Forcing, Transition Algebras, and Calculi

Go Hashimoto
Kyushu University, Fukuoka, Japan

Daniel Găină
Kyushu University, Fukuoka, Japan

Ionuţ Ţuţu
Simion Stoilow Institute of Mathematics of the Romanian Academy, Bucharest, Romania

Abstract

We bring forward a logical system of *transition algebras* that enhances many-sorted first-order logic using features from dynamic logics. The sentences we consider include compositions, unions, and transitive closures of transition relations, which are treated similarly to the actions used in dynamic logics in order to define necessity and possibility operators. This leads to a higher degree of expressivity than that of many-sorted first-order logic. For example, one can finitely axiomatize both the finiteness and the reachability of models, neither of which are ordinarily possible in many-sorted first-order logic. We introduce syntactic entailment and study basic properties such as compactness and completeness, showing that the latter does not hold when standard finitary proof rules are used. Consequently, we define proof rules having both finite and countably infinite premises, and we provide conditions under which completeness can be proved. To that end, we generalize the forcing method introduced in model theory by Robinson from a single signature to a category of signatures, and we apply it to obtain a completeness result for signatures that are at most countable.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Proof theory; Theory of computation → Equational logic and rewriting

Keywords and phrases Forcing, institution theory, calculi, algebraic specification, transition systems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.143

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2404.16111` [23]

Funding The work presented in this paper has been partially supported by Japan Society for the Promotion of Science, grant number 23K11048.

1 Introduction

Algebraic specification is one of the main approaches to formal methods that supports both the formal specification of software and hardware systems and the formal verification of their requirements. The underlying logic of an algebraic specification language is often presented as an institution [16], a category-theoretic formalization of the intuitive notion of logic that includes its syntax, semantics and the satisfaction relation between them. A lot of theoretical computer science has been developed within institution theory [9, 31, 10] based on the principle that formal specification should be based rigorously upon a concrete institution. Two notable specification languages have been designed by following this principle: CafeOBJ in Japan [11] and CASL in Europe [1]. However, there also is an exception given by the Maude system [3], which was originally developed at SRI International in the United States. The underlying logic of Maude, called rewriting logic [25], is not given as an institution, which has led to a series of developments that diverged from mainstream institution-theoretic approaches to topics such as modularization and heterogeneity [4].

Motivation. The main goal of the present study is to apply a body of methods and principles developed within institutional model theory for defining a denotational semantics of algebraic specification languages that are executable by term rewriting such that:

1. it enjoys the modular properties of the logic underlying CafeOBJ such as the *satisfaction condition* for signature morphisms, and therefore it can be formalized as an *institution*;
2. it has the rich expressivity of rewriting logic, in the sense that it can provide a semantics for the Maude language, in general, and for its strategy language [12], in particular.

In algebraic specification languages executable by rewriting such as Maude and CafeOBJ, systems are specified using two kinds of atomic statements: (a) *equations*, which define an algebraic structure on system states, with constructors and derived operations, for example; and (b) *transition rules*, which capture the behaviour of a system by telling us how the states may change as a result of certain actions. In the present contribution, we propose a logic of *transition algebras* where the models consist of many-sorted algebras equipped with binary relations that give semantics to the transition rules. From transition rules one can construct *actions* by applying composition, union, and the Kleene star (i.e., the reflexive and transitive closure of a relation). For the sake of simplicity, we omit the subsorting relation [14].

Many-sorted logical systems. Many-sorted logics are widely acknowledged as being suitable for applications in computer science. However, in pure mathematical logic, they tend to be classified as "inessential variations" [28] of their unsorted forms. This might be true w.r.t. some classical aspects such as compactness or axiomatizability. However, in general, moving from the unsorted to the many-sorted case is a far from trivial task. Allowing for multiple sorts, and thus for multiple carriers in models, some of which may be empty, alters the properties of the logics and significantly increases the complexity of proofs.

An important example of logical property that does not have a straightforward many-sorted generalization is Craig interpolation [7]. This property generally holds in unsorted first-order logic, but fails to hold in the many-sorted variant of the logic; a counterexample can be found, for example, in [2]. Finding the most general criteria for Craig interpolation property was an open problem originally stated in [33]. A solution based on techniques advanced in institutional model theory was provided in [22] after nearly two decades.

Moreover, as noticed in [13], if we admit models with potentially empty carrier sets, then proof rules for unsorted (or single-sorted) first-order logic may be unsound for its many-sorted counterpart. This already suggests that generalizations to other variants of many-sorted first-order logic may pose difficulties. The completeness results proved in an institutional setting, such as [29, 21, 18] are applicable to logical systems where models interpret sorts as non-empty sets. In fact, we are not aware of any completeness result for many-sorted first-order logic in which models interpret sorts as possibly empty sets.

Forcing. In the present contribution, we prove the completeness of the many-sorted logic of transition algebras by applying *forcing*. This technique was originally introduced by Paul Cohen [5, 6] in set theory to show the independence of the continuum hypothesis from the other axioms of Zermelo-Fraenkel set theory. Robinson [30] developed an analogous theory of forcing in model theory. In our setting, forcing is a technique used for constructing expanded models of consistent sets of sentences. More specifically, it allows one to expand a set of sentences while preserving satisfiability even if compactness does not hold in the underlying logical system. Transition algebra is not compact for the same reason classical dynamic propositional logic is not compact. Therefore, the classical Henkin method for proving completeness, which relies on compactness, is not applicable to transition algebra. Another issue arises when proving completeness from the fact that we work with models with empty carriers, because the addition of constants does not necessarily preserve the consistency of theories. Therefore, we generalize the forcing method from a single signature to a category of signatures using ideas from institutional model theory such that the so-called Henkin constants can be added when needed in a way that preserves consistency.

An extended version of this work, which includes all proofs, is available on arXiv [23].

2 Transition algebra

In this section, we define *the logic of many-sorted transition algebras*, or *transition algebra* (TA), for short. We present, in order: signatures, models, sentences, and the TA satisfaction relation.

Signatures. The signatures we consider are ordinary algebraic signatures endowed with polymorphic transition labels and monotonic function symbols. We denote them by tuples of the form $(S, F \supseteq M, L)$, where:

- (S, F) is a many-sorted algebraic signature consisting of a set S of *sorts* and a family $F = \{F_{w,s} \mid w \in S^*, s \in S\}$ of sets of *function symbols*;
- M is a family of subsets $M_{w,s} \subseteq F_{w,s}$ of *monotonic* function symbols; and
- L is a set whose elements we call *transition labels*.

We often write $\sigma\colon w \to s \in F$ to indicate that $\sigma \in F_{w,s}$, and we refer to $w \in S^*$ and $s \in S$ as the *arity* and *sort*, respectively, of the symbol σ. Under this notation, F can also be regarded as an ordinary set consisting of *function declarations* of the form $\sigma\colon w \to s$. When w is the empty arity, we may speak of $\sigma\colon\ \to s$ as a *constant* (symbol) of sort s.

Throughout the paper, we let Σ, Σ', and Σ_i range over arbitrary signatures of the form $(S, F \supseteq M, L)$, $(S', F' \supseteq M', L')$, and $(S_i, F_i \supseteq M_i, L_i)$, respectively.

As usual in institution theory [9, 31], important constructions such as signature extensions with constants as well as open formulae and quantifiers are realized in a multi-signature setting, so moving between signatures is common. A *signature morphism* $\chi\colon \Sigma \to \Sigma'$ consists of an ordinary algebraic signature morphism $\chi\colon (S, F) \to (S', F')$ such that $\chi(M) \subseteq M'$ together with a function $L \to L'$, which we typically denote using the same symbol, χ.

▶ Remark 1. Signature morphisms compose componentwise. Their composition has identities and is associative, thus leading to a category Sig of signatures.

Models. Given a signature Σ, a Σ-*model* $\mathfrak{A}$ is an (S, F)-algebra $\mathfrak{A}$ that interprets every label $\lambda \in L$ as a *many-sorted transition relation* $\lambda^{\mathfrak{A}} \subseteq \mathfrak{A} \times \mathfrak{A}$ (that is, $\lambda^{\mathfrak{A}} = \{\lambda^{\mathfrak{A}}_s \mid s \in S\}$ and $\lambda^{\mathfrak{A}}_s \subseteq \mathfrak{A}_s \times \mathfrak{A}_s$ for all sorts $s \in S$) that respects monotonic function symbols (that is, for all function symbols $\sigma\colon s_1 \cdots s_n \to s$ in M, all tuples $(a_1, \ldots, a_n) \in \mathfrak{A}_{s_1} \times \cdots \times \mathfrak{A}_{s_n}$, all indices $k \in \{1, \ldots, n\}$, and all elements $b \in \mathfrak{A}_{s_k}$, if $\langle a_k, b\rangle \in \lambda^{\mathfrak{A}}_{s_k}$ then $\langle \sigma^{\mathfrak{A}}(a_1 \ldots, a_k, \ldots, a_n), \sigma^{\mathfrak{A}}(a_1 \ldots, b, \ldots, a_n)\rangle \in \lambda^{\mathfrak{A}}_s$).

A *homomorphism* $h : \mathfrak{A} \to \mathfrak{B}$ over a signature Σ is an algebraic (S, F)-homomorphism that preserves transitions: $h(\lambda^{\mathfrak{A}}) \subseteq \lambda^{\mathfrak{B}}$ for all $\lambda \in L$. It is easy to see that Σ-homomorphisms form a category, which we denote by $\mathsf{Mod}(\Sigma)$, under their obvious componentwise composition.

▶ Remark 2. Every signature morphism $\chi\colon \Sigma \to \Sigma$ determines a *model-reduct functor* $\_\!\upharpoonright_\chi\colon \mathsf{Mod}(\Sigma') \to \mathsf{Mod}(\Sigma)$ such that:

- for every Σ'-model $\mathfrak{A}'$, $(\mathfrak{A}'\!\upharpoonright_\chi)_s = \mathfrak{A}'_{\chi(s)}$ for each sort $s \in S$, $\sigma^{\mathfrak{A}'\upharpoonright_\chi} = \chi(\sigma)^{\mathfrak{A}'}$ for each symbol $\sigma \in F$, and $\lambda^{\mathfrak{A}'\upharpoonright_\chi} = \chi(\lambda)^{\mathfrak{A}'}$ for each label $\lambda \in L$; and
- for every Σ'-homomorphism $h'\colon \mathfrak{A}' \to \mathfrak{B}'$, $(h'\!\upharpoonright_\chi)_s = h'_{\chi(s)}$ for each $s \in S$.

Moreover, the mapping $\chi \mapsto \_\!\upharpoonright_\chi$ is functorial.

For any signature morphism $\chi : \Sigma \to \Sigma'$, any Σ-model $\mathfrak{A}$ and any Σ'-model $\mathfrak{A}'$ if $\mathfrak{A} = \mathfrak{A}' \upharpoonright_\chi$, we say that $\mathfrak{A}$ is the *χ-reduct* of $\mathfrak{A}'$, and that $\mathfrak{A}'$ is a *χ-expansion* of $\mathfrak{A}$. For example, for any many-sorted set X (say, of variables) that is disjoint from the set of constant-function symbols in Σ, consider the inclusion morphism $\iota_X : \Sigma \hookrightarrow \Sigma[X]$, where $\Sigma[X] = (S, F[X] \supseteq M, L)$ is the signature obtained from $\Sigma = (S, F \supseteq M, L)$ by adding the elements of X to F as new constant-operation symbols of appropriate sort. Then an expansion of a Σ-model $\mathfrak{A}$ along ι_X can be seen as a pair $\langle \mathfrak{A}, g : X \to \mathfrak{A} \rangle$, where g is a valuation of X in $\mathfrak{A}$.

As in many-sorted algebra, there is a special, initial model in $\mathsf{Mod}(\Sigma)$, which we denote by T_Σ, whose elements are ground terms built from function symbols, and whose transition relations are all empty. The Σ-model $T_\Sigma(X)$ of terms with variables from X is defined as the ι_X-reduct of $T_{\Sigma[X]}$; i.e., $T_\Sigma(X) = T_{\Sigma[X]} \upharpoonright_{\iota_X}$. The following property is an immediate consequence of the initiality of T_Σ.

▶ Remark 3. Any signature morphism $\chi \colon \Sigma \to \Sigma'$ determines uniquely a Σ-homomorphism $T_\Sigma \to T_{\Sigma'} \upharpoonright_\chi$. In order to simplify notations later on, we denote that homomorphism by $\chi \colon T_\Sigma \to T_{\Sigma'} \upharpoonright_\chi$; therefore, for any Σ-term $\sigma(t_1, t_2, \ldots, t_n)$, we have $\chi(\sigma(t_1, t_2, \ldots, t_n)) = \chi(\sigma)(\chi(t_1), \chi(t_2), \ldots, \chi(t_n))$.

Sentences. The *actions* over a signature Σ are defined by the following grammar:

$$\mathfrak{a} ::= \lambda \mid \mathfrak{a} \mathbin{⨟} \mathfrak{a} \mid \mathfrak{a} \cup \mathfrak{a} \mid \mathfrak{a}^*$$

where λ is a transition label of Σ. We let A denote the set of all actions obtained from transition labels declared in a signature Σ, and we extend the notational convention that we use for the components of signatures to their corresponding sets of actions; that is, we usually denote by A' the set of actions over a signature Σ', by A_i the set of actions over a signature Σ_i, and so on. Moreover, through a slight abuse of notation, we also denote by $\chi \colon A \to A'$ the canonical map determined by a signature morphism $\chi \colon \Sigma \to \Sigma'$.

To define sentences, we assume a countably infinite set of *variable names* $\{v_i \mid i < \omega\}$. A *variable* for a signature Σ is a triple $\langle v_i, s, \Sigma \rangle$, where v_i is a variable name and s is a sort in Σ – the third component is used only to ensure that variables are distinct from the constant-operation symbols declared in Σ, which is essential when dealing with quantifiers. The set $\mathsf{Sen}(\Sigma)$ of *sentences* over Σ is given by the following grammar:

$$\phi ::= t_1 = t_2 \mid t_1 \overset{\mathfrak{a}}{\Rightarrow} t_2 \mid \neg\phi \mid \bigvee \Phi \mid \exists X \cdot \phi'$$

where (a) t_1 and t_2 are (S, F)-terms of the same sort; (b) $\mathfrak{a} \in A$ is an action; (c) Φ is a finite set of Σ-sentences; and (d) X is a finite set of variables for Σ and ϕ' is a $\Sigma[X]$-sentence.

When $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$, we may write $\phi_1 \vee \phi_2 \vee \cdots \vee \phi_n$ instead of $\bigvee \Phi$. Besides the above core connectives, we also make use of the following convenient (and standard) abbreviations: $\bigwedge \Phi := \neg \bigvee \{\neg\phi \mid \phi \in \Phi\}$ for finite conjunctions; $\bot := \bigvee \emptyset$ for falsity; $\top := \bigwedge \emptyset = \neg\bot$ for truth; $\phi_1 \to \phi_2 := \neg\phi_1 \vee \phi_2$ for implications; and $\forall X \cdot \phi' := \neg \exists X \cdot \neg\phi'$ for universally quantified sentences.

▶ Remark 4. Any signature morphism $\chi \colon \Sigma \to \Sigma'$ can be canonically extended to a *sentence-translation function* $\chi \colon \mathsf{Sen}(\Sigma) \to \mathsf{Sen}(\Sigma')$ given by:

- $\chi(t_1 = t_2) = (\chi(t_1) = \chi(t_2))$;
- $\chi(t_1 \overset{\mathfrak{a}}{\Rightarrow} t_2) = \chi(t_1) \overset{\chi(\mathfrak{a})}{\Longrightarrow} \chi(t_2)$;
- $\chi(\neg\phi) = \neg\chi(\phi)$;
- $\chi(\bigvee \Phi) = \bigvee \chi(\Phi)$; and
- $\chi(\exists X \cdot \phi') = \exists X' \cdot \chi'(\phi')$, where $X' = \{\langle x, \chi(s), \Sigma' \rangle \mid \langle x, s, \Sigma \rangle \in X\}$ and $\chi' : \Sigma[X] \to \Sigma'[X']$ is the extension of χ mapping each variable $\langle x, s, \Sigma \rangle \in X$ to $\langle x, \chi(s), \Sigma' \rangle \in X'$.

Moreover, this sentence-translation mapping is functorial in χ.

For the sake of simplicity, we identify variables only by their name and sort, provided that there is no danger of confusion. Using this convention, each inclusion morphism $\iota\colon \Sigma \hookrightarrow \Sigma'$ determines an inclusion function $\iota\colon \mathsf{Sen}(\Sigma) \hookrightarrow \mathsf{Sen}(\Sigma')$, which corresponds to the approach of classical model theory. This simplifies the presentation greatly. A situation when we cannot apply this convention arises when translating a Σ-sentence $\exists X \cdot \phi$ along the inclusion $\iota_X : \Sigma \hookrightarrow \Sigma[X]$.

Satisfaction relation. Actions are interpreted as binary transition relations in models. Given a model $\mathfrak{A}$ over a signature Σ, and actions $\mathfrak{a}, \mathfrak{a}_1, \mathfrak{a}_2 \in A$, we have:

- $(\mathfrak{a}_1 \fatsemi \mathfrak{a}_2)^{\mathfrak{A}} = \mathfrak{a}_1^{\mathfrak{A}} \fatsemi \mathfrak{a}_2^{\mathfrak{A}}$ (i.e., diagrammatic composition of binary relations);
- $(\mathfrak{a}_1 \cup \mathfrak{a}_2)^{\mathfrak{A}} = \mathfrak{a}_1^{\mathfrak{A}} \cup \mathfrak{a}_2^{\mathfrak{A}}$ (the union of binary relations); and
- $(\mathfrak{a}^*)^{\mathfrak{A}} = (\mathfrak{a}^{\mathfrak{A}})^*$ (the reflexive and transitive closure of binary relations).

We define the *satisfaction relation* between models and sentences as follows:

- $\mathfrak{A} \models t_1 = t_2$ iff $t_1^{\mathfrak{A}} = t_2^{\mathfrak{A}}$;
- $\mathfrak{A} \models t_1 \overset{\mathfrak{a}}{\Rightarrow} t_2$ iff $(t_1^{\mathfrak{A}}, t_2^{\mathfrak{A}}) \in \mathfrak{a}^{\mathfrak{A}}$;
- $\mathfrak{A} \models \neg\phi$ iff $\mathfrak{A} \not\models \phi$,
- $\mathfrak{A} \models \bigvee \Phi$ iff $\mathfrak{A} \models \phi$ for some sentence $\phi \in \Phi$, and
- $\mathfrak{A} \models \exists X \cdot \phi'$ iff $\mathfrak{A}' \models \phi'$ for some expansion $\mathfrak{A}'$ of $\mathfrak{A}$ to the signature $\Sigma[X]$.

For the sake of simplicity, we write $d_1 \xRightarrow{\mathfrak{a}} d_2$ if $\langle d_1, d_2 \rangle \in \mathfrak{a}^{\mathfrak{A}}$.

Let ϕ, ϕ' be sets of Σ-sentences, $\mathfrak{A}$ a Σ-model. We also use the following notations:

- $\mathfrak{A} \models \Phi$ iff $\mathfrak{A} \models \phi$ for all sentences $\phi \in \Phi$;
- $\Gamma \models \Phi$ iff $\mathfrak{A} \models \Gamma$ implies $\mathfrak{A} \models \Phi$ for all Σ-models $\mathfrak{A}$.

In particular, we write $\Gamma \models \phi$ instead of $\Gamma \models \{\phi\}$ for any set of sentences Γ and any single sentence ϕ.

▶ **Proposition 5.** *For all signature morphisms $\chi : \Sigma \to \Sigma'$, all Σ'-models $\mathfrak{A}$ and all sentences $\phi \in \mathsf{Sen}(\Sigma)$ we have: $\mathfrak{A}{\restriction}_\chi \models \phi \quad$ iff $\quad \mathfrak{A} \models \chi(\phi)$.*

▶ **Example 6** (CCS)**.** To illustrate the expressivity of transition algebra, we refer to Robin Milner's *calculus of communicating systems* (CCS) [26, 27], which is emblematic of a broad family of formal languages used for modelling and reasoning about concurrency. In a nutshell, CCS is a process calculus that enables syntactic descriptions of concurrent systems to be written, and subsequently manipulated and analysed, based on two kinds of atomic entities – process identifiers and channel names – and a handful of composition operators.

To start, we assume two sets: *PI* of *process identifiers*, and *CN* of so-called *channel names*, which capture the interaction capabilities of processes. Take, for instance, the following famous quote by Alfréd Rényi: "A mathematician is a machine for turning coffee into theorems" [32]. This can be modelled in CCS as an interaction between two processes, a mathematician and a coffee vending machine, that trade coffee (in exchange, perhaps, of coins or some other form of payment) in order to jointly produce theorems. Hence, we can consider theorems, coffee, and coins as types of interactions between the two processes.

For each channel name $c \in \mathit{CN}$, we let $\overline{c}$ be a new symbol, distinct from all channel names, called the *co-name* of c. We also let $\overline{\mathit{CN}} = \{\overline{c} \mid c \in \mathit{CN}\}$ be the set of all co-names, and $L = \mathit{CN} \cup \overline{\mathit{CN}}$ be the set of *labels*. Intuitively, we may regard the symbols in CN as inputs of some process, and the symbols in $\overline{\mathit{CN}}$ as outputs. Besides labels, we also consider an additional *silent-action* symbol, denoted τ, that indicates an internal, unobservable behaviour of the system under consideration. Altogether, we refer to the symbols in $A = L \cup \{\tau\}$ as CCS *actions*.[1] *Processes* over CN and PI are defined according to the following grammar:

[1] Although they share the name "action", CCS actions are conceptually very different from the actions we have defined for transition algebra. To distinguish the two, we always prefix the former by CCS.

$$P \ ::= \ 0 \mid \pi \mid a\,.\,P \mid P + P \mid P\,\text{‘}|\text{’}\,P \mid P \setminus k$$

where (a) 0 denotes a special terminal, inactive process; (b) $\pi \in PI$ is a process identifier; (c) $a \in A$ is a CCS action, which can be used to prefix a process P in order to form a new process, $a\,.\,P$, that intuitively performs a then continues as P; (d) ‘+’ denotes the non-deterministic choice between two processes; (e) ‘|’ denotes the parallel composition of two processes; and (f) $k \in CN$ is a channel name, which can be used in expressions like $P \setminus k$ to form a new, restricted process with the same interaction capabilities as P except for the labels k and $\overline{k}$. To simplify the presentation, we omit the relabelling operator because it plays no role in the examples we consider in this paper and it could be added with ease if needed. For a comprehensive account of CCS, see for example [27].

A CCS *context*, or *program*, is a set of *declarations* of the form $\pi ::= P$, where $\pi \in PI$ is a process identifier and P is a process conforming to the grammar introduced above, such that any two distinct declarations have distinct left-hand sides; in other words, we do not admit multiple declarations of the same process identifier within a given context.

Using this syntax, the interaction between mathematicians and coffee vending machines announced in Alfréd Rényi's quote can be formalized as a parallel composition of processes over $PI = \{\texttt{Mathematician}, \texttt{CoffeeVM}\}$ and $CN = \{\texttt{coin}, \texttt{coffee}, \texttt{theorem}\}$, which we write as $\texttt{Mathematician} \mid \texttt{CoffeeVM}$, where `Mathematician` and `CoffeeVM` are process identifiers "defined" recursively according to the following context:

- $\texttt{Mathematician} ::= \overline{\texttt{coin}}\,.\,\texttt{coffee}\,.\,\overline{\texttt{theorem}}\,.\,\texttt{Mathematician}$,
- $\texttt{CoffeeVM} ::= \texttt{coin}\,.\,\overline{\texttt{coffee}}\,.\,\texttt{CoffeeVM}$.

Thanks to the expressivity of transition algebra, we can easily capture both the syntax and the operational semantics of CCS. For the syntax of processes, it suffices to consider a many-sorted TA signature with $S = \{\texttt{Channel}, \texttt{Action}, \texttt{Process}\}$ and with F given by the following function symbols (which employ OBJ's [15] and Maude's [3] mixfix notation):

- $0\colon \ \to \texttt{Process}$,
- $\pi\colon \ \to \texttt{Process}$ for each process identifier $\pi \in PI$,
- $a\colon \ \to \texttt{Action}$ for each CCS action $a \in A$,
- $\_\,.\,\_\colon \texttt{Action Process} \to \texttt{Process}$,
- $\_+\_\,,\ \_\,|\,\_\colon \texttt{Process Process} \to \texttt{Process}$,
- $k\colon \ \to \texttt{Channel}$ for each channel name $k \in CN$,[2]
- $\_\setminus\_\colon \texttt{Process Channel} \to \texttt{Process}$.

The parallel-composition operator is the only monotonic function symbol of this example. For convenience, we also declare the parallel-composition operator and the non-deterministic choice as associative, commutative, and with identity 0. These properties can be presented – as usual in algebraic specification – using plain equations. To capture and reason about the behaviour of processes, we regard each CCS action as an atomic action (i.e., a transition label) in transition algebra – and those are the only TA labels that we consider here.

The transitional semantics of a CCS program *Pgm* is given by the following collection of axioms. The transition-algebra sentences below are all universally quantified over variables P, P', Q, Q' of sort `Process` and k of sort `Channel`; however, we drop the quantifiers in order to simplify the notation. We also use names for the axioms (at the beginning of each line) that are indicative of the transition rules defined in [27].

[2] To avoid subsorting, we overload channel names, which can be seen either as constants of sort `Channel` or as constants of sort `Action` depending on the context in which they are used.

- (Act) $\quad a\,.\,P \overset{a}{\Rightarrow} P \quad$ for all $a \in A$,
- (Sum) $\; P \overset{a}{\Rightarrow} P' \to P + Q \overset{a}{\Rightarrow} P' \quad$ for all $a \in A$,
- (Com) $\; P \overset{c}{\Rightarrow} P' \wedge Q \overset{\overline{c}}{\Rightarrow} Q' \to P \mid Q \overset{\tau}{\Rightarrow} P' \mid Q' \quad$ for all $c \in \mathit{CN}$,
- (Res) $\; P \overset{a}{\Rightarrow} P' \wedge a \neq k \wedge \overline{a} \neq k \to P \setminus k \overset{a}{\Rightarrow} P' \setminus k \quad$ for all $a \in A$,[3]
- (Con) $\; P \overset{a}{\Rightarrow} P' \to \pi \overset{a}{\Rightarrow} P' \quad$ for all $\pi ::= P \in \mathit{Pgm}$.

To simplify some of the notations used later on in the paper, for any process P and any non-empty and finite sequence $K = (k_i \mid 1 \leq i \leq n)$ of channel names, we also write $P \setminus K$ in place of $P \setminus k_1 \setminus \cdots \setminus k_n$ and we consider the following derived form of the axiom (Res):

- (Res^*) $\; P \overset{a}{\Rightarrow} P' \wedge \bigwedge\{a \neq k_i \wedge \overline{a} \neq k_i \mid 1 \leq i \leq n\} \to P \setminus K \overset{a}{\Rightarrow} P' \setminus K \quad$ for all $a \in A$.

Similar encodings of CCS in languages that support transitions can be found in the rewriting-logic literature, notably in [24, 34, 8]. But the encodings presented therein rely on a notion of *derivative* of a process instead of reasoning about plain processes, which is usually because labelled transitions cannot be used in the conditions of Horn clauses such as *Sum* and *Com*. That is, the axiomatization is done in terms of pairs $\langle \alpha, P \rangle$, where P is a process and α is a CCS action or a sequence of CCS actions leading to P. In TA, this additional step can be avoided because the use of labelled transitions is unrestricted, which allows our axioms to be nearly to-the-letter transcriptions of Robin Milner's rules for CCS.

3 Entailment relations

In this section, we define the proof-theoretic properties necessary for proving our results such as entailment relation, soundness and completeness. Before we proceed, let us recall an example from [13], which shows that classical rules of first-order deduction are not sound.

▶ **Example 7.** Let $\Sigma = (S, F)$ be an algebraic signature consisting of:

- two sorts, that is, $S = \{\mathit{Elt}, \mathit{Bool}\}$, and
- five function symbols $F = \{\mathit{true} :\to \mathit{Bool}, \mathit{false} :\to \mathit{Bool}, \sim\_ : \mathit{Bool} \to \mathit{Bool}, \_\&\_ : \mathit{Bool}\ \mathit{Bool} \to \mathit{Bool}, \_+\_ : \mathit{Bool}\ \mathit{Bool} \to \mathit{Bool}, \mathit{foo} : \mathit{Elt} \to \mathit{Bool}\}$.

Let Γ be a set of sentences over Σ which consists of the following sentences:

- $\sim \mathit{true} = \mathit{false}$ and $\sim \mathit{false} = \mathit{true}$,
- $\forall y \cdot y \ \&\ \sim y = \mathit{false}$ and $\forall y \cdot y \ \&\ y = y$,
- $\forall y \cdot y \ +\ \sim y = \mathit{true}$ and $\forall y \cdot y \ +\ y = y$, and
- $\forall x \cdot\ \sim \mathit{foo}(x) = \mathit{foo}(x)$.

Using the ordinary rules of first-order deduction, we can show that

$$\begin{aligned} \mathit{true} &= \mathit{foo}(x) + \sim \mathit{foo}(x) \\ &= \mathit{foo}(x) + \mathit{foo}(x) \\ &= \mathit{foo}(x) \\ &= \mathit{foo}(x)\ \&\ \mathit{foo}(x) \\ &= \mathit{foo}(x)\ \&\ \sim \mathit{foo}(x) \\ &= \mathit{false} \end{aligned} \tag{1}$$

As a result, one would expect $\mathit{true} = \mathit{false}$ to hold in all algebras satisfying Γ. But that is not the case. To see why, suppose $\mathfrak{A}$ is the algebra obtained from T_Σ through a factorization under the congruence relation $\equiv_\Gamma$ generated by Γ, that is:

[3] As usual in CCS, we extend the over-line notation employed for channel co-names to a bijection $\bar{\cdot}\colon A \to A$ given by $\overline{\overline{c}} = c$ for all channel names c and by $\overline{\tau} = \tau$ for the silent action.

- $\mathfrak{A}_{Bool} = \{true/_{\equiv_\Gamma}, false/_{\equiv_\Gamma}\}$ and $\mathfrak{A}_{Elt} = \emptyset$,
- $\sim$ is interpreted as the negation, & as the conjunction and $+$ as the disjunction, and
- $foo^{\mathfrak{A}}$ is the empty function.

Clearly, the algebra $\mathfrak{A}$ satisfies the sentences in Γ referring to the negation, conjunction, and disjunction of booleans. Moreover, since there is no function from $\{x\}$ to $\mathfrak{A}_{Elt} = \emptyset$, we have $\mathfrak{A} \models_\Sigma \forall x \cdot \sim foo(x) = foo(x)$. It follows that $\mathfrak{A} \models_\Sigma \Gamma$ but $\mathfrak{A} \not\models_\Sigma true = false$.

This shows that moving from the unsorted to the many-sorted case is not as straightforward as one might expect. Since a model may have some empty domains, one needs to design proof rules that take into account changes of signatures.

▶ **Definition 8** (Entailment relation). *An entailment relation* $\vdash = \{\vdash_\Sigma\}_{\Sigma \in |\mathsf{Sig}^{\mathsf{TA}}|}$ *is a family of binary relations between sets of sentences indexed by signatures, that is,* $\vdash_\Sigma \subseteq \mathcal{P}(\mathsf{Sen}(\Sigma)) \times \mathcal{P}(\mathsf{Sen}(\Sigma))$ *for all first-order signatures* Σ*, such that the following properties are satisfied:*

$$(Monotonicity)\ \frac{\Gamma \supseteq \Phi}{\Gamma \vdash_\Sigma \Phi} \qquad (Transitivity)\ \frac{\Gamma \vdash_\Sigma \Phi \qquad \Phi \vdash_\Sigma \Psi}{\Gamma \vdash_\Sigma \Psi}$$

$$(Union)\ \frac{\Gamma \vdash_\Sigma \phi \text{ for all } \phi \in \Phi}{\Gamma \vdash_\Sigma \Phi} \qquad (Translation)\ \frac{\Gamma \vdash_\Sigma \Phi}{\chi(\Gamma) \vdash_{\Sigma'} \chi(\Phi)} \text{ where } \chi : \Sigma \to \Sigma'$$

For the sake of simplicity, we write $\Gamma \vdash_\Sigma \phi$ rather than $\Gamma \vdash_\Sigma \{\phi\}$. Also, we omit the subscript Σ from the notation $\vdash_\Sigma$ when it is clear from the context. An example of entailment relation is $\models$. It is straightforward to prove that $\models$ satisfies $(Monotonicity)$, $(Transitivity)$, $(Union)$ and $(Translation)$.

▶ **Definition 9** (Entailment properties). *An entailment relation* $\vdash$ *is* sound (complete) *if* $\vdash \subseteq \models$ ($\models \subseteq \vdash$)*. An entailment relation* $\vdash$ *is* α-compact*, where* α *is an infinite cardinal, if*

$$\Gamma \vdash_\Sigma \phi \text{ implies } \Gamma_\alpha \vdash_\Sigma \phi \text{ for some subset } \Gamma_\alpha \subseteq \Gamma \text{ of cardinality } \mathsf{card}(\Gamma_\alpha) < \alpha,$$

for all signatures Σ*, all sets of* Σ*-sentences* Γ *and all* Σ*-sentences* ϕ*. If* $\alpha = \omega$*, we say, simply, that* $\vdash$ *is compact.*

The dynamic entailment relation is defined in two steps. Firstly, we define an entailment relation to reason about the logical consequences of atomic sentences, given as equations or relations. Secondly, we define the dynamic entailment relation by adding proof rules to deal with actions, Boolean connectives and quantifiers.

3.1 Basic entailment relation

The fragment obtained from TA by restricting the sentences to atoms is studied in this section.

▶ **Definition 10** (Basic entailment relation). The basic entailment relation $\vdash^b$ *is the least entailment relation closed under the following basic proof rules:*

$$(R)\ \frac{}{\Gamma \vdash_\Sigma t = t} \qquad (S)\ \frac{\Gamma \vdash_\Sigma t_1 = t_2}{\Gamma \vdash_\Sigma t_2 = t_1} \qquad (T)\ \frac{\Gamma \vdash_\Sigma t_1 = t_2 \qquad \Gamma \vdash_\Sigma t_2 = t_3}{\Gamma \vdash_\Sigma t_1 = t_3}$$

$$(F)\ \frac{\Gamma \vdash_\Sigma t_i = t_i' \quad \text{for } 1 \leq i \leq n}{\Gamma \vdash_\Sigma \sigma(t_1, \ldots, t_n) = \sigma(t_1', \ldots, t_n')} \qquad (P)\ \frac{\Gamma \vdash_\Sigma t_1 = t_1' \quad \Gamma \vdash_\Sigma t_2 = t_2' \quad \Gamma \vdash_\Sigma t_1 \overset{\lambda}{\Longrightarrow} t_2}{\Gamma \vdash_\Sigma t_1' \overset{\lambda}{\Longrightarrow} t_2'}$$

$$(M)\ \frac{\Gamma \vdash_\Sigma t_j \overset{\lambda}{\Longrightarrow} u_j}{\Gamma \vdash_\Sigma f(t_1, \ldots, t_j, \ldots, t_n) \overset{\lambda}{\Longrightarrow} f(t_1, \ldots, u_j, \ldots, t_n)} \quad \text{where } f \in M$$

▶ **Lemma 11** (Basic compactness). *The basic entailment relation is compact.*

Any set of atomic sentences E defined over a signature Σ determines a congruence $\equiv^E := \{t_1 \equiv^E t_2 \mid E \vdash_\Sigma t_1 = t_2\}$ on T_Σ. One can construct a model $\mathfrak{A}^E$ from the initial model of terms T_Σ factorized by the congruence $\equiv^E$ interpreting each transition label λ in Σ as the set $\{(t_1, t_2) \mid t_1, t_2 \in T_{\Sigma,s}$ and $E \vdash_\Sigma t_1 \overset{\lambda}{\Longrightarrow} t_2\}$.

▶ **Lemma 12.** *Let E be a set of atomic sentences defined over a signature Σ. For all Σ-models $\mathfrak{A}$, we have $\mathfrak{A} \models E$ iff there exists a unique homomorphism $\mathfrak{A}^E \to \mathfrak{A}$.*

Lemma 12 says that the satisfaction of E by a model $\mathfrak{A}$ is equivalent with the existence of a unique homomorphism from $\mathfrak{A}^E$ to $\mathfrak{A}$. In particular, $\mathfrak{A}^E$ is the initial model of E. See [9] for a proof of Lemma 12.

▶ **Proposition 13** (Basic completeness). *For any set of atomic sentences E and any atomic sentence φ defined over a signature Σ, the following are equivalent:*

$$(a)\ E \models \varphi, \qquad (b)\ \mathfrak{A}^E \models \varphi, \qquad \textit{and} \qquad (c)\ E \vdash^b \varphi.$$

3.2 Dynamic entailment relation

The dynamic entailment relation is built on top of basic entailment relation by adding the proof rules to reason about actions, Boolean connectives, and first-order quantifiers.

▶ **Definition 14** (Dynamic entailment relation). The dynamic entailment relation $\vdash$ *is the least entailment relation closed under the basic proof rules presented in Definition 10 and the following proof rules:*

Proof rules for actions

$$(Comp_I)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_1}{\Longrightarrow} t \qquad \Gamma \vdash_\Sigma t \overset{\mathfrak{a}_2}{\Longrightarrow} t_2}{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_1 \mathbin{;} \mathfrak{a}_2}{\Longrightarrow} t_2}$$

$$(Comp_E)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_1 \mathbin{;} \mathfrak{a}_2}{\Longrightarrow} t_2 \quad \Gamma \cup \{t_1 \overset{\mathfrak{a}_1}{\Longrightarrow} x, x \overset{\mathfrak{a}_2}{\Longrightarrow} t_2\} \vdash_{\Sigma[x]} \phi}{\Gamma \vdash_\Sigma \phi}$$

$$(Union_I)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_i}{\Longrightarrow} t_2}{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_1 \cup \mathfrak{a}_2}{\Longrightarrow} t_2} \qquad (Union_E)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_1 \cup \mathfrak{a}_2}{\Longrightarrow} t_2 \quad \Gamma \cup \{t_1 \overset{\mathfrak{a}_i}{\Longrightarrow} t_2\} \vdash_\Sigma \phi \text{ for all } i \in \{1,2\}}{\Gamma \vdash_\Sigma \phi}$$

$$(Star_I)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}^n}{\Longrightarrow} t_2}{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}^*}{\Longrightarrow} t_2} \qquad (Star_E)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}^*}{\Longrightarrow} t_2 \quad \Gamma \cup \{t_1 \overset{\mathfrak{a}^n}{\Longrightarrow} t_2\} \vdash_\Sigma \phi \text{ for all } n \in \omega}{\Gamma \vdash_\Sigma \phi}$$

Proof rules for Boolean connectives

$$(Neg_D)\ \frac{\Gamma \vdash_\Sigma \neg\neg\phi}{\Gamma \vdash_\Sigma \phi} \qquad (False)\ \frac{\Gamma \vdash_\Sigma \bot}{\Gamma \vdash_\Sigma \phi}$$

$$(Neg_I)\ \frac{\Gamma \cup \{\phi\} \vdash_\Sigma \bot}{\Gamma \vdash_\Sigma \neg\phi} \qquad (Neg_E)\ \frac{\Gamma \vdash_\Sigma \neg\phi}{\Gamma \cup \{\phi\} \vdash_\Sigma \bot}$$

$$(Disj_I)\ \frac{\Gamma \vdash_\Sigma \phi}{\Gamma \vdash_\Sigma \vee\Phi} \text{ where } \phi \in \Phi \qquad (Disj_E)\ \frac{\Gamma \vdash_\Sigma \vee\Phi \quad \Gamma \cup \{\phi\} \vdash_\Sigma \gamma \text{ for all } \phi \in \Phi}{\Gamma \vdash_\Sigma \gamma}$$

Proof rules for first-order quantifiers

$$(Quant_I)\ \frac{\Gamma \cup \{\phi\} \vdash_{\Sigma[X]} \gamma}{\Gamma \cup \{\exists X \cdot \phi\} \vdash_{\Sigma} \gamma} \qquad (Quant_E)\ \frac{\Gamma \cup \{\exists X \cdot \phi\} \vdash_{\Sigma} \gamma}{\Gamma \cup \{\phi\} \vdash_{\Sigma[X]} \gamma}$$

$$(Subst)\ \frac{\Gamma \vdash_{\Sigma} \theta(\phi)}{\Gamma \vdash_{\Sigma} \exists X \cdot \phi} \qquad \text{where } \theta : X \to T_{\Sigma} \text{ is a substitution}$$

▶ **Proposition 15** (ω_1-compactness). *We have that*

1. *the dynamic entailment relation $\vdash$ is ω_1-compact, and*
2. *the satisfaction relation $\models$ is not ω_1-compact.*

The first statement holds because the dynamic entailment relation is generated by proof rules with an at most countable number of premises. For uncountable signatures Σ, the satisfaction relation $\models_{\Sigma}$ is not ω_1-compact. It follows that the dynamic logic proposed in this contribution, TA, is not complete. However, the restriction of TA to countable signatures, TA^c, is complete. Since TA^c is not compact, the Henkin method for proving completeness is not applicable.

▶ **Example 16** (Analysis of CCS programs). Recall the CCS description of the interaction between mathematicians and coffee vending machines discussed in Section 2, and let `Institute` be an abbreviation for the following process:

$$(\texttt{Mathematician} \mid \texttt{CoffeeVM}) \setminus (\texttt{coin}, \texttt{coffee}).$$

In this context, can we check, as an example, that the process `Institute` is able to continuously output theorems? The property can be formalized in TA as a transition

$$\texttt{Institute} \xRightarrow{\tau^* \,\mathbin{;}\, \overline{\texttt{theorem}} \,\mathbin{;}\, \tau^*} \texttt{Institute}$$

whose τ-components correspond to internal communications between sub-processes of the institute – i.e., mathematicians and vending machines. Therefore, we need to check an entailment of the form $\Gamma \vdash_{\Sigma} \phi$, where (a) Σ is the TA-signature that consists of the process identifiers `Mathematician` and `CoffeeVM` together with the CCS process-building operators for action prefixing, non-deterministic choice, parallel composition, etc., discussed in Section 2; (b) Γ is the set of Σ-sentences given by the axiom schemas *Act*, *Sum*, *Com*, *Res**, and *Con* listed on page 7 together with equations pertaining to the axiomatization of CCS actions (e.g., $\overline{\texttt{theorem}} \neq \texttt{coffee}$), as well as equations that capture elementary properties of processes such as the associativity, commutativity, and identity element of the non-deterministic-choice and parallel-composition operators; and (c) ϕ is the transition written above.

The proof mimics the following chain of CCS transitions:

$$\begin{aligned}\texttt{Institute} &\xRightarrow{\tau} (\texttt{coffee} \,.\, \overline{\texttt{theorem}} \,.\, \texttt{Mathematician} \mid \overline{\texttt{coffee}} \,.\, \texttt{CoffeeVM}) \setminus (\texttt{coin}, \texttt{coffee}) \\ &\xRightarrow{\tau} (\overline{\texttt{theorem}} \,.\, \texttt{Mathematician} \mid \texttt{CoffeeVM}) \setminus (\texttt{coin}, \texttt{coffee}) \\ &\xRightarrow{\overline{\texttt{theorem}}} (\texttt{Mathematician} \mid \texttt{CofeeVM}) \setminus (\texttt{coin}, \texttt{coffee}) = \texttt{Institute}\end{aligned}$$

To shorten the presentation of the proof, we use the following derived proof rule:

$$(GMP)\ \frac{\Gamma \vdash_{\Sigma} \forall X \cdot \bigwedge \Phi \to \gamma \quad \Gamma \vdash_{\Sigma} \theta(\Phi)}{\Gamma \vdash_{\Sigma} \theta(\gamma)} \qquad \text{where } \theta : X \to T_{\Sigma} \text{ is a substitution.}$$

In addition, we simplify the notations by writing down only the conclusions of entailments and by abbreviating `Mathematician` as `M`, `CoffeeVM` as `CVM`, and the sequence of channel names $(\texttt{coin}, \texttt{coffee})$ as K. This leads us to the (sketch of) proof tree depicted in Figure 1.

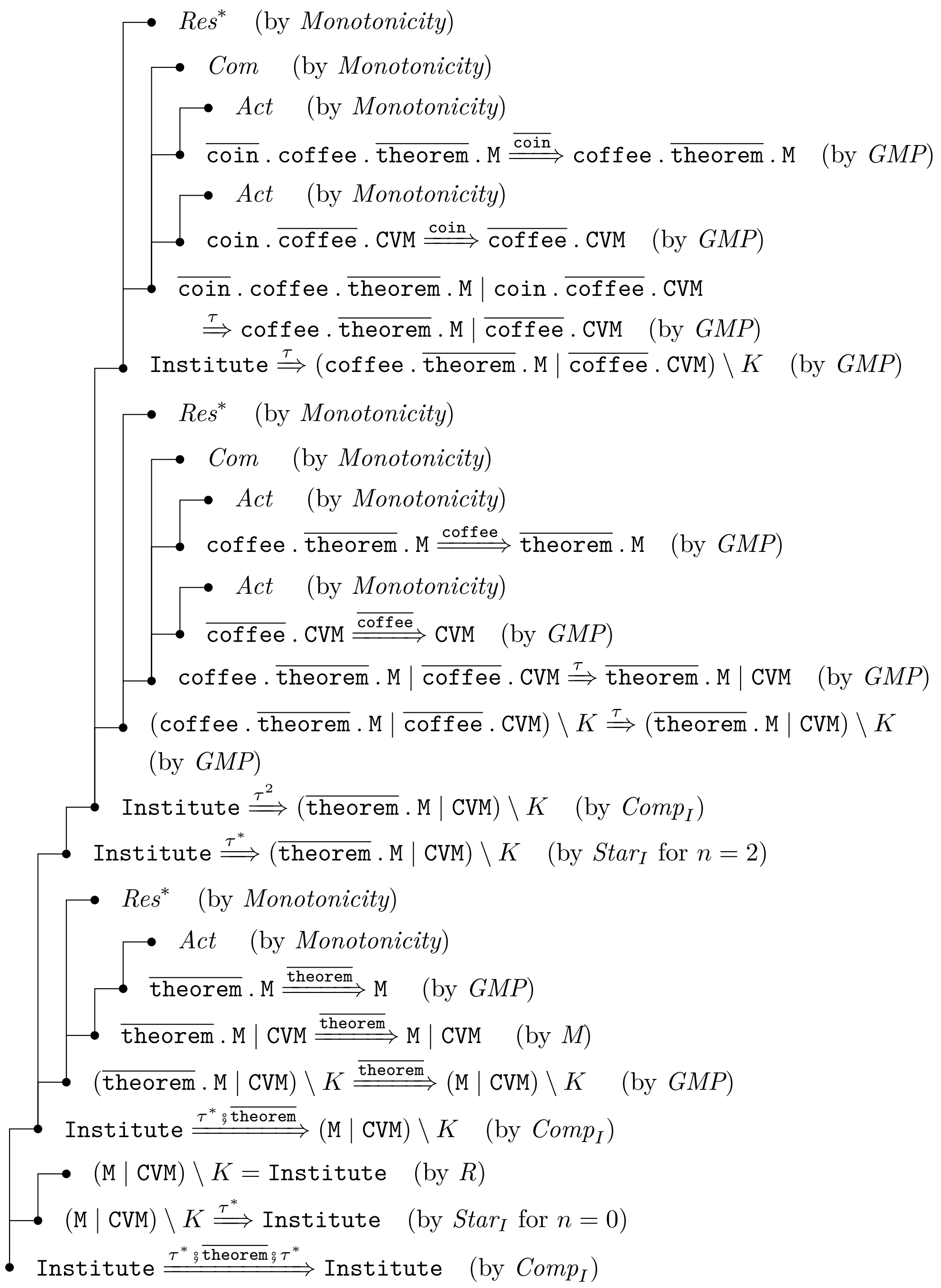
Res^* (by $Monotonicity$)
Com (by $Monotonicity$)
Act (by $Monotonicity$)
$\overline{\mathtt{coin}}\,.\,\mathtt{coffee}\,.\,\overline{\mathtt{theorem}}\,.\,\mathtt{M} \xRightarrow{\overline{\mathtt{coin}}} \mathtt{coffee}\,.\,\overline{\mathtt{theorem}}\,.\,\mathtt{M}$ (by GMP)
Act (by $Monotonicity$)
$\mathtt{coin}\,.\,\overline{\mathtt{coffee}}\,.\,\mathtt{CVM} \xRightarrow{\mathtt{coin}} \overline{\mathtt{coffee}}\,.\,\mathtt{CVM}$ (by GMP)
$\overline{\mathtt{coin}}\,.\,\mathtt{coffee}\,.\,\overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \mathtt{coin}\,.\,\overline{\mathtt{coffee}}\,.\,\mathtt{CVM} \xRightarrow{\tau} \mathtt{coffee}\,.\,\overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \overline{\mathtt{coffee}}\,.\,\mathtt{CVM}$ (by GMP)
$\mathtt{Institute} \xRightarrow{\tau} (\mathtt{coffee}\,.\,\overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \overline{\mathtt{coffee}}\,.\,\mathtt{CVM}) \setminus K$ (by GMP)
Res^* (by $Monotonicity$)
Com (by $Monotonicity$)
Act (by $Monotonicity$)
$\mathtt{coffee}\,.\,\overline{\mathtt{theorem}}\,.\,\mathtt{M} \xRightarrow{\mathtt{coffee}} \overline{\mathtt{theorem}}\,.\,\mathtt{M}$ (by GMP)
Act (by $Monotonicity$)
$\overline{\mathtt{coffee}}\,.\,\mathtt{CVM} \xRightarrow{\overline{\mathtt{coffee}}} \mathtt{CVM}$ (by GMP)
$\mathtt{coffee}\,.\,\overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \overline{\mathtt{coffee}}\,.\,\mathtt{CVM} \xRightarrow{\tau} \overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \mathtt{CVM}$ (by GMP)
$(\mathtt{coffee}\,.\,\overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \overline{\mathtt{coffee}}\,.\,\mathtt{CVM}) \setminus K \xRightarrow{\tau} (\overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \mathtt{CVM}) \setminus K$ (by GMP)
$\mathtt{Institute} \xRightarrow{\tau^2} (\overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \mathtt{CVM}) \setminus K$ (by $Comp_I$)
$\mathtt{Institute} \xRightarrow{\tau^*} (\overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \mathtt{CVM}) \setminus K$ (by $Star_I$ for $n = 2$)
Res^* (by $Monotonicity$)
Act (by $Monotonicity$)
$\overline{\mathtt{theorem}}\,.\,\mathtt{M} \xRightarrow{\overline{\mathtt{theorem}}} \mathtt{M}$ (by GMP)
$\overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \mathtt{CVM} \xRightarrow{\overline{\mathtt{theorem}}} \mathtt{M} \mid \mathtt{CVM}$ (by M)
$(\overline{\mathtt{theorem}}\,.\,\mathtt{M} \mid \mathtt{CVM}) \setminus K \xRightarrow{\overline{\mathtt{theorem}}} (\mathtt{M} \mid \mathtt{CVM}) \setminus K$ (by GMP)
$\mathtt{Institute} \xRightarrow{\tau^* \,\S\, \overline{\mathtt{theorem}}} (\mathtt{M} \mid \mathtt{CVM}) \setminus K$ (by $Comp_I$)
$(\mathtt{M} \mid \mathtt{CVM}) \setminus K = \mathtt{Institute}$ (by R)
$(\mathtt{M} \mid \mathtt{CVM}) \setminus K \xRightarrow{\tau^*} \mathtt{Institute}$ (by $Star_I$ for $n = 0$)
$\mathtt{Institute} \xRightarrow{\tau^* \,\S\, \overline{\mathtt{theorem}} \,\S\, \tau^*} \mathtt{Institute}$ (by $Comp_I$)

Figure 1 Proof tree for the continuous output of theorems by the process `Institute`.

4 Forcing

In this section, we develop a forcing technique for proving completeness which extends in a non-straightforward way the classical forcing from one signature to a category of signatures.

▶ **Definition 17** (Forcing property). A forcing property *is a tuple* $\mathbb{P} = (P, \leq, \Delta, f)$*, where:*

$$\begin{tikzcd}
(P,\leq) \arrow[rr, bend left, "f"] \arrow[dr, bend right, "\Delta"'] & {} \arrow[d, Rightarrow, "\subseteq"] & \mathbb{S}\mathsf{et} \\
& \mathsf{Sig} \arrow[ur, bend right, "\mathsf{Sen}_b"'] &
\end{tikzcd}$$

1. $(P, \leq)$ *is a partially ordered set with a least element* 0.
   The elements of P *are traditionally called conditions.*
2. $\Delta : (P, \leq) \to \mathsf{Sig}$ *is a functor, which maps each arrow* $(p \leq q) \in (P, \leq)$ *to an inclusion* $\Delta_p \subseteq \Delta_q$.
3. $f : (P, \leq) \to \mathbb{S}\mathsf{et}$ *is a functor from the small category* $(P, \leq)$ *to the category of sets* $\mathbb{S}\mathsf{et}$ *such that* $f \subseteq \Delta ; \mathsf{Sen}_b$ *is a natural transformation, that is:* (a) $f(p) \subseteq \mathsf{Sen}_b(\Delta_p)$ *for all conditions* $p \in P$*, and* (b) $f(p) \subseteq f(q)$ *for all arrows* $(p \leq q) \in (P, \leq)$.
4. *If* $f(p) \models \phi$ *then* $\phi \in f(q)$ *for some* $q \geq p$*, for all atoms* $\phi \in \mathsf{Sen}_b(\Delta_p)$.

A classical forcing property is a particular case of forcing property such that $\Delta_p = \Delta_q$ for all conditions $p, q \in P$.

▶ **Example 18** (Syntactic forcing). Let Σ be a base signature and C an S-sorted set of new constants such that $\mathsf{card}(C_s) = \omega$ for all $s \in S$. Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property defined as follows:

- P is the set of presentations of the form $p = (\Delta_p, \Gamma_p)$, where (a) Δ_p is obtained from Σ by adding a finite set C_p of constants from C, and (b) $\Gamma_p \subseteq \mathsf{Sen}(\Delta_p)$ is consistent, that is, $\Gamma_p \nvdash_{\Delta_p} \bot$.
- $p \leq q$ iff $\Delta_p \subseteq \Delta_q$ and $\Gamma_p \subseteq \Gamma_q$, for all conditions $p, q \in P$.
- Δ is the forgetful functor which maps each condition $p \in P$ to Δ_p.
- $f(p) = \Gamma_p \cap \mathsf{Sen}_b(\Delta_p)$, for all conditions $p \in P$.

The syntactic forcing described in the example above is used to prove completeness. The constants from C are traditionally called Henkin constants and are used as witnesses for existentially quantified sentences obtained by extending an initial theory to a maximally consistent set of sentences.

As usual, forcing properties determine suitable relations between conditions and sentences.

▶ **Definition 19** (Forcing relation). *Let* $\mathbb{P} = \langle P, \leq, \Delta, f \rangle$ *be a forcing property.* The forcing relation $\Vdash$ *between conditions* $p \in P$ *and sentences from* $\mathsf{Sen}(\Delta_p)$ *is defined by induction on the structure of sentences, as follows:*

- $p \Vdash \varphi$ *if* $\varphi \in f(p)$*, for all atomic sentences* $\varphi \in \mathsf{Sen}_b(\Delta_p)$.
- $p \Vdash t_1 \overset{\mathfrak{a}_1 ; \mathfrak{a}_2}{\Longrightarrow} t_2$ *if* $p \Vdash t_1 \overset{\mathfrak{a}_1}{\Longrightarrow} t$ *and* $p \Vdash t \overset{\mathfrak{a}_2}{\Longrightarrow} t_2$ *for some* $t \in T_{\Delta_p}$.
- $p \Vdash t_1 \overset{\mathfrak{a}_1 \cup \mathfrak{a}_2}{\Longrightarrow} t_2$ *if* $p \Vdash t_1 \overset{\mathfrak{a}_1}{\Longrightarrow} t_2$ *or* $p \Vdash t_1 \overset{\mathfrak{a}_2}{\Longrightarrow} t_2$.
- $p \Vdash t_1 \overset{\mathfrak{a}^*}{\Longrightarrow} t_2$ *if* $p \Vdash t_1 \overset{\mathfrak{a}^n}{\Longrightarrow} t_2$ *for some natural number* $n \in \omega$.
- $p \Vdash \neg\phi$ *if there is no* $q \geq p$ *such that* $q \Vdash \phi$.
- $p \Vdash \vee\Phi$ *if* $p \Vdash \phi$ *for some* $\phi \in \Phi$.
- $p \Vdash \exists X \cdot \phi$ *if* $p \Vdash \theta(\phi)$ *for some substitution* $\theta : X \to T_{\Delta_p}$.

The relation $p \Vdash \phi$ *in* $\mathbb{P}$*, is read as* p *forces* ϕ*. We say that* p *weakly forces* ϕ*, in symbols,* $p \Vdash^w \phi$*, if* $p \Vdash \neg\neg\phi$.

A few basic properties of forcing are presented below.

▶ **Lemma 20** (Forcing properties). *Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property. For all conditions $p \in P$ and all sentences $\phi \in \mathsf{Sen}(\Delta_p)$ we have:*

1. *$p \Vdash \neg\neg\phi$ iff for each $q \geq p$ there is a condition $r \geq q$ such that $r \Vdash \phi$.*
2. *If $p \leq q$ and $p \Vdash \phi$ then $q \Vdash \phi$.*
3. *If $p \Vdash \phi$ then $p \Vdash \neg\neg\phi$.*
4. *We can not have both $p \Vdash \phi$ and $p \Vdash \neg\phi$.*

The second property stated in the above lemma shows that the forcing relation is preserved along inclusions of conditions. The fourth property shows that the forcing relation is consistent, that is, a condition cannot force all sentences. The remaining conditions are about negation.

▶ **Definition 21** (Generic set). *Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property. A subset of conditions $G \subseteq P$ is generic if*

1. *G is an ideal, that is: (a) for all $p \in G$ and all $q \leq p$ we have $q \in G$, and (b) for all $p, q \in G$ there exists $r \in G$ such that $p \leq r$ and $q \leq r$; and*
2. *for all conditions $p \in G$ and all sentences $\phi \in \mathsf{Sen}(\Delta_p)$ there exists a condition $q \in G$ such that $q \geq p$ and either $q \Vdash \phi$ or $q \Vdash \neg\phi$ holds.*

We write $G \Vdash \phi$ if $p \Vdash \phi$ for some $p \in G$.

A generic set G describes a reachable model which satisfies all sentences forced by the conditions in G.

▶ **Lemma 22** (Existence). *Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property. If any signature in $\{\Delta_p\}_{p \in P}$ is countable then every $p \in P$ belongs to a generic set.*

Proof sketch. Let pair : $\omega \times \omega \to \omega$ be a bijective function defined by $\mathrm{pair}(i, j) := \big((i + j)(i + j + 1) + 2j\big)/2$ for all $i, j \in \omega$. For all conditions $p \in P$, let $\psi_p : \omega \to \mathsf{Sen}(\Delta_p)$ be a bijective mapping, which gives an enumeration of $\mathsf{Sen}(\Delta_p)$. We define an increasing chain of conditions $p_0 \leq p_1 \leq \ldots$ in P recursively. Let $p = p_0$. For the induction step, we assume that we have defined p_n and we define p_{n+1}. Notice that there are unique natural numbers $i, j \in \omega$ such that $n = \mathrm{pair}(i, j)$ and $i, j \leq n$.

- If there is $q \geq p_n$ such that $q \Vdash \psi_{p_i}(j)$, then let $p_{n+1} := q$.
- Otherwise, $p_{n+1} := p_n$, which means that $p_{n+1} \Vdash \neg\psi_{p_i}(j)$.

Then $G := \{q \in P \mid q \leq p_n \text{ for some } n \in \omega\}$ is generic and contains p. ◀

Lemma 22 is the key for a modular approach to forcing and it is the equivalent of the Lindenbaum's lemma from Henkin's method for proving completeness.

▶ Remark 23. Since $\Delta : (G, \leq) \to \mathsf{Sig}$ is a directed diagram of signature inclusions, one can construct a co-limit $\mu : \Delta \Rightarrow \Delta_G$ of the functor $\Delta : (G, \leq) \to \mathsf{Sig}$ such that $\mu_p : \Delta_p \to \Delta_G$ is an inclusion for all $p \in G$.

The results which leads to completeness are developed over the signature Δ_G. If $\mathbb{P}$ is the syntactic forcing described in Example 18, then Δ_G is obtain from the base signature Σ by adding all Henkin constants from $\{\Delta_p\}_{p \in G}$. In general, Δ_G does not contain all Henkin constants from C, which is one of the major differences between the classical approach and the present developments.

▶ **Definition 24** (Generic model). *Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property and $G \subseteq P$ a generic set. A model $\mathfrak{A}$ defined over Δ_G is a* generic model *for G iff for every sentence $\phi \in \bigcup_{p \in G} \mathsf{Sen}(\Delta_p)$, we have $\mathfrak{A} \models \phi$ iff $G \Vdash \phi$.*

The notion of generic model is the semantic counterpart of the definition of generic set. The following result shows that every generic set has a generic model.

▶ **Theorem 25** (Generic Model Theorem). *Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property and $G \subseteq P$ a generic set. Then there is a generic model $\mathfrak{A}$ for G which is countable and reachable.*

Proof sketch. We define the set of all atomic sentences $B := \{\phi \in \mathsf{Sen}_b(\Delta_G) \mid G \Vdash \phi\}$ forced by the generic set G. The basic model $\mathfrak{A}^B$ given by Lemma 12 is the generic model for G. ◀

5 Completeness

The logical framework in which the results are developed in this section is the fragment TA^c obtained from TA by restricting the syntax to at most countable signatures. The syntactic forcing property defined in Example 18 is the starting point for proving completeness. Therefore, throughout this section, we let $\mathbb{P} = (P, \leq, \Delta, f)$ be a syntactic forcing property in TA^c as described in Example 18. In particular, all signatures in $\{\Delta\}_{p \in P}$ are at most countable. For the sake of simplicity, we write $p \vdash \phi$ iff $\Gamma_p \vdash_{\Delta_p} \phi$, for all conditions $p = (\Delta_p, \Gamma_p)$ in P.

▶ **Theorem 26.** *For all $p \in P$ and all $\phi \in \mathsf{Sen}(\Delta_p)$, we have $p \Vdash^w \phi$ iff $p \vdash \phi$.*

The above theorem says that a sentence is entailed by a condition if and only if it is weakly forced by that condition. In other words, the entailment relation is the weak forcing relation. Now, we can interpret Lemma 22 in the present context given by the syntactic forcing property $\mathbb{P}$ set above. The following result is a direct consequence of Theorem 26 and Lemma 22.

▶ **Corollary 27** (Lindenbaum's lemma). *Assume the following:*

- *a condition $p^\circ = (\Delta_{p^\circ}, \Gamma_{p^\circ})$ from P, and*
- *a generic set G which contains p° (by Lemma 22).*

Let Δ_G be the vertex of the co-limit $\mu : \Delta \to \Delta_G$ of the functor $\Delta : (G, \leq) \to \mathsf{Sig}$ defined in Remark 23. Then $\Gamma_G = \bigcup_{p \in G} \Gamma_p$ is a maximally consistent set which includes Γ_{p°.

The following example shows that Δ_G does not contain all Henkin constants defined for the base signature.

▶ **Example 28.** Let Σ be the signature defined by: $S := \{s_i \mid i \in \omega\}$, $F := \{c :\to s_0, d :\to s_0\}$, $M := \emptyset$, and $P := \{\lambda\}$. Let Γ be the set of sentences over Σ which consists of: (a) $c \overset{\lambda^*}{\Longrightarrow} d$, and (b) $(\exists x_n \cdot \top) \to \neg(c \overset{\lambda^n}{\Longrightarrow} d)$ for all $n \in \omega$, where x_n is a variable of sort s_n.

The first sentence says that there is a transition from c to d in a finite number of steps. For each natural number n, the sentence $(\exists x_n \cdot \top) \to \neg(c \overset{\lambda^n}{\Longrightarrow} d)$ says that if the sort s_n is not empty then there is no transition from c to d in exactly n steps. Recall that $C = \{C_{s_n}\}_{n \in \omega}$ is the set of all Henkin constants, and $\mathsf{card}(C_{s_n}) = \omega$ for all natural numbers n. Notice that $p^\circ = (\Sigma, \Gamma)$ is consistent, but $q = (\Sigma[C], \Gamma)$ is not consistent. By Corollary 27, one can extend Γ to a maximally consistent set of sentences Γ_G. Unlike in classical first-order logic, Γ_G does not contain all Henkin constants from C.

▶ **Theorem 29** (Downwards Löwenheim-Skolem Theorem). *For any consistent set of sentences Γ defined over a countable signature Σ, there exists a countable Σ-model $\mathfrak{A}$ that satisfies Γ.*

Proof. Let $\mathbb{P} = (P, \leq, \Delta, f)$ be the forcing property described in Definition 17. Notice that $p := (\Sigma, \Gamma)$ is a condition from P. Since all signatures are countable, by Lemma 22, p belongs to a generic set G. By Theorem 25, G has a generic model $\mathfrak{B}$ which is countable and reachable. In particular, $\mathfrak{B} \models_{\Delta_G} \Gamma$. Let $\mathfrak{A} := \mathfrak{B}\upharpoonright_{\Sigma}$, and by the satisfaction condition, $\mathfrak{A} \models_{\Sigma} \Gamma$. ◀

▶ **Theorem 30** (Completeness). *For all sets of sentences Γ and all sentences ϕ defined over a countable signature Σ, we have: $\Gamma \vdash_{\Sigma} \phi$ iff $\Gamma \models_{\Sigma} \phi$.*

Proof. The forward implication holds because all proof rules are sound. For the backwards implication assume $\Gamma \nvdash_{\Sigma} \phi$. We have $\Gamma \cup \{\neg\phi\} \nvdash_{\Sigma} \bot$. By Theorem 29, there is a countable Σ-model $\mathfrak{A}$ such that $\mathfrak{A} \models_{\Sigma} \Gamma \cup \{\neg\phi\}$. Therefore, $\Gamma \not\models_{\Sigma} \phi$. ◀

6 Conclusions

In this study, we have defined an extension of many-sorted first-order logic, called transition algebra, that offers explicit support for state transitions; furthermore, we have investigated its logical properties in order to apply the institutional model theory approach to new algebraic specification languages based on this logic, and with a greater expressivity than Maude and CafeOBJ. Transition algebra satisfies desirable properties such as truth invariance under change of signature, and has an expressive power that goes beyond that of ordinary first-order logic, which is important for formal-verification purposes. Our efforts have focused on two main aspects of transition algebra: first, on its formal-specification capabilities, i.e., to show that it forms a proper extension of first-order equational logic; and second, on support for formal verification, for which we have studied a number of model-theoretic properties, syntactic entailment and, most importantly, soundness and completeness results.

Concerning its formal-specification capabilities, transition algebra blends features of dynamic logic with features of many-sorted first-order logic. From the former, it borrows the idea of expressing the dynamics of a system by means of actions, which are built from atomic transitions using composition, iteration, and so on. The iteration of an action is a key feature because it allows us to express reachability, which is not possible in ordinary first-order logic. From the latter, our logic borrows term-building operators and quantifiers. This allows us to capture system states as terms, and hence to reason about the structure of states more freely and in a more complex manner than it is possible in dynamic logic.

For verification purposes, our contribution is twofold: on one hand, we have introduced a sound proof system for transition algebra; and on the other hand, we have developed a new general method for proving completeness based on forcing. The latter is highly important, because it has enabled us to circumvent the lack of compactness of transition algebra, which prevents the use of readily available methods for proving completeness. Moreover, it also overcomes a significant limitation of existing forcing techniques, namely their reliance on models with non-empty carriers, which is another basic property (like compactness) that does not hold for transition algebra. We have demonstrated the use of this extended forcing technique to show that the proof system for transition algebra is complete. We aim to further develop and apply this technique to extensions of transition algebra that take into account, for example, subsorting – to which we have already alluded in this paper. Furthermore, future research includes applying forcing to prove omitting types theorem for logical systems that interpret sorts as sets, possibly empty, thus upgrading the results from [17, 20]. Subsequently, the application of omitting types theorem to Robinson consistency property and interpolation, as demonstrated in [19], remains a feasible avenue for exploration.

References

1 Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella, and Andrzej Tarlecki. CASL: The common algebraic specification language. *Theoretical Computer Science*, 286(2):153–196, 2002. doi:10.1016/S0304-3975(01)00368-1.

2 Tomasz Borzyszkowski. Generalized interpolation in CASL. *Information Processing Letters*, 76(1-2):19–24, 2000. doi:10.1016/S0020-0190(00)00120-4.

3 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude – A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007. doi:10.1007/978-3-540-71999-1.

4 Mihai Codescu, Till Mossakowski, Adrián Riesco, and Christian Maeder. Integrating Maude into Hets. In Michael Johnson and Dusko Pavlovic, editors, *Algebraic Methodology and Software Technology - 13th International Conference, AMAST 2010, Lac-Beauport, QC, Canada, June 23-25, 2010. Revised Selected Papers*, volume 6486 of *LNCS*, pages 60–75. Springer, 2010. doi:10.1007/978-3-642-17796-5_4.

5 Paul J. Cohen. The independence of the continuum hypothesis. *Proceedings of the National Academy of Sciences of the United States of America*, 50(6):1143–1148, December 1963. doi:10.1073/pnas.50.6.1143.

6 Paul J. Cohen. The independence of the continuum hypothesis, II. *Proceedings of the National Academy of Sciences of the United States of America*, 51(1):105–110, January 1964. doi:10.1073/pnas.51.1.105.

7 William Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 1957. doi:10.2307/2963593.

8 Pierpaolo Degano, Fabio Gadducci, and Corrado Priami. A causal semantics for CCS via rewriting logic. *Theoretical Computer Science*, 275(1-2):259–282, 2002. doi:10.1016/S0304-3975(01)00165-7.

9 Răzvan Diaconescu. *Institution-Independent Model Theory.* Studies in Universal Logic. Birkhäuser, 2008.

10 Răzvan Diaconescu. Three decades of institution theory. In Jean-Yves Béziau, editor, *Universal Logic: An Anthology*, pages 309–322. Springer, 2012. doi:10.1007/978-3-0346-0145-0_25.

11 Răzvan Diaconescu and Kokichi Futatsugi. Logical foundations of CafeOBJ. *Theoretical Computer Science*, 285(2):289–318, 2002. doi:10.1016/S0304-3975(01)00361-9.

12 Steven Eker, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Alberto Verdejo. The Maude strategy language. *Journal of Logical and Algebraic Methods in Programming*, 134:100887, 2023. doi:10.1016/j.jlamp.2023.100887.

13 Joseph Goguen and José Meseguer. Completeness of many-sorted equational logic. *ACM SIGPLAN Notices*, 17(1):9–17, 1982. doi:10.1145/947886.947887.

14 Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992. doi:10.1016/0304-3975(92)90302-V.

15 Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. *Introducing OBJ*, pages 3–167. Springer, 2000. doi:10.1007/978-1-4757-6541-0_1.

16 Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specification and programming. *J. ACM*, 39(1):95–146, 1992. doi:10.1145/147508.147524.

17 Daniel Găină. Forcing, downward Löwenheim-Skolem and omitting types theorems, institutionally. *Logica Universalis*, 8(3-4):469–498, 2014. doi:10.1007/S11787-013-0090-0.

18 Daniel Găină. Forcing and calculi for hybrid logics. *Journal of the Association for Computing Machinery*, 67(4):1–55, 2020. doi:10.1145/3400294.

19 Daniel Găină, Guillermo Badia, and Tomasz Kowalski. Robinson consistency in many-sorted hybrid first-order logics. In David Fernández-Duque, Alessandra Palmigiano, and Sophie Pinchinat, editors, *Advances in Modal Logic, AiML 2022, Rennes, France, August 22-25, 2022*, pages 407–428. College Publications, 2022. URL: http://www.aiml.net/volumes/volume14/25-Gaina-Badia-Kowalski.pdf.

20 Daniel Găină, Guillermo Badia, and Tomasz Kowalski. Omitting types theorem in hybrid dynamic first-order logic with rigid symbols. *Annals of Pure and Applied Logic*, 174(3):103212, 2023. doi:10.1016/J.APAL.2022.103212.

21 Daniel Găină and Marius Petria. Completeness by forcing. *Journal of Logic and Computation*, 20(6):1165–1186, 2010. doi:10.1093/LOGCOM/EXQ012.

22 Daniel Găină and Andrei Popescu. An institution-independent proof of the Robinson consistency theorem. *Studia Logica*, 85(1):41–73, 2007. doi:10.1007/S11225-007-9022-4.

23 Go Hashimoto, Daniel Găină, and Ionuţ Ţuţu. Forcing, transition algebras, and calculi (extended version), 2024. arXiv:2404.16111.

24 Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. In José Meseguer, editor, *First International Workshop on Rewriting Logic and its Applications, RWLW 1996, Asilomar Conference Center, Pacific Grove, CA, USA, September 3-6, 1996*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 190–225. Elsevier, 1996. doi:10.1016/S1571-0661(04)00040-4.

25 José Meseguer. Conditional rewriting logic: Deduction, models and concurrency. In Stéphane Kaplan and Mitsuhiro Okada, editors, *Conditional and Typed Rewriting Systems, 2nd International CTRS Workshop, Montreal, Canada, June 11-14, 1990, Proceedings*, volume 516 of *LNCS*, pages 64–91. Springer, 1990. doi:10.1007/3-540-54317-1_81.

26 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980. doi:10.1007/3-540-10235-3.

27 Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

28 J. Donald Monk. *Mathematical Logic*, volume 37 of *Graduate Texts in Mathematics*. Springer-Verlag New York, 1976.

29 Marius Petria. An institutional version of Gödel's completeness theorem. In Till Mossakowski, Ugo Montanari, and Magne Haveraaen, editors, *Algebra and Coalgebra in Computer Science, Second International Conference, CALCO 2007, Bergen, Norway, August 20-24, 2007, Proceedings*, volume 4624 of *LNCS*, pages 409–424. Springer, 2007. doi:10.1007/978-3-540-73859-6_28.

30 Abraham Robinson. Forcing in model theory. *Symposia Mathematica*, 5:69–82, 1971.

31 Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012. doi:10.1007/978-3-642-17336-3.

32 Bruce Schechter. *My Brain is Open: The Mathematical Journeys of Paul Erdös*. Simon & Schuster, 2000.

33 Andrzej Tarlecki. Bits and pieces of the theory of institutions. In Klaus Drosten, Hans-Dieter Ehrich, Martin Gogolla, and Udo W. Lipeck, editors, *Proceedings of the 4st Workshop on Abstract Data Type, 1986. University of Braunschweig, Germany*, 1986.

34 Alberto Verdejo and Narciso Martí-Oliet. Implementing CCS in Maude 2. In Fabio Gadducci and Ugo Montanari, editors, *Fourth International Workshop on Rewriting logic and Its Applications, WRLA2002, Pisa, Italy, 19-21, 2002*, volume 71 of *Electronic Notes in Theoretical Computer Science*, pages 282–300. Elsevier, 2002. doi:10.1016/S1571-0661(05)82540-X.