An Algebraic Approach to Formal System Metatheory

Francesco Gavazzo
University of Padova, Italy

Abstract

We introduce an algebraic approach to the metatheory of formal systems of term assertions akin to those of structural and natural operational semantics, type theories, rewriting systems, and equational theories. We rest on *Term Relation Algebras*, *viz.* pointfree algebras of structurally-defined predicates on terms, to give a common algebraic semantics to different kinds of formal systems, regardless of their inferential mechanisms and underlying term structures. This enables abstract reasoning about formal systems, and their metatheory, through the algebra of the term predicates they define. We give a faithfully *term-free* algebraization of semantically-relevant term predicates, such as parallel reduction, big-step evaluation, and applicative bisimilarity. We extend this algebraization also to fundamental metatheoretical properties of these notions – including congruence of applicative bisimilarity, determinacy of evaluation, and confluence of reduction – and prove these properties using algebraic methods.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases Relation Algebra, Term Relations, Language Metatheory, Formal System Metatheory

Digital Object Identifier 10.4230/LIPIcs.LICS.2026.50

Funding This work was partially supported by the 2026 STARS Grants@Unipd CoG *RELATIONS* and by PRIN 2022 PNRR Project No. P2022HXNSC.

1 Introduction

The situation in the semantics of programming languages today bears a striking resemblance to the early landscape of program logics – and, we shall argue, calls for a similar remedy.

By the late 1980s and 1990s, the landscape of program logics was highly fragmented, characterised by a diversity of formal systems – *Hoare logic* [72], *dynamic logic* [125, 65], *predicate transformer semantics* [42, 41] – each with its own specialised syntax, deductive apparatus, and metatheory. A turning point came with the discovery that all these systems share a common *algebraic semantics*, as embodied by fragments of *relation algebra* [97, 143] – notably (extensions of) *Kleene algebras* [32, 85, 83, 86] and *dynamic algebras* [127, 30]. These algebras not only provided an abstract, *syntax-independent* framework for studying different formal systems *uniformly*, but also enabled the use of algebraic methods to conduct metatheory, yielding general results that, proved *once*, apply *directly* to all systems, and establishing novel relationships between apparently unrelated metalogical properties – so-called "bridge theorems".

Today, a similar fragmentation afflicts the semantics of programming languages. Formal systems of term assertions – *structural and natural operational semantics* [79, 106, 105, 122], *type theories* [99], *rewriting systems* [74, 108], *reduction semantics* [47], and *equational theories* [27] – are routinely used to give semantics to programming languages, while their metatheoretical properties – *congruence of semantic equivalence* [107, 22, 33], *subject reduction and expansion* [33, 148] (*type safety* [104]), *confluence* and *normalization* [108, 33], and *parametricity* [130] – provide ways to ensure different forms of semantic correctness. Each of these formal systems comes with its own syntax of assertions and terms (first-order [29],


© Francesco Gavazzo;
licensed under Creative Commons License CC-BY 4.0

second-order terms [7, 82], nominal terms [52, 115], string diagrams [135]), as well as its specialised deductive apparatus, so that one routinely has to reprove essentially the same metatheoretical results for any new language or variation in the underlying term structure. Since the semantics of a programming language is not given by a single formal system but by an entire ecosystem of them, this situation calls for more abstract, uniform accounts of these assertional systems and their metalogical properties. This becomes even more desirable once one observes that *different* metatheorems – even across different types of formal systems – are typically established by resting on similar, if not identical, constructions – *parallel reduction* [14, 141], *Howe's method* [73, 114], and *logical relations* [130, 121], *etc.* – which points to nontrivial relationships between metatheorems and calls for precisely such bridge theorems.

And indeed, various abstract accounts have been given for *specific* metatheorems and formal systems, notable examples including confluence results on rewriting systems over large classes of term structures [7, 82, 74] and abstract congruence theorems for (applicative) bisimilarity [73, 145, 26, 71, 60]. Yet, a general theory targeting *all* such systems and their metatheories *uniformly*, within a single formal framework, remains missing. Luckily, the history of program logics suggests where to look. Much as Hoare distilled the success of algebraization in program logic into the slogan "*algebra unifies calculi of programming*", we ask: *can it unify calculi of language semantics, too?* In this work, following up on a previous paper by the author [53], we provide evidence for an affirmative answer.

1.1 Algebras of Term Predicates

At the heart of this answer lies a threefold observation.

- First, all the aforementioned formal systems admit a uniform "provability semantics" by interpreting concrete terms into abstract term structures [59] – *abstract syntax* [103, 33] – and term assertions as predicates on such structures – *(abstract) term predicates*. This suggests that, similarly to the case of program logics, a unifying account of formal systems can be given semantically, through the abstract term predicates these systems induce (reduction, evaluation, typing, *etc.*).[1] As further evidence, one observes that even meta-theoretical notions, such as semantic equivalences and refinements, are naturally interpreted as (abstract) term predicates, too.
- Second, under this semantic perspective, the inferential mechanisms used by the various types of formal system (syntax-directed rules, contextual rules, *etc.*) and their underlying term manipulations (substitution, renaming, pattern-matching) provide syntactic realisations of *structural* predicate definitions, so that those term predicates arising from the semantics and metatheory of formal systems correspond to those defined using *logical* and *structural* principles only.
- Third, many metatheoretical properties of formal systems are naturally expressed as structural and logical properties of term predicates, such as coherence of typing and evaluation, well-foundedness of reduction, and congruence of bisimilarity.

Altogether, these observations suggest that the unifying account that we aim for can be given in the form of an *algebra of term predicates*, namely an algebra of predicates on (syntactically) *structured domains* whose operations account for predicate definitions by both

[1] This is very close to the *Abstract Algebraic Logic* [21] approach to logical systems, where, following insights by Tarski, a logical system is not viewed as a proof-theoretic structure, but identified with the consequence relation on the (term) algebra of formulas it induces.

logical operations (conjunction, disjunction, universal quantification, *etc.*) and structural manipulations (structural recursion, substitution, pattern-matching, *etc.*), and whose laws encompass both logical inference and structural reasoning (structural induction, equational theories of substitution, *etc.*).[2]

Such algebras are the main contribution of a previous work by the author [53] where, taking a *relational* view on predicates, it is shown that not only virtually any term structure, whether abstract or concrete, naturally carries an algebra of (structurally and logically defined) predicates on its terms, but also that the operations and laws defining such an algebra are the *same* across *all* term structures. That is to say, all term structures (unscoped and well-scoped first-order terms, nameless and nominal terms, string diagrams, *etc.*), no matter how their basic manipulations are defined (structural recursion, natural number recursion, rule induction, *etc.*), they all share essentially the same algebra of term predicates, whose structure can be expressed by means of operations and laws that make no reference to the underlying terms. This defines a novel class of algebraic structures – dubbed *Term Relation Algebras* (TRAs) – that axiomatise algebras of term predicates in a complete *syntax-independent* fashion, as *pointfree* – *term-free* – algebras of relations extended with operations and laws distilling the essence of structural definition and proof principles.

1.2 Pointfree Metatheory of Formal Systems

In this work, we show that TRAs provide the unifying account of formal systems of term assertions that we seek. More precisely, TRAs offer a *pointfree* theory of formal systems, and their metatheory, close in spirit to *pointfree topology* [147]: much like the latter studies spaces through the algebra of their open sets, without reference to a predefined underlying set of points, TRAs enable the study of formal systems through the algebra of the term relations these systems define, without reference to any underlying term or syntax structure.

We will show that TRAs can express many term predicates defined by various formal systems, including forms of reduction and evaluation given by rewriting systems, reduction semantics, and structural and natural operational semantics, together with the semantic equivalences they induce (bisimilarity, contextual equivalence, Kleene equality). Crucially, even if term-free, the algebraization of these formal systems is faithful, in the sense that the algebraic operations of TRAs faithfully account for (*termwise*) structural definitions realised by the formal systems' syntactic and inferential machinery. Such algebraization extends to metatheoretical properties of formal systems: we will prove that many fundamental metalogical properties of operational semantics, rewriting systems, equational theories, and reduction semantics (other systems that will be treated in future work include type theories and proof calculi) can be recast as properties of term predicates and expressed by purely algebraic means – mostly through equations and quasi-equations – in the language of TRAs. As a payoff, we will leverage algebraic methods to prove keystone metatheorems, such as confluence of orthogonal reduction [74, 7] and congruence of applicative bisimilarity [73, 3], in complete abstraction from any term structure and syntactic presentation. Importantly, once such results have been proved – something done *once and for all* within the setting of TRAs

[2] Notice that while there is an extensive literature on algebras of predicates – both relationally [143, 51, 142] and functionally [67, 91] conceived – all such algebras model predicates over *abstract*, *unstructured* domains, accounting for predicate definition and reasoning without any reference to the internal structure of the objects (terms) on which predicates act. (This is indeed *logicality* in the sense of Tarski [144].)

– they immediately apply to *any* term structure (and formal system over that structure) carrying a TRA structure – and, as already mentioned, virtually all term structures used in the semantics of programming languages do carry such a structure.[3]

The Algebraic Lens. To achieve the aforementioned results, we will recast algebraically, in the setting of TRAs, many syntactic and term-based notions. It was already noticed [53] that the very defining operations and laws of TRAs shed light on various syntactic constructions, especially those belonging to so-called syntactic methods in rewriting (*e.g.* parallel reduction [141]), operational semantics (*e.g.* Howe's method [73, 62, 89]), and concurrency (*e.g.* up-to techniques [132, 123]). Here, we considerably extend the reach of that observation by showing that many fine-grained syntactic notions – such as ones of an open and closed term – that are usually presented termwise by rather *ad hoc* definitions, once seen through the algebraic lens of TRAs, not only take the form of general algebraic constructions, but also reveal deep structures in syntax. A beautiful example of that is the one mentioned earlier of the open and closed terms:[4] at the level of TRAs, these notions are recast as an *adjoint cylinder* [90], *viz.* an adjoint triple of modalities, that form an *axiomatic cohesion* [92].

Unifying Metatheory. Another notable example of the just described phenomenon is given by the notions of *introduction* and *elimination form* [99], which in turn leads to the algebraic formulation of such fundamental semantic principles [66, 134] as Gentzen's *Inversion Principle* and *Conservation Principle* [57]. This algebraization reveals novel and deep relationships in the metatheory of *distinct* formal systems, giving evidence of the unifying character of TRAs. In that setting, many metalogical properties of formal systems are expressed as "local-to-global" implications asserting that *local coherence* propagates to *global coherence* along those combinations of term relation forming operations that describe semantic definitions. More concretely, local correctness properties of ground semantics (primitive reductions, equational axioms, *etc.*) yield global correctness of complete semantic notions (full reduction, equality, semantic equivalence).

While global correctness typically capture the true metatheorems one is interested in (bisimilarity is a congruence, reduction is confluent, typing and evaluation are coherent, *etc.*) on concrete systems, these results typically depend on plenty of local conditions – such as linearity and orthogonality in rewriting systems [74, 18] and rule formats in structural operational semantics [5] – that are specific to the formal system considered, and thus usually each other unrelated. Remarkably, TRAs unify all these local conditions, showing that they can all be seen as distinct "syntactic realisations" of the same fundamental algebraic laws, embodied, in this case, by the aforementioned Inversion and Conservation Principle. As a consequence, we obtain *bridge theorems* showing that superficially unrelated metatheorems follow from the same basic algebraic laws. As a main example of such bridge theorems, we shall prove that whenever a (term relation standing for a) primitive computation rule satisfies both the aforementioned inversion and conservation principles, then: *(i)* its parallel reduction is confluent; *(ii)* a large family of big-step operational semantics over it (including traditional call-by-name and call-by-value ones [119]) are deterministic; *(iii)* the notion of applicative bisimilarity built upon the latter is a congruence.

---

[3] TRA are not merely a convenient algebraic language for carrying out syntactic reasoning. Their purpose is not to replace termwise proofs with algebraic ones, but rather to *eliminate the need for such proofs altogether*

[4] Another example is the one of one-hole contexts and sequentiality, which correspond to algebraic notions of derivative [77, 102, 2] and preservation of relation joins, respectively [110].

Notice, finally, the practical value of this theoretical result. Since TRAs are term-free, the just mentioned bridge theorem is freely available on *any* term structure carrying a TRA structure. This means that once having such a term structure (most of which being provided in this paper, as well as [53]), it is enough to show that the local conditions of the bridge theorem hold to obtain three major metatheoretical results that would otherwise require, even appealing to state-of-the-art general metatheorems, the verification of multiple, diverse syntax-dependent conditions on different formal systems. In the case of second order terms, for example, we obtain a unification of Aczel's confluence theorem [7] and Howe's congruence of applicative bisimilarity [73]. Once we show that the ground semantics on, in this case, a second-order language (typically a form of ground computation, such as the β-rule in the λ-calculus) satisfies the inversion and conservation principle, a body of diverse metatheoretical properties (including the aforementioned confluence and congruence) automatically follow.

1.2.1 Contribution List

Here is a detailed list of the contribution of the present paper.

- Even if TRAs are introduced in [53], that work does not give an explicit axiomatisation of TRAs. As a minor contribution, we give such axiomatisation.
- We algebrize the "syntactic" notions of an open and closed term, introduction and elimination form, value, and program. In doing so, we make explicit some (to the best of the author's knowledge) novel structure on syntax, most notably the modal, axiomatic cohesion structure of open and closed expression. We also algebrize Gentzen's Inversion Principle and Conservation Principle. To the best of the author's knowledge, this is the first formal, language-independent account of such principles.
- We use the resulting algebraic notions to give a faithful algebraization of various classes of natural semantics and rewriting systems by expressing within TRAs term predicates for big-step evaluation, parallel reduction, and many of the semantic equivalences and refinements they induce, most notably conversion, applicative bisimilarity, and Kleene equality. While some results on rewriting systems have already been proved in [53], this work improves such results by relating them to the inversion and conservation principles. Developments in the direction of operational semantics and semantic equivalence, instead, are a completely novel contribution of this work. In the case of big-step semantics, our algebraization uniformly covers different evaluation strategies, including call-by-name and call-by-value ones [119]. This is in agreement with Martin-Löf [99, 4] pre-theoretical account of operational semantics. Due to space constraints we omit algebraization of small-step semantics (structural operational semantics). Similarly, we focus on the algebraization of *applicative bisimilarity* [3], omitting ones of *contextual equivalence* [107], *CIU equivalence* [101], and *Kleene equivalence* [119].
- We use the algebraic machinery to prove three fundamental metatheorems in rewriting, operational semantics, and program equivalence, namely *confluence of orthogonal reduction*, *determinism of program evaluation*, and *congruence of applicative bisimilarity*. We discover that these results follow from essentially the same hypothesis, which boil down to the inversion and conservation principles. This bridge theorem, the main result of the work, gives a novel, unifying perspective on the metatheory of formal systems. Other results we have proved but that we omit due to space constraints include further confluence results (*e.g.* via critical pair-like conditions [110]), equivalence between small-step and big-step semantics, congruence of Kleene and CIU equivalence.

1.3 A Walkthrough Example

Before diving into technical developments, we give an informal introduction to some of the key features of TRAs by commenting one of the metatheorems we shall prove. Here is the (formal statement of the) theorem:

$$a^\circ ; a \leq \Delta \;\&\; a \leq \langle \overline{\Delta} \rangle ; a \implies a^{\Downarrow\circ} ; a^{\Downarrow} \leq \overline{\Delta} \tag{†}$$

Informally, theorem (†) states that if a ground reduction a – such as the β-rule – is deterministic ($a^\circ ; a \leq \Delta$) and satisfies Gentzen's Inversion Principle ($a \leq \langle \overline{\Delta} \rangle ; a$), then the big-step semantics it induces ($a^{\Downarrow}$), evaluates programs to at most one value ($a^{\Downarrow\circ} ; a^{\Downarrow} \leq \overline{\Delta}$). Let us comment on that.

The first observation to make is that we are modelling predicates as binary relations, viewed as abstract algebraic entities, and not as relations of arbitrary ranks or (propositional) functions on terms.[5] Being algebraic, relations are synthetically built from "primitive" relations using various operations, which are conveniently classified into two families.

Logical Operations. Operations such as $\cdot^\circ$, $\cdot_1 ; \cdot_2$, and (the constant) Δ are examples of *logical operations*. These describe, on an algebraic level, the logical structure of relations, *viz.* the one that is completely independent of (the internal structure of) the related objects [143]. For instance, the aforementioned operations correspond to relation converse, composition, and identity (or diagonal), respectively.

Structural Operations. The second class of operations, the one characteristic of TRAs, is the one of *structural operations*. These operations encapsulate how core syntactic manipulations can be used to define relations. Accordingly, they describe the syntactic structure of relations, the one reflecting the internal (syntactic) structure of the objects on which relations act. At its most fundamental level, syntactic terms are recursively built from variables using operators. Variables, moreover, act as placeholders, and thus they can be substituted for terms. All of that is captured by three operations:

- A constant operations Δ_η, which morally relates two terms if they both are a variable, and the same one.
- A unary operation $\widetilde{\cdot}$ with the following intended meaning: $\widetilde{a}$ relates two terms if they are built using the same outermost operator and their immediate sub-terms are pairwise a-related. TRAs "force" this intended interpretation *operationally*, through axioms expressing how syntactic operations interact with logical ones – for instance, we require $\widetilde{a ; b} = \widetilde{a} ; \widetilde{b}$ – and with each other. Crucially, these axioms – which we mathematically justify in Section 3 – constrain the possible interpretations of (otherwise arbitrary) relations to relations on terms. In particular, we can identify the "operational" abstract syntax of terms on which relation acts with the diagonal relation Δ. For example, the equality $\Delta = \widehat{\Delta}$, where we write $\widehat{\cdot}$ for $\Delta_\eta \vee \widetilde{\cdot}$,[6] asserts that terms are either variables or compound terms made of operators and operands (cf. *regularity* of abstract

[5] Apparently, Peirce himself though of his *logic of relatives* [128] as a genuine algebra of predicates rather than as a calculus of binary relations [136]. Korselt (see [142]), however, proved that such a logic (and its evolution as Tarski's relation algebra [143]) has the same expressive power than classical first-order logic with three distinct variables, and thus cannot fully account for ordinary predicates. Nevertheless, the work by Givant and Tarski [142] give evidence that algebras of binary relations are "[...] strong enough to be called a truly first-order (as opposed to propositional) algebraic logic" [9]. Furthermore, the development of *Fork algebras* [51, 16] showed that algebras of binary relations can recover the full expressive power of first-order through the addition of a duplication operation.

[6] $\vee$ is the usual join operation, whereas $\leq$ is its corresponding ordering.

syntax [103]). Asking Δ to be not just one solution of the equation $x = \widehat{x}$, but the least such solution then amounts to make terms recursively generated from variables using operators. This also gives the law [89] $a \leq \widehat{a} \implies \Delta \leq a$ which expresses *structural induction*.

- A binary operation $\cdot_1[\cdot_2]$ with the following intended meaning: two terms are related by $a[b]$ if the terms are obtained from a-related terms by applying pointwise b-related substitutions on them. The usual equational, monoid-like, theory of substitution is forced precisely by requiring $\cdot_1[\cdot_2]$ to form a monoid on relations, with Δ_η acting as unit (*e.g.* $\Delta_\eta[a] = a = a[\Delta_\eta]$). More interestingly, we can ensure compatibility of substitution with term forming operators through the distributivity law $\widetilde{a}[b] \leq \widetilde{a[b]}$. Finally, the parametric action of substitution corresponds to the existence of the b-parametric family right adjoints $\cdot[b] \dashv b \pitchfork \cdot$.

Using the just described basic operations and the algebraic machinery, we can account for even finer syntactic descriptions. For example, in big-step semantics one splits term constructs into *introductory forms* and *eliminatory forms*, making evaluation reduce programs until an introductory form is reached. Algebraically, this amount to give a *cocartesian decomposition* of $\widetilde{\cdot}$. as $\overline{\cdot} \vee \langle \cdot \rangle$, where $\overline{\cdot}$ and $\langle \cdot \rangle$ restrict the behaviour of $\widetilde{\cdot}$ to introductory forms and eliminatory forms, respectively. This also shows in what sense the algebraization of formal systems is faithful: we can account for the structural character of various inferential mechanisms (contextual rules, syntax-directed rules, *etc.*) through composition of TRA operations. For example, rule-based definitions of big-step semantics over a can be algebraically understood as syntactic presentations of $a^{\Downarrow}$, which in turn is the least solution to the algebraic equation $x = \overline{\Delta} \vee \langle x \rangle \,;\, a \,;\, x$.

Coherence Conditions. Let us turn to theorem (†). The conclusion of the implication, $a^{\Downarrow \circ} \,;\, a^{\Downarrow} \leq \overline{\Delta}$, expresses the metatheoretical property of interest, namely that the big-step semantics $a^{\Downarrow}$ is deterministic and evaluates terms to introductory forms. (This may look cryptic at the moment, but once acquainted basic familiarity with algebras of relations it will become crystal clear.) The antecedent, instead, expresses two local conditions that ensure this property. The first requires that ground reduction is, as expected, itself deterministic. The second is the already mentioned *Gentzen's Inversion Principle*. This principle asserts that computation arises when an eliminatory form has an introductory form as major argument (a bell should ring for readers familiar with Proposition-as-Types). Notice that these conditions are indeed *local*, as they express properties of a (primitive computation) and, indirectly, of the syntactic form of terms.

Applicability. At this point, one may have concerns about the concrete applicability of theorem (†). As already mentioned, virtually all the main term structures used in program semantics carry a TRA structure. These encompass *free* and many *quotient*, scoped (*i.e.* context-indexed) and unscoped, term algebras over a large class of signatures – *algebraic* [59], *binding* [7, 82], *nominal* [115], and *monoidal* [78, 135], as well as their multi-sorted refinements – endowed with simultaneous substitution, defined either by structural [118, 49] or natural number [33] recursion, rule induction [116], or even imperatively [138]. More generally, we can canonically construct a TRA without relation substitution for any initial algebra of a polynomial endofuctor on a topos [59, 88]; and for many of the substitution structures studied in the literature (Σ-monoids [49, 48], modules over monads [69], *etc.*), this construction extends to full TRAs [53]. Whenever presenting the syntax of a language with one of those structures, one immediately obtains a large body of notions and results directly from the

general theory TRAs. For example, once a deterministic ground computation on one of such term structures is defined, notions of reduction, big-step semantics, and applicative bisimilarity over it readily follow. In addition, these come with various metatheoretical results, such as theorem (†). The latter, for example, allows us to reduce the proof of determinism of big-step semantics to checking that ground semantics satisfies the inversion principle; similarly, we shall reduce congruence of bisimilarity and confluence and parallel reduction to checking that ground reduction satisfies Gentzen's inversion and conservation principles (Theorem 37).

How difficult is it to check these local conditions? And how restrictive are they? Quite remarkably, checking the local conditions we deal with in this paper is typically an easy task, and many times it can be done by syntax inspection. Furthermore, there are plenty of interesting examples of formal systems satisfying the local conditions of our main metatheorems. These examples include: many λ-calculi [15, 13], such as the untyped call-by-name and call-by-value λ-calculus, and their extensions with additional operators as well as simple, recursive, polymorphic, and dependent types [137, 112]; object calculi, such as the ς-calculus [1] and Featherweight Java [75]; and reduction calculi on string diagrams [23, 24, 25]. While, due to space constraints, we will able to show only but a few of these examples, readers should be able to apply our results to the remaining calculi mentioned.

2 Preliminaries

We assume familiarity with basic category theory [96] and order theory [39]. In particular, we take the order-theoretic notions of poset, meet, join, (complete) lattice, and monotone functions for granted, as well as basic category-theoretic ones. In particular, given monotone functions $F : (A, \leq_A) \rightleftarrows (B, \leq_B) : G$ between posets, we say that F is *left adjoint* to G (and G is *right adjoint* to F), notation $F \dashv G$, if $F(x) \leq_B y \iff x \leq_A G(y)$. We also say that a function F is *join-preserving* (resp. *meet-preserving*) if $F(\bigvee x_i) = \bigvee_i F(x_i)$ (resp. $F(\bigwedge x_i) = \bigwedge_i F(x_i)$). These notions are related by the (posetal, or special) *Adjoint Functor Theorem* [39, 96]: a function $F : A \to A$ is join-preserving (resp. *meet-preserving*) if and only if it has a right (resp. left) adjoint. This theorem is particularly useful because it allows us to express continuity conditions without resting on infinitary operations simply by requiring the existence of suitable adjoints.

Finally, we recall that, by Knaster-Tarski Theorem [39], any monotone function $F : A \to A$ on a complete lattice has both *least* and *greatest* fixed points, denoted by μF (or $\mu x.F(x)$) and νF (or $\nu x.F(x)$), respectively. In particular, we can express that an element m is a least fixed point of F using the fixed point law $F(m) = m$ and fixed point induction principle $F(x) \leq x \implies m \leq x$ (and similarly for greatest fixed points).

2.1 Allegories and Relation Algebras

We also assume some familiarity with the basic theory of *allegories* [50] and *relation algebras* [97, 133]. We shall use the former as a compact and canonical vocabulary for the latter, viewing algebras of relations as various kinds of one-object allegory, *i.e.* allegories devoid of proper categorical structure. For that reason, we tacitly collapse most definitions to the one-object case.

▶ **Definition 1.** *An* allegory *[50] is a set $\mathcal{A}$ endowed with: (i) a monoid multiplication $;$, called composition, with unit Δ; (ii) a partial order $\leq$ and a meet operation $\wedge$ making composition monotone; (ii) an order-preserving contravariant involution $-^\circ$ (so that $a^{\circ\circ} = a$, $(a\,;b)^\circ = b^\circ\,;a^\circ$, and $(a \wedge b)^\circ = a^\circ \wedge b^\circ$). All these data, additionally, have to obey the so-called* modular law*: $a\,;b \wedge c \leq (a \wedge c\,;b^\circ)\,;b$.*

(One-object) allegories correspond to *positive* fragments of relation algebras, namely algebras of relations without complement (logical negation). For that reason, we often use a "relational vernacular" to talk about allegories, referring to elements of an allegory as *relations*, to Δ as the identity (or diagonal) relation, and to relations a° as the *converse* of a. As usual, we say that a relation a is *reflexive*, *coreflexive*, *symmetric*, and *transitive* if $\Delta \leq a$, $a \leq \Delta$, $a^\circ \leq a$, and $a \,;\, a \leq a$, respectively.

Coreflexive relations (just coreflexives) correspond to unary relations. Thinking about an allegory as an algebra of relations over some object A, coreflexives simulate relationally 'subsets of A', the identity relation corresponding to the whole A itself. Indeed, given two coreflexives p, q, we see that $p \,;\, q$ coincides with $p \wedge q$. In this way, any property P on the intended domain A on which relations act can be regarded as a coreflexive p (on set-theoretic relations, *i.e.* elements in $\wp(W \times W)$, for example, any $P \in \wp(W)$ defines a coreflexive $p \triangleq \{(x, y) \mid x = y \;\&\; P(x)\}$). Proving that any element has the property P then amounts to showing $\Delta \leq p$.

2.1.1 Locally-Complete Allegories

Allegories as they are offer a rather limited algebra of relations, and for our purposes the notion of a *locally complete allegory* [50] is more appropriate (although we may also work with finitary algebras, such as *division allegories* [50]).

▶ **Definition 2.** *A* locally complete allegory *is an allegory $\mathcal{A}$ endowed with a complete lattice such that composition and finite intersection distributing over arbitrary joins.*

We denote by $a \vee b$ the relation $\bigvee\{a, b\}$ and by $\bot$ the relation $\bigvee \emptyset$. Notice that in a locally-complete allegory we have $a \,;\, \bigvee_i b_i = \bigvee_i a \,;\, b_i$ and $(\bigvee_i a_i) \,;\, b = \bigvee_i a_i \,;\, b$ (and thus also $a \,;\, \bot = \bot = \bot \,;\, a$).

Since a locally-complete allegory is a complete lattice, we can define relations on it (co)inductively using the Knaster-Tarski theorem, as well as exploit adjunction relationships between relations. As an example of the former, we see that for any relation $a \in \mathcal{A}$, the monotone parametric function $\Delta \vee a \,;\, \cdot$ has a least fixed point a, denoted by a^*. This gives a uniform fixed point operation $\cdot^*$ which is referred to as *iteration*. As an example of the latter, observe that any locally-complete allegory is also a *division* allegory [50], which means that for all relations $a, b \in \mathcal{A}$, we have *right implication* $a \to b$ and *left implication* $b \leftarrow a$ (the latter actually being $(a^\circ \to b^\circ)^\circ$) which give right adjoints of relation composition (most notably, $a \,;\, b \leq c \iff b \leq a \to c \iff a \leq c \leftarrow b$).[7]

For simplicity, we refer to locally complete allegories simply as *relation algebras* (RAs, for short).

▶ **Example 3.**

1. The prototypical example of a RA is given by set-theoretic endorelations. For any set W, the powerset $\wp(W \times W)$ carries the structure of a RA. In particular, we have [126]: $\phi \to \psi = \{(w, v) \mid \forall u \in W.\ (u, w) \in \phi \Rightarrow (u, v) \in \psi\}$ and $\psi \leftarrow \phi = \{(w, v) \mid \forall u \in W.\ (v, u) \in \phi \Rightarrow (w, u) \in \psi\}$.
2. Given a set I and a set W, we consider *dependent sets*, namely families $A \in \prod_{i \in \wp_{\mathrm{fin}}(I)} \wp(W)$ of sets indexed by finite subsets of I. As usual, we sometimes write a dependent set A as the sequence $(A(i) \mid i \in \wp_{\mathrm{fin}}(I))$. A *dependent relation* ϕ between dependent sets A and

[7] Cf. *action algebras* [126, 84] and *quantales* [131, 113].

B is an element in $\prod_i \wp(A(i), B(i))$ *closed under renaming*: for any function (renaming) $f : i \to j$, if $(\mathtt{t}, \mathtt{s}) \in \phi(i)$, then $(\mathtt{t}, \mathtt{s}) \in \phi(j)$. With these definitions, the RA structure of $\wp(A(i), A(i))$ extends pointwise to $\prod_i \wp(A(i), A(i))$, endowing the collection of dependent endo-relations over a dependent set with a RA structure, too [113].

2.2 Relators

Many useful notions on (term) relation algebras can be uniformly organised as structure-preserving maps on allegories. Such maps have been extensively studied in the theory of allegories, where they go under the name of a *relator* [81, 19, 11, 12].

Recall that an allegory $\mathcal{A}$ being a monoid, it is a (one-object) category. Therefore, we have notions of a *functor*, *lax functor*, and *colax functor* between allegories (a colax functor satisfying $F\Delta \leq \Delta$ and $F(a\,;b) \leq Fa\,;Fb$, and a lax functor being defined dually).

▶ **Definition 4.** *A* relator *between allegories $\mathcal{A}$ and $\mathcal{B}$ is a monotone functor $F : \mathcal{A} \to \mathcal{B}$ that preserves converse. Replacing functors with colax functors (resp. lax functor), we obtain the notion of a* colax relator *(resp.* lax relator*). A colax relator which is strict on composition ($F(a\,;b) = Fa\,;Fb$) is called a* semirelator*.*

In particular, a relator F satisfies the law $F(a^\circ) = (Fa)^\circ$, so that we can unambiguously write Fa°. We say that a relator F is ω-cocontinuous if it preserves joins of ω-chains [8].

We will extensively use semirelators to define certain decompositions of relators that capture coproduct structure on a base in terms of operations on its algebra of relations.

▶ **Definition 5.** *A* cocartesian decomposition *of a relator F is a pair of semirelators F_l, F_r such that $F = F_l \vee F_r$ and and $F_l\,;F_r \leq \bot$, where relation operations are extended to relators pointwise. We refer to the triple (F, F_l, F_r) as a* cocartesian relator triple*.*

All these notions generalise to n-ary relators in the natural way.

3 Term Relation Algebras: Axioms

Term Relation Algebras arise as a pointfree axiomatisation of calculi (or logic) of relations on abstract term structures. Such structures are meant to capture the essential syntactic structure of terms devoid of inessential details, an objective typically achieved through categorical structures such as *initial algebras* [59, 88], *substitution monoids* [49, 48], *etc.*, which are characterised by suitable universal properties. Many such structures have been proposed over the years: initial algebras on the category of sets and relations, for instance, capture the first-order syntax structure of unscoped terms [59], whereas a similar treatment can be given for well-scoped terms by moving to categories of presheaves. Substitution, in both these cases, is modelled by the multiplication of the free monad induced by the initial algebra. The situation becomes more fragmented for second-order syntax (variable binding), as well as for more articulated syntax structures, where various, equally valid, structures concur as models of capture-avoiding substitution – notable examples include Σ-*monoids* [49, 48], *nominal sets* [52, 115], and monads and modules [69, 70].

Despite these (and other) differences, we observe abstract term structures are modelled by a structured – usually universal – object $\mathcal{S}$ in a universe category $\mathcal{E}$. Under mild conditions [50, 76], the universe category induces a category of relations $\mathcal{A}_\mathcal{E}$ (an allegory, in the truly category-theoretic sense), so that we can identify term relations on $\mathcal{S}$ with relations in $\mathcal{A}_\mathcal{E}(\mathcal{S}, \mathcal{S})$. When $\mathcal{E}$ is a topos (but one can even take a weaker structure), this set carries a RA structure, which provides a calculus (or logic) of "pure" relations on $\mathcal{S}$. Notice that

the RA structure is determined by $\mathcal{E}$ only, regardless of the structure of $\mathcal{S}$. However, one can extend the RA structure with "domain-specific" operations by noticing that the very (term) structure of $\mathcal{S}$ canonically gives rise to a small collection of operations on $\mathcal{A}_{\mathcal{E}}(\mathcal{S},\mathcal{S})$ that internalise $\mathcal{S}$-structural definitions of relations, and whose algebraic laws capture fundamental principles of structural reasoning on $\mathcal{S}$. Crucially, those operations are precisely abstract counterparts of those introduced in Section 1.3.

Overall, we obtain an extended calculus of relations on $\mathcal{S}$ whose operations and laws account for both logical and structural reasoning and definition principles.[8] The key observation behind [53] is that not only the just outlined construction can be given *mutatis mutandis* on many different abstract term structures, but that in *all* these cases we obtain essentially the *same* calculus of relations, independently of the underlying term structure. *Term Relation Algebras* axiomatise this invariant. That is, they axiomatise the common algebraic structure shared by the various hom-set $\mathcal{A}_{\mathcal{E}}(\mathcal{S},\mathcal{S})$ abstracting away from the term structure $\mathcal{S}$ and the universe category $\mathcal{E}$.

▶ **Definition 6.** *A TRA is a locally-complete allegory (a RA) $\mathcal{A}$ extended with a cocartesian relator triple $(\widehat{\cdot}, \Delta_\eta, \widetilde{\cdot})$ and a colax (bi)relator $\cdot_1[\cdot_2]$ such that:*

1. *The semirelator $\widetilde{\cdot}$ is ω-cocontinuous.*
2. *For any $a \in \mathcal{A}$, the equation $x = \widehat{x}\,;a$ has a uniform unique solution, denoted by $a^{\mathbf{H}}$, in such a way that $\Delta^{\mathbf{H}} = \Delta$.*
3. *$(\mathcal{A}, \Delta_\eta, \cdot_1[\cdot_2])$ is a closed monoid.*
4. *$\cdot[b]$ distributes over $\widetilde{\cdot}$. That is: $\widetilde{a}[b] \leq \widetilde{a[b]}$.*

The first two conditions in Definition 6 capture the initial algebra structure of syntax: morally, a TRA is an algebra of relations over an initial $V + \Sigma(-)$-algebra $\mathcal{S}V$, with V acting as the object of variables and Σ as the signature functor. The coreflexive Δ_η relates those terms in $\mathcal{S}V$ that come from the variable object V through the mono $\eta : V \to \mathcal{S}V$. Similarly, a relation $\widetilde{a}$ relates terms in $\mathcal{S}V$ that are related (through a relational extension of Σ) in $\Sigma(\mathcal{S}V)$ along the mono $\sigma : \Sigma(\mathcal{S}V) \to \mathcal{S}V$. The cocartesian relator triple structure then captures the coproduct structure $\mathcal{S}V \cong V + \Sigma(\mathcal{S}V)$ purely in terms of the operations Δ_η and $\widetilde{\cdot}$. Similarly, ω-cocontinuity of $\widetilde{\cdot}$ reflects ω-cocontinuity of Σ [8].

The second condition in Definition 6 expresses initiality of $\mathcal{S}V$. Indeed, regarding a TRA $\mathcal{A}$ as a one-object (categorically-intended) allegory, we see that we are requiring that the unique object $\bullet$ of the allegory (morally an abstraction of $\mathcal{S}V$) is the initial $\widehat{\cdot}$-algebra, and that its algebra structure is given by $\Delta : \bullet \to \bullet$. In that sense, $a^{\mathbf{H}}$ precisely is the unique extension of a given by initiality. By Knaster-Tarski theorem, that unique solution is also the *least* solution to $\widehat{x}\,;a \leq x$ (as well as the greatest solution to $a \leq \widehat{x}\,;a$), from which the following induction principle follows: $\widehat{x}\,;a \leq x \implies a^{\mathbf{H}} \leq x$. As we shall see, initiality gives definitions of relations by structural recursion which, when declined syntactically, corresponds to the well-known *Howe's construction* [73]. Notice also that from $\Delta = \Delta^{\mathbf{H}}$ it follows that $\Delta = \widehat{\Delta}$ as well as the *structural induction* principle [89]: $\widehat{x} \leq x \implies \Delta \leq x$.

The third condition in Definition 6 expresses the monoidal structure of substitution. The colax relator $\cdot_1[\cdot_2]$ captures relation substitution, with the monoidal structure accounting for the usual monoid properties of substitution. The closed structure amounts to an adjunction

[8] Notice that this construction is not specific to syntax structures, and it can be potentially used to internalise (non-algebraic) structure on objects as (algebraic) structure on its predicates/relations. Indeed, various algebras of logic can be seen as arising in this way. Modal algebras [20], for instance, arise as algebras of propositions on relational domains (Kripke frames) by internalising the relational structure as operations on predicates (modalities). Fork algebras [51] can be seen similarly, this time internaising the product structure of a base category as a fork operation on relations over that base.

$\cdot\,[b] \dashv b \mathbin{\gg} \cdot$ which, by the posetal Adjoint Functor Theorem, gives join-preservation of $\cdot\,[b]$, which in turn captures the parametric action of substitution. Finally, the fourth requirement states that substitution distributes over syntax structure. Notice that these conditions indeed distill the "operative" essence of a substitution structure: all different proposals of a substitution structure, to qualify as such, must indeed give the just described structure on term relations (and indeed, in a dual perspective, many of the technical subtleties behind the definitions of these structure can be understood as arising precisely to induce the desired structure on term relations).

▶ Remark 7. Notice that even if TRAs axiomatise the invariant structure of relations over different abstract term structures, they give a *synthetic* framework that can be directly used on concrete term structures, without going through their interpretation as abstract objects. This has various practical advantages, as one often deals with concrete term structures that are difficult to model as initial algebras and the like, but that can nonetheless be endowed with a TRA structure. Another advtange of the synthetic approach is that it naturally underpins the study of potentially interesting fragments of TRAs without caring too much about the underlying term structures. For example, we may restrict the underlying RA structure of a TRA to a, *e.g.*, Kleene algebra so as to study regular programs that operate on syntactic structures (*e.g.* interpreters and evaluators).

3.1 Examples

Before studying the expressiveness of TRAs, we introduce some familiar examples. Through the rest of this work, we refer to Δ_η, $\widetilde{\cdot}$, $\widehat{\cdot}$, $\cdot^{\mathbf{H}}$, and $\cdot_1[\cdot_2]$ as the *variable coreflexive* [53], *strict compatible refinement* [93, 53], *compatible refinement* [62, 89], *Howe extension* [73], and *relation substitution* [89], respectively, for reasons that will become clear soon.

3.1.1 First-Order Syntax

Let us fix a first-order signature Σ (*i.e.* a collection of operation symbols $\mathbf{op}$ together with an arity function $|\mathbf{op}| \in \mathbb{N}$) and a disjoint collection V of variables. We refer to elements in $\wp_{\mathrm{fin}}(V)$ as *contexts* and to maps $\rho : \Gamma \to \Delta$ between contexts Γ, Δ as *renaming* of Γ into Δ.

▶ **Definition 8.** *The dependent set of* terms in context $\mathcal{T}_\Sigma$ *is defined inductively, via the following rules, in which we write* $\Gamma \vdash \mathtt{t}$ *in place of* $\mathtt{t} \in \mathcal{T}_\Sigma(\Gamma)$*. The collection* $\mathcal{T}_\Sigma(V)$ *of all terms is defined as* $\bigcup_\Gamma \mathcal{T}_\Sigma(\Gamma)$*.*

$$\frac{\mathtt{x} \in \Gamma}{\Gamma \vdash \mathbf{var}\ \mathtt{x}} \qquad \frac{\Gamma \vdash \mathtt{t}_1 \quad \cdots \quad \Gamma \vdash \mathtt{t}_{|\mathbf{op}|} \quad \mathbf{op} \in \Sigma}{\Gamma \vdash \mathbf{op}(\mathtt{t}_1, \ldots, \mathtt{t}_{|\mathbf{op}|})}$$

Notice that the collection variables itself induces a dependent set $V(\Gamma) \triangleq \Gamma$, and that any context renaming $\rho : \Gamma \to \Delta$ extends to a term renaming $-[\rho] : \mathcal{T}_\Sigma(\Gamma) \to \mathcal{T}_\Sigma(\Delta)$. Finally, rule induction [6] provides the familiar principles of structural induction and recursion on $\mathcal{T}_\Sigma(V)$.

A *substitution* is a map $\sigma : V \to \mathcal{T}_\Sigma(V)$ with finite support, meaning that there are only finitely many variables $\mathtt{x}$ such that $\sigma(\mathtt{x}) \neq \mathbf{var}\ \mathtt{x}$). Using either rule induction or structural recursion, one can extend the action of substitution to terms in the usual way (see, *e.g.*, [116]), thus obtaining a map $-[\sigma] : \mathcal{T}_\Sigma(V) \to \mathcal{T}_\Sigma(V)$. Writing id for the identity substitution $id(\mathtt{x}) \triangleq \mathbf{var}\ \mathtt{x}$ and $(\sigma \cdot \tau)(\mathtt{x}) \triangleq \sigma(\mathtt{x})[\tau]$ for the composition of substitutions σ, τ, we obtain the monoid equational theory of substitution: $(\mathbf{var}\ \mathtt{x})[\sigma] = \sigma(\mathtt{x})$, $\mathtt{t}[id] = \mathtt{t}$, and $\mathtt{t}[\sigma])[\tau] = \mathtt{t}[\sigma \cdot \tau]$. Furthermore, we see that substitution is *syntax-preserving*: $\mathbf{op}(\mathtt{t}_1, \ldots, \mathtt{t}_{|\mathbf{op}|})[\sigma] = \mathbf{op}(\mathtt{t}_1[\sigma], \ldots, \mathtt{t}_{|\mathbf{op}|}[\sigma])$.

By Example 3, we know that we have a relation algebra on (first-order) terms in context. Notice how invariance under renaming forces relations on terms to be *term relations*, *i.e.* relations on the abstract syntax structure of terms (and thus invariant under irrelevant syntactic details). To obtain a TRA, we first observe that the very definition of terms gives a definition of the variable coreflexive and strict compatible refinement thus (we write $\Gamma \vdash \mathtt{t}\ \phi\ \mathtt{s}$ in place of $(\mathtt{t},\mathtt{s}) \in \phi(\Gamma)$):

$$\frac{\mathtt{x} \in \Gamma}{\Gamma \vdash \mathbf{var}\ \mathtt{x}\ \Delta_\eta\ \mathbf{var}\ \mathtt{x}} \qquad \frac{\Gamma \vdash \mathtt{t}_1\ \phi\ \mathtt{s}_1 \quad \cdots \quad \Gamma \vdash \mathtt{t}_{|\mathbf{op}|}\ \phi\ \mathtt{s}_{|\mathbf{op}|}}{\Gamma \vdash \mathbf{op}(\mathtt{t}_1,\dots,\mathtt{t}_{|\mathbf{op}|})\ \widetilde{\phi}\ \mathbf{op}(\mathtt{s}_1,\dots,\mathtt{s}_{|\mathbf{op}|})}$$

Notice that these definitions are *not* inductive, but they rather describe a single step in the inductive construction of terms, the latter relationally corresponding to Δ. This precisely makes Δ the least fixed point of $\widehat{\cdot}$. The law $\widehat{\phi} \leq \phi \implies \Delta \leq \phi$ thus captures structural induction [89]. In fact, taking ϕ as a coreflexive, the above quasi-equation states that to prove that any term has the property ϕ (*i.e.* $\Delta \leq \phi$), it is enough to show that any variable satisfies ϕ (*i.e.* $\Delta_\eta \leq \phi$) and that the collection of terms satisfying ϕ is closed under term constructors (*i.e.* $\widetilde{\phi} \leq \phi$).

Let us now move to the Howe extension operation. By instantiating its least fixed point characterisation, we obtain the the following inductive definition:

$$\frac{\Gamma \vdash \mathbf{var}\ \mathtt{x}\ \phi\ \mathtt{t}}{\Gamma \vdash \mathbf{var}\ \mathtt{x}\ \phi^{\mathbf{H}}\ \mathtt{t}} \qquad \frac{\Gamma \vdash \mathtt{t}_1\ \phi^{\mathbf{H}}\ \mathtt{s}_1 \quad \cdots \quad \Gamma \vdash \mathtt{t}_{|\mathbf{op}|}\ \phi^{\mathbf{H}}\ \mathtt{s}_{|\mathbf{op}|} \quad \Gamma \vdash \mathbf{op}(\mathtt{s}_1,\dots,\mathtt{s}_{|\mathbf{op}|})\ \phi\ \mathtt{s}}{\Gamma \vdash \mathbf{op}(\mathtt{t}_1,\dots,\mathtt{t}_{|\mathbf{op}|})\ \phi^{\mathbf{H}}\ \mathtt{t}}$$

Readers familiar with *Howe's method* [73], will probably recognise that $\phi^{\mathbf{H}}$ is precisely the *Howe extension* [73, 114, 62] of the relation ϕ. Readers familiar with rewriting, will probably see that $\phi^{\mathbf{H}}$ follows the same pattern of *parallel reduction* [141] in the style of Tait and Martin-Löf [15]. Indeed, these constructions are instances of the same general notion, which is, as previously seen, a form of relational initiality (see also [53]).

Finally, the action of substitution naturally extends from terms to relations, giving rise to the relation substitution operation:

$$\Delta \vdash \mathtt{t}[\tau]\ \phi[\psi]\ \mathtt{s}[\sigma] \iff \exists \Gamma.\ \Delta \vdash \mathtt{t}\ \phi\ \mathtt{s}\ \&\ \forall \mathtt{x} \in \Gamma.\ \Delta \vdash \tau(\mathtt{x})\ \psi\ \sigma(\mathtt{x})$$

With this definition, the monoid laws of substitution determine an actual monoid structure $(\mathcal{A}, \Delta_\eta, \cdot_1[\cdot_2])$ on relations. Syntax-preservation of substitution then gives the law $\widetilde{a}[b] \leq \widetilde{a[b]}$, whereas $\Gamma \vdash \mathtt{t}\ (\phi \mathbin{»} \psi)\ \mathtt{s}$ is defined by

$$\forall \tau, \sigma, \Delta.[(\forall \mathtt{x} \in \Gamma.\ \Delta \vdash \tau(\mathtt{x})\ \phi\ \sigma(\mathtt{x})) \implies \Delta \vdash \mathtt{t}[\tau]\ \psi\ \mathtt{s}[\sigma]]$$

▶ Remark 9. The reader may have recognised the pattern behind logical relations. Indeed, logical relations seem to be closely related to a solution to the equation $x = x \mathbin{»} x$. Notice, however, that, since $»$ is antitone in the first argument, the existence of such a solution is not ensured by standard fixed point arguments. We leave this for future research.

3.1.2 Second-Order Syntax

Second-order syntax [49, 7] enriches first-order syntax with binding structure. Loosely, this means that term constructs can bind variables, and that terms are identified modulo renaming of bound variables. In this example, we model second-order syntax through *binding signatures* [7], an extension of first-order signatures where the arity of an operation symbol is an element of $\mathbb{N}^*$. The intended meaning of an arity $|\mathbf{op}| = (k_1,\dots,k_n)$ (notation $\mathbf{op} : |\mathbf{op}|$)

is that operation **op** takes n arguments and binds k_i variables in the i-th argument. Because we shall deal with renaming of bound variables, it is necessary to have an unlimited supply of variables. Accordingly, from now on, we tacitly assume that V is countably infinite.

▶ **Definition 10.** *The collection* $\mathcal{T}_\Sigma(V) \triangleq \bigcup_\Gamma \mathcal{T}_\Sigma(\Gamma)$ *of* raw terms *in context is defined similarly to the first-order case, by the following rules, where the notation* $\Gamma, \mathtt{x}$ *stands for* $\Gamma \cup \{\mathtt{x}\}$*, provided that* $\mathtt{x} \notin \Gamma$*, and each* $\vec{\mathtt{x}}_i$ *has the right length* k_i*:*

$$\frac{\mathtt{x} \in \Gamma}{\Gamma \vdash \mathbf{var}\ \mathtt{x}} \qquad \frac{\Gamma, \vec{\mathtt{x}}_1 \vdash \mathtt{t}_1 \quad \cdots \quad \Gamma, \vec{\mathtt{x}}_n \vdash \mathtt{t}_n \quad |\mathbf{op}| = (k_1, \ldots, k_n)}{\Gamma \vdash \mathbf{op}(\vec{\mathtt{x}}_1.\mathtt{t}_1, \ldots, \vec{\mathtt{x}}_n.\mathtt{t}_n)}$$

As usual, in a raw term of the form $\mathbf{op}(\vec{\mathtt{x}}_1.\mathtt{t}_1, \ldots, \vec{\mathtt{x}}_n.\mathtt{t}_n)$, we say that variables $\vec{\mathtt{x}}_i$ are *binding* in **op** and *bound* (if present) in $\mathtt{t}_i$; variables not bound in a term are said to be *free*, whereas a variable not appearing neither in Γ nor in $\mathtt{t}$ is said to be *fresh* for the judgment $\Gamma \vdash \mathtt{t}$

▶ **Example 11.** *(i)* As a running example, we consider the signature of the λ-calculus, made of two operations – `lam` and `app` – of arity (1) and $(0, 0)$, respectively. We will employ more familiar notation, writing, *e.g.*, $\lambda\mathtt{x}.\mathtt{t}$ and $\mathtt{ts}$ in place of $\mathtt{lam}(\mathtt{x}.\mathtt{t})$ and $\mathtt{app}(\mathtt{t}, \mathtt{s})$, respectively. *(ii)* It is easy to extend this signature to more sophisticated languages, such as calculi with dependent product types and natural numbers [99, 100]. For instance, we may consider the binding signature containing operators : $\mathtt{Pi} : (0, 1)$ (in more familiar notation, $\Pi(\mathtt{x} : \mathtt{t}).\mathtt{s}$, with the first term standing for a type); $\mathtt{lam} : (0, 1)$ ($\lambda(\mathtt{x} : \mathtt{t}).\mathtt{s}$, with $\mathtt{t}$ acting as a type) $\mathtt{eq} : (0, 0, 0)$; $\mathtt{refl} : (0)$; $\mathtt{eqrec} : (0, 0)$; $\mathtt{nat} : ()$; $\mathtt{z} :: ()$; $\mathtt{succ} : (0)$; $\mathtt{natrec} : (0, 0, 2)$.

Any binding signature induces a notion of *variable renaming* on raw terms and a corresponding *equivalence under variable-renaming* – so-called α-equivalence. Both these notions are defined by rule induction (the reader can find all details in the lecture notes by Pitts [116]), and give a context-dependent equivalence relation $\sim_\alpha$ on raw terms.

▶ **Definition 12.** *We define the dependent set of* terms *in context* Γ *as the* quotient $\mathcal{T}^\alpha_\Sigma(\Gamma) \triangleq \mathcal{T}_\Sigma(\Gamma)/{\sim_\alpha}$*, and the collection of terms as* $\mathcal{T}^\alpha_\Sigma(V) \triangleq \bigcup_\Gamma \mathcal{T}^\alpha_\Sigma(\Gamma)$*. As before, dependent relations on dependent sets* $\mathcal{T}^\alpha_\Sigma$ *form a relation algebra.*

As a notational convention, we do not distinguish between raw terms $\mathtt{t}$ and their $\sim_\alpha$-equivalence classes – *terms* – $[\mathtt{t}]_{\sim_\alpha}$. Most definitions on terms are actually given on raw terms, and then tacitly extended to terms by (equally tacitly) observing that these definitions are invariant under $\sim_\alpha$. For example, we define $\mathcal{T}^\alpha_\Sigma(V)$ strict compatible refinement first on raw terms, and then observe that the resulting notion is invariant under $\sim_\alpha$. The definition of Δ_η is given similarly, and it is essentially identical to the one given on first-order terms:

$$\frac{\Gamma, \vec{\mathtt{x}}_1 \vdash \mathtt{t}_1\ \phi\ \mathtt{s}_1 \quad \cdots \quad \Gamma, \vec{\mathtt{x}}_n \vdash \mathtt{t}_n\ \phi\ \mathtt{s}_n \quad |\mathbf{op}| = (k_1, \ldots, k_n)}{\Gamma \vdash \mathbf{op}(\vec{\mathtt{x}}_1.\mathtt{t}_1, \ldots, \vec{\mathtt{x}}_n.\mathtt{t}_n)\ \widetilde{\phi}\ \mathbf{op}(\vec{\mathtt{x}}_1.\mathtt{s}_1, \ldots, \vec{\mathtt{x}}_n.\mathtt{s}_n)}$$

Moving to substitution, the shift from first-order to second-order syntax requires us to deal with capture-avoidance. A substitution is a map $\sigma : V \to \mathcal{T}^\alpha_\Sigma(V)$ with finite support, and we would like to extend it to a map $-[\sigma] : \mathcal{T}^\alpha_\Sigma(V) \to \mathcal{T}^\alpha_\Sigma(V)$ satisfying monoid laws and syntax-preservation (that is one of the reasons why capture-avoidance is needed). Since maps $-[\sigma]$ correspond to maps $-\{\sigma\} : \mathcal{T}_\Sigma(V) \to \mathcal{T}^\alpha_\Sigma(V)$ that are invariant under $\sim_\alpha$ (*i.e.* $\mathtt{t} \sim_\alpha \mathtt{s} \implies \mathtt{t}\{\sigma\} = \mathtt{s}\{\sigma\}$), it is enough to extend σ to $-\{\sigma\}$.

It is well-known that naive attempts to that by structural recursion on $\mathcal{T}_\Sigma(V)$ fail, so that one has to rest on different techniques. Nominal sets [52] (see below) provide an elegant solution to the problem, offering structural definitions of substitution by α-structural

recursion [118]. Working with binding signature, we can define substitution as an inductive relation on raw terms, then showing, by rule induction, that the resulting relation gives a function on $\sim_\alpha$-equivalence classes of raw terms [116]. We could even work with $\sim_\alpha$-equivalence classes of abstract syntax trees and define substitution by natural number recursion, as customary in classic textbooks on the λ-calculus [15, 87, 68], or following the clever, *first-order*, approach by Stoughton [138]. In all these cases, we obtain a definition of capture-avoiding substitution upon which we define relation substitution as in the previous example.

Nominal Syntax

Nominal sets [115] provide a general approach to second-order syntax based on the notions of permutation and finite support. In a nutshell, given a group G, a *nominal set* consists of a set X endowed with a G-action $\cdot : G \times X \to X$ on it, such that all elements of X are finitely supported. When it comes to syntax, one considers the group of *permutations* $\mathsf{Sym}(V)$ of the set of variables V. This immediately makes V a nominal set, with $\mathsf{Sym}(V)$-action given by function application.

▶ **Definition 13.** *Given a binding signature Σ and a set of variables V, we define the $\mathsf{Sym}(V)$-action on $\mathcal{T}_\Sigma(V)$ by: $\pi \cdot \mathbf{op}(\vec{\mathtt{x}}_1.\mathtt{t}, \dots, \vec{\mathtt{x}}_n.\mathtt{t}_n) \triangleq \mathbf{op}(\overrightarrow{\pi(\mathtt{x})}_1.(\pi \cdot \mathtt{t}), \dots, \overrightarrow{\pi(\mathtt{x})}_n.(\pi \cdot \mathtt{t}_n))$ and $\pi \cdot \mathbf{var}\ \mathtt{x} \triangleq \pi(\mathtt{x})$. This way, we obtain a* nominal set *of raw terms.*

The natural notion of a term relation on nominal raw terms is the one of an *equivariant finitely supported relation* [115, 117], which gives the desired TRA structure on $\mathcal{T}_\Sigma(V)$ by the so-called Finite Support Principle (see [115] and Theorem 3.5 in [118]). More interestingly, nominal sets give a natural notion of α-equivalence $\sim_\alpha$, and the quotient set $\mathcal{T}^\alpha_\Sigma(V)$ is a nominal set itself, with action $\pi \cdot [\mathtt{t}]_\alpha \triangleq [\pi \cdot \mathtt{t}]_\alpha$. Equivariant finitely supported relations on $\mathcal{T}^\alpha_\Sigma(V)$, as expected, carry a complete TRA structure.

▶ Remark 14 (Further Examples). Due to space constraints, we have to omit further examples. Among those, we mention multisorted refinements of first-order and second-order terms. Those can be used to model various typed structures, especially if taking sorts modelled by second-order terms themselves. Also monoidal terms can be modelled using multisorted terms by taking natural numbers as sorts and quotienting terms according to the usual equational laws of string diagrams [135].

3.2 Expressiveness

Having examples of TRAs, we now investigate how many "semantic" notions can be expressed within them. As a warm up, we notice that we can express basic, albeit useful, syntax-based notions.

▶ **Definition 15.** *We say that a relation a is: (i)* compatible *if $\widehat{a} \leq a$; (ii)* closed under term constructs *if $\widetilde{a} \leq a$; (iii) a* (pre)congruence *if it is compatible equivalence (preorder); (iv)* closed under (substitution) instances *(CUI) if $a[\Delta] \leq a$; (v)* substitutive *if $a[a] \leq a$. (vi) We say that a satisfies* Leibniz's Substitution Law *[64] if $\Delta[a] \leq a$.*

Notice that any compatible relation is reflexive. Moreover, for any a, the relation $a[\Delta]$ (the closure under substitution instances of a) is always CUI. Moreover, compatibility ($\widehat{a} \leq a$) implies Leibniz's law, but the converse is in general false.

3.2.1 Substitution and Cohesion

Many syntax-based systems build upon the distinction between *complete* – *open* – and *incomplete* – *closed* – terms [99], and operations of restriction to closed terms and extension to open ones. Mathematically, these distinctions and operations are usually left in the background and treated syntactically. The algebraic machinery of TRAs reveals, instead, a rich structure behind these notions.

Let us begin by uncovering the (relational) structure of complete terms. To fix ideas, let us consider first-order terms (the example holds *mutatis mutandis* for second-order syntax) $\mathcal{T}_\Sigma(V)$ $(= \bigcup_\Gamma \mathcal{T}_\Sigma(\Gamma))$. In this case, we say that a term $\mathtt{t} \in \mathcal{T}_\Sigma(\Gamma)$ is *closed* if $\mathtt{t} \in \mathcal{T}_\Sigma(\varnothing)$. Writing $\mathcal{A}$ and $\mathcal{A}_0$ for the relation algebras of relations over terms and closed terms, respectively, we see that these two algebras are related by the adjoint triple $\iota \dashv \flat \dashv \sharp$, where $\iota, \sharp : \mathcal{A}_0 \to \mathcal{A}$ and $\flat : \mathcal{A} \to \mathcal{A}_0$ are defined by $\flat(\phi) \triangleq \{(\mathtt{t}, \mathtt{s}) \mid \varnothing \vdash \mathtt{t}\ \phi\ \mathtt{s}\}$, $\iota(r)(\Gamma) \triangleq \{(\mathtt{t}, \mathtt{s}) \mid \mathtt{t}\ r\ \mathtt{s}\}$, and $\sharp(r)(\Gamma) \triangleq \{(\mathtt{t}, \mathtt{s}) \mid \mathtt{t}, \mathtt{s} \in \mathcal{T}_\Sigma(\emptyset) \implies \mathtt{t}\ r\ \mathtt{s}\}$. These adjunctions create an (axiomatic) *cohesion* [92], whereby the (co)monadic *modalities* $\Box \triangleq \iota \circ \flat$, $\blacklozenge \triangleq \sharp \circ \flat$ on $\mathcal{A}$ form an adjunction $\Box \dashv \blacklozenge$ themselves [80]. Making some calculations, we obtain the following explicit definition on relations: $\Gamma \vdash \mathtt{t}\ \Box\phi\ \mathtt{s}$ iff $\varnothing \vdash \mathtt{t}\ \phi\ \mathtt{s}$, and $\Gamma \vdash \mathtt{t}\ \blacklozenge\phi\ \mathtt{s}$ iff $(\varnothing \vdash \mathtt{t}, \mathtt{s} \implies \varnothing \vdash \mathtt{t}\ \phi\ \mathtt{s})$.

Since the modalities $\Box$ and $\blacklozenge$ are themselves obtain through the composition of adjoint maps, they satisfy (co)monadic laws. For instance, $\Box$ being a comonad, it is an $S4$-*modality* [20], meaning that it is monotone, and satisfies $\Box\phi \subseteq \phi$ and $\Box(\phi\,;\psi) = \Box\phi\,;\Box\psi$. It also distributes over relation converse and composition. Furthermore, we see that whether two terms are related by $\Box\phi$ boils down to the shape of the terms (they must be complete) and the relation ϕ itself (they must be ϕ-related). This is captured by the law $\Box\Delta\,;\phi\,;\Box\Delta \subseteq \Box\phi$. Finally, as expected, we have the laws $\Box\Delta_\eta = \bot$ (no variable is a complete term) and $(\Box\phi)[\psi] \subseteq \Box\phi$ (substitution has no effect on complete terms).

Closed Relations

While all the aforementioned laws have been obtained in the context of a specific example, their very structure suggests that the modalities $\Box$ and $\blacklozenge$ should be definable solely in terms of substitution. This is indeed that case, we see that defining $\Box a \triangleq a[\bot]$ and $\blacklozenge a \triangleq a \mathbin{»} \bot$ we indeed obtain all the aforementioned laws.

Using the modality $\Box$, we obtain an image of the algebra of complete relations within usual TRAs simply by considering *closed relations*, *viz.* relations a such that $a \leq \Box a$ (and thus $\Box a = a$). Closed relations play a crucial role in term evaluation, as the latter acts on complete terms only. Since evaluation and related forms of syntax dynamics are typically defined recursively, as least solutions $a^{\mathbf{F}}$ of parametric equations of the form $x = F(x, a)$, we would like to ensure that whenever a is closed, and F is of a suitable form, then $a^{\mathbf{F}}$ is closed too.

▶ **Lemma 16.** *Let F be a monotone function on a TRA $\mathcal{A}$. We say that F is* closed *if $\Box F(x) \leq F(\Box x)$. If F is closed then $\Box\mu F = \mu x.\Box F(x)$. In particular, $\Box a^{\mathbf{F}} = (\Box a)^{\mathbf{F}}$ and thus $a^{\mathbf{F}} = \Box a^{\mathbf{F}}$ if $a = \Box a$.*

Thinking about F as the extension of a term function to relations, the requirement that F is closed means that if two complete terms are related by $\Box F(a)$, then all the a-related subterms involved are complete themselves. Reading all of that as term evaluation, closedness means that evaluating a compound complete term requires to evaluate its complete sub-terms only (cf. with *weak* evaluation, where one does not evaluate under binders precisely because of that).

Finally, we can extend relations on closed terms to open ones through the *open extension* modality [89]. In the case of first-order terms, this is given by the relation $\Diamond\phi$ where $\Gamma \vdash \mathtt{t} \ \Diamond\phi \ \mathtt{s}$ holds iff $\forall\sigma.\ (\forall \mathtt{x} \in \Gamma.\ \sigma(\mathtt{x}) \in \mathcal{T}_\Sigma(\varnothing)) \implies \varnothing \vdash \mathtt{t}[\sigma] \ \phi \ \mathtt{s}[\sigma]$ Abstracting from first-order syntax, we see that the key point of open extension is to behave as the right adjoint $\Box(-[\Delta]) \dashv \Diamond$, which is actually expressible in the language of TRAs thus: $\Diamond a \triangleq \Delta \mathbin{\rightarrowtriangle} \blacklozenge a$. As we shall see, open extension will play an important role in the definition of bisimilarity.

4 Pointfree Metatheory, I: Reduction

It is finally time to put TRAs to work and develop some metatheory with (and within) them. We first show some examples of metatheory of *reduction systems*.

▶ **Definition 17.** *We say that a relation a is a* reduction *if $\Delta_\eta \,;\, a = \bot$. This condition [18] ensures nontriviality with respect to (substitution)* instances *of the rule, i.e.term relations obtained from a by instantiating the variables of a-related terms using the same substitution.*

▶ **Example 18.** Given a collection of first-order terms $\mathcal{T}_\Sigma(V)$, a *Term Rewriting System* (TRS) [18, 74] is given by a reduction ϕ in the TRA of relations over $\mathcal{T}_\Sigma(V)$. For example, for the signature of *combinatory logic* [33] $\Sigma \triangleq \{\mathtt{k} : 0, \mathtt{s} : 0, \mathtt{A} : 2\}$, the (closure under extended renaming of the) term relation $\mathtt{x}, \mathtt{y} \vdash \mathtt{A}(\mathtt{A}(\mathtt{k}, \mathtt{x}), \mathtt{y}) \mapsto \mathtt{x}$ is a reduction (and similarly for the corresponding reduction for $\mathtt{s}$). We extend the reduction action of $\mathtt{k}$ from variables to arbitrary terms by considering $\mapsto[\Delta]$, while taking $(\mapsto[\Delta] \vee \Delta)^{\mathbf{H}}$ gives parallel reduction. Sequential reduction can be modelled by considering so-called *derivatives* [110]

▶ **Example 19.** Let us consider the signature of the dependent λ-calculus. We see that β-reduction gives a relation defined by $\Gamma \vdash (\lambda(x : \mathtt{t}).\mathtt{s})\mathtt{u} \ \beta \ \mathtt{s}[\mathtt{u}/\mathtt{x}]$. Notice that $\Delta_\eta \,;\, \beta = \bot$ and $\beta[\Delta] \leq \beta$. Taking the Howe extension of its reflexive closure $(\beta \cup \Delta)^{\mathbf{H}}$ we obtain *parallel β-reduction* [141] $\Rrightarrow_\beta$, which we can inductively characterised thus:

$$\frac{}{\Gamma, \mathtt{x} \vdash \mathtt{x} \Rrightarrow_\beta \mathtt{x}} \qquad \frac{\Gamma, \mathtt{x} \vdash \mathtt{t} \Rrightarrow_\beta \mathtt{t}' \quad \Gamma \vdash \mathtt{s} \Rrightarrow_\beta \mathtt{s}'}{\Gamma \vdash (\lambda \mathtt{x} : \mathtt{u}.\mathtt{t})\mathtt{s} \Rrightarrow_\beta \mathtt{t}'[\mathtt{s}'/\mathtt{x}]} \qquad \frac{\Gamma, \mathtt{x} \vdash \mathtt{t}_i \Rrightarrow_\beta \mathtt{s}_i}{\Gamma \vdash \lambda \mathtt{x} : \mathtt{t}_1.\mathtt{t}_2 \Rrightarrow_\beta \lambda \mathtt{x} : \mathtt{s}_1.\mathtt{s}_2}$$

$$\frac{\Gamma \vdash \mathtt{t} \Rrightarrow_\beta \mathtt{t}' \quad \Gamma \vdash \mathtt{s} \Rrightarrow_\beta \mathtt{s}'}{\Gamma \vdash \mathtt{t}\mathtt{t}' \Rrightarrow_\beta \mathtt{s}\mathtt{s}'} \qquad \frac{\Gamma \vdash \mathtt{t} \Rrightarrow_\beta \mathtt{t}' \quad \Gamma, \mathtt{x} \vdash \mathtt{s} \Rrightarrow_\beta \mathtt{s}'}{\Gamma \vdash \Pi(\mathtt{x} : \mathtt{t}).\mathtt{s} \Rrightarrow_\beta \Pi(\mathtt{x} : \mathtt{t}').\mathtt{s}'}$$

Finally, taking $(\beta \vee \beta^\circ)^{\mathbf{H}^*}$ gives *β-convertibility* $=_\beta$.

Abstracting from the previous example, we define the *parallel reduction* over a as $a^{\Rrightarrow} \triangleq (\Delta \vee a[\Delta])^{\mathbf{H}}$ and convertibility as $a^{\approx} \triangleq (a \vee a^\circ)^{\mathbf{H}^*}$. Standard algebraic TRA calculations show that $a^{\approx} \triangleq (a^{\Rrightarrow} \vee a^{\Rrightarrow\circ})^*$.

A central problem in rewriting is the one of determining consistency of convertibility, which often reduces to prove the so-called *Church-Rosser property* [18].

▶ **Definition 20.** *We say that a relation a has the* Church-Rosser property *if $(a \vee a^\circ)^* = a^*;a^{*\circ}$, and that it has the* diamond property *if $a^\circ \,;\, a \leq a \,;\, a^\circ$. We say that a is* confluent *if a^* has the diamond property.*[9]

[9] Even if we will not deal with termination, we mention, for the pure sake of beauty, the particularly pleasant laws [44] expressing that a relation a is *inductive* – $a \to x \leq x \implies \top \leq x$ – and *well-founded* – $x \leq x \,;\, a \implies x \leq \bot$.

Easy algebraic calculations show that if a has the diamond property, then it is confluent; and that a relation has the Church-Rosser property iff it is confluent [139, 140]. In light of that, to prove that $a^{\Rightarrow}$ has the Church-Rosser it is sufficient to prove that $a^{\Rightarrow}$ has the diamond property. Using the language of TRAs, we can algebraically express an easy-to-check "semantic" notion of *orthogonality* and calculationally infer confluence from it.

▶ **Theorem 21** ([53]). *A reduction a is* orthogonal *if $a[\Delta]^\circ \,;\, a[\Delta] \leq \Delta$ and $a[\Delta]^\circ \,;\, \widetilde{a[\Delta]^{\Rightarrow}} \leq a^\circ[a^{\Rightarrow}]$. If a is orthogonal, then $a^{\Rightarrow}$ is diamond, and thus confluent.*

The first condition in orthogonality states that substitution instance of (ground) reduction is deterministic – a relation a is deterministic when $a^\circ \,;\, a \leq \Delta$. The second condition asserts that even reducing substerms of a redex does not destroy the redex structure and, the new nested redexes it creates can be uniformly reduced.

▶ **Example 22.** The relation β is orthogonal: $\Gamma \vdash \mathtt{t[s/x]}\ \beta^\circ\ (\lambda\mathtt{x.t})\mathtt{s}$ and $\Gamma \vdash (\lambda\mathtt{x.t})\mathtt{s} \Rrightarrow_\beta (\lambda\mathtt{x.t'})\mathtt{s'}$ entail $\Gamma \vdash \mathtt{t[s/x]} \Rrightarrow_\beta \mathtt{t'[s'/x]}$ and $\Gamma \vdash \mathtt{t'[s'/x]}\ \beta^\circ\ (\lambda\mathtt{x.t'})\mathtt{s'}$, since $\Rrightarrow_\beta$. This follows since, in general, $a^{\Rightarrow}$ is substitutive ($a^{\Rightarrow}[a^{\Rightarrow}] \leq a^{\Rightarrow}$). By Theorem 21, we conclude confluence.

While our notion of orthogonality is typically easy to check, it is not a genuine local condition, as it speaks of $a^{\Rightarrow}$. In the next section, we will give a finer, completely local conditions and show that such condition implies Theorem 21.

5 Pointfree Metatheory: Operational Semantics

Operational semantics involves diverse "syntactic" notions: program, values, context, lazy and eager evaluation, reduction strategies, and many others, that are typically left, implicitly or explicitly, to term representations. Much of these notions can be uniformly understood following (the informal account by) Martin-Löf [99] (see also the nice explanation given in [4]). Accordingly, term constructs are divided into *constructors* and *destructors*: the former construct pieces of structured data; the latter consume them. Arguments of a destructor are furthermore divided into *major* – or *eager* – *minor* – or *lazy*. Terms whose outermost operator is a constructor (resp. destructor) are *introduction forms* (resp. *elimination forms*). Computation is triggered when there is an elimination form with introduction forms as major arguments: this is the so-called *Gentzen's Inversion Principle* [57].[10]

Closed terms – *programs* – are evaluated until a *canonical form* – *value* – is reached. In a lazy discipline, canonical forms are identified with introduction forms (for example, we do not reduce under lambdas); in eager disciplines, other notions of canonical form may be considered (*e.g.* introduction forms whose arguments are canonical themselves). Languages with "eager canonical forms" can be turned into lazy ones by adopting a *reduced syntax* [89], or *fine-grain* syntax [94]. For simplicity, we will also identify canonical forms with introduction forms.

Finally, evaluation proceeds thus: if a closed term is an introduction form (a value), then just return it. Otherwise the term is an elimination form; evaluate its major sub-terms – which has to be closed, otherwise the evaluation fails – until an introduction form is reached. At this point, by Inversion Principle, if the introduction forms correspond to the elimination form to be evaluated, a true computation step is performed and then evaluation is resumed.

[10] Notice that following this view, the purpose of a type discipline is precisely to ensure that elimination forms interact only with their corresponding introduction forms.

5.1 Algebraization of Operational Semantics

All these notions can be easily expressed, algebraically, in the language of TRA. In fact, at the heart of the just outlined pre-theory of operational semantics lie certain distinctions between term constructs, that can be naturally recast in terms of cocartesian decomposition triples.

▶ **Definition 23.** *Given a TRA, an* operational decomposition *is given by a decomposition of strict compatible refinement as a cocartesian relator triple* $(\widetilde{\cdot}, \overline{\cdot}, \langle \cdot_1, \cdot_2 \rangle)$ *such that* $\cdot[b]$ *distributes over both* $\overline{\cdot}$ *and* $\langle \cdot_1, \cdot_2 \rangle$ *(e.g.* $\langle a, b\rangle[c] \leq \langle a[c], b[c]\rangle$*, and* $\Box$ *distributes over* $\langle \cdot, b\rangle$ *(i.e.* $\Box\langle a, b\rangle \leq \langle \Box a, b\rangle$*).*

The intuitive meaning of an operational decomposition should be clear. Both $\overline{a}$ and $\langle a, b\rangle$ refine strict compatible refinement by restricting outermost operators to constructors, in the case of $\overline{a}$, and destructors, in the case of $\langle a, b\rangle$. Furthermore, $\langle a, b\rangle$ requires that *major* subterms are pairwise a-related and all other (*minors*) are pairwise b-related. Thinking of TRAs as algebras of relations over initial $(V + \Sigma(-))$-algebras, an operational decomposition signals that the signature functor can be factorized as $\Sigma_{\mathsf{intro}}(\cdot_1) + \Sigma_{\mathsf{elim}}(\cdot_1, \cdot_2)$.

Using these operations, we define the coreflexives of introduction and elimination forms as $\overline{\Delta}$ and $\langle \Delta, \Delta\rangle$, and the coreflexive of *values* as complete introduction forms: $\Delta_\kappa \triangleq \Box\overline{\Delta}$. Notice that the cocartesian relatotr triple $(\widetilde{\cdot}, \overline{\cdot}, \langle \cdot_1, \cdot_2 \rangle)$ entails $\overline{a}\,;\langle b, c\rangle \leq \bot$, meaning that constructors and destructors are disjoint. Furthermore, if we identify canonical forms with introduction forms, as we do, then we see that terms with an outermost constructors are always values (*i.e.* evaluation must be lazy, or weak). Although this is always the case in call-by-name semantics, it is not so in call-by-value one: for example, a pair $\mathtt{pair}(\mathtt{t}, \mathtt{s})$ is a value if both $\mathtt{t}$ and $\mathtt{s}$ are. We can easily overcome this issue by presenting syntax in *reduced* or *fine-grain* form [94, 89]. Finally, distributivity of $\Box$ over $\langle \cdot, b\rangle$ ensures that major arguments of a closed term are closed themselves.

5.1.1 Inversion Principle

Let us now algebrize Gentzen's Inversion Principle. Since we will extensively relate elimination forms through their major arguments, we introduce the abbreviation $\langle a\rangle \triangleq \langle a, \Delta\rangle$. Intuitively, two elimination forms are related by $\langle a\rangle$ if they have the same outermost destructor and minor arguments, and pairwise a-related major arguments.

▶ **Definition 24.** *We say that a relation* a *satisfies* Gentzen's Inversion Principle *(GIP) if* $a \leq \langle\overline{\Delta}\rangle\,;a$ *(and thus* $a = \langle\overline{\Delta}\rangle\,;a$*).*

Notice that all reductions met so far, as well as many others, satisfy (GIP). A stronger version of this principle – dubbed *Strong Gentzen's Inversion Principle* (SGIP) – can be obtained by introducing a *domain* operation dom giving, for each relation a, the coreflexive representing the domain of a [50, 133, 44]. Assuming dom, we can express (SGIP) as $\mathsf{dom}(a) = \langle\overline{\Delta}\rangle$. This equation, which states that any elimination form whose major arguments are introduction forms is reducible, holds on *e.g.*, β, although it already fails when moving to simple extensions, such as PCF, due to stuck terms (*e.g.* $\mathsf{app}(\mathtt{z}, \mathtt{z})$). Typing disciplines, however, typically ensure the validity of SGIP.

5.1.2 Examples

A large family of examples can be obtained by splitting (first-order, binding, nominal, *etc.*) signatures Σ into two (first-order, binding, nominal, *etc.*) *disjoint* signatures Σ_{intro} and Σ_{elim} containing constructors and destructors, respectively. Notice that, as just discussed, we require that constructors and destructors are disjoint.

Next, we refine the notion of arity of a destructor so as to specify which operands are *major* and *minor* arguments. This can be done by labelling specific notions of arity. For example, in a binding signature with $|\mathbf{op}| = (k_1, \ldots, k_n)$, we label those k_js corresponding to major positions, provided that these do *not* bind variables. Finally, we use a "correspondence relation" $\sim \subseteq \Sigma_{\mathsf{elim}} \times \Sigma_{\mathsf{intro}}$ relating destructors with their corresponding constructors. As already mentioned, type systems can be understood as a way to constrain term formation according to $\sim$.

▶ **Example 25.** We refine the signature of the λ-calculus with dependent types and natural numbers thus: $\Sigma_{\mathsf{intro}} \triangleq \{\mathtt{lam} : (1), \mathtt{Pi} : (0,1), \mathtt{refl} : (0), \mathtt{nat} : (), \mathtt{z} :: (), \mathtt{succ} : (0)\}$, $\Sigma_{\mathsf{elim}} \triangleq \mathtt{app} : (\underline{0}, 0), \mathtt{eqrec} : (\underline{0}, 0), \mathtt{natrec} : (\underline{0}, 0, 2)\}$, where we underline major positions. The relation $\sim$ is defined by: $\mathtt{app} \sim \mathtt{lam}$, $\mathtt{eqrec} \sim \mathtt{refl}$, $\mathtt{natrec} \sim \mathtt{z}$, and $\mathtt{natrec} \sim \mathtt{succ}$. Reduction is then determined by the scheme: $(\mathtt{app}(\mathtt{lam}(-), -), \mathtt{eqrec}(\mathtt{refl}(-), -), \mathtt{natrec}(\mathtt{z}, -, -)$, and $\mathtt{natrec}(\mathtt{succ}(-), -, -)$. Other interactions between destructors and constructors following this pattern give so-called stuck terms.

Notice that the choice of major premises in a destructors determines the evaluation order of a term. In the example above, for instance, the destructor $\mathtt{app} : (\underline{0}, 0)$ stipulates that the evaluation of a term $\mathtt{app}(\mathtt{t}, \mathtt{s})$ proceeds by evaluating $\mathtt{t}$ until a corresponding introduction form is reached, *viz.* $\mathtt{lam}(\mathtt{x}.\mathtt{t}')$, and then performing β-reduction according Gentzen's principle – hence reducing $\mathtt{app}(\mathtt{lam}(\mathtt{x}.\mathtt{t}'), \mathtt{s})$ to $\mathtt{t}'[\mathtt{s}/\mathtt{x}]$ – *without* evaluating $\mathtt{s}$. In this sense, our syntax underpins a *call-by-name* evaluation strategy.

▶ **Example 26.** The *call-by-value* λ-calculus is given by the signatures $\Sigma_{\mathsf{intro}} \triangleq \{\mathtt{lam} : (1)\}$ and $\Sigma_{\mathsf{elim}} \triangleq \mathtt{app} : (\underline{0}, \underline{0})\}$. Accordingly, the evaluation of $\mathtt{app}(\mathtt{t}, \mathtt{s})$ requires to evaluate *both* $\mathtt{t}$ and $\mathtt{s}$.

Constructing examples of operational decomposition is now straightforward. On binding signature, for instance, we define $\overline{\phi}$ and $\langle \phi, \psi \rangle$ thus, where, for a destructor $\mathbf{op} : (k_1, \ldots, k_n)$, we have $m_{\mathbf{op}}(i, \phi, \psi) \triangleq \phi$ if k_i major, and $m_{\mathbf{op}}(i, \phi, \psi) \triangleq \psi$, otherwise.

$$\frac{\Gamma, \vec{\mathtt{x}}_1 \vdash \mathtt{t}_1 \ \phi \ \mathtt{s}_1 \quad \cdots \quad \Gamma, \vec{\mathtt{x}}_n \vdash \mathtt{t}_n \ \phi \ \mathtt{s}_n \quad \mathbf{op} : (k_1, \ldots, k_n) \in \Sigma_{\mathsf{intro}}}{\Gamma \vdash \mathbf{op}(\vec{\mathtt{x}}_1.\mathtt{t}_1, \ldots, \vec{\mathtt{x}}_n.\mathtt{t}_n) \ \overline{\phi} \ \mathbf{op}(\vec{\mathtt{x}}_1.\mathtt{s}_1, \ldots, \vec{\mathtt{x}}_n.\mathtt{s}_n)}$$

$$\frac{(\forall 1 \leq i \leq n) \ \Gamma, \vec{\mathtt{x}}_i \vdash \mathtt{t}_i \ m_{\mathbf{op}}(i, \phi, \psi) \ \mathtt{s}_i \quad \mathbf{op} : (k_1, \ldots, k_n) \in \Sigma_{\mathsf{elim}}}{\Gamma \vdash \mathbf{op}(\vec{\mathtt{x}}_1.\mathtt{t}_1, \ldots, \vec{\mathtt{x}}_n.\mathtt{t}_n) \ \langle \phi, \psi \rangle \ \mathbf{op}(\vec{\mathtt{x}}_1.\mathtt{s}_1, \ldots, \vec{\mathtt{x}}_n.\mathtt{s}_n)}$$

5.1.2.1 Structural Recursion, Revisited

The refinement of $\widetilde{\cdot}$ (and, consequently, of $\widehat{\cdot}$) into the operations $\overline{\cdot}$ and $\langle \cdot, \cdot \rangle$ opens the door to a number of refinements of the operation $\cdot^{\mathbf{H}}$, each corresponding to various forms of structural recursion on, *e.g.*, elimination forms, evaluation context, *etc.* In light of applications (and due to space constraint), here we consider only the operator $\cdot^{\mathbf{F}}$, which captures the recursive pattern needed to define term evaluation: where $a^{\mathbf{H}}$ relates variables, $a^{\mathbf{F}}$ relates introduction forms; and where $a^{\mathbf{H}}$ recursively applies on the operands of a compound term,

$a^{\mathbf{H}}$ recursively applies on the *major* operands of an elimination form. Consequently, we see that $a^{\mathbf{F}}$ arises as the least solution (which indeed exists) to the equation $x = (\overline{\Delta} \vee \langle x \rangle)\,;a$, (cf. with $x = \widehat{x}\,;a$). It is easy to relate $\cdot^{\mathbf{F}}$ to initiality, as done for $\widehat{\cdot}$.

5.1.2.2 Big-Step Semantics

Let us now algebrize the informal notion of evaluation *á la* Martin-Löf. Let us fix a TRA $\mathcal{A}$ with an operational decomposition, and agree that, from now on, any reduction tacitly satisfies (GIP). Notice, in particular, that this implies $\overline{x}\,;a \leq \bot$, for any reduction a.

▶ **Definition 27.** *Given a reduction a, we define the* one-step evaluation *induced by a as* $a^{\mathbf{E}} \triangleq (a \vee \overline{\Delta})^{\mathbf{F}}$.

Intuitively, $a^{\mathbf{E}}$ either relates introduction forms with themselves or it recursively evaluates major arguments of an elimination form, then a-reducing the result obtained. In this sense, we see that it refines parallel reduction following the reduction strategy imposed by the operational decomposition.

We now define evaluation by iterating $a^{\mathbf{E}}$ until a canonical form is reached.

▶ **Definition 28.** *Given a reduction a, we define the* evaluation – *or* big-step semantics – *induced by a as* $a^{\Downarrow} \triangleq a^{\mathbf{E}*}\,;\overline{\Delta}$.

As expected, we can characterise $a^{\Downarrow}$ inductively.

▶ **Proposition 29.** *Let a be a reduction. Then $a^{\Downarrow}$ is the least solution to the equation $x = \overline{\Delta} \vee \langle x \rangle\,;a\,;x$. Furthermore, $\Box a^{\Downarrow} = (\Box a)^{\Downarrow}$. In particular, if a is closed, then $a^{\Downarrow}$ is closed too, and it is the least solution of the equation $x = \Delta_\kappa \vee \langle x \rangle\,;a\,;x$.*

Proof Sketch. Observe that $a^{\mathbf{E}}$ is the least solution to the equation $x = \overline{\Delta} \vee \langle x \rangle\,;a$, since $(\overline{\Delta} \vee \langle x \rangle)\,;(a \vee \overline{\Delta}) = (\overline{\Delta}\,;a) \vee \overline{\Delta} \vee (\langle x \rangle\,;a) \vee (\langle x \rangle\,;\overline{\Delta}) = \overline{\Delta} \vee \langle x \rangle\,;a$. As a consequence, $\Box a^{\mathbf{E}} = (\Box a)^{\mathbf{E}}$. In particular, if a is closed, then $a^{\mathbf{E}}$ is closed too, and it is the least solution of the equation $x = \Delta_\kappa \vee \langle x \rangle\,;a$. Then exploit the least fixed point characterisations of a^*. ◀

▶ **Example 30.** Let us consider the call-by-value λ-calculus (Example 26), and let β_v be call-by-value β-reduction ($\Gamma \vdash (\lambda \mathtt{x.t})\mathtt{v}\ \beta\ \mathtt{t}[\mathtt{v}/\mathtt{x}]$). Then, $\Box\beta^{\Downarrow}$ coincides with the usual big-step semantics $\Downarrow_{\beta_v}$.

Now the main metatheorem of this section: evaluation is deterministic whenever its underlying reduction is.

▶ **Theorem 31** (Determinism). *Let a be a closed reduction. Then:* $a;a^\circ \leq \Delta \implies a^{\Downarrow\circ};a^{\Downarrow} \leq \Delta_\kappa$

Proof Sketch. The proof is composed of different parts. First, we show that GIP implies the well-known *inversion lemma* [111], which is relationally given by $\langle b \rangle\,;a^{\Downarrow} \leq \langle b\,;a^{\Downarrow} \rangle\,;a^{\Downarrow}$. We use the latter to show that, under the assumption $a\,;a^\circ \leq \Delta$, we have $a^\circ\,;a^{\Downarrow} \leq a^{\Downarrow}$. We then prove that $a^\circ\,;a^{\Downarrow} \leq a^{\Downarrow}$ implies $a^{\Downarrow\circ}\,;a^{\Downarrow} \leq \Delta_\kappa$. ◀

6 Pointfree Metatheory: Semantic Equivalence

In this last section, we outline a further application of TRA s to the metatheory of *semantic equivalence*. For many formal systems, it is interesting (and useful) to study the various notions of semantic equivalence (and refinement) they induce on terms. Notable such equivalences include *contextual equivalence* [107] and *applicative (bi)similarity* [3], which

arise in the context of systems of operational semantics, and *conversion* [31, 108] and *Kleene equivalence* [119], which are useful to study rewriting systems (and equational theories). Classic examples of metatheorems on semantic equivalences include *congruence of bisimilarity*, whereby bisimilarity is closed under term formation operations, *Church-Rosser*, which recasts conversion bidirectional (iterated) reduction and leads to *consistency*, and *decidability*, which is obtained by combining Church-Rosser with normalisation. In addition to those, various results relating different semantic equivalences (*e.g.* Kleene equivalence is contained in applicative bisimilarity) are usually of interest.

Many of these notions can be defined and studied in the context of TRAs, including contextual equivalence, applicative bisimilarity, CIU equivalence [101], Kleene equivalence, and conversion. Due to space constraints, here we focus on applicative (bi)similarity only, this notion being known to be meta-theoretically challenging.

6.1 Applicative (Bi)Similarity

(Applicative) similarity is a *coinductively* defined preorder on complete terms. Intuitively, we say that a complete $\mathtt{t}$ is similar to $\mathtt{t}'$ if whenever $\mathtt{t}'$ ($a^{\Downarrow}$)-evaluates to a canonical term $\mathtt{v}'$, then also $\mathtt{t}$ evaluates to a canonical term $\mathtt{v}$, and $\mathtt{v}$ and $\mathtt{v}'$ have the same outermost constructor (*e.g.* they are both lambdas, pairs, *etc.*) with pairwise *similar* operands. Similarity is defined coinductively as the greatest simulation.

We can then "semi-formally" say that similarity is the largest complete relation $\gtrapprox$ such that, whenever $\mathtt{t} \gtrapprox \mathtt{t}'$ (with $\mathtt{t}, \mathtt{t}'$ complete) and $\mathtt{t}'\ a^{\Downarrow}\ \mathtt{v}'$, then $\mathtt{t}\ a^{\Downarrow}\ \mathtt{v}$ and $\mathtt{v}\ \overline{\Diamond \gtrapprox}\ \mathtt{v}'$. Notice that even if similarity is defined on complete terms, we need to take its open extension when it comes to relate sub-terms of introduction forms, as such sub-terms may not be complete themselves. It is an easy exercise to check that, on concrete examples, we obtain Abramsky's applicative similarity [3] and its generalisation by Howe [73].

▶ **Definition 32.** *Let b be a closed reduction. We define $B_b(a)$ as $(b^{\gtrapprox}\,;\overline{a}) \leftarrow b^{\gtrapprox}$, and say that a is a b-*simulation *if $a \leq B_b(\Diamond a)$, i.e. $a\,;b^{\gtrapprox} \leq b^{\gtrapprox}\,;\overline{\Diamond a}$.*

Let b be fixed from now on, so that we can omit subscripts. It is easy to see that B is monotone, and that $B(a)$ is always complete, *i.e.* $\Box B(a) = B(a)$. This means that the equation $x = B(\Diamond x)$ has a greatest solution, which we denote by $b^{\gtrapprox}$ and call *(b-)similarity*. Notice that $b^{\gtrapprox}$ is complete (*i.e.* $\Box b^{\gtrapprox} = b^{\gtrapprox}$), and comes with a corresponding *coinduction proof principle*: $a \leq B(a) \implies a \leq b^{\gtrapprox}$. Using the latter, we see that $b^{\gtrapprox}$ is a preorder on closed terms: $\Box\Delta \leq b^{\gtrapprox}$ and $b^{\gtrapprox}\,;b^{\gtrapprox} \leq b^{\gtrapprox}$. Furthermore, its (open) extension $\Diamond b^{\gtrapprox}$ to incomplete terms – known as *open similarity* – still admits a coinductive characterisation as the greatest solution of the equation $x = \Diamond B(x)$. From that it follows that $\Diamond b^{\gtrapprox}$ is a preorder. Finally, we define *(applicative) bisimilarity* $b^{\cong}$ coinductively as $\nu x.\Diamond x \wedge B(\Diamond x^{\circ})^{\circ}$, and observe that it is an equivalence.

6.1.1 Congruence

The main meta-theoretical result on bisimilarity is congruence. The so-called "transitive-closure trick" [114], which can be easily generalised to TRAs, actually reduces congruence of bisimilarity to *compatibility* and *substitutivity* of similarity. These results, which are highly nontrivial, are typically proved using *Howe's method* [73, 114]. We now outline an algebraic and language-independent proof of compatibility and substitutivity.

Given our "dualised" definition of simulation it is more convenient to use a dual version of the Howe extension operator – *op-Howe extension* – to be applied to closed relations: $a^{\S} \triangleq (\Diamond a)^{\mathbf{H}\circ}$. By the general theory of TRAs, we see that, for any closed preorder a, $a^{\S}$ is *reflexive*, *substitutive*, *compatible* ($\widehat{a^{\S}} \leq a^{\S}$), and satisfies the so-called *quasi-transitivity* law: $\Diamond a \,;\, a^{\S} \leq a^{\S}$.

To prove compatibility, we need to spell out an important hypothesis, one that is hardly detectable without the algebraic vocabulary of TRAs and that expresses a fundamental principle of language design [66]: the *Gentzen Conservation Principle*.

▶ **Definition 33** (Gentzen Conservation Principle)**.** *We say that a closed reduction a satisfies* Gentzen's Conservation Principle *(GCP) if for any compatible relation x, we have $\langle \widehat{x}, x\rangle \,;\, a \leq a \,;\, x[x]$.*

Although cryptic at first, Definition 33 is deeply connected with the proof-theoretic principle of *harmony* [45, 129], as well as with Plotkin's account of a *structural rule* [122]. The equation $\langle \widehat{x}, x\rangle ; a \leq a ; x[x]$, an embryonic version of which is due to Lassen [89], states two facts: *(i)* that the behaviour of a reduction a is determined by the outermost operator of the term to which it is applied, along with the outermost shape of its operands, being parametric in all the rest; *(ii)* that computation proceeds by substitution.[11] To see that, consider the example of the call-by-value λ-calculus: if two terms are related by $\langle \widehat{\phi}, \phi\rangle \,;\, \beta_v$, then we must have $\varnothing \vdash (\lambda \mathtt{x.p})\mathtt{w}\ \langle \widehat{\phi}, \phi\rangle\ (\lambda \mathtt{x.u})\mathtt{v}$, with $\varnothing \vdash \mathtt{w}\ \phi\ \mathtt{v}$ and $\mathtt{x} \vdash \mathtt{p}\ \phi\ \mathtt{u}$. Those give $\mathtt{p}[\mathtt{w}/\mathtt{x}]\ \phi[\phi]\ \mathtt{u}[\mathtt{v}/\mathtt{x}]$ which, together with $(\lambda \mathtt{x.p})\mathtt{w}\ \beta_v\ \mathtt{p}[\mathtt{w}/\mathtt{x}]$, precisely gives $\langle \widehat{\phi}, \phi\rangle \,;\, \beta_v \leq (\beta_v \,;\, \phi[\phi])$.

▶ **Lemma 34** (Key Lemma)**.** *Let a be a closed reflexive and transitive simulation. If b satisfies (GCP), then $a^{\S} \leq \Diamond B(a^{\S})$.*

Sketch. First, an easy calculation shows that $a^{\S} \leq \Diamond B(a^{\S})$ is equivalent to $b^{\gtrsim} \leq \Box a^{\S} \to (b^{\gtrsim} \,;\, \widehat{a^{\S}})$. The latter is proved by fixed point induction on $b^{\gtrsim}$. The proof is then given by a (long) equational calculation. ◀

▶ **Theorem 35.** *$\Diamond b^{\gtrsim}$ is substitutive and compatible, and thus $\Diamond b^{\cong}$ is a congruence.*

Proof Sketch. By Lemma 34, $b^{\gtrsim\S}$ is an open simulation, hence $b^{\gtrsim\S} \leq \Diamond b^{\gtrsim}$, by coinduction. Since $\Diamond b^{\gtrsim} \leq b^{\gtrsim\S}$, we conclude $b^{\gtrsim\S} = \Diamond b^{\gtrsim}$ and thus compatibility of substitutivity of $\Diamond b^{\gtrsim}$. ◀

6.2 Unified Metatheory

A remarkable property of having elucidated both the conversion (GCP) and inversion (GIP) principles is that they not only ensure congruence of (bi)similarity but also confluence of parallel reduction, thereby unifying different metatheoretical results via natural local conditions.

▶ **Lemma 36.** *Let a be a reduction. If a satisfies (GCP) and (GIP), then $a[\Delta]^{\circ} \,;\, \widetilde{a[\Delta]^{\Rightarrow}} \leq a^{\circ}[a^{\Rightarrow}]$.*

Proof Sketch. Observe that $\overline{x} \,;\, a^{\Rightarrow} \leq \overline{x \,;\, a^{\Rightarrow}}$. ◀

We can finally state what we have previously referred to as a *bridge theorem*, which establishes a novel connection between superficially unrelated metatheoretical results.

[11] This is the case for many, but not all, languages. One can consider alternatives to the substitution model of computation, and formulate GCP-like laws of the form $\langle \widehat{x}, x\rangle \,;\, a \leq a \,;\, F(x)$, with F describing how computation is performed.

▶ **Theorem 37.** *Let a be a reduction. If $a[\Delta]$ is deterministic and satisfies (GIP) and (GCP) (and thus, so does $\Box a$) then: (i) $a^{\Rightarrow}$ is confluent; (ii) $(\Box a)^{\Downarrow}$ is deterministic; (iii) $\Diamond((\Box a)^{\cong})$ is a congruence.*

As already mentioned in Section 1.3, we can now easily apply Theorem 37 to various systems. Examples include the call-by-name and call-by-value λ-calculus, and many of their typed extensions that we have omitted. Among those, we mention combined recursive and polymorphic types (System $\mathsf{F}\mu$) [112], PCF [120] possibly extended with type structure [116], and dependent types [100]. Another source of examples is given by calculi for object-oriented computation, most notably Featherweight Java [75] and the ς-calculus [1]. Gordon's relational analysis of the latter [63], for instance, can be read precisely in this sense. (Parts of) Theorem 37 can also be applied to some (but not all) calculi of control, such as the λ_v-C-calculus [46] and the Δ-calculus [137]. More generally, many term and diagram calculi used in the context of the Curry-Howard correspondence [137, 58] provide sources of examples.

7 Related Work and Conclusion

Related Work. The idea of employing algebraic methods, and specifically algebras of relations, to study those formal systems used in the semantics of programming languages is not new. Starting with the work by Bäumer [17], (fragments) of relation algebras have been used to study abstract rewriting systems [74], producing algebraic analyses of abstract confluence and termination [43, 44, 139, 140, 40]. None of these proposals, however, has scaled to term-based systems.

To the best of the author's knowledge, Lassen [89], was the first to explicitly advocate the use of an *algebra of term relations* to study semantic equivalences. Following a previous work by Gordon [62, 61], Lassen introduced various term relation operations on specific λ-calculi. Many crucial operations, however, are not present in Lassen's system, and that prevented the development of algebraic accounts of reduction and evaluation, thereby leaving reasoning markedly syntactic. Even when relational operations are available, the most significant part of proofs has to be carried out syntactically. Nevertheless, it is fair to say that TRAs are conceptually due to Lassen.

Another prominent line of research, as embodied by, *e.g.*, *mathematical operational semantics* [145] and *categorical rewriting* [28, 124, 95], studies formal systems as abstract categorical structures, resting on the unifying vocabulary of category theory to relate different notions. While many abstract results have been achieved in the context of specific systems and abstract term structures (just think about the vast literature on congruence theorems for bisimilarity [145, 26, 71, 146, 60]), bridge theorems in the spirit of Theorem 37 seem lacking. A further difference from TRAs, finally, is that mainstream categorical approaches seem to be inherently "structure-wise", necessitating (and thus depending) of an underlying (abstract) term structure.

Conclusion. This work constitutes a first foundational step in a research programme aimed at a general algebraic theory of various classes of formal systems of term assertions. Systems that have not been treated in this paper but that will be subject of future work include type theories and proof calculi, along with metatheoretical results akin to logical relations and normalisation. Another line of research involves *monadic* algebras of term relations, in the direction of effectful systems [55, 37, 34, 56], as well as quantitative term relation algebras, in the direction of coeffectful systems [38, 98, 109, 54, 35, 36, 10]. Finally, we observe that one can also view TRAs as a (highly expressive) algebraic logic over certain structured domains.

That the domain is a syntactic one, from this perspective, is not particularly relevant; one could come up with similar logics over monoids or groups. It is then interesting to understand this class of (algebraic) logics – logics of predicates over structured domains that logically internalise structural reasoning – in its own right.

References

1 Martín Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer, 1996. doi:10.1007/978-1-4419-8598-9.

2 Michael Gordon Abbott, Thorsten Altenkirch, Neil Ghani, and Conor McBride. Derivatives of containers. In Martin Hofmann, editor, *Proc. of TLCA 2003*, volume 2701 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2003. doi:10.1007/3-540-44904-3_2.

3 S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.

4 Samson Abramsky. Computational interpretations of linear logic. *Theor. Comput. Sci.*, 111(1&2):3–57, 1993. doi:10.1016/0304-3975(93)90181-R.

5 Luca Aceto, Wan Fokkink, and Frits Vaandrager. Structural operational semantics. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 197–292. Elsevier, 2001. doi:10.1016/B978-044482830-9/50021-7.

6 Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *HANDBOOK OF MATHEMATICAL LOGIC*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 739–782. Elsevier, 1977. doi:10.1016/S0049-237X(08)71120-0.

7 Peter Aczel. A general church-rosser theorem. *Draft, Manchester*, 1978.

8 Jiří Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 015(4):589–602, 1974.

9 Hajnal Andréka, Judit X. Madarász, and István Németi. Algebras of relations of various ranks, some current trends and applications. *Journal on Relational Methods in Computer Science*, 1:27–49, 2004. Special volume (JoRMiCS).

10 Martin Avanzini and Akihisa Yamada. Weighted rewriting. In René Thiemann and Christoph Weidenbach, editors, *Proc. of FroCoS '25*, Lecture Notes in Computer Science, pages 191–208. Springer, 2025. doi:10.1007/978-3-032-04167-8_11.

11 Roland Carl Backhouse, Peter J. de Bruin, Paul F. Hoogendijk, Grant Malcolm, Ed Voermans, and Jaap van der Woude. Polynomial relators (extended abstract). In *Proc. of (AMAST '91*, Workshops in Computing, pages 303–326. Springer, 1991.

12 Roland Carl Backhouse and Paul F. Hoogendijk. Elements of a relational theory of datatypes. In *Formal Program Development – IFIP TC2/WG 2.1 State-of-the-Art Report*, pages 7–42, 1993. doi:10.1007/3-540-57499-9_15.

13 Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in Logic. Cambridge University Press, 2013. doi:10.1017/CBO9781139032636.

14 Henk P. Barendregt. *Some Extensional Term Models for Combinatory Logics and Lambda Calculi*. PhD thesis, Utrecht University, Utrecht, The Netherlands, 1971.

15 H.P. Barendregt. *The lambda calculus: its syntax and semantics*. Studies in logic and the foundations of mathematics. North-Holland, 1984.

16 Gabriel A. Baum, Armando M. Haeberer, and Paulo A. S. Veloso. On the representability of the ∇-abstract relational algebra. *Journal of the IGPL*, 1:3–4, 1992.

17 Han Bäumer. On the use of relation algebra in the theory of reduction systems. In *CSN*, volume 92, page 5464, 1992.

18 M. Bezem, J.W. Klop, E. Barendsen, R. de Vrijer, and Terese. *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003.

19 Richard S. Bird and Oege de Moor. *Algebra of programming*. Prentice Hall International series in computer science. Prentice Hall, 1997.

20 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001. doi:10.1017/CBO9781107050884.

21 Willem J. Blok and Don Pigozzi. *Algebraizable Logics*, volume 396 of *Memoirs of the American Mathematical Society*. American Mathematical Society, Providence, 1989.

22 Corrado Böhm. Alcune proprietà delle forme $\beta\eta$-normali del λk-calcolo. *Pubblicazioni dell'Istituto per le Applicazioni del Calcolo*, 696, 1968.

23 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory I: Rewriting with Frobenius structure. *Journal of the ACM*, 2022.

24 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory II: Rewriting with symmetric monoidal structure. *Mathematical Structures in Computer Science*, 2022.

25 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory III: Confluence with and without Frobenius. *Mathematical Structures in Computer Science*, 2022.

26 Peio Borthelle, Tom Hirschowitz, and Ambroise Lafont. A cellular howe theorem. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *Proc. of LICS 2020*, pages 273–286. ACM, 2020. doi:10.1145/3373718.3394738.

27 Stanley Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Springer, New York, 1981.

28 A. Burroni. Higher-dimensional word problems with applications to equational logic. *Theoretical Computer Science*, 115(1):43–62, 1993. doi:10.1016/0304-3975(93)90054-W.

29 Rod M. Burstall. Proving properties of programs by structural induction. *Comput. J.*, 12(1):41–48, 1969. doi:10.1093/COMJNL/12.1.41.

30 Ross Casley, Roger F. Crew, José Meseguer, and Vaughan R. Pratt. Temporal structures. In David H. Pitt, David E. Rydeheard, Peter Dybjer, Andrew M. Pitts, and Axel Poigné, editors, *Category Theory and Computer Science, Manchester, UK, September 5-8, 1989, Proceedings*, volume 389 of *Lecture Notes in Computer Science*, pages 21–51. Springer, 1989. doi:10.1007/BFB0018343.

31 Alonzo Church and J. Barkley Rosser. Some properties of conversion. *Trans. AMS*, 39:472–482, 1936.

32 J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall mathematics series. Dover Publications, Incorporated, 2012.

33 H.B. Curry and R. Feys. *Combinatory Logic*. Number v. 1 in Combinatory Logic. North-Holland Publishing Company, 1958.

34 Francesco Dagnino, Paola Giannini, and Elena Zucca. Monadic type-and-effect soundness. In Jonathan Aldrich and Alexandra Silva, editors, *Proc. of ECOOP 2025*, LIPIcs, pages 7:1–7:31. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ECOOP.2025.7.

35 Francesco Dagnino and Fabio Pasquali. Logical foundations of quantitative equality. In Christel Baier and Dana Fisman, editors, *Proc. of LICS 2022*, pages 16:1–16:13. ACM, 2022. doi:10.1145/3531130.3533337.

36 Francesco Dagnino and Fabio Pasquali. Quantitative equality in substructural logic via lipschitz doctrines. *Log. Methods Comput. Sci.*, 21(1), 2025. doi:10.46298/LMCS-21(1:7)2025.

37 U. Dal Lago, F. Gavazzo, and P.B. Levy. Effectful applicative bisimilarity: Monads, relators, and howe's method. In *Proc. of LICS 2017*, pages 1–12, 2017.

38 Ugo Dal Lago and Francesco Gavazzo. A relational theory of effects and coeffects. *Proc. ACM Program. Lang.*, 6(POPL):1–28, 2022. doi:10.1145/3498692.

39 B.A. Davey and H.A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 1990.

40 Jules Desharnais, Bernhard Möller, and Georg Struth. Algebraic notions of termination. *Log. Methods Comput. Sci.*, 7(1), 2011. doi:10.2168/LMCS-7(1:1)2011.

41 Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975. doi:10.1145/360933.360975.

42 Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

43 Henk Doornbos and Roland Carl Backhouse. Algebra of program termination. In Roland Carl Backhouse, Roy L. Crole, and Jeremy Gibbons, editors, *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction, International Summer School and Workshop, Oxford, UK, April 10-14, 2000, Revised Lectures*, volume 2297 of *Lecture Notes in Computer Science*, pages 203–236. Springer, 2000. doi:10.1007/3-540-47797-7_6.

44 Henk Doornbos, Roland Carl Backhouse, and Jaap van der Woude. A calculational approach to mathematical induction. *Theor. Comput. Sci.*, 179(1-2):103–135, 1997. doi:10.1016/S0304-3975(96)00154-5.

45 Michael Dummett. *The Logical Basis of Metaphysics*. Harvard University Press, Cambridge, 1991.

46 Matthias Felleisen. The theory and practice of first-class prompts. In Jeanne Ferrante and Peter Mager, editors, *Proc. of POPL '88*, pages 180–190. ACM Press, 1988. doi:10.1145/73560.73576.

47 Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. The MIT Press, 1st edition, 2009.

48 Marcelo P. Fiore. Second-order and dependently-sorted abstract syntax. In *Proc. of LICS 2008*, pages 57–68. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.38.

49 Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 193–202. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782615.

50 Peter J. Freyd and Andre Scedrov. *Categories, allegories*, volume 39 of *North-Holland mathematical library*. North-Holland, 1990.

51 Marcelo F. Frias. *Fork Algebras in Algebra, Logic and Computer Science*, volume 2 of *Advances in Logic*. World Scientific, 2002. doi:10.1142/4899.

52 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 214–224. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782617.

53 Francesco Gavazzo. Allegories of symbolic manipulations. In *Proc. of LICS*, pages 1–15, 2023. doi:10.1109/LICS56636.2023.10175807.

54 Francesco Gavazzo and Cecilia Di Florio. Elements of quantitative rewriting. *Proc. ACM Program. Lang.*, 7(POPL), January 2023. doi:10.1145/3571256.

55 Francesco Gavazzo and Claudia Faggian. A relational theory of monadic rewriting systems, part I. In *Proc. of LICS 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470633.

56 Francesco Gavazzo, Riccardo Treglia, and Gabriele Vanoni. Monadic intersection types, relationally. In Stephanie Weirich, editor, *Proc. of ESOP 2024*, volume 14576 of *Lecture Notes in Computer Science*, pages 22–51. Springer, 2024. doi:10.1007/978-3-031-57262-3_2.

57 Gerhard Gentzen. Untersuchungen Über das logische schliessen. i. *Mathematische Zeitschrift*, 35:176–210, 1935.

58 Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Cambridge University Press, USA, 1989.

59 Joseph A. Goguen, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24(1):68–95, 1977. doi:10.1145/321992.321997.

60 Sergey Goncharov, Stefan Milius, Lutz Schröder, Stelios Tsampas, and Henning Urbat. Towards a higher-order mathematical operational semantics. *Proc. ACM Program. Lang.*, 7(POPL):632–658, 2023. doi:10.1145/3571215.

61 A.D. Gordon. A tutorial on co-induction and functional programming. In *Workshops in Computing*, pages 78–95. Springer London, September 1994.

62 A.D. Gordon. Bisimilarity as a theory of functional programming. In Stephen D. Brookes, Michael G. Main, Austin Melton, and Michael W. Mislove, editors, *Proc. of MFPS 1995*, volume 1 of *Electronic Notes in Theoretical Computer Science*, pages 232–252. Elsevier, 1995. doi:10.1016/S1571-0661(04)80013-6.

63 Andrew D. Gordon. Operational equivalences for untyped and polymorphic object calculi. In Andrew D. Gordon and Andrew M. Pitts, editors, *Higher Order Operational Techniques in Semantics*. Cambridge University Press, Cambridge, UK, 1998.

64 David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math*. Texts and Monographs in Computer Science. Springer, 1993. doi:10.1007/978-1-4757-3837-7.

65 D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. Foundations of Computing. MIT Press, 2000.

66 Robert Harper. *Practical Foundations for Programming Languages (2nd. Ed.)*. Cambridge University Press, 2016. URL: https://www.cs.cmu.edu/%7Erwh/pfpl/index.html.

67 Leon Henkin, J. Donald Monk, and Alfred Tarski. *Cylindric Algebras. Part I*, volume 64 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1971.

68 J.R. Hindley and J.P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press, 2008.

69 André Hirschowitz and Marco Maggesi. Modules over monads and linearity. In Daniel Leivant and Ruy J. G. B. de Queiroz, editors, *Proc. of WoLLIC 2007*, volume 4576 of *Lecture Notes in Computer Science*, pages 218–237. Springer, 2007. doi:10.1007/978-3-540-73445-1_16.

70 André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Inf. Comput.*, 208(5):545–564, 2010. doi:10.1016/j.ic.2009.07.003.

71 Tom Hirschowitz and Ambroise Lafont. A categorical framework for congruence of applicative bisimilarity in higher-order languages. *Log. Methods Comput. Sci.*, 18(3), 2022. doi:10.46298/lmcs-18(3:37)2022.

72 C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969. doi:10.1145/363235.363259.

73 D.J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2):103–112, 1996. doi:10.1006/INCO.1996.0008.

74 Gérard P. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, 1980. doi:10.1145/322217.322230.

75 Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight java: A minimal core calculus for java and GJ. In Brent Hailpern, Linda M. Northrop, and A. Michael Berman, editors, *Proc. of OOPSLA 1999*, pages 132–146. ACM, 1999. doi:10.1145/320384.320395.

76 P.T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium: Volume 2*. Oxford Logic Guides. Clarendon Press, 2002.

77 André Joyal. Foncteurs analytiques et espèces de structures. *Comptes rendus de l'Académie des sciences. Série I. Mathématique*, 298(15):393–396, 1984.

78 André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, July 1991. doi:10.1016/0001-8708(91)90003-P.

79 Gilles Kahn. Natural semantics. In *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '87, pages 22–39, Berlin, Heidelberg, 1987. Springer-Verlag. doi:10.1007/BFB0039592.

80 G. A. Kavvos. Modalities, cohesion, and information flow. *Proc. ACM Program. Lang.*, 3(POPL):20:1–20:29, 2019. doi:10.1145/3290333.

81 Yasuo Kawahara. Notes on the universality of relational functors. *Memoirs of the Faculty of Science, Kyushu University. Series A, Mathematics*, 27(2):275–289, 1973.

82 Jan Willem Klop. *Combinatory reduction systems*. PhD thesis, Univ. Utrecht, 1980.

83 Dexter Kozen. On kleene algebras and closed semirings. In Branislav Rovan, editor, *Proc. of MFCS 1990*, volume 452 of *Lecture Notes in Computer Science*, pages 26–47. Springer, 1990. doi:10.1007/BFb0029594.

84 Dexter Kozen. On action algebras. *DAIMI Report Series*, 21(381), January 1992. doi:10.7146/dpb.v21i381.6613.

85 Dexter Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1065.

86 Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, 1997. doi:10.1145/256167.256195.

87 Jean-Louis Krivine. *Lambda-Calculus, Types and Models*. Ellis Horwood Series in Computers and Their Applications. Ellis Horwood, New York, 1993. URL: https://dl.acm.org/doi/10.5555/167408.

88 Thomas Lamiaux and Benedikt Ahrens. An introduction to different approaches to initial semantics. *CoRR*, abs/2401.09366, 2024. doi:10.48550/arXiv.2401.09366.

89 S.B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Dept. of Computer Science, University of Aarhus, May 1998.

90 F. W. Lawvere. Cohesive toposes and cantor's "Lauter Einsen". *Philosophia Mathematica*, 2(1):5–15, 1994. doi:10.1093/philmat/2.1.5.

91 F. William Lawvere. Adjointness in Foundations. *Dialectica*, 23(3-4):281–296, 1969. doi:10.1111/j.1746-8361.1969.tb01194.x.

92 F.William Lawvere. Axiomatic cohesion. *Theory and Applications of Categories [electronic only]*, 19:41–49, 2007.

93 P.B. Levy. Infinitary howe's method. *Electr. Notes Theor. Comput. Sci.*, 164(1):85–104, 2006. doi:10.1016/J.ENTCS.2006.06.006.

94 P.B. Levy, J. Power, and H. Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2):182–210, 2003. doi:10.1016/S0890-5401(03)00088-9.

95 C. Lüth. *Categorical Term Rewriting: Monads and Modularity*. PhD thesis, University of Edinburgh, 1998.

96 S. MacLane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.

97 Roger D. Maddux. Relation algebras. In Chris Brink, Wolfram Kahl, and Gunther Schmidt, editors, *Relational Methods in Computer Science*, Advances in computing science, pages 22–38. Springer, 1997. doi:10.1007/978-3-7091-6510-2_2.

98 Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. Quantitative algebraic reasoning. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proc. of LICS 2016*, pages 700–709. ACM, 2016. doi:10.1145/2933575.2934518.

99 Per Martin-Löf. Constructive mathematics and computer programming. In L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter Podewski, editors, *Logic, Methodology and Philosophy of Science VI*, volume 104 of *Studies in Logic and the Foundations of Mathematics*, pages 153–175. Elsevier, 1982. doi:10.1016/S0049-237X(09)70189-2.

100 Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

101 Ian A. Mason and Carolyn L. Talcott. Equivalence in functional languages with effects. *J. Funct. Program.*, 1(3):287–327, 1991. doi:10.1017/S0956796800000125.

102 Conor Mcbride. The derivative of a regular type is its type of one-hole contexts (extended abstract), 2009.

103 John McCarthy. Towards a mathematical science of computation. In *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 – September 1, 1962*, pages 21–28. North-Holland, 1962.

104 Robin Milner. A theory of type polymorphism in programming. *J. Comput. Syst. Sci.*, 17(3):348–375, 1978. doi:10.1016/0022-0000(78)90014-4.

105 Robin Milner and Mads Tofte. Co-induction in relational semantics. *Theor. Comput. Sci.*, 87(1):209–220, 1991. doi:10.1016/0304-3975(91)90033-X.

106 Robin Milner, Mads Tofte, and Robert Harper. *Definition of standard ML*. MIT Press, 1990.

107 J. Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, MIT, 1969.

108 M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

109 Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. Quantitative program reasoning with graded modal types. *Proc. ACM Program. Lang.*, 3(ICFP):110:1–110:30, 2019. doi:10.1145/3341714.

110 Lorenzo Pace. A relation algebra for term rewriting: A differential approach to sequential reduction (revised version). *CoRR*, abs/2312.02996, 2023. doi:10.48550/arXiv.2312.02996.

111 F. Pfenning. *Computation and Deduction*. Cambridge University Press, 2016. URL: https://books.google.it/books?id=N6YkPQAACAAJ.

112 Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

113 A. M. Pitts. Applications of sup-lattice enriched category theory to sheaf theory. *Proc. London Math. Soc.*, 57:433–480, 1988.

114 A.M. Pitts. Howe's method for higher-order languages. In D. Sangiorgi and J. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*, pages 197–232. Cambridge University Press, 2011.

115 A.M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2013.

116 Andrew M. Pitts. Some notes on inductive and co-inductive techniques in the semantics of functional programs. Technical Report BRICS-NS-94-5, BRICS, Department of Computer Science, University of Aarhus, December 1994. vi+135 pp, draft version. URL: https://cs.au.dk/~gerth/MC/.

117 Andrew M. Pitts. Equivariant syntax and semantics. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 32–36. Springer, 2002. doi:10.1007/3-540-45465-9_3.

118 Andrew M. Pitts. Alpha-structural recursion and induction. *J. ACM*, 53(3):459–506, 2006. doi:10.1145/1147954.1147961.

119 G.D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

120 G.D. Plotkin. Lcf considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977. doi:10.1016/0304-3975(77)90044-5.

121 Gordon Plotkin. Lambda-definability and logical relations. Technical Report SAI-RM-4, School of A.I., University of Edinburgh, 1973.

122 Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. DAIMI FN-19. Computer Science Department, Aarhus University, 1981.

123 Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In Davide Sangiorgi and Jan J. M. M. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, Cambridge tracts in theoretical computer science, pages 233–289. Cambridge University Press, 2012.

124 A.J. Power. An abstract formulation for rewrite systems. In D.H. Pitt, D.E. Rydehard, P. Dybjer, A.M. Pitts, and A. Poigné, editors, *Category Theory and Computer Science*, volume 389, pages 300–312. Springer, 1989. doi:10.1007/BFB0018358.

125 Vaughan R. Pratt. Semantical considerations on floyd-hoare logic. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 109–121. IEEE, 1976. doi:10.1109/SFCS.1976.27.

126 Vaughan R. Pratt. Action logic and pure induction. In Jan van Eijck, editor, *Logics in AI, European Workshop, JELIA '90, Amsterdam, The Netherlands, September 10-14, 1990, Proceedings*, volume 478 of *Lecture Notes in Computer Science*, pages 97–120. Springer, 1990. doi:10.1007/BFb0018436.

127 Vaughan R. Pratt. Dynamic algebras: Examples, constructions, applications. *Stud Logica*, 50(3-4):571–605, 1991. doi:10.1007/BF00370685.

128 Vaughan R. Pratt. Origins of the calculus of binary relations. In *Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS '92), Santa Cruz, California, USA, June 22-25, 1992*, pages 248–254. IEEE Computer Society, 1992. doi:10.1109/LICS.1992.185537.

129 Dag Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Dover Publications, Mineola, N.Y., 1965.

130 J.C. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP Congress*, pages 513–523, 1983.

131 K.I. Rosenthal. *Quantales and their applications*. Pitman research notes in mathematics series. Longman Scientific & Technical, 1990.

132 Davide Sangiorgi. On the proof method for bisimulation (extended abstract). In *Proc. of MFCS'95*, Lecture Notes in Computer Science, pages 479–488. Springer, 1995. doi:10.1007/3-540-60246-1_153.

133 Gunther Schmidt. *Relational Mathematics*, volume 132 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2011.

134 Peter Schroeder-Heister. Proof-Theoretic Semantics. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2024 edition, 2024.

135 P. Selinger. *A Survey of Graphical Languages for Monoidal Categories*, pages 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-12821-9_4.

136 Paul Shields. *Charles S. Peirce on the Logic of Number*. Docent Press, 2012. Originally a doctoral dissertation, Fordham University, 1981.

137 M.H.B. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Rapport (Københavns universitet. Datalogisk institut). Datalogisk Institut, Københavns Universitet, 1998. URL: https://books.google.it/books?id=mm73GQAACAAJ.

138 Allen Stoughton. Substitution revisited. *Theoretical Computer Science*, 59(3):317–325, 1988. doi:10.1016/0304-3975(88)90149-1.

139 Georg Struth. Calculating church-rosser proofs in kleene algebra. In Harrie C. M. de Swart, editor, *Proc. of RelMICS 2001*, volume 2561 of *Lecture Notes in Computer Science*, pages 276–290. Springer, 2001. doi:10.1007/3-540-36280-0_19.

140 Georg Struth. Abstract abstract reduction. *J. Log. Algebraic Methods Program.*, 66(2):239–270, 2006. doi:10.1016/j.jlap.2005.04.001.

141 Masako Takahashi. Parallel reductions in lambda-calculus. *Inf. Comput.*, 118(1):120–127, 1995. doi:10.1006/INCO.1995.1057.

142 A. Tarski and S.R. Givant. *A Formalization of Set Theory Without Variables*. Number v. 41 in A formalization of set theory without variables. American Mathematical Soc., 1988.

143 Alfred Tarski. On the calculus of relations. *J. Symb. Log.*, 6(3):73–89, 1941. doi:10.2307/2268577.

144 Alfred Tarski. What are logical notions? *History and Philosophy of Logic*, 7(2):143–154, 1986. doi:10.1080/01445348608837096.

145 Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 – July 2, 1997*, pages 280–291. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614955.

146 Henning Urbat, Stelios Tsampas, Sergey Goncharov, Stefan Milius, and Lutz Schröder. Weak similarity in higher-order mathematical operational semantics. *CoRR*, abs/2302.08200, 2023. doi:10.48550/arXiv.2302.08200.

147 S. Vickers. *Topology Via Logic*. Cambridge Tracts in Theoretica. Cambridge University Press, 1996.

148 Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115(1):38–94, 1994. doi:10.1006/inco.1994.1093.