = c) \in \mathrm{D}$, and sequential composition.

In the first case, we can use the assumption of the lemma that $\mathrm{D} \vdash c$ guarded, which, by rule application, immediately implies that $\mathrm{D} \vdash f(\mathbf{a})$ guarded.

In the second case we have the induction hypotheses $\mathrm{D} \vdash c_1$ guarded and $\mathrm{D} \vdash c_2$ guarded. Now, either $\mathrm{D} \vdash c_1$ nullable or $\mathrm{D} \nvdash c_1$ nullable. In either case, we have a rule for concluding that $\mathrm{D} \vdash c_1; c_2$ guarded. □

Proof (Theorem 2) (Sketch)

1. We show $\mathrm{D}, \delta \vdash_D c \xrightarrow{e} c' \Longrightarrow \mathrm{D}, \delta \models e \backslash c = c'$ by induction on derivations of $\mathrm{D}, \delta \vdash_D c \xrightarrow{e} c'$. Each case follows immediately from Lemma 1. In the case of sequential composition we also require Proposition 1.
2. Note that, by Lemma 3, $\mathrm{D} \vdash c$ guarded if D is guarded. It is sufficient to show $\mathrm{D} \vdash c$ guarded $\Longrightarrow \forall\delta\forall e\exists c'.\mathrm{D}, \delta \vdash_D c \xrightarrow{e} c'$. The fact that c' is guarded in context D follows from Lemma 3, and it is a routine matter to extend the proof cases with a check of uniqueness of c'.

   We cannot prove $\mathrm{D} \vdash c$ guarded $\Longrightarrow \forall\delta\forall e\exists c'.\mathrm{D}, \delta \vdash_D c \xrightarrow{e} c'$ by induction on the definition of guarded contracts, however, as the induction hypothesis is not strong enough in the case of contract application: we would require that $c[\mathbf{v}/\mathbf{X}]$ has a residual contract for arbitrary δ, e, but the induction hypothesis only yields that that holds for c. Consequently, we strengthen the lemma and prove $\mathrm{D} \vdash c$ guarded $\Longrightarrow \forall\delta, e, \mathbf{X}, \mathbf{v}\exists c'.\mathrm{D}, \delta \vdash_D c[\mathbf{v}/\mathbf{X}] \xrightarrow{e} c'$.

   All cases are straightforward except the second rule for sequential composition: to make the induction proof go through we require $\mathrm{D} \nvdash c[/\mathbf{X}]$ nullable the last deterministic reduction rule, but the case only carries the assumption $\mathrm{D} \nvdash c$ nullable. Consequently, if we can show that $\mathrm{D} \vdash c[/\mathbf{X}]$ nullable $\Longrightarrow \mathrm{D} \vdash c$ nullable, we are done.

   Claim: $\mathrm{D} \vdash c[/\mathbf{X}]$ nullable $\Longrightarrow \mathrm{D} \vdash c$ nullable.

   Proof of claim: By structural induction on c. All cases are straightforward except the rule for contract application. In that case we need to show $\mathrm{D} \vdash f(\mathbf{a}[\mathbf{v}/\mathbf{X}])$ nullable $\Longrightarrow \mathrm{D} \vdash f(\mathbf{a})$ nullable. Assume $\mathrm{D} \vdash f(\mathbf{a}[\mathbf{v}/\mathbf{X}])$ nullable. By inspection of the rules for nullability we can see that this must have been concluded from $\mathrm{D} \vdash c$ nullable where $(f(\mathbf{Y}) = c) \in \mathrm{D}$. By the same rule we can infer, however, $\mathrm{D} \vdash f(\mathbf{a})$ nullable, and we are done.

□

Proof (Theorem 3) We prove the two statements

1. If $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} c'$ then $\mathrm{D}, \delta \models c' \subseteq e \backslash c$
2. If $\mathrm{D}, \delta \vdash_N c \xrightarrow{\tau} c'$ then $\mathrm{D}, \delta \models c' \subseteq c$

by induction on the height of the derivation of $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} c'$ and $\mathrm{D}, \delta \vdash_N c \xrightarrow{\tau} c'$, respectively. We use definitions in Figs. 7, 8 and 12. Assume..." is used as short-hand for "Assume a derivation with the conclusion...". Finally, γ abbreviates $\mathcal{D}[\![\mathrm{D}]\!]^\delta$ in the following.

Proving 1:

– Assume $\mathrm{D}, \delta \vdash_N \mathrm{Success} \xrightarrow{e} \mathrm{Failure}$. To show $\mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash\mathrm{Success}]\!]^{\gamma;\delta} = \mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta}$. Done.
– Assume $\mathrm{D}, \delta \vdash_N \mathrm{Failure} \xrightarrow{e} \mathrm{Failure}$.
To show $\mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash\mathrm{Failure}]\!]^{\gamma;\delta} = \mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta}$. Done.
– Assume $\mathrm{D}, \delta \vdash_N \mathrm{transmit}(\mathbf{X} \mid P).\, c \xrightarrow{\mathrm{transmit}(\mathbf{v})} c[\mathbf{v}/\mathbf{X}]$ and also $\delta \oplus \mathbf{X} \mapsto \mathbf{v} \models P$ where $\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^\delta$.
To show $\mathcal{C}[\![c[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![(\mathrm{transmit}(\mathbf{v})\backslash\mathrm{transmit}(\mathbf{X} \mid P).\, c)]\!]^{\gamma;\delta} = \mathcal{C}[\![c[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta}$. Done.
– Assume $\mathrm{D}, \delta \vdash_N \mathrm{transmit}(\mathbf{X} \mid P).\, c \xrightarrow{\mathrm{transmit}(\mathbf{v})} \mathrm{Failure}$ and also $\delta \oplus \mathbf{X} \mapsto \mathbf{v} \not\models P$ where $\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^\delta$.
To show $\mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![(\mathrm{transmit}(\mathbf{v})\backslash\mathrm{transmit}(\mathbf{X} \mid P).\, c)]\!]^{\gamma;\delta} = \mathcal{C}[\![\mathrm{Failure}]\!]^{\gamma;\delta}$. Done.
– Assume $\mathrm{D}, \delta \vdash_N c \parallel c' \xrightarrow{e} d \parallel c'$ and $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} d$. To show

$$\mathcal{C}[\![d \parallel c']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash(c \parallel c')]\!]^{\gamma;\delta} = \mathcal{C}[\![e\backslash c \parallel c' + c \parallel e\backslash c']\!]^{\gamma;\delta}$$
$$= \mathcal{C}[\![e\backslash c \parallel c']\!]^{\gamma;\delta} \cup \mathcal{C}[\![c \parallel e\backslash c']\!]^{\gamma;\delta}$$

By the IH, $\mathcal{C}[\![d]\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash c]\!]^{\gamma;\delta}$ so in particular $\mathcal{C}[\![d \parallel c']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash c \parallel c']\!]^{\gamma;\delta}$, which is sufficient.
– Assume $\mathrm{D}, \delta \vdash_N c \parallel c' \xrightarrow{e} c \parallel d'$ and $\mathrm{D}, \delta \vdash_N c' \xrightarrow{e} d'$. Analogous to the above case.
– Assume $\mathrm{D}, \delta \vdash_N c; c' \xrightarrow{e} d; c'$ and also $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} d$. To show $\mathcal{C}[\![d; c']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash(c; c')]\!]^{\gamma;\delta}$. If $\mathrm{D}, \delta \models \mathrm{Success} \subseteq c$ then $\mathcal{C}[\![e\backslash(c; c')]\!]^{\gamma;\delta} = \mathcal{C}[\![(e\backslash c; c') + e\backslash c']\!]^{\gamma;\delta} = \mathcal{C}[\![e\backslash c; c']\!]^{\gamma;\delta} \cup \mathcal{C}[\![e\backslash c']\!]^{\gamma;\delta}$. By the IH, $\mathcal{C}[\![d]\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash c]\!]^{\gamma;\delta}$ so in particular $\mathcal{C}[\![d; c']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash c; c']\!]^{\gamma;\delta}$, which is sufficient.
– Assume $\mathrm{D}, \delta \vdash_N c \xrightarrow{\tau} c'$ and $\mathrm{D}, \delta \vdash_N c' \xrightarrow{e} c''$. By the IH, we have $\mathcal{C}[\![c']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![c]\!]^{\gamma;\delta}$ and $\mathcal{C}[\![c'']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash c']\!]^{\gamma;\delta}$. We need to show $\mathcal{C}[\![c'']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash c]\!]^{\gamma;\delta}$. But this follows from the IH and monotonicity of residuation: $\mathcal{C}[\![c'']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash c']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![e\backslash c]\!]^{\gamma;\delta}$.

Proving 2:

– Assume $\mathrm{D}, \delta \vdash_N f(\mathbf{a}) \xrightarrow{\tau} c[\mathbf{V}/\mathbf{X}]$ where $(f(\mathbf{X}) = c) \in \mathrm{D}$ and $\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^\delta$. To show $\mathcal{C}[\![c[\mathbf{v}/\mathbf{X}]]\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![f(\mathbf{a})]\!]^{\gamma;\delta} = \gamma(f)(\mathbf{v})$, which holds by definition of γ and Lemma 2.
– Assume $\mathrm{D}, \delta \vdash_N c + c' \xrightarrow{\tau} c$. To show $\mathcal{C}[\![c]\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![c + c']\!]^{\gamma;\delta} = \mathcal{C}[\![c]\!]^{\gamma;\delta} \cup \mathcal{C}[\![c']\!]^{\gamma;\delta}$. Done.
– Assume $\mathrm{D}, \delta \vdash_N c + c' \xrightarrow{\tau} c'$. Analogous to the above case.
– Assume $\mathrm{D}, \delta \vdash_N c \parallel c' \xrightarrow{\tau} d \parallel c'$ and $\mathrm{D}, \delta \vdash_N c \xrightarrow{\tau} d$. To show $\mathcal{C}[\![d \parallel c']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![c \parallel c']\!]^{\gamma;\delta}$, which follows easily by the IH.
– Assume $\mathrm{D}, \delta \vdash_N c \parallel c' \xrightarrow{\tau} c \parallel d'$ and $\mathrm{D}, \delta \vdash_N c' \xrightarrow{\tau} d'$. To show $\mathcal{C}[\![c \parallel d']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![c \parallel c']\!]^{\gamma;\delta}$, which follows easily by the IH.
– Assume $\mathrm{D}, \delta \vdash_N \mathrm{Success} \parallel c \xrightarrow{\tau} c$. To show $\mathcal{C}[\![c]\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![\mathrm{Success} \parallel c]\!]^{\gamma;\delta}$, holds trivially.
– Assume $\mathrm{D}, \delta \vdash_N c \parallel \mathrm{Success} \xrightarrow{\tau} c$. To show $\mathcal{C}[\![c]\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![c \parallel \mathrm{Success}]\!]^{\gamma;\delta}$, holds trivially.
– Assume $\mathrm{D}, \delta \vdash_N \mathrm{Success}; c' \xrightarrow{\tau} c'$. To show $\mathcal{C}[\![c']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![\mathrm{Success}; c']\!]^{\gamma;\delta}$, holds trivially.
– Assume $\mathrm{D}, \delta \vdash_N c; c' \xrightarrow{\tau} d; c'$ and $\mathrm{D}, \delta \vdash_N c \xrightarrow{\tau} d$. To show $\mathcal{C}[\![d; c']\!]^{\gamma;\delta} \subseteq \mathcal{C}[\![c; c']\!]^{\gamma;\delta}$, which follows easily by the IH.

□

Proof (Theorem 4) The proof is by induction on the derivation of $\mathrm{D}, \delta \vdash_D c \xrightarrow{e} c'$.

– $\mathrm{D}, \delta \vdash_D \mathrm{Success} \xrightarrow{e} \mathrm{Failure}$. Clearly no τ-transitions can be taken in the nondeterministic reduction system. However, there is just one contract c_1 such that $\mathrm{D}, \delta \vdash_N \mathrm{Success} \xrightarrow{e} c_1$ which is Failure. We must then show: $\mathrm{D}, \delta \models \mathrm{Failure} \subseteq \mathrm{Failure}$. By definition $\mathrm{D}, \delta \models \mathrm{Failure} = \emptyset$, so we must show $\emptyset \subseteq \emptyset$ which is trivially true.
– $\mathrm{D}, \delta \vdash_D \mathrm{Failure} \xrightarrow{e} \mathrm{Failure}$. Again no τ-transitions are possible. There is just one contract c_1 such that $\mathrm{D}, \delta \vdash_N \mathrm{Failure} \xrightarrow{e} c_1$ namely Failure. We must show $\mathrm{D}, \delta \models \mathrm{Failure} \subseteq \mathrm{Failure}$, which is true since $\emptyset \subseteq \emptyset$.
– $\mathrm{D}, \delta \vdash_D \mathrm{transmit}(\mathbf{X}|P).\, c \xrightarrow{\mathrm{transmit}(\mathbf{v})} c[\mathbf{v}/\mathbf{X}]$ where $\delta \oplus \mathbf{X} \mapsto \mathbf{v} \models P$ and $\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^\delta$. In this case we can only do the reduction $\mathrm{D}, \delta \vdash_N \mathrm{transmit}(\mathbf{X}|P).\, c \xrightarrow{\mathrm{transmit}(\mathbf{v})} c[\mathbf{v}/\mathbf{X}]$. Now we must show $\mathrm{D}, \delta \models c[\mathbf{v}/\mathbf{X}] \subseteq c[\mathbf{v}/\mathbf{X}]$, which is obviously true.

- $\mathrm{D},\delta \vdash_D \mathrm{transmit}(\mathbf{X}|P).\, c \xrightarrow{\mathrm{transmit}(\mathbf{v})} \mathrm{Failure}$ and $\delta \oplus \mathbf{X} \mapsto \mathbf{v} \models P$ where $\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta}$. No τ-transitions are possible and only one contract c_1 exists such that

  $\mathrm{D},\delta \vdash_D \mathrm{transmit}(\mathbf{X}|P).\, c \xrightarrow{\mathrm{transmit}(\mathbf{v})} c_1,$

  so $c_1 = \mathrm{Failure}$. This means we must show $\mathrm{D},\delta \models c[\mathbf{v}/\mathbf{X}] \subseteq c[\mathbf{v}/\mathbf{X}]$ which clearly holds.
- $\mathrm{D},\delta \vdash_D f(\mathbf{a}) \xrightarrow{e} c'$. This implies that $(f(\mathbf{X}) = c) \in \mathrm{D}$ and $\mathrm{D},\delta \vdash_D c[\mathbf{v}/\mathbf{X}] \xrightarrow{e} c'$ with $\mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^{\delta}$. By a derivation of $\mathrm{D},\delta \vdash_D c[\mathbf{v}/\mathbf{X}] \xrightarrow{e} c'$ we use the IH to get contracts $c_1, \dots, c_n$ such that $\mathrm{D},\delta \vdash_N c[\mathbf{v}/\mathbf{X}] \xrightarrow{\tau *} c_i'' \xrightarrow{e} c'$ and $\mathrm{D},\delta \models c' \subseteq \sum_{i=1}^{n} c_i$. However we need to show $\mathrm{D},\delta \vdash_N f(\mathbf{a}) \xrightarrow{\tau *} c_i'' \xrightarrow{e} c_i$ and $c' \subseteq \sum_{i=1}^{n} c_i$, the latter of which follows directly from the IH. By the nondeterministic reduction rules, $f(\mathbf{a})$ has just one reduction $\mathrm{D},\delta \vdash_N f(\mathbf{a}) \xrightarrow{\tau} c[\mathbf{v}/\mathbf{X}]$. Thus we can extend all reductions of $\mathrm{D},\delta \vdash_N c[\mathbf{v}/\mathbf{X}] \xrightarrow{\tau *} c_i'' \xrightarrow{e} c_i$ with one more τ-transition giving reductions $\mathrm{D},\delta \vdash_N f(\mathbf{a}) \xrightarrow{\tau *} c_i'' \xrightarrow{e} c_i$ for all $0 < i \leq n$.
- $\mathrm{D},\delta \vdash_D c + c' \xrightarrow{e} d + d'$. This implies that $\mathrm{D},\delta \vdash_D c \xrightarrow{e} d$ and $\mathrm{D},\delta \vdash_D c' \xrightarrow{e} d'$. From the nondeterministic reduction rules we see that $c + c'$ may be reduced by a τ-transition into either c or c'. By the IH we then have contracts $d_0, \dots, d_n$ and $d_0', \dots, d_m'$ such that $\mathrm{D},\delta \vdash_N c \xrightarrow{\tau *} d_i'' \xrightarrow{e} d_i$ for $0 < i \leq n$, $\mathrm{D},\delta \models d \subseteq \sum_{i=1}^{n} d_i$ and $\mathrm{D},\delta \vdash_N c' \xrightarrow{\tau *} d_j''' \xrightarrow{e} d_j'$ for $0 < j \leq m$, $\mathrm{D},\delta \models d \subseteq \sum_{j=1}^{m} d_j$. Thus we can extend the nondeterministic reductions of c and c' to get reductions of $c + c$ into contracts $c_0, \dots, c_{n+m}$. That is: there are contracts, c_i such that $\mathrm{D},\delta \vdash_N c + c' \xrightarrow{\tau *} c_i'' \xrightarrow{e} c_i$ with $0 < i \leq m + n$. As seen from the IH we know that $\mathrm{D},\delta \models d \subseteq \sum_{i=1}^{n} d_i$ and $\mathrm{D},\delta \models d \subseteq \sum_{j=1}^{m} d_j$. Taking the union of these we get $\mathrm{D},\delta \models d \bigcup d' \subseteq \sum_{i=1}^{n} d_i + \sum_{j=1}^{m} d_j$. By definition this is $\mathrm{D},\delta \models d + d' \subseteq \sum_{i=1}^{m+n} d_i$ (given proper enumeration of contracts in d_i and d_j which is the desired goal.
- $\mathrm{D},\delta \vdash_D c \parallel c' \xrightarrow{e} d \parallel c' + c \parallel d'$. By a derivation $\mathrm{D},\delta \vdash_D c \xrightarrow{e} d$ we use the IH to get contracts d_i such that $c \xrightarrow{\tau *} c_i'' \xrightarrow{e} d_i$ and $\mathrm{D},\delta \models d \subseteq \sum_{i=1}^{n} d_i$. Then use the left $\cdot \parallel \cdot$-introduction rule to get contracts $d_i \parallel c'$ such that $c \parallel c' \xrightarrow{\tau *} c_i'' \parallel c' \xrightarrow{e} d_i \parallel c'$ and $\mathrm{D},\delta \models d \parallel c' \subseteq \sum_{i=1}^{n} d_i \parallel c'$. By a derivation $\mathrm{D},\delta \vdash_D c' \xrightarrow{e} d'$ we now again use the IH to get contracts d_i' such that $c' \xrightarrow{\tau *} c_i''' \xrightarrow{e} d_i'$ and $\mathrm{D},\delta \models d' \subseteq \sum_{i=1}^{m} d_i'$. Then use the right $\cdot \parallel \cdot$-introduction rule to get contracts $c \parallel d_i'$ such that $c \parallel c' \xrightarrow{\tau *} c \parallel c_i''' \xrightarrow{e} c \parallel d_i'$ and $\mathrm{D},\delta \models c \parallel d' \subseteq \sum_{i=1}^{m} c \parallel d_i'$. Taking all contracts $d_i \parallel c'$ and $c \parallel d_i'$ we need to show $\mathrm{D},\delta \models d \parallel c' + c \parallel d' \subseteq \sum_{i=1}^{n} d_i \parallel c + \sum_{i=1}^{m} c \parallel d_i'$ which follows directly from the above.
- $\mathrm{D},\delta \vdash_D c; c' \xrightarrow{e} d; c' + d'$ and $\mathrm{D} \vdash c$ nullable. There are two possible reductions of $c; c'$ under the nondeterministic reduction rules. Either $\mathrm{D},\delta \vdash_N c \xrightarrow{\tau *} \mathrm{Success}$ and so $\mathrm{D},\delta \vdash_N \mathrm{Success}; c' \xrightarrow{\tau} c'$ or $\mathrm{D},\delta \vdash_N c \xrightarrow{\tau *} c_p \xrightarrow{e} d_p$ where $c_p \neq \mathrm{Success}$ and then
  $\mathrm{D},\delta \vdash_N c; c' \xrightarrow{\tau *} c_f \xrightarrow{e} d_p$.

  In the former case, by a derivation of $\mathrm{D},\delta \vdash_D c' \xrightarrow{e} d$ we get by the IH that there exist contracts d_i' such that $c' \xrightarrow{\tau *} c'' \xrightarrow{e} d_i'$ and $\mathrm{D},\delta \models d' \subseteq \sum_{i=1}^{n} d_i'$. Taking $c_p = c''$ and $d_p = d$.
  In the latter case there is no sequence of τ-transitions that makes $c = \mathrm{Success}$ so all contracts d_q such that $c; c' \xrightarrow{\tau *} c_q \xrightarrow{e} d_q$ must have the form $d_i; c'$. By a derivation $\mathrm{D},\delta \vdash_D c \xrightarrow{e} d$ the IH gives that there are contracts d_i such that $c \xrightarrow{\tau *} c''' \xrightarrow{e} d_i$ and $\mathrm{D},\delta \models d \subseteq \sum_{i=1}^{m} d_i$. This implies $\mathrm{D},\delta \vdash_N c; c' \xrightarrow{\tau *} c'''' \xrightarrow{e} d_i; c'$ for $0 < i \leq m$ and furthermore that $\mathrm{D},\delta \models d; c' \subseteq \sum_{i=1}^{m} d_i; c'$.
  We have thus shown that there are contracts c_i such that $\mathrm{D},\delta \vdash_N c; c' \xrightarrow{\tau *} c'' \xrightarrow{e} c_i$ and that c_i is either $d'i$ or $d_i; c'$. We still need to show that $\mathrm{D},\delta \models d; c' + d \subseteq \sum_{i=1}^{k} c_i$; that is: $\mathrm{D},\delta \models d; c' + d \subseteq \sum_{i=1}^{m} d_i; c' + \sum_{i=1}^{n} d_i'$. This follows directly from the already noted fact that by the IH $\mathrm{D},\delta \models d; c' \subseteq \sum_{i=1}^{m} d_i; c'$ and $\mathrm{D},\delta \models d' \subseteq \sum_{i=1}^{n} d_i'$
- $\mathrm{D},\delta \vdash_D c; c' \xrightarrow{e} d; c'$ and $\mathrm{D} \nvdash c$ nullable. To show: $\mathrm{D},\delta \vdash_N c; c' \xrightarrow{\tau *} c_i'' \xrightarrow{e} c_i$ and $\mathrm{D},\delta \models d; c' \subseteq \sum_{i=1}^{n} c_i$. By a derivation $\mathrm{D},\delta \vdash_D c \xrightarrow{e} d$ the IH yields contracts $d_0, \dots, d_n$ for $0 < i \leq n$ such that $\mathrm{D},\delta \vdash_N c \xrightarrow{\tau *} c_i''' \xrightarrow{e} d_i$ and $\mathrm{D},\delta \models d \subseteq \sum_{i=1}^{n} d_i$. As $\mathrm{D} \nvdash c$ nullable $c \neq \mathrm{Success}$ Success, no number of τ-reductions can make $c; c' = c'$. The form of all c_i must then be $d_i; c'$. The goal is then to show $\mathrm{D},\delta \models d; c \subseteq d_i; c'$ which follows by $\mathrm{D},\delta \models d \subseteq \sum_{i=1}^{n} d_i$.

□

Proof (Proposition 2) By induction on the derivation of $\mathrm{D},\delta \vdash_C c \xrightarrow{\tau} c'$.

$\boxed{\frac{(f(\mathbf{X})=c)\in \mathrm{D}, \mathbf{v}=\mathcal{Q}[\![\mathbf{a}]\!]^{\delta}}{\mathrm{D},\delta \vdash_C f(\mathbf{a}) \xrightarrow{\tau} c[\mathbf{v}/\mathbf{X}]}}$ Here, we have

$\mathcal{C}[\![f(\mathbf{a})]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} = \mathcal{D}[\![\mathrm{D}]\!]^{\delta}(f)(\mathcal{Q}[\![\mathbf{a}]\!]^{\delta}) = \mathcal{C}[\![c[\mathbf{a}/\mathbf{X}]]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta},$
as desired.

$\boxed{\frac{\mathrm{D},\delta \vdash_C c \xrightarrow{\tau} d}{\mathrm{D},\delta \vdash_C c + c' \xrightarrow{\tau} d + c'}}$ In this case, $\mathcal{C}[\![d + c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} = \mathcal{C}[\![d]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} \cup \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} = \mathcal{C}[\![c]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} \cup \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}$, where the last equality follows from the IH. But $\mathcal{C}[\![c]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} \cup \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} = \mathcal{C}[\![c + c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}$, concluding the proof of the case.

$\boxed{\frac{\mathrm{D},\delta \vdash_C c' \xrightarrow{\tau} d'}{\mathrm{D},\delta \vdash_C c + c' \xrightarrow{\tau} c + d'}}$ As the previous case.

$\boxed{\frac{\mathrm{D},\delta \vdash_C c \xrightarrow{\tau} d}{\mathrm{D},\delta \vdash_C c \parallel c' \xrightarrow{\tau} d \parallel c'}}$ We have that $\mathcal{C}[\![d \parallel c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}$ equals

$$\left\{ s : s \in Tr \mid \exists s_1 \in \mathcal{C}[\![d]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} s_2 \in \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}.\, (s_1, s_2) \rightsquigarrow s \right\}$$

By the IH, we gather that $\{t \in \mathcal{C}[\![d]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta}}\}$ equals $\{s' \in \mathcal{C}[\![c]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta}}\}$, whence

$\{s : s \in Tr \mid \exists s_1 \in \mathcal{C}[\![d]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} s_2 \in \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}.\, (s_1, s_2) \rightsquigarrow s\}.$
equals

$$\left\{ s : s \in Tr \mid \exists s_1 \in \mathcal{C}[\![c]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} s_2 \in \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}.\, (s_1, s_2) \rightsquigarrow s \right\}$$

as desired.

$\boxed{\frac{\mathrm{D},\delta \vdash_C c' \xrightarrow{\tau} d'}{\mathrm{D},\delta \vdash_C c \parallel c' \xrightarrow{\tau} c \parallel d'}}$ As the previous case.

$\boxed{\mathrm{D},\delta \vdash_C \mathrm{Success} \parallel c \xrightarrow{\tau} c}$ We have $\mathcal{C}[\![\mathrm{Success}]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} = \{\langle\rangle\}$, and thus obtain $\{s : s \in Tr \mid \exists s_1 \in \mathcal{C}[\![\mathrm{Success}]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} s_2 \in \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}.\, (s_1, s_2) \rightsquigarrow s\} = \{s' : s' \in \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}\}$
$= \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}$, as desired.

$\boxed{\mathrm{D},\delta \vdash_C c \parallel \mathrm{Success} \xrightarrow{\tau} c}$ As the previous case.

$\boxed{\frac{\mathrm{D},\delta \vdash_C c \xrightarrow{\tau} d}{\mathrm{D},\delta \vdash_C c; c' \xrightarrow{\tau} d; c'}}$ We have $\mathcal{C}[\![c; c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} = \{ss' : s \in Tr, s' \in Tr \mid s \in \mathcal{C}[\![c]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} \wedge s' \in \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}\}$. But by the IH, we gather that $\mathcal{C}[\![c]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} = \mathcal{C}[\![d]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}$, whence $\{ss' : s \in Tr, s' \in Tr \mid s \in \mathcal{C}[\![c]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} \wedge s' \in \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}\} = \{ss' : s \in Tr, s' \in Tr \mid s \in \mathcal{C}[\![d]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta} \wedge s' \in \mathcal{C}[\![c']\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta}\}$.

$\boxed{\mathrm{D}, \delta \vdash_C \mathrm{Success}; c' \xrightarrow{\tau} c'}$ As the previous case, noting that $\mathcal{C}\llbracket \mathrm{Success} \rrbracket^{\mathcal{D}\llbracket \mathrm{D} \rrbracket^{\delta};\delta} = \{\langle\rangle\}$.

$\boxed{\frac{\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} c'}{\delta \vdash_C \mathrm{letrec}\, \mathrm{D} \text{ in } c \xrightarrow{\tau} \mathrm{letrec}\, \mathrm{D} \text{ in } c'}}$ Here, $\mathcal{C}\llbracket \mathrm{letrec}\, \mathrm{D}' \text{ in } c \rrbracket^{\delta}$ $= \mathcal{C}\llbracket c \rrbracket^{\mathcal{D}\llbracket \mathrm{D} \rrbracket^{\delta};\delta}$ for some D'. By the IH, we have $\mathcal{C}\llbracket c \rrbracket^{\mathcal{D}\llbracket \mathrm{D} \rrbracket^{\delta};\delta} = \mathcal{D}\llbracket c' \rrbracket^{\mathcal{D}\llbracket \mathrm{D} \rrbracket^{\delta};\delta}$ and hence $\mathcal{C}\llbracket \mathrm{letrec}\, \mathrm{D} \text{ in } c \rrbracket^{\delta} = \mathcal{C}\llbracket \mathrm{letrec}\, \mathrm{D} \text{ in } c' \rrbracket^{\delta}$, as desired.

□

Proof (Proposition 3) "If". To show: For all $\mathrm{D}, \delta, c, c'$: $\mathrm{D}, \delta \vDash c = \mathrm{Success}$ if $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau^*} \mathrm{Success}$.

A trivial induction on the length of the τ-reduction sequences using Proposition 2 furnishes $\mathcal{C}\llbracket c \rrbracket^{\mathcal{D}\llbracket \mathrm{D} \rrbracket^{\delta};\delta} = \mathcal{C}\llbracket \mathrm{Success} \rrbracket^{\mathcal{D}\llbracket \mathrm{D} \rrbracket^{\delta};\delta}$, and the result follows.

"Only if": To show: For all $\mathrm{D}, \delta, c, c'$: $\mathrm{D}, \delta \vDash c = \mathrm{Success}$ only if $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau^*} \mathrm{Success}$. Note that $\mathrm{D}, \delta \vDash c = \mathrm{Success}$ implies $\mathrm{D} \models c$ nullable and, by Proposition 1, $\mathrm{D} \vdash c$ nullable. Consequently, the result follows if we can prove $\mathrm{D} \vdash c$ nullable $\Longrightarrow$ $(\mathrm{D}, \delta \vDash c = \mathrm{Success} \Longrightarrow \mathrm{D}, \delta \vdash_C c \xrightarrow{\tau^*} \mathrm{Success})$.

Claim: The set of derivations of $\mathrm{D} \vdash c$ nullable is finite.

Proof of claim: Observe that all contracts c' that can occur in a derivation of $\mathrm{D} \vdash c$ nullable must occur in either D or c. Furthermore no contract can occur twice on any path in a derivation tree. Thus the depth of any derivation tree of $\mathrm{D} \vdash c$ nullable is bounded by the sum of the sizes of D and c. Furthermore, as the outdegree of derivation trees is bounded by 2, we can conclude that the set of derivation trees for $\mathrm{D} \vdash c$ nullable is finite.

Let us define the *maximal derivation depth* of a derivable judgment $\mathrm{D} \vdash c$ nullable to be the maximal depth of any of the derivations of $\mathrm{D} \vdash c$ nullable. By the claim above this is well defined.

We shall now prove by Noetherian (well-founded) induction on the maximal derivation depth of $\mathrm{D} \vdash c$ nullable that $\mathrm{D}, \delta \vDash c = \mathrm{Success}$ implies $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau^*} \mathrm{Success}$. We do this by cases on the syntax of c.

– Success.
  In this case, we have $\mathrm{D}, \delta \vdash_C \mathrm{Success} \xrightarrow{\tau^0} \mathrm{Success}$ and we are done.
– $c_1 + c_2$.
  Let $\mathrm{D} \vdash c_1 + c_2$ nullable with maximal derivation depth n. Assume $\mathrm{D}, \delta \vDash c_1 + c_2 = \mathrm{Success}$. It follows that both $\mathrm{D}, \delta \vDash c_1 = \mathrm{Success}$ and $\mathrm{D}, \delta \vDash c_1 = \mathrm{Success}$ and thus $\mathrm{D} \models c_1$ nullable and $\mathrm{D} \models c_2$ nullable. By Proposition 1 we thus have that $\mathrm{D} \vdash c_1$ nullable and $\mathrm{D} \vdash c_2$ nullable. As both $\mathrm{D} \vdash c_1$ nullable and $\mathrm{D} \vdash c_2$ nullable yield a derivation of $\mathrm{D} \vdash c_1 + c_2$ nullable it follows that the maximal derivation depths of $\mathrm{D} \vdash c_1$ nullable and $\mathrm{D} \vdash c_2$ nullable are less than n. Consequently we can apply the induction hypotheses to them and obtain that $\mathrm{D}, \delta \vdash_C c_1 \xrightarrow{\tau^*} \mathrm{Success}$ and $\mathrm{D}, \delta \vdash_C c_2 \xrightarrow{\tau^*} \mathrm{Success}$. By induction on the combined length of the two reductions, it can now be shown that $\mathrm{D}, \delta \vdash_C c_1 + c_2 \xrightarrow{\tau^*} \mathrm{Success} + \mathrm{Success}$. Now, we can apply Rule $\mathrm{D}, \delta \vdash_C \mathrm{Success} + \mathrm{Success} \xrightarrow{\tau} \mathrm{Success}$ and we are done.
– $c_1 \parallel c_2$.
  Let $\mathrm{D} \vdash c_1 \parallel c_2$ nullable with maximal derivation depth n. Assume $\mathrm{D}, \delta \vDash c_1 \parallel c_2 = \mathrm{Success}$. It follows that both $\mathrm{D}, \delta \vDash c_1 = \mathrm{Success}$ and $\mathrm{D}, \delta \vDash c_1 = \mathrm{Success}$.
  $\mathrm{D} \vdash c_1 \parallel c_2$ nullable can only be derived from $\mathrm{D} \vdash c_1$ nullable and $\mathrm{D} \vdash c_2$ nullable, each of which consequently has maximal derivation depth less than n. We can thus apply the induction hypothesis to $\mathrm{D} \vdash c_1$ nullable and $\mathrm{D} \vdash c_2$ nullable, which yield that $\mathrm{D}, \delta \vdash_C c_1 \xrightarrow{\tau^*} \mathrm{Success}$ and $\mathrm{D}, \delta \vdash_C c_2 \xrightarrow{\tau^*} \mathrm{Success}$. By induction on the combined length of the two reductions it can now be shown that $\mathrm{D}, \delta \vdash_C c_1 \parallel c_2 \xrightarrow{\tau^*} \mathrm{Success} \parallel \mathrm{Success}$. Using one of the two rules for eliminating a parallel Success, we thus arrive at $\mathrm{D}, \delta \vdash_C c_1 \parallel c_2 \xrightarrow{\tau^*} \mathrm{Success}$ and we are done.
– $c_1; c_2$.
  Similar to above.
– $f(\mathbf{a})$.
  Let $\mathrm{D} \vdash f(\mathbf{a})$ nullable with maximal derivation depth n where $(f(\mathbf{X}) = c) \in \mathrm{D}$. Assume $\mathrm{D}, \delta \vDash f(\mathbf{a}) = \mathrm{Success}$. It follows that $\mathrm{D}, \delta \vDash c[\mathbf{v}/\mathbf{X}] = \mathrm{Success}$ where $\mathbf{v} = \mathcal{Q}\llbracket \mathbf{a} \rrbracket^{\delta}$. As $\mathrm{D} \vdash f(\mathbf{a})$ nullable can only be derived from $\mathrm{D} \vdash c$ nullable it follows that the maximal derivation depth of $\mathrm{D} \vdash c$ nullable is less than n. Furthermore, it can be shown that for each derivation of $\mathrm{D} \vdash c$ nullable there is a derivation of $\mathrm{D} \vdash c[\mathbf{v}/\mathbf{X}]$ nullable equal depth. Consequently the maximal derivation depth of $\mathrm{D} \vdash c[\mathbf{v}/\mathbf{X}]$ nullable is also less than n, and we can apply the induction hypothesis to obtain that $\mathrm{D}, \delta \vdash_C c[\mathbf{v}/\mathbf{X}] \xrightarrow{\tau^*} \mathrm{Success}$. Prefixing this reduction sequence with Rule $\mathrm{D}, \delta \vdash_C f(\mathbf{a}) \xrightarrow{\tau^*} c[\mathbf{v}/\mathbf{X}]$ we arrive at $\mathrm{D}, \delta \vdash_C f(\mathbf{a}) \xrightarrow{\tau^*} \mathrm{Success}$ and we are done.
– Other cases. In all other cases $\mathrm{D} \vdash c$ nullable is not derivable.

□

Proof (Lemma 4) We show the lemma by proving the stronger result that τ-reduction is normalizing and confluent.
First we show that all guarded contracts are τ-normalizing, i.e., there exists a (τ-normal form) c' s.t. $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau^*} c'$ and for no c'' $\mathrm{D}, \delta \vdash_C c' \xrightarrow{\tau} c''$. Proof by induction on the (minimal) height of the derivation of guardedness of c. Use Fig. 10.

– Assume $\mathrm{D} \vdash \mathrm{Success}$ guarded. Clearly, there is no rule such that Success reduces via τ, so we must already have a τ-normal form.
– Assume $\mathrm{D} \vdash \mathrm{Failure}$ guarded. Analogous to the case above.
– Assume $\mathrm{D} \vdash \mathrm{transmit}(\mathbf{X} \mid P).\, c$ guarded. Analogous to the cases above.
– Assume $\frac{\mathrm{D} \vdash c \text{ guarded} \quad (f(\mathbf{X}) = c) \in \mathrm{D}}{\mathrm{D} \vdash f(\mathbf{a}) \text{ guarded}}$. As $(f(\mathbf{X}) = c) \in \mathrm{D}$ we can build a derivation of
  $\mathrm{D}, \delta \vdash_C f(\mathbf{a}) \xrightarrow{\tau} c[\mathbf{v}/\mathbf{X}]$ where $\mathbf{v} = \mathcal{Q}\llbracket \mathbf{a} \rrbracket^{\delta}$. It is left to show that $c[\mathbf{v}/\mathbf{X}]$ is τ-normalizing.
  Claim: For any derivation of $\mathrm{D} \vdash c$ guarded there is a derivation of $\mathrm{D} \vdash c[/\mathbf{X}]$ guarded of equal height.
  Proof of claim: By induction on guardedness.
  By the above claim it follows that the height of derivation of $\mathrm{D} \vdash c[/\mathbf{X}]$ guarded is the same as the height of $\mathrm{D} \vdash c$ guarded, which is less than the height of $\mathrm{D} \vdash f(\mathbf{a})$ guarded. Applying the induction hypothesis to $\mathrm{D} \vdash c[/\mathbf{X}]$ guarded we get that $c[/\mathbf{X}]$ is τ-normalizing and as $\mathrm{D}, \delta \vdash_C f(\mathbf{a}) \xrightarrow{\tau} c[/\mathbf{X}]$ also that $f(\mathbf{a})$ is τ-normalizing.
– Assume $\frac{\mathrm{D} \vdash c \text{ guarded} \quad \mathrm{D} \vdash c' \text{ guarded}}{\mathrm{D} \vdash c + c' \text{ guarded}}$. There are three cases to consider.
  1. Suppose $\frac{\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} d}{\mathrm{D}, \delta \vdash_C c + c' \xrightarrow{\tau} d + c'}$. By the IH we are done.
  2. Suppose $\frac{\mathrm{D}, \delta \vdash_C c' \xrightarrow{\tau} d'}{\mathrm{D}, \delta \vdash_C c + c' \xrightarrow{\tau} c + d'}$. Again by the IH we are done.
  3. Suppose $\mathrm{D}, \delta \vdash_C \mathrm{Success} + \mathrm{Success} \xrightarrow{\tau} \mathrm{Success}$. But Success is already a τ-normal form so we are done.
– Assume $\frac{\mathrm{D} \vdash c \text{ guarded} \quad \mathrm{D} \vdash c' \text{ guarded}}{\mathrm{D} \vdash c \parallel c' \text{ guarded}}$. There are four cases to consider.
  1. Suppose $\frac{\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} d}{\mathrm{D}, \delta \vdash_C c \parallel c' \xrightarrow{\tau} d \parallel c'}$. By the IH on the two premises of the derivation of guardedness of $c \parallel c'$, we obtain what was required.

2. Suppose $\frac{\mathrm{D},\delta\vdash_C c' \xrightarrow{\tau} d'}{\mathrm{D},\delta\vdash_C c\|c' \xrightarrow{\tau} c\|d'}$. Analogous to the case just shown.
3. Suppose $\mathrm{D}, \delta \vdash_C$ Success $\|\ c \xrightarrow{\tau} c$. By assumption $\mathrm{D} \vdash c$ guarded and by the IH we are done.
4. Suppose $\mathrm{D}, \delta \vdash_C c \,\|$ Success $\xrightarrow{\tau} c$. By assumption, we have $\mathrm{D} \vdash c'$ guarded and by the IH we are done.

– Assume $\frac{\mathrm{D}\vdash c \text{ guarded} \quad \mathrm{D}\vdash c' \text{ guarded}}{\mathrm{D}\vdash c;c' \text{ guarded}}$. There are two cases to consider.
  1. Suppose $\frac{\mathrm{D},\delta\vdash_C c \xrightarrow{\tau} d}{\mathrm{D},\delta\vdash_C c;c' \xrightarrow{\tau} d;c'}$. Easy, by the IH.
  2. Suppose $\mathrm{D}, \delta \vdash_C$ Success; $c' \xrightarrow{\tau} c'$. Immediate by the IH.

Second, we prove confluence by showing that the diamond property holds for τ-reduction, i.e., if $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} c'$ and $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} c''$, then there exists a d with $\mathrm{D}, \delta \vdash_C c' \xrightarrow{\tau^=} d$ and $\mathrm{D}, \delta \vdash_C c'' \xrightarrow{\tau^=} d$. The proof is by induction on the derivation of $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} c'$.

– $\frac{(f(\mathbf{X})=c)\in\mathrm{D}, \mathbf{v}=\mathcal{Q}[\![\mathbf{a}]\!]^\delta}{\mathrm{D},\delta\vdash_C f(\mathbf{a}) \xrightarrow{\tau} c[\mathbf{v}/\mathbf{X}]}$.
  No other rules match any subterm of $f(\mathbf{a})$, and we must hence have $c' = c''$, whence the result follows.
– $\frac{\mathrm{D},\delta\vdash_C c_1 \xrightarrow{\tau} d_1}{\mathrm{D},\delta\vdash_C c_1+c_2 \xrightarrow{\tau} d_1+c_2}$.
  If the τ-rewrite step $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} c''$ takes place inside c, we have $c'' = c_1'' + c_2$, and the IH furnishes a d_1' such that $\mathrm{D}, \delta \vdash_C c_1'' \xrightarrow{\tau^=} d_1'$ and $\mathrm{D}, \delta \vdash_C d_1 \xrightarrow{\tau^=} d_1'$. We hence have $\mathrm{D}, \delta \vdash_C c' \xrightarrow{\tau^=} d_1' + c_2$ and $\mathrm{D}, \delta \vdash_C c'' \xrightarrow{\tau^=} d_1' + c_2$, as desired.
– $\frac{\mathrm{D},\delta\vdash_C c_2 \xrightarrow{\tau} d_2}{\mathrm{D},\delta\vdash_C c_1+c_2 \xrightarrow{\tau} c_1+d_2}$.
  As the previous case.
– $\mathrm{D}, \delta \vdash_C$ Success + Success $\xrightarrow{\tau}$ Success.
  In this case, we must have $c' = c''$, and the result follows.
– $\frac{\mathrm{D},\delta\vdash_C c \xrightarrow{\tau} d}{\mathrm{D},\delta\vdash_C c\|c' \xrightarrow{\tau} d\|c'}$.
  Exactly as the case $\frac{\mathrm{D},\delta\vdash_C c_1 \xrightarrow{\tau} d_1}{\mathrm{D},\delta\vdash_C c_1+c_2 \xrightarrow{\tau} d_1+c_2}$.
– $\frac{\mathrm{D},\delta\vdash_C c' \xrightarrow{\tau} d'}{\mathrm{D},\delta\vdash_C c\|c' \xrightarrow{\tau} c\|d'}$. As the previous case.
– $\mathrm{D}, \delta \vdash_C$ Success $\|\ d \xrightarrow{\tau} d$.
  In this case, $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} c''$ must be an application of either of the rules $\mathrm{D}, \delta \vdash_C c \,\|$ Success $\xrightarrow{\tau} c$, or $\frac{\mathrm{D},\delta\vdash_C d \xrightarrow{\tau} d'}{\mathrm{D},\delta\vdash_C c_1+d \xrightarrow{\tau} c_1+d'}$. In the first case, we have $c' =$ Success $= c''$, and we are done. in the second case, we have $c_1 =$ Success, and thus $\mathrm{D}, \delta \vdash_C d \xrightarrow{\tau} d'$ and $\mathrm{D}, \delta \vdash_C c_1 + d \xrightarrow{\tau} d'$, as desired.
– $\mathrm{D}, \delta \vdash_C c \,\|$ Success $\xrightarrow{\tau} c$.
  Symmetric to the previous case.
– $\frac{\mathrm{D},\delta\vdash_C c_1 \xrightarrow{\tau} d_1}{\mathrm{D},\delta\vdash_C c_1;c_2 \xrightarrow{\tau} d_1;c_2}$.
  If the reduction step $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} c''$ takes place inside c_1, then $c'' = d_1''; c_2$, and the IH furnishes existence of a d_1' such that $\mathrm{D}, \delta \vdash_C d_1 \xrightarrow{\tau^=} d_1'$ and $\mathrm{D}, \delta \vdash_C d_1 \xrightarrow{\tau^=} d_1'$. Then, $d_1'; c_2$ is a common τ-reduct of c' and c'', and the desired result follows. Otherwise, $\mathrm{D}, \delta \vdash_C c \xrightarrow{\tau} c''$ is an application of the rule $\mathrm{D}, \delta \vdash_C$ Success; $c' \xrightarrow{\tau} c'$, which is impossible, as Success is a τ-normal form, i.e., it cannot be the case that $\mathrm{D}, \delta \vdash_C c_1 \xrightarrow{\tau} d_1$.
– $\mathrm{D}, \delta \vdash_C$ Success; $c' \xrightarrow{\tau} c'$.
  Symmetric to the previous case.

□

Proof (Theorem 5) "If": By induction on the height of the derivation of $\mathrm{D}, \delta \vdash_C c \xrightarrow{\mathbf{d}e} c'$. "Only if": By induction on the height of the derivation of $\mathrm{D}, \delta \vdash_N c \xrightarrow{e} c'$.

Proving "Only if": (note that we only consider non-τ-derivations)

– Assume $\mathrm{D}, \delta \vdash_N$ Success $\xrightarrow{e}$ Failure. From the reduction semantics we see that there is just one possible reduction $\mathrm{D}, \delta \vdash_C$ Success $\xrightarrow{e} c'$ giving $c' =$ Failure so $c'' =$ Failure.
– Assume $\mathrm{D}, \delta \vdash_N$ Failure $\xrightarrow{e}$ Failure. Analogous.
– Assume $\frac{\delta\oplus\mathbf{X}\mapsto\mathbf{v}\models P, \mathbf{v}=\mathcal{Q}[\![\mathbf{a}]\!]^\delta}{\mathrm{D},\delta\vdash_N \mathrm{transmit}(\mathbf{X}|P).\, c \overset{\mathrm{transmit}(\mathbf{v})}{\longrightarrow} c[\mathbf{v}/\mathbf{X}]}$.
  Again we see that there is a unique reduction of the transmit$(\mathbf{X}|P).\,c$-contract and we have,
  $$\frac{\delta \oplus \mathbf{X} \mapsto \mathbf{v} \models P, \mathbf{v} = \mathcal{Q}[\![\mathbf{a}]\!]^\delta}{\mathrm{D}, \delta \vdash_C \mathrm{transmit}(\mathbf{X} \mid P).\, c \overset{\mathrm{transmit}(\mathbf{v})}{\longrightarrow} c[\mathbf{v}/\mathbf{X}]}$$
  by which we conclude $c'' = c[\mathbf{v}/\mathbf{X}]$.
– Assume $\frac{\delta\oplus\mathbf{X}\mapsto\mathbf{v}\models P, \mathbf{v}=\mathcal{Q}[\![\mathbf{a}]\!]^\delta}{\mathrm{D},\delta\vdash_N \mathrm{transmit}(\mathbf{X}|P).\, c \overset{\mathrm{transmit}(\mathbf{v})}{\longrightarrow} \mathrm{Failure}}$. Analogous.
– Assume $\frac{\mathrm{D},\delta\vdash_N c \xrightarrow{e} d}{\mathrm{D},\delta\vdash_N c\|c' \xrightarrow{e} d\|c'}$. By the IH we gather that $\mathrm{D}, \delta \vdash_C c \xrightarrow{\mathbf{d}e} d$. We can extend $\mathbf{d}$ with l and build a *unique* derivation $\frac{\mathrm{D},\delta\vdash_C c \xrightarrow{\mathbf{d}e} d}{\mathrm{D},\delta\vdash_C c\|c' \xrightarrow{l\mathbf{d}e} d\|c'}$.
– Assume $\frac{\mathrm{D},\delta\vdash_N c' \xrightarrow{e} d'}{\mathrm{D},\delta\vdash_N c\|c' \xrightarrow{e} c\|d'}$. Analogously by extending $\mathbf{d}$ with r.
– Assume $\frac{\mathrm{D},\delta\vdash_N c \xrightarrow{e} d}{\mathrm{D},\delta\vdash_N c;c' \xrightarrow{e} d;c'}$. By the IH we have a derivation $\mathrm{D}, \delta \vdash_C c \xrightarrow{e} d$. Thus we can construct the unique derivation $\frac{\mathrm{D},\delta\vdash_C c \xrightarrow{e} d}{\mathrm{D},\delta\vdash_C c;c' \xrightarrow{e} d;c'}$.

Proving "If":

– Assume $\mathrm{D}, \delta \vdash_C$ Success $\xrightarrow{e}$ Failure. There is no $\mathbf{d}$ in this case, and we can immediately build $\mathrm{D}, \delta \vdash_N$ Success $\xrightarrow{e}$ Failure, also choosing no τ-transitions for the first part.
– Assume $\mathrm{D}, \delta \vdash_C$ Failure $\xrightarrow{e}$ Failure. Analogous.
– Assume $\frac{\delta\oplus\mathbf{X}\mapsto\mathbf{v}\models P, \mathbf{v}=\mathcal{Q}[\![\mathbf{a}]\!]^\delta}{\mathrm{D},\delta\vdash_C \mathrm{transmit}(\mathbf{X}|P).\, c \overset{\mathrm{transmit}(\mathbf{v})}{\longrightarrow} c[\mathbf{v}/\mathbf{X}]}$. Take no $\mathbf{d}$ and choose no
  τ-transitions. Then immediate.
– Assume $\frac{\delta\oplus\mathbf{X}\mapsto\mathbf{v}\not\models P, \mathbf{v}=\mathcal{Q}[\![\mathbf{a}]\!]^\delta}{\mathrm{D},\delta\vdash_C \mathrm{transmit}(\mathbf{X}|P).\, c \overset{\mathrm{transmit}(\mathbf{a})}{\longrightarrow} \mathrm{Failure}}$. Analogous.
– Assume $\frac{\mathrm{D},\delta\vdash_C c \xrightarrow{\mathbf{d}e} c'}{\mathrm{D},\delta\vdash_C c+d \xrightarrow{f\mathbf{d}e} c'}$. Must build a derivation of $\mathrm{D}, \delta \vdash_N c + d \xrightarrow{\tau^*} c'' \xrightarrow{e} c'$. By IH: $\mathrm{D}, \delta \vdash_N c \xrightarrow{\tau^*} c'' \xrightarrow{e} c'$. So we just need the first part. Clearly, we have $\mathrm{D}, \delta \vdash_N c + d \xrightarrow{\tau} c$. Thus, by choosing $c = c''$ and exactly one τ-transition, we are done.
– Assume $\frac{\mathrm{D},\delta\vdash_C d \xrightarrow{\mathbf{d}e} d'}{\mathrm{D},\delta\vdash_C c+d \xrightarrow{s\mathbf{d}e} d'}$. Analogous.
– Assume $\frac{\mathrm{D},\delta\vdash_C c \xrightarrow{\mathbf{d}e} d}{\mathrm{D},\delta\vdash_C c\|c' \xrightarrow{l\mathbf{d}e} d\|c'}$. By using the IH, taking $c = c''$, and making no τ-transitions in the first part, we are done.
– Assume $\frac{\mathrm{D},\delta\vdash_C c' \xrightarrow{\mathbf{d}e} d'}{\mathrm{D},\delta\vdash_C c\|c' \xrightarrow{r\mathbf{d}e} c\|d'}$. Analogous.
– Assume $\frac{\mathrm{D},\delta\vdash_C c \xrightarrow{e} d}{\mathrm{D},\delta\vdash_C c;c' \xrightarrow{e} d;c'}$. Analogous.
– Assume $\frac{\mathrm{D},\delta\vdash_C c \xrightarrow{e} c'}{\delta\vdash_C \mathrm{letrec}\,\mathrm{D} \text{ in } c \xrightarrow{e} \mathrm{letrec}\,\mathrm{D} \text{ in } c'}$. Analogous.

It is obvious that, if $\mathbf{d}$ exists in the above case, it is unique. Furthermore, for all c'' such that $\mathrm{D}, \delta \vdash_C c \xrightarrow{\mathbf{d}e} c''$ we have $c' = c''$. Again, it is obvious.

□

References

1. Andersen, J., Elsborg, E.: Compositional specification of commercial contracts. M.S. term project, December (2003)
2. Arkin, A.: Business process modeling language, (2002)
3. Baeten, J.C.M., Middelburg, C.A.: Process Algebra with Timing. Springer,Berlin Heidelberg New York (2002)
4. Baeten, J.C.M., Weijland, W.P.: Process Algebra. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge (1990)
5. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM 31(3),560–599 (1984)
6. Conway, J.H.: Regular Algebra and Finite Machines. Chapman & Hall,London (1971)
7. Eber, J.-M.: Personal communication, June (2002)
8. Geerts, G., McCarthy, W.E.: The ontological foundations of rea enterprise information systems. Unpublished, August (2000)
9. Hennessy, M.: Algebraic Theory of Processes. MIT Press,Cambridge (1988)
10. Hoare, C.A.R.: Communicating Sequential Processes. International Series in Computer Science. Prentice-Hall,Englewood Cliffs (1985)
11. Jones, S.P., Eber, J.-M.: How to write a financial contract. In Gibbons, J., de Moor, O. (eds) The Fun of Programming. Palgrave Macmillan, Australia (2003)
12. Jones, S.P., Eber, J.-M., Seward, J.: Composing contracts: an adventure in financial engineering (functional pearl). In: Proceedings of the fifth ACM SIGPLAN International Conference on Functional Programming, pp 280–292. ACM Press,Washington (2000)
13. Kristoffersen, K.J., Pedersen, C., Andersen, H.R.: Runtime verification of timed LTL using disjunctive normalized equation systems. Unpublished, September (2003)
14. Kristoffersen, K.J., Pedersen, C., Andersen, H.R.: Checking temporal business rules. In: Proceedings of the First International REA Workshop (2004)
15. McCarthy, W.E.: The REA accounting model: a generalized framework for accounting systems in a shared data environment. Account. Rev. LVII(3),554–578 (1982)
16. Milner, R.: Communication and Concurrency. International Series in Computer Science. Prentice-Hall,Englewood Cliffs (1989)
17. Milner, R.: Communicating and Mobile Systems: The π-Calculus. Cambridge University Press,Cambridge (1999)
18. Milner, R., Parrow, J., D. Walker: A calculus of mobile processes, parts I and II. Inf. Comput. 100(1),1–77 (1992)
19. Singh, M.P., Meredith, G., Tomlinson, C., Attie, P.C.: An event algebra for specifying and scheduling workflows. In: Database Systems for Advanced Applications, pp 53–60 (1995)
20. van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., Wohed, P.: Pattern-based analysis of BPML (and WSCI). Technical Report FIT-TR-2002-05, Queensland University (2002)
21. van der Aalst, W., van Hee, K.: Workflow Management—Models, Methods, and Systems. MIT Press,Cambridge (2002)
22. Winskel, G.: The Formal Semantics of Programming Languages. MIT Press,Cambridge (1993)