Computer Science Journal of Moldova, vol.**24**, no.3(72), 2016

Propositional inquisitive logic: a survey[*]

Ivano Ciardelli

Abstract

This paper provides a concise survey of a body of recent work on propositional inquisitive logic. We review the conceptual foundations of inquisitive semantics, introduce the propositional system, discuss its relations with classical, intuitionistic, and dependence logic, and describe an important feature of inquisitive proofs.

Keywords: questions, inquisitive logic, dependency, intermediate logics, proofs-as-programs.

1 Introduction

Inquisitive semantics stems from a line of work which, going back to [12], has aimed at providing a uniform semantic foundation for the interpretation of both statements and questions. The approach was developed in an early version, based on pairs of models, in [13], [16]; it reached the present form, based on information states, in [3], [9], where the associated propositional logic was also investigated. An algebraic underpinning for the inquisitive treatment of logical operators was given in [19]. The foundations of the inquisitive approach have been motivated starting from a language-oriented perspective in [11], and starting from logic-oriented perspective in [7], [8].

The aim of this paper is to provide a short survey of the work done on propositional inquisitive logic, drawing mostly on [3],[6],[8],[9]. More precise pointers to the literature will be provided when discussing specific topics. We will start in Section 2 by showing at a general level how questions can be brought within the scope of logic by means of a simple

[*] Financial support from the Netherlands Organization for Scientific Research (NWO) is gratefully acknowledged.

but fundamental shift in the way semantics is viewed. In Section 3, we instantiate this general approach in the propositional setting, introducing propositional inquisitive logic. In Sections 4, 5, and 6, we examine the connections of this logic to the propositional versions of classical logic, intuitionistic logic, and dependence logic. In Section 7 we discuss inquisitive proofs and their constructive content. In Section 8, we present an extension and a generalization of propositional inquisitive logic. Section 9 wraps up and concludes.

2 Bringing question into the logical landscape

Traditionally, logical entailment captures relations such as the one exemplified by (1): the information that Alice and Bob live in the same city, combined with the information that Alice lives in Amsterdam, yields the information that Bob lives in Amsterdam.

(1) Alice and Bob live in the same city
Alice lives in Amsterdam
―――――――――――――――
Bob lives in Amsterdam

Inquisitive logic brings questions into this standard picture, broadening the notion of entailment so as to encompass patterns which we might write as in (2): the information that Alice and Bob live in the same city, combined with the information on where Alice lives, yields the information on where Bob lives.

(2) Alice and Bob live in the same city
Where Alice lives
―――――――――――――――
Where Bob lives

Notice the crucial difference between the two examples: in (1) we are concerned with a relation holding between three specific pieces of information. The situation is different in (2): given the information that

Alice and Bob live in the same city, *any* given piece of information on Alice's city of residence yields some corresponding information on Bob's city of residence. We may say that what is at play in (2) are two *types* of information, which we may see as labeled by the questions *where Alice lives* and *where Bob lives*. Entailment captures the fact that, given the assumption that Alice and Bob live in the same city, information of the first type yields information of the second type.

The entrance of questions into the logical arena is made possible by a fundamental shift in the way the semantics of a sentence is construed. In classical logic, the meaning of a sentence is given by laying out in what states of affairs the sentence is *true*; however, this truth-conditional view does not seem suitable in the case of questions. In inquisitive logic, by contrast, the meaning of a sentence is given by laying out what information is needed in order for a sentence to be *supported*. Accordingly, sentences are evaluated relative to objects called *information states*, which formally encode bodies of information.

Unlike truth-conditional approach, the support approach is applicable to both statements and questions. To give concrete examples, a statement like (3-a) is supported by an information state s if the information available in s implies that Alice lives in Amsterdam; on the other hand, a question like (3-b) is supported by an information state s if the information available in s determines where Alice lives.

(3) a. Alice lives in Amsterdam.
b. Where does Alice live?

This more general semantic approach comes with a corresponding notion of entailment, understood as preservation of support: an entailment holds if the conclusion is supported whenever all the premises are. Assuming a natural connection between the truth-conditions of a statement and its support conditions—namely, that a state supports a statement iff it implies that the statement is true—this notion of entailment coincides with the truth-conditional one as far as statements are concerned. The novelty, however, lies in the fact that now, questions can also participate in entailment relations. Thus, for example, we can indeed capture the pattern in (2) as a case of logical entailment. To see this, suppose an information state s supports the premises of (2): this means that the information available in s implies that Alice and Bob live in the same city, and also determines in which city Alice lives; clearly, then, the information available in the state determines in which city Bob lives, which means that the conclusion of (2) is supported.

The one discussed in this section is a very general approach to logic, which can be instantiated by a range of concrete systems, differing with respect to their logical language and to the relevant notion of information states. Just as for classical logic, we have inquisitive logics of different sorts: propositional, modal, first-order, etc. The remaining sections of the paper provide an overview of the results obtained in the most basic and best understood setting—the propositional one.[1,2]

3 Propositional inquisitive logic

The language of propositional inquisitive logic, InqB, is the propositional language built up from a set of atomic sentences and $\bot$ by means of conjunction, $\wedge$, implication, $\to$, and inquisitive disjunction, $\mathbin{\rotatebox[origin=c]{-90}{$\geqslant$}}$.

$$\phi \ ::= \ p \mid \bot \mid \phi \wedge \phi \mid \phi \to \phi \mid \phi \mathbin{\rotatebox[origin=c]{-90}{$\geqslant$}} \phi$$

Negation and classical disjunction are defined by setting $\neg\phi := \phi \to \bot$, and $\phi \vee \psi := \neg(\neg\phi \wedge \neg\psi)$. Formulas that contain no occurrence of $\mathbin{\rotatebox[origin=c]{-90}{$\geqslant$}}$ are called *classical* formulas.

In the propositional setting, an information state is construed as a set of propositional valuations. The idea here is that a set s encodes the information that the actual state of affairs corresponds to one of the valuations in s. This means that if $t \subseteq s$, then t contains at least as much information as s, and possibly more.

The clauses defining the relation of *support* relative to an information state are the following ones:

[1]For discussion on the semantic foundations of the inquisitive approach, on the role of questions in logic, and on the relation between truth and support, see [6], [8].

[2]The research in inquisitive modal logic and inquisitive first-order logic has also been growing rapidly in these last few years. Recent work includes [4], [6], [10], [22].

- $s \models p \iff w(p) = 1$ for all $w \in s$
- $s \models \bot \iff s = \emptyset$
- $s \models \phi \wedge \psi \iff s \models \phi$ and $s \models \psi$
- $s \models \phi \vee\!\!\!\vee \psi \iff s \models \phi$ or $s \models \psi$
- $s \models \phi \rightarrow \psi \iff \forall t \subseteq s : t \models \phi$ implies $t \models \psi$.

A key feature of the semantics is *persistency*: if ϕ is supported by an information state s, then it is also supported by any state $t \subseteq s$ which contains at least as much information. This means that as information grows, more and more formulas become supported. In the information state $\emptyset$, which represents the state of *inconsistent* information, every formula is supported. This may be regarded as a semantic analogue of the *ex falso quodlibet* principle.

4 Relations with classical logic

In inquisitive logic, the fundamental semantic notion is that of support relative to an information state. However, the notion of *truth* relative to a particular valuation w can be recovered by setting: $w \models \phi \iff \{w\} \models \phi$. It is then easy to check that all classical formulas receive the standard truth-conditions.

For some formulas, support at a state simply amounts to truth at each world in the state. If this is the case, we say that the formula is *truth-conditional*. More formally, ϕ is truth-conditional in case for all states s: $s \models \phi \iff \forall w \in s, w \models \phi$. We regard truth-conditional formulas as corresponding to *statements*. The intuition is that there is only one way for an information state s to support a statement: the information available in s must imply that the statement is true.

As a matter of fact, large classes of formulas in InqB are truth-conditional. In particular, all classical formulas are.

Proposition 1. All classical formulas are truth-conditional.

This means that all classical formulas receive essentially the same treatment as in classical propositional logic: their semantics is fully determined by their truth-conditions, which in turn are the standard ones. This is reflected by the relation of entailment among these formulas.

Proposition 2 (Conservativity over classical logic)**.**
Entailment restricted to classical formulas coincides with entailment in classical propositional logic.

This means that the classical fragment of InqB can be identified for all intents and purposes with classical propositional logic, and our logic may be regarded as a conservative extension of classical propositional logic with an inquisitive disjunction operator.

Formulas formed by means of inquisitive disjunction are typically not truth-conditional. We take such formulas to correspond to *questions*. For instance, the formula $p \vee\!\!\!\vee \neg p$, abbreviated as $?p$, corresponds to the question *whether p or not p*. An information state can support this formula in two different ways: either by implying that p is true, or by implying that p is false. Similarly, the formula $p \vee\!\!\!\vee q$ can be regarded as encoding the question *whether p or q*, which can be supported either by establishing that p is true, or by establishing that q is true.[3]

Like in classical logic, formulas in inquisitive logic can be written in a very constrained normal form: namely, any formula of InqB can be written as an inquisitive disjunction of classical formulas.

Theorem 1 (Inquisitive normal form)**.**
Recursively on ϕ, we can define a set $\mathcal{R}(\phi) = \{\alpha_1, \dots, \alpha_n\}$ of classical formulas, called the *resolutions* of ϕ, such that $\phi \equiv \alpha_1 \vee\!\!\!\vee \dots \vee\!\!\!\vee \alpha_n$.

Intuitively, we can regard the resolutions of a formula as capturing the different ways in which the formula may be supported. If ϕ is a classical formula, then it can be supported in only one way, by establishing that it is true; accordingly, we have $\mathcal{R}(\phi) = \{\phi\}$. On the other hand, if ϕ stands for a question, there will be multiple ways of

[3]An exclusive reading of the question *whether p or q* can be formalized as well, by translating the question as $(p \wedge \neg q) \vee\!\!\!\vee (q \wedge \neg p)$.

supporting the formula, and thus multiple resolutions; for instance, we have $\mathcal{R}(?p) = \{p, \neg p\}$, and $\mathcal{R}(p \vee\!\!\!\vee\, q) = \{p, q\}$. Any formula in InqB can thus be construed as offering a (possibly trivial) choice among classical formulas.

We saw that entailments among classical formulas amount to entailments in classical logic. On the other hand, we saw that our language also includes formulas which can be regarded as questions. Instances of entailment which involve such formulas capture interesting logical relations that lack a counterpart in classical logic; notably, entailments involving both question assumptions and question conclusions capture relations of *logical dependency* among these questions, possibly within the context of certain statements. For instance, the following entailment captures the fact that, given the information that $r \leftrightarrow p \wedge q$, the question $?r$ is completely determined by the questions $?p$ and $?q$.

$$r \leftrightarrow p \wedge q,\ ?p,\ ?q\ \models ?r.$$

Summing up, then, propositional inquisitive logic can be regarded as a conservative extension of classical propositional logic with an inquisitive disjunction: while the classical fragment of the language coincides with classical logic, by means of the operator $\vee\!\!\!\vee$ we can build formulas which express propositional questions, and capture dependencies among such questions as special cases of the relation of entailment.[4]

5 Relations with intuitionistic logic

In the previous section, we saw that InqB can be viewed as a conservative extension of classical logic if $\vee\!\!\!\vee$ is regarded as an additional, non-standard connective. In this section we will show that if, on the other hand, we regard $\vee\!\!\!\vee$ as the standard disjunction of the system, then InqB turns out to be a special kind of *intermediate* logic, i.e., a logic sitting in between intuitionistic and classical logic.

The first step in this direction is to notice that our semantics can be regarded as a case of intuitionistic Kripke semantics on a particular

[4]For more on the relations between inquisitive logic and classical logic, see [6], [8].

Kripke model, having consistent information states as its elements, the relation $\supseteq$ as accessibility relation, and the valuation function $V(p) = \{s \mid w(p) = 1 \text{ for all } w \in s\}$. Since Kripke semantics is sound for intuitionistic logic, this implies that anything that can be falsified in inquisitive logic can be falsified in intuitionistic propositional logic, IPL. On the other hand, it is easy to see that singleton information states $\{w\}$ behave just like the corresponding propositional valuation w: this ensures that anything that can be falsified in classical logic, CPL, can also be falsified in inquisitive logic. If we identify a logic with the corresponding set of validities, we can sum up our findings as follows.

Proposition 3. $\mathsf{IPL} \subseteq \mathsf{InqB} \subseteq \mathsf{CPL}$

Thus, from this perspective InqB is a logic stronger than intuitionistic logic, but weaker than classical logic. It is not, however, an *intermediate logic* in the usual sense of the term. This is because InqB is not closed under the rule of *uniform substitution*: in particular, the double negation law is valid for propositional atoms, but invalid when atoms are replaced by questions: $\neg\neg p \to p \in \mathsf{InqB}$, but $\neg\neg ?p \to ?p \notin \mathsf{InqB}$. The conceptual point here is that atoms in InqB are not intended as placeholders for arbitrary sentences, but only placeholders for arbitrary *statements*. As we saw, statements are truth-conditional, and as such they validate the double negation law, which is not generally valid. It is worth emphasizing that this is not an accident, but a deliberate architectural choice (see pp. 66-67 of [6]). This choice (i) enables InqB to retain a classical fragment, which encodes the underlying logic of statements; (ii) allows for a recursive decomposition of questions into resolutions; and (iii) makes a neat proof system possible.

Besides this classical feature of atoms, inquisitive logic differs from intuitionistic logic in that the space of information states has a special structure, which renders valid some non-intuitionistic principles. The best known of these is the Kreisel-Putnam scheme, first studied in [14]:

$$(\mathsf{KP}) \qquad (\neg\phi \to \psi \mathbin{\vee\!\!\!\vee} \chi) \to (\neg\phi \to \psi) \mathbin{\vee\!\!\!\vee} (\neg\phi \to \chi)$$

While this principle may look mysterious at first, it can be shown (see p. 80 of [6]) to encode a fundamental relation between statements and questions: a statement only counts as resolving a question if it entails a specific resolution to the question.

As shown in [9], the classicality of atoms and the validity of the KP scheme, together with the underlying intuitionistic base, suffice to characterize inquisitive propositional logic completely. More formally, InqB can be characterized as the set of formulas obtained by extending IPL with all instances of KP and with $\neg\neg p \to p$ for all atoms p, and closing the resulting set under *modus ponens*.

Theorem 2. $\mathsf{InqB} = \mathsf{IPL} + \mathsf{KP} + \neg\neg p \to p$

In fact, besides the Kreisel-Putnam logic axiomatized by the scheme KP, there is a whole range of intermediate logics which, when extended with classical atoms, yield inquisitive logic: as shown in [9], this range consists exactly of those intermediate logics which include Maksimova's logic [15] and are included in Medvedev's logic of finite problems [17], [18]. In particular, Medvedev's logic is the largest standard intermediate logic included in InqB.

An important aspect of the relation between inquisitive logic and intuitionistic logic can be observed based on the normal form result given by Theorem 1. This result guarantees that any formula can be written as an inquisitive disjunction of classical formulas. Since classical formulas behave as in classical logic, they are logically equivalent to their own double negation. Thus, it follows that in InqB, any formula ϕ is equivalent to an inquisitive disjunction of negations $\phi^{\mathsf{DNT}} = \neg\psi_1 \mathbin{\vee\!\!\!\vee} \ldots \mathbin{\vee\!\!\!\vee} \neg\psi_n$. Now, the following theorem shows that the map $(\cdot)^{\mathsf{DNT}}$ is a translation of inquisitive logic into intuitionistic logic.

Theorem 3. $\Phi \models \psi \iff \Phi^{\mathsf{DNT}} \models_{\mathsf{IPL}} \psi^{\mathsf{DNT}}$

This result can be extended to show that the Lindenbaum-Tarski algebra for InqB is isomorphic to the sub-algebra of the Lindenbaum-Tarski algebra for IPL consisting of equivalence classes of disjunctions of negations. Thus, while classical propositional logic can be regarded as the negative fragment of intuitionistic logic, propositional inquisitive logic can be regarded as the *disjunctive-negative fragment* of intuitionistic logic—the fragment consisting of disjunctions of negations.[5]

6 Relations with dependence logic

We mentioned above that in inquisitive logic, entailments involving questions capture logical dependencies. The relation of dependency is also the focus of recent work in the framework of *dependence logic* [23]. Dependence logic and inquisitive logic are tightly connected frameworks, as discussed in detail in [7]. In the propositional setting, full translations are possible between the two [25]. In both propositional systems, formulas are interpreted relative to sets of assignments; while propositional inquisitive logic enriches classical propositional logic with questions, propositional dependence logic enriches it with formulas called *dependence atoms*, written $=\!(p_1,\dots,p_n,q)$, which capture the fact that the truth-value of an atomic proposition q is determined by the truth-values of other atomic propositions $p_1,\dots,p_n$. The semantics of these atoms is given by the following clause:

$$s \models {=}(p_1,\dots,p_n,q) \iff \forall w,w'\in s: \text{ if } w(p_i)=w'(p_i) \text{ for all } i, \text{ then } w(q)=w'(q)$$

It is easy to check that such a dependence atom can be expressed in InqB by means of the formula $?p_1 \wedge \cdots \wedge ?p_n \to ?q$. This is not an accident: as shown in [7], in inquisitive logic, the fact that a question ν is fully determined by questions $\mu_1,\dots,\mu_n$ is generally captured by the implication $\mu_1\wedge\cdots\wedge\mu_n \to \nu$. More precisely, the formula $\mu_1\wedge\cdots\wedge\mu_n \to \nu$ is supported at a state s in case relative to s, any way of resolving the questions $\mu_1,\dots,\mu_n$ determines a corresponding way to resolve the question ν. What a dependence atom expresses is that the question $?q$ is determined by the questions $?p_1,\dots,?p_n$, hence the representation $?p_1 \wedge \cdots \wedge ?p_n \to ?q$.

Realizing that dependencies can be captured generally as implications between questions is interesting for various reasons. The first kind

[5]For more on the relations between propositional inquisitive logic, intuitionistic logic, and intermediate logics, see [3] and [9].

of reason is proof-theoretic: in inquisitive logic, all the connectives, including those involved in a dependence formula, can be handled by essentially standard inference rules. Thus, for instance, a dependency $?p \to ?q$ may be formally proved to hold by assuming the question $?p$ and trying to conclude the question $?q$. In fact, this perspective brings out the fact that the *Armstrong axioms* for functional dependency [2] used in database theory are essentially nothing but the axioms of implication in disguise—a fact that was first noted in [1].

Moreover, realizing that dependencies can be generally captured as implications between questions allows us to see that dependence atoms are a particular case of a more general pattern. Not just for atomic polar questions of the form $?p$, but for all sorts of questions $\mu_1, \ldots, \mu_n, \nu$ expressible in the system—in fact, in *any* inquisitive system—the fact that ν is determined by $\mu_1, \ldots, \mu_n$ is expressed by $\mu_1, \ldots, \mu_n \to \nu$.

Finally, realizing that dependencies can be expressed as implications among questions allows us to use inquisitive logics to investigate the logical properties of the notion of dependency. For example, consider the valid entailment $?p,\ ?p \wedge ?q \to ?r \models ?q \to ?r$. This captures the fact that given the information whether p, from a dependency of $?r$ on both $?p$ and $?q$ we can always compute a dependency of $?r$ on $?q$. If we think of a dependency as encoded by a function (cf. the notion of *dependence function* in §2 of [6]), this amounts to the fact that we can saturate one of the arguments of this function.[6]

7 Questions in proofs

An important feature of inquisitive logic is that it shows that questions can meaningfully be manipulated in logical inferences, and that their logical behavior is in fact rather familiar. In the propositional setting, a natural deduction system for inquisitive logic is obtained by extending a system for intuitionistic logic with the following two inference rules, where α ranges over classical formulas, and ϕ, ψ over arbitrary formulas.

[6]For more on the relations between inquisitive and dependence logic, see [6], [7], [26].

$$\frac{\alpha \to (\phi \mathbin{\vee\!\!\!\vee} \psi)}{(\alpha \to \phi) \mathbin{\vee\!\!\!\vee} (\alpha \to \psi)}\ (\mathsf{split}) \qquad\qquad \frac{\neg\neg\alpha}{\alpha}\ (\mathsf{dne})$$

The second of these rules captures the fact that classical formulas are truth-conditional, and thus behave exactly as in classical logic. The first—related to the Kreisel-Putnam scheme discussed above—captures the interaction among statements and questions, stipulating that if a statement resolves a question, it must do so by yielding a particular resolution to it. The completeness of this system for InqB, proved in [6], implies in particular that any valid propositional dependency can be formally proved by making inferences with propositional questions in this system. Thus, questions are interesting proof-theoretic tools: by making inferences with them, we can establish the existence of certain logical dependencies. Moreover, the following theorem, proved in [6], shows that a proof of a dependency does not just *witness* that the dependency holds, but actually encodes a method for computing it.

Theorem 4 (Constructive content of inquisitive proofs)**.**
Suppose P is a natural deduction proof having assumptions $\phi_1, \dots, \phi_n$ and conclusion ψ. Recursively on P, we can define a procedure f_P which, when given as input resolutions $\alpha_1, \dots, \alpha_n$ of the assumptions, outputs a resolution $f_P(\alpha_1, \dots, \alpha_n)$ of the conclusion with the property that $\alpha_1, \dots, \alpha_n \models f_P(\alpha_1, \dots, \alpha_n)$.

What this theorem shows is that proofs in inquisitive logic have a specific kind of constructive content: they encode methods for turning any given resolutions of the question assumptions into a resolution of the conclusion which is determined by them. This is reminiscent of the proofs-as-programs interpretation of intuitionstic logic, and it shows once more that, while our logic coincides with classical logic on statements, encoded by classical formulas, the logic of questions has a constructive flavor to it.[7]

[7]For more on the role of questions in inference and on the constructive content of inquisitive proofs, see [5], [6], [8].

8 Extensions and generalization

In the last couple of years, the work on propositional inquisitive logic presented in the previous sections has been extended in several directions. First of all, it has been taken as the basis for logics that go beyond the propositional realm, such as the modal logics given in [4], [6], [10], and the first-order logics given in [6], [7]. Presenting these richer logics goes beyond the scope of the present survey. However, in this section I want to briefly discuss an extension and a generalization of InqB, both due to Vít Punčochář, that remain within the domain of propositional logic.

First, the system InqB is extended in [20] with a *weak negation* connective, denoted $\sim$, which allows us to express the fact that a certain formula fails to be supported at the evaluation state.

$$s \models \sim\phi \iff s \not\models \phi$$

Evidently, the addition of this connective results in a system in which support is no longer persistent: a formula $\sim\phi$ may be supported by a state s, yet it may fail to be supported by a stronger state $t \subseteq s$. One reason why such a system is interesting is that—while remaining within the propositional inquisitive setting—it allows for the definition of formulas $\Diamond\phi$ which express the fact that the state of evaluation can be extended consistently to support ϕ. Interestingly, this logic is axiomatized by means of a proof system which allows for two different modes of hypothetical proofs. In one mode, making the assumption ϕ corresponds to supposing that the current information state supports ϕ. In the other mode, it corresponds to supposing that the current information state is extended so as to support ϕ. In this second mode, only some formulas from outside the hypothetical context can be appealed to when reasoning within the hypothetical context.

A generalization of propositional inquisitive logic is explored in [21]. This paper defines an operation G which, given a logic Λ with IPL $\subseteq \Lambda \subseteq$ CPL, returns a corresponding logic G(Λ), called the *global variant* of Λ. Logics of the form G(Λ) are called G-logics.[8] Intuitively,

[8]Here, the logic Λ is assumed to be closed under *modus ponens*, but not neces-

G(Λ) is a logic obtained by extending the $\mathbin{\vee\!\!\!\vee}$-free fragment of Λ with an inquisitive disjunction connective. In Section 4, we saw that InqB can be seen as arising from extending classical logic with inquisitive disjunction. And indeed, we have $\mathsf{InqB} = \mathrm{G}(\mathsf{CPL})$, which means that inquisitive logic is the greatest of all G-logics. The smallest G-logic, G(IPL), is the logic $\mathsf{IPL}{+}\mathsf{H}$ axiomatized by extending intuitionistic logic with the following scheme, where ϕ, ψ range over arbitrary formulas, and α ranges over Harrop formulas, and closing under *modus ponens*.[9]

$$(\mathsf{H}) \qquad (\alpha \to \phi \mathbin{\vee\!\!\!\vee} \psi) \to (\alpha \to \phi) \mathbin{\vee\!\!\!\vee} (\alpha \to \psi)$$

All other G-logics fall in between $\mathsf{IPL} + \mathsf{H}$ and InqB, and share many of the core features of inquisitive logic. All of them have the disjunction property, meaning that a disjunction $\phi \mathbin{\vee\!\!\!\vee} \psi$ can only be valid if either ϕ or ψ is valid. None of them is closed under uniform substitution. All of them coincide with the base logic Λ in their $\mathbin{\vee\!\!\!\vee}$-free fragment, and allow for an analogue of Theorem 1, stating that any formula is equivalent to a disjunction $\alpha_1 \mathbin{\vee\!\!\!\vee} \ldots \mathbin{\vee\!\!\!\vee} \alpha_n$ of classical formulas. Finally, all G-logics can be characterized axiomatically in a uniform way: G(Λ) amounts to the logic obtained by extending intuitionistic logic with the scheme H and all $\mathbin{\vee\!\!\!\vee}$-free formulas which are valid in Λ, and closing this set under *modus ponens*.

9 Conclusion

In this paper I have tried to give a bird's eye view of propositional inquisitive logic, including its conceptual underpinnings, its main mathematical features, and its relations to other logics. My hope is that this survey, together with the pointers scattered through the paper, will provide a valuable guide to the growing literature on the subject.

sarily under uniform substitution.

[9]A Harrop formula is defined as a formula in which disjunction is only allowed to occur within the antecedent of an implication.

References

[1] S. Abramsky, and J. Väänänen, "From IF to BI," *Synthese*, vol. 167, no. 2, pp. 207–230, 2009.

[2] W. Armstrong, "Dependency structures of data base relationships," In *Proceedings of the IFIP Congress*, J. L. Rosenfeld, Ed. North-Holland, 1974, pp. 580–583.

[3] I. Ciardelli, "Inquisitive semantics and intermediate logics," MSc Thesis, University of Amsterdam, 2009.

[4] I. Ciardelli, "Modalities in the realm of questions: axiomatizing inquisitive epistemic logic," in *Advances in Modal Logic 10*, R. Goré, B. Kooi, A. Kurucz, Eds. College Publications, 2014, pp. 94–113.

[5] I. Ciardelli, "Interrogative dependencies and the constructive content of inquisitive proofs," In *Logic, Language, Information and Computation. Proceedings of WoLLIC 2014*, ser. Lecture Notes in Computer Science, U. Kohlenbach, P. Barceló, and R. de Queiroz, Eds. Springer, 2014 pp. 109–123.

[6] I. Ciardelli, "Questions in Logic," PhD Thesis, University of Amsterdam, 2016.

[7] I. Ciardelli, "Dependency as Question Entailment," in *Dependence Logic: theory and applications*, S. Abramsky, J. Kontinen, J. Väänänen and H. Vollmer, Eds. Springer, 2016, pp. 129–181.

[8] I. Ciardelli, "Questions as Information Types," *Synthese*, to be published.

[9] I. Ciardelli, and F. Roelofsen, "Inquisitive Logic," *Journal of Philosophical Logic*, vol. 40, no. 1, pp. 55–94, 2011.

[10] I. Ciardelli, and F. Roelofsen, "Inquisitive Dynamic Epistemic Logic," *Synthese*, vol. 192, no. 6, pp. 1643–1687, 2015.

[11] I. Ciardelli, J. Groenendijk, and F. Roelofsen, "Inquisitive semantics: a new notion of meaning," *Language and Linguistics Compass*, vol. 7, no. 9, pp. 459–476, 2013.

[12] J. Groenendijk, "The logic of interrogation," In *Semantics and Linguistic Theory*, T. Matthews and D. Strolovitch, Eds. 1999, pp. 109–126.

[13] J. Groenendijk, "Inquisitive semantics: Two possibilities for disjunction," In *Seventh International Tbilisi Symposium on Language, Logic, and Computation*, P. Bosch, D. Gabelaia, and J. Lang, Eds. 2009, pp. 80–94.

[14] G. Kreisel, and H. Putnam, "Eine Unableitbarkeitsbeweismethode für den intuitionistischen Aussagenkalkül," *Archiv für Mathematische Logik und Grundlagenforschung*, vol. 3, pp. 74–78, 1957.

[15] L. Maksimova, "On maximal intermediate logics with the disjunction property," *Studia Logica*, vol. 45, pp. 69–75, 1986.

[16] S. Mascarenhas, "Inquisitive semantics and logic," MSc Thesis, University of Amsterdam, 2009.

[17] J. T. Medvedev, "Finite problems," *Soviet Mathematics Doklady*, vol. 3, pp. 227–230, 1962.

[18] J. T. Medvedev, "Interpretation of logical formulas by means of finite problems," *Soviet Mathematics Doklady*, vol. 7, pp. 857–860, 1966.

[19] F. Roelofsen, "Algebraic foundations for the semantic treatment of inquisitive content," *Synthese*, vol. 190, no. 1, pp. 79–102, 2013.

[20] V. Punčochář, "Weak negation in inquisitive semantics," *Journal of Logic, Language and Information*, vol. 23, pp. 47–59, 2015.

[21] V. Punčochář, "A generalization of inquisitive semantics," *Journal of Philosophical Logic*, 2015. [Online]. Available: DOI:10.1007/s10992-015-9379-1.

[22] V. Punčochář, “A new semantic framework for modal logic,” *Philosophical Alternatives*, vol. 23, pp. 47–59, 2014.

[23] J. Väänänen, *Dependence Logic: A New Approach to Independence Friendly Logic*, Cambridge University Press, 2007.

[24] J. Väänänen, “Modal dependence logic,” In *New Perspectives on Games and Interaction*, K. Apt and R. van Rooij, Eds. Amsterdam University Press, 2008.

[25] F. Yang, “On extensions and variants of dependence logic: a study of intuitionistic connectives in the team semantics setting,” PhD thesis, University of Helsinki, 2014.

[26] F. Yang, and J. Väänänen, “Propositional logics of dependence,” *Annals of Pure and Applied Logic*, vol. 167, no. 7, pp. 557–589, 2016.

Ivano Ciardelli

Received May 27, 2016

ILLC, University of Amsterdam
E–mail: `i.a.ciardelli@uva.nl`

Computer Science Journal of Moldova, vol.**24**, no.3(72), 2016

"The absence of the difference from a pot is potness" – Axiomatic Proofs of Theorems Concerning Negative Properties in Navya-Nyāya

Eberhard Guhe

Abstract

The present paper deals with an aspect of the Navya-Nyāya "logic of property and location" (Matilal) in classical Indian philosophy, namely the so-called "absences" (*abhāva*). Following George Bealer (*Quality and Concept*, Oxford 1982) we may regard these negative properties as the result of applying certain algebraic operations to property terms, which Bealer names after their corresponding propositional or first-order operations ("negation of a property", "conjunction of properties", "existential generalization of a property" etc.). Bealer introduces these operations in his property theories in order to explain how the denotation of a complex property term can be determined from the denotation(s) of the relevant syntactically simpler term(s). An interesting case in Navya-Nyāya is the "conjoint absence" (*ubhayābhāva*), which can be regarded as the Sheffer stroke applied to property terms.

We will show that an extension of Bealer's axiomatic system T1 may serve to prove some of the Navya-Naiyāyikas' intuitions concerning iterated absences, such as "the relational absence of the difference from a pot", "the relational absence of the relational absence of a pot" or "the relational absence of the relational absence of the relational absence of a pot". The former, e.g., was claimed to be identical to the universal "potness".

Keywords: Indian logic, Navya-Nyāya, intensionality, property theories, negation.

1 Introduction

The present paper is about late Indian logic. More specifically, we will deal with a type of logic which originated in a school of classical Indian philosophy called “Navya-Nyāya” (“New Logic”?). Its early beginnings date back to the 12th or 13th century with authors such as Śaśadhara and Maṇikaṇṭha Miśra.[1] Gaṅgeśa’s magnum opus Tattvacintāmaṇi (14th century) was seminal for the development of the typical style of the Navya-Naiyāyikas’ approach to logical and epistemological issues. In order to define their concepts with utmost precision they designed an ideal language, a kind of Leibnizian characteristica universalis based on a canonical form of Sanskrit, which serves to explicate the objective content of verbalized and unverbalized cognitions and to disambiguate sentences formulated in ordinary Sanskrit. The school reached its peak in the works of authors such as Raghunātha Śiromaṇi (16th century), Jagadīśa and Gādādhara (17th century) and remained active through to the 19th century.

Navya-Nyāya logic was dubbed a “logic of property and location” by Matilal. In order to demonstrate what this means, let us take an empty pot as an example. Even though the pot is empty, Navya-Naiyāyikas claimed that there are lots of items in this pot and also all around it. The universals (*jāti*) “substanceness” (*dravyatva*) and “potness” (*ghaṭatva*), e.g., are in the pot and all around it. There are some other properties attached to the pot, such as “being created” (*kṛtatva*) and “being non-eternal” (*anityatva*), which were unlike universals not counted as elementary constituents of empirical reality in Navya-Nyāya. As nominal properties (*upādhi*) they were nevertheless considered to be part of the actual world. Navya-Naiyāyikas reified universals and nominal properties, i.e., they treated them as individuals.

In the present paper we will focus on negative properties, the so-called “absences” (*abhāva*), which were also regarded as individuals. There are two types of absence. In order to illustrate them we can

[1]There is good reason to believe that the works of Udayana (11th century) already mark the advent of Navya-Nyāya (cf. [13], p. 9f).

again refer to our example of an empty pot. Since the pot is different from a cloth, it is a locus of the "mutual absence" (*anyonyābhāva*) *of* or the "difference" (*bheda*) *from* a cloth. Since there is no food in the pot, it is also a locus of the "absence of food". This is another type of negative property. More accurately, it is called "relational absence" (*saṃsargābhāva*), since this property characterizes a locus as being unrelated to the absentee in terms of one of the relations enunciated in the Navya-Nyāya system of ontological categories. Since food has no "contact" (*saṃyoga*) with the pot, the latter is a locus of the "absence of food having contact with the pot".[2] The specification of the relation whereby an absentee fails to reside in a locus was regarded as crucial: Although potness resides in a pot via a relation called "inherence" (*samavāya*), a pot is a locus of the absence of potness having contact with a pot. A difference is construed as a denial of a further type of relation, namely "identity" (*tādātmya*).

Let us look again at the properties "being created" and "being non-eternal". Navya-Naiyāyikas assumed that whatever is created is non-eternal and vice versa. Therefore both properties were believed to share the same loci, i.e., they were regarded as equi-locatable. Nevertheless, they were considered to be distinct properties. This coincides with our intuition, because we can imagine a logically possible world in which something is created, but eternal.

What about potness and the absence of the difference from a pot? Differences were assumed to be related to their loci via the so-called "peculiar relation" (*svarūpasaṃbandha*). So, we may understand this absence as an absence whose absentee, i.e., the difference from a pot, is unrelated to a locus in the sense that it has no peculiar relation to that locus. But no matter in what way we specify the relation here, the difference from a pot is always absent from every pot, since every pot is not different from some pot ($\forall x(Px \rightarrow \exists y(Py \land \neg(x \neq y)))$). So, the relational absence of the difference from a pot resides in every pot. On the other hand, every pot is a locus of potness. Hence, the relational absence of the difference from a pot is equi-locatable with

[2]To be more precise, such an absence was said to be "limited" (*avacchinna*) by contact.

potness. In this case we might have the feeling that the expressions “the relational absence of the difference from a pot” and “potness” refer to the same property. The former expression seems to be a kind of logically equivalent circumlocution of the latter. Actually, the Navya-Nyāya logician Mathurānātha equates “potness” with “the relational absence of the difference from a pot”.

The example of the properties “being created” and “being non-eternal” shows that in the Navya-Nyāya logic of property and location properties do not conform to an extensionality principle. Unlike sets, which are identical if they have the same members, properties need not be identical if they have the same loci.

On the other hand, the example of the properties “potness” and “the relational absence of the difference from a pot” shows that an appropriate formal reconstruction of the Navya-Nyāya logic of property and location should be equipped with a criterion for the identification of properties. An obvious idea is to model the Navya-Nyāya intuitions about the identity of properties by regarding identity as tantamount to necessary equivalence. This is expressed in [2] in the form of axiom A8 as part of Bealer’s intensional property theory T1, as we will see below.

2 Towards a formal reconstruction of the logic of Navya-Nyāya

2.1 G. Bealer’s calculus T1 as a basic framework

T1 harmonizes well with the reification of properties in Navya-Nyāya, since Bealer denotes properties by means of terms. For that purpose he adds square brackets to a first-order language along with the following formation rule:

If A is a formula and $v_1, \ldots, v_m$ $(0 \leq m)$ are distinct variables, then $[A]_{v_1 \ldots v_m}$ is a term.

A term of the form $[A]_{v_1 \ldots v_m}$ denotes . . .

a) a proposition, if $m = 0$ ("that A").
b) a property, if $m = 1$ ("being a v_1 of which A is true").
c) an m-ary relation, if $m \geq 2$ ("the relation which holds between $v_1, \dots, v_m$ *iff* A applies to them").

If Px translates into "x is a pot", then $[Px]_x$ (which can be read as "being an x such that x is P") is an analytical expression for "potness". The function of the index variable x is to bind the free occurrence of x in Px. The square brackets around Px indicate an intensional context. If one substitutes an expression within the bracketed part by another one which is extensionally equivalent, one might change the reference of the property term. Such restrictions concerning the substitutability of extensionally equivalent expressions generally distinguish intensional from extensional logical systems.

The language of T1 includes also the modal operators $\Box$ and $\Diamond$, but as defined symbols. An expression of the form $\Box A$ is adopted as a convenient abbreviation of expressions such as $N[A]$, where N is a one-place predicate expressing "... is necessary". The semantic model structure for T1 (cf. [2], p. 49) contains a condition which ensures that there is only one necessary truth (cf. [2], p. 52f). Since $[x = x]$ is a trivial necessary truth for any proposition x, $[A]$ can be identified with it if A is necessarily true. Therefore it is possible to define the modal operator $\Box$ simply by means of the square brackets: $\Box A :\leftrightarrow [A] = [[A] = [A]]$ (A is necessarily true *iff* the proposition "that A" is identical to a trivial necessary truth.) As usual, $\Diamond A :\leftrightarrow \neg \Box \neg A$.

Bealer shows that we obtain a sound and complete calculus by axiomatizing T1 in the following way (cf. [2], p. 58f):

A1: Truth-functional tautologies

A2: $\forall v_i A(v_i) \rightarrow A(t)$, where t is free for v_i in A, i.e., no free occurrence of v_i in A lies within the scope of a quantifier or a sequence of index variables in a term $[\dots]_{v_1 \dots v_m}$ which would bind a variable occurring in t.

A3: $\forall v_i (A \rightarrow B) \rightarrow (A \rightarrow \forall v_i B)$, where v_i is not free in A.

A4: $v_i = v_i$

A5: $v_i = v_j \rightarrow (A(v_i, v_i) \leftrightarrow A(v_i, v_j))$, where $A(v_i, v_j)$ is a formula that arises from $A(v_i, v_i)$ by replacing some (but not necessarily all) free occurrences of v_i by v_j, and v_j is free for the occurrences of v_i that it replaces.

A6: $[A]_{u_1 \dots u_p} \neq [B]_{v_1 \dots v_q}$, where $p \neq q$.

A7: $[A(u_1, \dots, u_p)]_{u_1 \dots u_p} = [A(v_1, \dots, v_p)]_{v_1 \dots v_p}$, where these two terms are alphabetic variants.

A8: $[A]_{u_1 \dots u_p} = [B]_{u_1 \dots u_p} \leftrightarrow \Box \forall u_1 \dots \forall u_p (A \leftrightarrow B)$

A9: $\Box A \rightarrow A$

A10: $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$

A11: $\Diamond A \rightarrow \Box \Diamond A$

R1: If $\vdash A$ and $\vdash (A \rightarrow B)$, then $\vdash B$.

R2: If $\vdash A$, then $\vdash \forall v_i A$.

R3: If $\vdash A$, then $\vdash \Box A$.

A1 – A5 along with R1 and R2 constitute an axiomatization of first-order predicate logic including identity. A6 – A8 determine how to deal with the intensional abstracts in T1. A8 furnishes a criterion for the identification of intensional abstracts. It captures the idea that identity is tantamount to necessary equivalence. A9 – A11 and R3 are the modal part of the axiomatic system S5 of propositional modal logic.

2.2 Extensions of T1 which function as alternatives to set theories

2.2.1 The naive property abstraction in Navya-Nyāya

In order to prove some of the Navya-Naiyāyikas' intuitions about negative properties we need an extension of this calculus. More specifically, we need a kind of comprehension principle for properties. The

Navya-Naiyāyikas themselves formulated such a principle in the following way: *tattvavat tad eva.* – "Anything which possesses the property 'being that' is that." (Cf. [9], p. 36)

In order to see how this rule works one might replace the Sanskrit word *tat* ("that"), which has the same function as a schematic variable here, by words like *ghaṭa* ("pot"). *ghaṭatvavān ghaṭa eva* means: "Anything which possesses the property 'potness' is a pot." Thus, the *tattvavat tad eva*-rule can be regarded as a kind of counterpart of the naive class abstraction in set theory:

$a \in \{x|A(x)\} \leftrightarrow A(a)$, where a is free for x in A and vice versa.

This equivalence can be transformed into a formal version of the naive property abstraction rule in Navya-Nyāya by replacing $\{x|A(x)\}$ by the corresponding property term in T1, i.e., $[A(x)]_x$. In order to express that something possesses or is a locus of $[A(x)]_x$ we can use Bealer's Δ-relation, which functions as a counterpart of the $\in$-relation in set theory (cf. [2], p. 96). Thus, if we understand the *tattvavat tad eva*-rule in the sense of an equivalence, we can formalize it in the following way[3]:

$(*)\ a\,\Delta\,[A(x)]_x \leftrightarrow A(a)$, where a is free for x in A and vice versa.

2.2.2 A property-theoretic variant of Zermelo-Russell's antinomy and its Sanskrit equivalent

Navya-Nyāya logicians were not aware that a variant of Zermelo-Russell's antinomy can be derived from the *tattvavat tad eva*-rule (cf. [5], p. 109 and [6], p. 144f):

Let us replace the word *tat* ("that") in the *tattvavat tad eva*-rule by *asvavṛttitva* ("being not resident in itself"). This property can easily

[3]The present interpretation of the *tattvavat tad eva*-rule as an equivalence is confirmed by Matilal, who characterizes the specific style of Navya-Nyāya texts in the following way: "Simple predicate formulations, such as 'x is F' are noted, but only to be rephrased as 'x has *F-ness*' (where '*F-ness*' stands for the property derived from 'F')." ([11], p. 115).

be formalized. If one admits $x \Delta x$ as a formal equivalent of "x resides in itself", "being not resident in itself" can be expressed as $[\neg x \Delta x]_x$. Let r be an abbreviation of this property.

(a) If r is resident in itself (i.e., if it is *svavṛtti*), then the property "being not resident in itself" (*asvavṛttitva*) resides in r. Therefore (according to the *tattvavat tad eva*-rule) r is not resident in itself (i.e., it is *asvavṛtti*). (Contradiction!)

This is the formal counterpart of the argument:

$$r \Delta r \Rightarrow \underbrace{r \Delta [\neg x \Delta x]_x}_{\text{can be substituted for } a \Delta [A(x)]_x \text{ in } (*)} \Rightarrow \neg r \Delta r$$

(b) If r is not resident in itself (i.e., if it is *asvavṛtti*), then (according to the *tattvavat tad eva*-rule) the property "being not resident in itself" (*asvavṛttitva*) resides in r. Therefore r is resident in itself (i.e., it is *svavṛtti*). (Contradiction!)

This is the formal counterpart of the argument:

$$\underbrace{\neg r \Delta r}_{\text{can be substituted for } A(a) \text{ in } (*)} \Rightarrow r \Delta [\neg x \Delta x]_x \Rightarrow r \Delta r$$

(a) and (b) together yield the following variant of Zermelo-Russell's antinomy:

$r \Delta r \leftrightarrow \neg r \Delta r$

2.2.3 An ST_2-style extension of T1 ("T1+") as an appropriate framework for a formal reconstruction of Navya-Nyāya logic

In order to modify $(*)$ in such a way that its paradoxical consequence disappears we can try to imitate the strategies which were pursued by the founders of set theories in order to safeguard the naive class abstraction rule against Zermelo-Russell's antinomy.

Certain restrictions in standard systems of set theory would, however, interfere with ontological commitments in Navya-Nyāya. In ZF (Zermelo-Fraenkel set theory), e.g., sets are the only objects in the domain of models of this system. However, since Navya-Naiyāyikas also talk about non-class-like objects, such as, e.g., pots, one needs a system which is similar to set theories with urelements.

Moreover, some logical arguments in Navya-Nyāya involve universal properties such as nameability, which can be regarded as the analogue of a proper class in set theory. Talking about proper classes like, e.g., $\{x \mid x = x\}$ ("the universal class") is admissible in NBG (Neumann-Bernays-Gödel set theory), but not in ZF. Therefore a property adaptation of NBG with urelements is preferable as a system which may serve to model logical inquiries concerning properties in Navya-Nyāya.

Mendelson incorporates urelements into the framework of NBG (cf. [12], p. 297f). He uses lower-case Latin letters (x, y, z) as restricted variables for sets, capital Latin letters (X, Y, Z) as restricted variables for classes (i.e., for sets and proper classes) and lower-case boldface Latin letters $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ as variables for classes and urelements alike (cf. [12], p. 297). In the present property adaptation of set-theory the same kinds of variables stand for set-like properties, class-like properties (i.e., set-like and properly class-like properties) and urelements, respectively. Lower-case boldface Latin letters as index variables in property terms refer to urelements and set-like individuals. Thus, $[A(\mathbf{x})]_{\mathbf{x}}$ has to be understood in the sense of "being an urelement or a set-like individual $\mathbf{x}$ such that A is true of $\mathbf{x}$". Without this restriction $[\mathbf{x} = \mathbf{x}]_{\mathbf{x}}$ might pass for a property of all properly class-like properties. However, the presumptive existence of such a property invokes a variant of Zermelo-Russell's antinomy in the present logical framework, as we will see below.

A property version of the NBG comprehension axiom seems to be still too restrictive, because it does not include impredicative instantiations, which a Navya-Naiyāyika might not want to rule out (cf. the example given in fn. 4). Since impredicative comprehension is admissible in QM (Quine-Morse set theory, also known as "Morse-Kelley set theory"), but not in NBG, the modification of $(*)$ should be patterned after the QM comprehension axiom. By means of the predicates $P_s\mathbf{x}$ (for "$\mathbf{x}$ is a set-like property") and $U\mathbf{x}$ (for "$\mathbf{x}$ is an urelement") it can be expressed in the following way:

(C) $\forall\mathbf{x}(P_s\mathbf{x} \vee U\mathbf{x} \rightarrow (\mathbf{x}\Delta[A(\mathbf{y})]_\mathbf{y} \leftrightarrow A(\mathbf{x})))$, where $\mathbf{x}$ is free for $\mathbf{y}$ in A and vice versa.[4]

There is still another constraint in standard systems of set theory which should not be reproduced in a formal reconstruction of Navya-Nyāya logic: It is commonly assumed that proper classes can never be elements of classes, i.e., (even finite) collections of proper classes do not exist.

In Navya-Nyāya, however, it is possible to apply the *-tva*-abstraction technique repeatedly, so that we might create an expression like *abhidheyatvatva* ("nameabilityness"), which denotes a property of nameability. The analogue of such a property in set theory would be the singleton of the universal class, something which does not exist according to standard systems of set theory. One might call it a "hyper-class" ([4], p. 142).[5]

An appropriate set-theoretic system on which one can model a formal reconstruction of Navya-Nyāya logic should endorse the existence of hyper-classes, hyper-hyper-classes (i.e., classes of hyper-classes) etc. In [4] (cf. p. 142f) the authors design such a system by combining

[4]Since (C) is impredicative, we can use it to formalize substitution instances of the *tattvavat tad eva*-rule, such as: "$\mathbf{x}$ is a locus of the property 'being a locus of some property which is equi-locatable with nameability' (*abhidheyatvasamaniyata-kiṃciddharmādhikaraṇatva*) iff $\mathbf{x}$ is a locus of some property which is equi-locatable with nameability." The symbolization key ...

$N\mathbf{x}$: "$\mathbf{x}$ is nameable"
$\mathbf{x}L\mathbf{y}$: "$\mathbf{x}$ is a locus of $\mathbf{y}$"
$\mathbf{x} \doteqdot \mathbf{y}$: "$\mathbf{x}$ is equi-locatable with $\mathbf{y}$", i.e., $\forall\mathbf{z}(\mathbf{z}L\mathbf{x} \leftrightarrow \mathbf{z}L\mathbf{y})$

... yields the following instantiation of (C):

$\forall\mathbf{x}(P_s\mathbf{x} \vee U\mathbf{x} \rightarrow (\mathbf{x}\Delta[\exists\mathbf{z}(\mathbf{x}\Delta\mathbf{z} \wedge \mathbf{z} \doteqdot [N\mathbf{y}]_\mathbf{y})]_\mathbf{x} \leftrightarrow \exists\mathbf{z}(\mathbf{x}\Delta\mathbf{z} \wedge \mathbf{z} \doteqdot [N\mathbf{y}]_\mathbf{y}))$

[5]The concept of a hyper-class should not be confounded with that of a hyperset (i.e., a non-wellfounded set) in non-wellfounded systems of set theory (cf. [1], p. 6).

the set theories of QM and ZF. The resulting system ST_2 can serve as a set-theoretic prototype of the Navya-Nyāya logic of property and location if we additionally take into account urelements. In ST_2 with urelements $S\mathbf{x}$ (read: "$\mathbf{x}$ is a set") functions as a primitive monadic predicate. The system comprises the following axioms:

(a) A sethood axiom: Every member of a set is a set or urelement.

(b) All the axioms of QM with urelements (with due regard to the above-mentioned notational convention for variables).

(c) The axioms of ZF with all variables replaced by upper case variables.

This is a two-tier set theory with sets and urelements in the bottom tier and classes in the upper tier. (c) warrants the existence of hyper-classes, hyper-hyper-classes etc. in ST_2. Due to the ZF-axiom of pairing with upper case variables we can, e.g., pair the universal class V with itself in order to obtain the hyper-class $\{V\}$. However, a hyper-class which contains all proper classes does not exist in ST_2. Since there is no universal set in ZF, there is also no way to obtain a corresponding universal hyper-class by means of the axioms in (c).[6]

Proper classes can be elements in ST_2, but they should still be distinguishable from sets. This is achieved by adding (a), which ensures that proper classes cannot be elements of sets.

A property-theoretic counterpart of ST_2 with urelements can be obtained by transforming (a), (b) and (c) into the corresponding property versions. Only the variants of the axiom of extensionality in (b) and (c) have to be exlcuded, because there is already a criterion for the identification of intensional abstracts in T1, namely A8.[7] The extension of T1 which includes the above-mentioned axioms of a property

[6]If $V = \{x \,|\, x = x\}$ were a set, then $\{x \in V \,|\, x \notin x\}$ would also be a set according to the ZF-comprehension axiom, i.e., $\{x \,|\, x = x \wedge x \notin x\} = \{x \,|\, x \notin x\} = Ru$ would be a set. Hence, $Ru \in Ru \leftrightarrow Ru \notin Ru$.

[7]The following list takes its cue from the property versions of NBG and ZF in [2] (cf. p. 265). For the sake of completeness we have included all the axioms ensuing from the property adaptation of ST_2 with urelements, although some of them might

adaptation of (a), (b) and (c) (exluding the variants of the axiom of extensionality in (b) and (c)) will be called “T1+” hereafter.[8]

be irrelevant to a formal reconstruction of Navya-Nyāya logic. A notable exception is a regularity axiom for properties, which does play an important role in logical inquiries concerning properties in Navya-Nyāya (cf. [8]).

(a)′ $\forall x \forall \mathbf{y}(\mathbf{y}\Delta x \rightarrow (P_s\mathbf{y} \vee U\mathbf{y}))$

(b)′ (Urelements) $\forall \mathbf{x}(U\mathbf{x} \rightarrow \forall \mathbf{y}(\mathbf{y} \not\Delta \mathbf{x}))$

(Comprehension) $\forall \mathbf{x}(P_s\mathbf{x} \vee U\mathbf{x} \rightarrow (\mathbf{x}\Delta[A(\mathbf{y})]_{\mathbf{y}} \leftrightarrow A(\mathbf{x})))$, where $\mathbf{x}$ is free for $\mathbf{y}$ in A and vice versa.

(Null) $\exists x \forall \mathbf{y}(\mathbf{y} \not\Delta x)$

(Pairing) $\forall \mathbf{x} \forall \mathbf{y}((P_s\mathbf{x} \vee U\mathbf{x}) \wedge (P_s\mathbf{y} \vee U\mathbf{y}) \rightarrow \exists z \forall \mathbf{w}(\mathbf{w}\Delta z \leftrightarrow (\mathbf{w} = \mathbf{x} \vee \mathbf{w} = \mathbf{y})))$

(Union) $\forall x \exists y \forall \mathbf{z}(\mathbf{z}\Delta y \leftrightarrow \exists \mathbf{w}(\mathbf{w}\Delta x \wedge \mathbf{z}\Delta \mathbf{w}))$

(Power) $\forall x \exists y \forall \mathbf{z}(\mathbf{z}\Delta y \leftrightarrow \forall \mathbf{w}(\mathbf{w}\Delta \mathbf{z} \rightarrow \mathbf{w}\Delta x))$

(Infinity) $\exists x([y \neq y]_y \Delta x \wedge \forall z(z\Delta x \rightarrow [w\Delta z \vee w = z]^z_w \Delta x))$

(Replacement) $\forall X \forall x(\forall \mathbf{u} \forall \mathbf{v} \forall \mathbf{w}((P_s\mathbf{u} \vee U\mathbf{u}) \wedge (P_s\mathbf{v} \vee U\mathbf{v}) \wedge (P_s\mathbf{w} \vee U\mathbf{w}) \rightarrow (<\mathbf{u},\mathbf{v}> \Delta X \wedge <\mathbf{u},\mathbf{w}> \Delta X \rightarrow \mathbf{v} = \mathbf{w})) \rightarrow \exists y \forall \mathbf{z}(\mathbf{z}\Delta y \leftrightarrow \exists \mathbf{w}(\mathbf{w}\Delta x \wedge <\mathbf{w},\mathbf{z}> \Delta X)))$

(Regularity) $\forall X(\exists \mathbf{y}(\mathbf{y}\Delta X) \rightarrow \exists \mathbf{y}(\mathbf{y}\Delta X \wedge \forall \mathbf{z}(\mathbf{z}\Delta X \rightarrow \mathbf{z} \not\Delta \mathbf{y})))$

(c)′ (Comprehension) $X\Delta[X\Delta Y \wedge A]^Y_X \leftrightarrow X\Delta Y \wedge A$

(Null) $X \not\Delta [X \neq X]_X$

(Pairing) $X\Delta[X = Y \vee X = Z]^{YZ}_X \leftrightarrow X = Y \vee X = Z$

(Union) $X\Delta[\exists Z(X\Delta Z \wedge Z\Delta Y)]^Y_X \leftrightarrow \exists Z(X\Delta Z \wedge Z\Delta Y)$

(Power) $X\Delta[\forall Z(Z\Delta X \rightarrow Z\Delta Y)]^Y_X \leftrightarrow \forall Z(Z\Delta X \rightarrow Z\Delta Y)$

(Infinity) $\exists X([Y \neq Y]_Y \Delta X \wedge \forall Z(Z\Delta X \rightarrow [W\Delta Z \vee W = Z]^Z_W \Delta X))$

(Replacement) $\forall X \forall Y \forall Z((A(X,Y) \wedge A(X,Z)) \rightarrow Y = Z) \rightarrow \forall Y(Y\Delta[\exists X(X\Delta W \wedge A(X,Y))]^W_Y \leftrightarrow \exists X(X\Delta W \wedge A(X,Y)))$

(Regularity) $\forall X(\exists Y(Y\Delta X) \rightarrow \exists Y(Y\Delta X \wedge \forall Z(Z\Delta X \rightarrow Z \not\Delta Y)))$

[8]In T1+ we can prove the following instantiation of the naive Navya-Nyāya property abstraction: “It is a locus of nameabilityness, iff it is nameability.” By means of the predicate $N\mathbf{x}$ (for “$\mathbf{x}$ is nameable”) “nameability” can be expressed as $[N\mathbf{x}]_{\mathbf{x}}$. The substitution of $[N\mathbf{x}]_{\mathbf{x}}$ for Y and Z in the (c)′-axiom of pairing yields the above-mentioned instantiation of the naive Navya-Nyāya property abstraction: $X\Delta[X = [N\mathbf{x}]_{\mathbf{x}}]_X \leftrightarrow X = [N\mathbf{x}]_{\mathbf{x}}$ The existence of the hyper-class-like property “nameabilityness”, i.e., $[X = [N\mathbf{x}]_{\mathbf{x}}]_X$ can be inferred from this as a corollary.

3 Negative properties

We are now prepared to formalize some of the Navya-Nyāya intuitions about negative properties and to prove them in T1+. First of all, we will translate the two types of "absence" (*abhāva*) into the language of T1+.

An absence can be regarded as the result of applying an operation to a property, which Bealer calls "negation". Following his terminology the term $[\neg F\mathbf{x}]_{\mathbf{x}}$ is the result of "negating" the term $[F\mathbf{x}]_{\mathbf{x}}$. Bealer introduces several operations on properties in order to explain how the denotation of a complex term $[A]_\alpha$ can be determined from the denotation(s) of the relevant syntactically simpler term(s) (cf. [2], p. 46f). Interestingly, some of these operations were also taken into account by Navya-Nyāya logicians.

3.1 Mutual absence

The term $[\neg F\mathbf{x}]_{\mathbf{x}}$ may serve as a formal representation of the "mutual absence" (*anyonyābhāva*), i.e., of the "difference" (*bheda*) from an F (more accurately: from anything which is an F). The indefinite article has been added here in front of F in order to facilitate a smooth English translation. In Sanskrit there is no article. A phrase like "the mutual absence of a cloth" is commonly expressed by means of a compound (*paṭānyonyābhāva*) and the literal meaning would be "cloth-mutual-absence". Similarly, "the relational absence of a pot" would be renderd as a compound which literally translates into "pot-relational-absence" (*ghaṭasaṃsargābhāva*). When asked to specify the absentees, the so-called "counterpositives" (*pratiyogin*) of these absences, a Navya-Naiyāyika might say "cloth" (*paṭa*) and "pot" (*ghaṭa*), where "cloth" and "pot" are meant in the sense of expressions which refer to any cloth or any pot, respectively.[9]

[9] In order to emphasize that no reference to one particular cloth or pot is intended here, a Navya-Naiyāyika might say that in these cases the counterpositiveness is "limited" (*avacchinna*) by clothness or potness, respectively. On the other hand, by adding a demonstrative like "this" (*etad*) in front of "cloth" or "pot" he might indicate that these expressions are meant in the sense of singular terms.

We can regard $[\neg F\mathbf{x}]_{\mathbf{x}}$ as a shorthand version of the following formalization which duly mirrors the fact that Navya-Naiyāyikas conceive of a mutual absence, i.e., of a difference, as a denial of an identity between the absentee and the locus of the absence:

(†) $[\neg\exists\mathbf{y}(F\mathbf{y}\wedge\mathbf{x}=\mathbf{y})]_{\mathbf{x}}$

3.2 Relational absence

The "relational absence" (*saṃsargābhāva*) of an F (more accurately: of anything which is an F) can be construed as a property which characterizes something as being devoid of (or: no locus of) anything which is an F. In order to formalize this property we will use the predicate $\mathbf{x}L\mathbf{y}$ with the intended meaning "$\mathbf{x}$ is a locus of $\mathbf{y}$". L is supposed to be a more general occurrence relation than the Δ-relation in the sense that $\mathbf{x}L\mathbf{y}$ might also be true if $\mathbf{y}$ is an urelement. Thus, we can infer $\mathbf{x}L\mathbf{y}$ from $\mathbf{x}\Delta\mathbf{y}$, but not vice versa.

Using the L-relation for the purpose of formalizing a relational absence is appropriate in cases where there is no specification of the occurrence relation which fails to subsist between the absentee and the locus of the absence. Thus, an unspecified relational absence can be formalized in the following way:

(‡) $[\neg\exists\mathbf{y}(F\mathbf{y}\wedge\mathbf{x}L\mathbf{y})]_{\mathbf{x}}$

If the relational absence is more specifically meant in the sense of a relational absence via contact or inherence etc., we can replace L by the corresponding symbols for these relations (such as, e.g., C or I).

Since the present formalization of relational absences has basically the same syntactic structure as a mutual absence, namely $[\neg\phi(\mathbf{x})]_{\mathbf{x}}$, where $\phi(\mathbf{x}) :\leftrightarrow \exists\mathbf{y}(F\mathbf{y}\wedge\mathbf{x}L\mathbf{y})$, we can also regard a relational absence as a negation, i.e., as the negation of the property "being a locus of an F" ($[\exists\mathbf{y}(F\mathbf{y}\wedge\mathbf{x}L\mathbf{y})]_{\mathbf{x}}$). Possessing a relational absence of an F means to be different from a locus of an F. So, a relational absence turns out to be a special case of a mutual absence. Both can be regarded as negations.

Navya-Naiyāyikas see the essential difference between the two types of absence in the relation by which the absentee fails to reside in the locus of the absence. In the case of mutual absence this relation is identity. In the case of relational absence it is some kind of occurrence relation. This distinctive feature is duly mirrored in the present formalizations (†) and (‡), because they differ only with respect to the relations (L and $=$).

3.3 Identities concerning iterated absences

[6] (p. 147f) contains a proof of the following identity concerning iterated absences, which is endorsed by Mathurānātha (cf. [9], p. 71 and [11], p. 152f):

(Id) The relational absence (*saṃsargābhāva*) of the difference (*bheda*) from a pot is identical to potness.

The difference from a pot can obviously be represented as ...

$[\neg P\mathbf{x}]_{\mathbf{x}}$, where $P\mathbf{x}$ translates into "$\mathbf{x}$ is a pot".

Since not only urelements other than pots, but also all class-like individuals are different from pots, one might be inclined to regard the difference from a pot as a property which applies to all class-like individuals including properly class-like properties. However, T1+ does not yield the existence of a hyper-class-like property which applies to every properly class-like property, as there is also no universal set in ZF. Therefore our formalization of the difference from a pot restricts the range of loci to urelements and set-like individuals.

In order to obtain a formal representation of the absence of the difference from a pot one might replace $F\mathbf{y}$ in (‡) by ...

$\mathbf{y} = [\neg P\mathbf{x}]_{\mathbf{x}}$.

Then the relational absence of the difference from a pot (*ghaṭabhedābhāva*) can be expressed as ...

$[\neg\exists\mathbf{y}(\mathbf{y} = [\neg P\mathbf{x}]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{y})]_{\mathbf{x}}$ (“being no locus of anything which is identical to the difference from a pot”).

If the difference from a pot is supposed to be a property of all set-like properties and urelements other than pots, one might argue that the relational absence of the difference from a pot applies to all properly class-like properties. However, as noted above, the existence of a hyper-class-like property which applies to every properly class-like property is not warranted by T1+. Therefore our formalization of the relational absence of the difference from a pot restricts the range of loci to urelements and set-like individuals.

Now (Id) can be rendered as a T1+ proposition and we can prove it in T1+:

Theorem (Id):
$[\neg\exists\mathbf{y}(\mathbf{y} = [\neg P\mathbf{x}]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{y})]_{\mathbf{x}} = [P\mathbf{x}]_{\mathbf{x}}$

The proof contains an application of the following instantiation of (C):

$$\forall\mathbf{x}(P_s\mathbf{x} \vee U\mathbf{x} \rightarrow (\mathbf{x}\Delta[\neg P\mathbf{x}]_{\mathbf{x}} \leftrightarrow \neg P\mathbf{x}))$$

Hence, a substitution of equivalents by means of the wff $\mathbf{x}\Delta[\neg P\mathbf{x}]_{\mathbf{x}} \leftrightarrow \neg P\mathbf{x}$ is admissible if we require the variable $\mathbf{x}$ to range over urelements and set-like properties:

Proof of (Id)[10]:

(A1)	$\neg\neg P\mathbf{x} \leftrightarrow P\mathbf{x}$
(C)	$\neg\mathbf{x}\Delta[\neg P\mathbf{x}]_{\mathbf{x}} \leftrightarrow P\mathbf{x}$
(1st-order logic)	$\neg\exists\mathbf{y}(\mathbf{y} = [\neg P\mathbf{x}]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{y}) \leftrightarrow P\mathbf{x}$
(R2, R3)	$\Box\forall\mathbf{x}(\neg\exists\mathbf{y}(\mathbf{y} = [\neg P\mathbf{x}]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{y}) \leftrightarrow P\mathbf{x})$
(A8, R1)	$[\neg\exists\mathbf{y}(\mathbf{y} = [\neg P\mathbf{x}]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{y})]_{\mathbf{x}} = [P\mathbf{x}]_{\mathbf{x}}$ ■

[10]In order to be very precise, one might want to add “$\wedge(P_s\mathbf{x} \vee U\mathbf{x})$” on each side of the equivalences in the first four lines. The last line of the proof would then be $[\neg\exists\mathbf{y}(\mathbf{y} = [\neg P\mathbf{x}]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{y}) \wedge (P_s\mathbf{x} \vee U\mathbf{x})]_{\mathbf{x}} = [P\mathbf{x} \wedge (P_s\mathbf{x} \vee U\mathbf{x})]_{\mathbf{x}}$ and this is, of course, equivalent to (Id).

Maheśa Chandra states two other identities concerning iterated absences, namely the following reduction rules, which are referred to as (Id′) and (Id″) below: *tathāhi dvitīyābhāvaḥ (ghaṭābhāvābhāvaḥ) pratiyogi(ghaṭa)svarūpas tṛtīyābhāvaḥ (ghaṭābhāvābhāvābhāvaḥ) prathamābhāva(ghaṭābhāva)svarūpa iti prathamābhāvasya (ghaṭābhāvasya) ghaṭa iva dvitīyābhāvo 'pi (ghaṭābhāvābhāvo 'pi) pratiyogī.* ([3], p. 15, 27f = [7], p. 81) – "So, the second absence (the absence of the absence of a pot) is essentially identical to the counterpositive (pot). The third absence (the absence of the absence of the absence of a pot) is essentially identical to the first absence (the absence of a pot). So, the second absence (the absence of the absence of a pot) is like 'pot' of the first absence (the absence of a pot) a counterpositive (author's note: The "second absence" *ghaṭābhāvābhāva* is the counterpositive of the "third absence" *ghaṭābhāvābhāvābhāva*.)."

(Id′) The relational absence of the relational absence of a pot is identical to "pot".[11]

(Id″) The relational absence of the relational absence of the relational absence of a pot is identical to the relational absence of a pot.[12]

[11]As noted by Matilal, the Navya-Naiyāyika Raghunātha Śiromaṇi rejected this identity. "Raghunātha, however, in his intensionalist vein, argued against the identification of x with $\sim\sim x$ (author's note: "$\sim$" is Matilal's abbreviation of "relational absence".). For, he thought, the notion of negation conveyed by the second can never be conveyed by the first, and hence it is difficult to think of them as non-distinct." ([10], p. 5) For the same reason Raghunātha would also not have been in favour of (Id), which was like (Id′) generally accepted in Navya-Nyāya (cf. [10], p. 7). Raghunātha's intuitions concerning properties are closer to Bealer's system T2 (cf. [2], p. 64f). In T2 axiom A8 of T1 is replaced by $\mathscr{A}8$: $[A]_\alpha = [B]_\alpha \rightarrow (A \leftrightarrow B)$ Moreover, T2 contains an axiom which coincides with Raghunātha's argument against (Id′), namely $\mathscr{A}9$: $t \neq r$ (where t and r are non-elementary complex terms of different syntactic kinds). According to our formalization techniques the left side of (Id′) is the negation of a property, whereas the right side should be interpreted as a non-negated property. Hence, they belong to different syntactic categories and therefore $\mathscr{A}9$ forces us to reject (Id′).

[12]Some other kinds of iterated absences might have been taken into account here, especially those starting with a difference, such as the difference from the absence of a pot or the difference from the difference from a pot. However, as noted by Mati-

In order to explicate the right side of (Id′) in an appropriate way one might substitute "pot" (*ghaṭa*) by "being a locus of a pot" (*ghaṭa-vattva*), since this is common practice in Navya-Nyāya (cf. [11], p. 115). After all, the property "being a locus of a pot" is equi-locatable with every pot. Even though the Navya-Naiyāyikas do regard expressions like *ghaṭa* and *ghaṭavattva* as interchangeable, this is not unproblematic, because a pot possesses potness, whereas the property "being a locus of a pot" does not.

Theorem (Id′):
$[\neg\exists\mathbf{z}(\mathbf{z} = [\neg\exists\mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{z})]_{\mathbf{x}} = [\exists\mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}}$

Since not only urelements other than loci of pots, but also all class-like individuals are no loci of pots, one might be inclined to regard the relational absence of a pot as a property which applies to all class-like individuals including properly class-like properties. However, as noted above, T1+ does not yield the existence of a hyper-class-like property which applies to every properly class-like property. Therefore our formalization of the relational absence of a pot restricts the range of loci to urelements and set-like individuals.

If the relational absence of a pot is supposed to be a property of all set-like properties and urelements other than loci of pots, one might argue that the relational absence of the relational absence of a pot applies to all properly class-like properties. However, since the existence of a hyper-class-like property which applies to every prop-

lal, these kinds of absences were ignored in Navya-Nyāya (cf. [10], p. 8), because they were regarded as "not very interesting" (ibid.). This is quite obvious from the perspective of our formalization techniques, because any iterated absence starting with a difference can be expressed as $[\mathbf{x} \neq \ldots]_{\mathbf{x}}$, where "..." stands for an absence or a difference. Only the dotted part might be reducible to a less complex term. A "difference" in the beginning of an iterated absence is invariant under the application of any reduction procedure. Hence, there is no coreferential syntactically simpler term which corresponds to "the difference from the absence of a pot" or "the difference from the difference from a pot". Each of these properties resides in everything except for one individual, namely the absence of a pot or the difference from a pot, respectively.

erly class-like property is not warranted by T1+, we have to impose the above-mentioned restriction on the formalization of the relational absence of the relational absence of a pot as well.

The proof of (Id′) contains an application of the following instantiation of (C):

$$\forall \mathbf{x}(P_s\mathbf{x} \lor U\mathbf{x} \rightarrow (\mathbf{x}\Delta[\neg\exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \leftrightarrow \neg\exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})))$$

Hence, a substitution of equivalents by means of the wff $\mathbf{x}\Delta[\neg\exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \leftrightarrow \neg\exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y}))$ is admissible if we require the variable $\mathbf{x}$ to range over urelements and set-like properties:

Proof of (Id′):

(A1)	$\neg\neg\exists \mathbf{y}(P\mathbf{x} \land \mathbf{x}L\mathbf{y}) \leftrightarrow \exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})$
(C)	$\neg\mathbf{x}\Delta[\neg\exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \leftrightarrow \exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})$
(1st-order logic)	$\neg\exists \mathbf{z}(\mathbf{z} = [\neg\exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \land \mathbf{x}\Delta\mathbf{z}) \leftrightarrow \exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})$
(R2, R3)	$\Box\forall \mathbf{x}(\neg\exists \mathbf{z}(\mathbf{z} = [\neg\exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \land \mathbf{x}\Delta\mathbf{z}) \leftrightarrow \exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y}))$
(A8, R1)	$[\neg\exists \mathbf{z}(\mathbf{z} = [\neg\exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \land \mathbf{x}\Delta\mathbf{z})]_{\mathbf{x}} = [\exists \mathbf{y}(P\mathbf{y} \land \mathbf{x}L\mathbf{y})]_{\mathbf{x}}$ ■

In order to prove (Id″) and any other reduction rule which states the identity of an uneven number of such relational absences to a single relational absence, it suffices to prove:

(Id*) The relational absence of the property "being a locus of a pot" is identical to the relational absence of a pot.

By adding one relational absence on both sides of (Id′) we can infer from (Id′) that the relational absence of the relational absence of the relational absence of a pot is identical to the relational absence of $\underline{\text{pot}}$ (where the underlined "pot" is supposed to be explicated in the sense of "the property 'being a locus of a pot'"). On account of (Id*) the relational absence of "pot", i.e., of the property "being a locus of a pot", is identical to the relational absence of a pot, and this proves (Id$''$).

THEOREM (Id*):
$[\neg\exists \mathbf{z}(\mathbf{z} = [\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{z})]_{\mathbf{x}} = [\neg\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}}$

The proof contains an application of the following instantiation of (C):

$$\forall \mathbf{x}(P_s\mathbf{x} \vee U\mathbf{x} \to (\mathbf{x}\Delta[\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \leftrightarrow \exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y}))).$$

It is plausible to asume that neither of the members of the equivalence $\mathbf{x}\Delta[\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \leftrightarrow \exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})$ in this formula is true of any $\mathbf{x}$ which fulfills the condition $\neg(P_s\mathbf{x} \vee U\mathbf{x})$, i.e., $\forall \mathbf{x}(\neg(P_s\mathbf{x} \vee U\mathbf{x}) \to \neg\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y}) \wedge \neg\mathbf{x}\Delta[\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}})$, because individuals which are neither set-like properties nor urelements are class-like properties, i.e., they are different from loci of pots and do not possess the property to be loci of pots. Hence, the equivalence $\mathbf{x}\Delta[\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \leftrightarrow \exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})$ can be applied unconditionally in this case.

PROOF of (Id*):

(A1)	$\neg\exists \mathbf{y}(P\mathbf{x} \wedge \mathbf{x}L\mathbf{y}) \leftrightarrow \neg\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})$
(C)	$\neg\mathbf{x}\Delta[\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \leftrightarrow \neg\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})$
(1st-order logic)	$\neg\exists \mathbf{z}(\mathbf{z} = [\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{z}) \leftrightarrow \neg\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})$
(R2, R3)	$\Box\forall \mathbf{x}(\neg\exists \mathbf{z}(\mathbf{z} = [\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{z}) \leftrightarrow \neg\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y}))$
(A8, R1)	$[\neg\exists \mathbf{z}(\mathbf{z} = [\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}} \wedge \mathbf{x}\Delta\mathbf{z})]_{\mathbf{x}} = [\neg\exists \mathbf{y}(P\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}}$ ■

3.4 Sheffer stroke applied to properties

In Mathurānātha Tarkavāgīśa's Vyāptipañcakarahasyam (quoted in [9], p. 64f) this operation is named "conjoint absence" (*ubhayābhāva*).

Maheśa Chandra characterizes it as "an absence due to prefixing 'being both'": *. . . paṭaghaṭobhayatvarūpeṇa vobhayatvapuraskāreṇābhāvo . . .* ([3], p. 14, 5f = [7], p. 76) – ". . . or an absence due to prefixing 'being both' in the form of 'being both, [i.e.] cloth and pot' . . . "

Following Bealer, who names property operators after their corresponding propositional operators, one can regard the Sheffer stroke applied to properties as the negation of the conjunction of properties. An example of such a property is the absence of both, cloth and pot, in a house where there is a cloth, but no pot. *evaṃ gṛhe kevalasya paṭasya sattve 'pi ghaṭasyābhāvena paṭaghaṭobhayasyāpy abhāvo 'sty eva. ekābhāvenobhayābhāvasyāvaśyaṃbhāvitvād . . .* ([3], p. 14, 8f = [7], p. 76) – "So, when there is only a cloth in the house, there is absence of both, a cloth and a pot <collectively>, because of the absence of a pot, because the absence of both <collectively> is necessary on account of the absence of one."

The condition that there is a cloth but no pot in the house can be formalized as . . .

$\exists\mathbf{y}(C\mathbf{y} \wedge hL\mathbf{y}) \wedge \neg\exists\mathbf{z}(P\mathbf{z} \wedge hL\mathbf{z})$ (where $C\mathbf{x}$ is to be read as "$\mathbf{x}$ is a cloth", $P\mathbf{x}$ as "$\mathbf{x}$ is a pot", $\mathbf{x}L\mathbf{y}$ as "$\mathbf{x}$ is a locus of $\mathbf{y}$" and h as "the house").

Now, if there is a cloth but no pot in the house, then it is not the case that there is a cloth and a pot in the house. This can be rendered as an implication with an alternative formalization of the consequent by means of the Sheffer stroke:

$$\exists\mathbf{y}(C\mathbf{y}\wedge hL\mathbf{y}) \wedge\neg\exists\mathbf{z}(P\mathbf{z}\wedge hL\mathbf{z}) \rightarrow \underbrace{\neg(\exists\mathbf{y}(P\mathbf{y} \wedge hL\mathbf{y}) \wedge \exists\mathbf{z}(C\mathbf{z} \wedge hL\mathbf{z}))}_{\exists\mathbf{y}(P\mathbf{y}\wedge hL\mathbf{y})\uparrow\exists\mathbf{z}(C\mathbf{z}\wedge hL\mathbf{z})}$$

Since h denotes an urelement, we can apply (C) and substitute the consequent by an equivalent formula which expresses the fact that the house is a locus of the negation of the conjunction of the properties $[\exists\mathbf{y}(C\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}}$ ("being a locus of a cloth") and $[\exists\mathbf{z}(P\mathbf{z} \wedge \mathbf{x}L\mathbf{z})]_{\mathbf{x}}$

("being a locus of a pot"):

$$h\Delta[\exists\mathbf{y}(C\mathbf{y} \wedge \mathbf{x}L\mathbf{y}) \uparrow \exists\mathbf{z}(P\mathbf{z} \wedge \mathbf{x}L\mathbf{z})]_{\mathbf{x}}$$

The negation of each of the properties $[\exists\mathbf{y}(C\mathbf{y} \wedge \mathbf{x}L\mathbf{y})]_{\mathbf{x}}$ and $[\exists\mathbf{z}(P\mathbf{z} \wedge \mathbf{x}L\mathbf{z})]_{\mathbf{x}}$ yields the term for the corresponding absence, i.e., the absence of a cloth and the absence of a pot, respectively. Therefore it makes sense to regard the negation of their conjunction, i.e., $[\exists\mathbf{y}(C\mathbf{y} \wedge \mathbf{x}L\mathbf{y}) \uparrow \exists\mathbf{z}(P\mathbf{z} \wedge \mathbf{x}L\mathbf{z})]_{\mathbf{x}}$, as a formal equivalent of the "conjoint absence" (*ubhayābhāva*) of a cloth and a pot.

References

[1] J. Barwise and L. Moss, *Vicious Circles. On the Mathematics of Non-Wellfounded Phenomena*, (CSLI Lecture Notes 60), Stanford, CA: Stanford University, Center for the Study of Language and Information, 1996. ISBN: 1-57586-009-0.

[2] G. Bealer, *Quality and Concept*, Oxford: Clarendon Press, 1982. ISBN: 0-19-824726-5.

[3] BN, *Brief Notes on the Modern Nyāya System of Philosophy and its Technical Terms.* By Mahāmahopādhyay Maheśa Chandra Nyāyaratna, Calcutta: Hare Press, 1891. ISBN-13: 978-5518533431.

[4] A. Fraenkel, Y. Bar-Hillel and A. Levy, *Foundations of Set Theory*, (Studies in Logic and Foundations of Mathematics 67), Amsterdam/London/New York/Oxford/Paris/Shannon/Tokyo: North Holland, 1973. ISBN: 0-7204-2270-1.

[5] E. Guhe, "Intensionale Aspekte der indischen Logik," *Berliner Indologische Studien*, vol. 13/14, pp. 105–116, 2000.

[6] E. Guhe, "George Bealer's Property Theories and their Relevance to the Study of Navya-Nyāya Logic," in *Logic, Navya-Nyāya & Applications* (Studies in Logic, vol. 15), M. K. Chakraborti et al.

Eds. London: College Publications, 2008, pp. 139–153. ISBN: 978-1-904987-44-4.

[7] E. Guhe, *Maheśa Chandra Nyāyaratna's "Brief Notes on the Modern Nyāya System of Philosophy and its Technical Terms"*, Shanghai: Fudan University Press, 2014. ISBN: 978-7-309-10193-5.

[8] E. Guhe, "The Problem of Foundation in Early Nyāya and in Navya-Nyāya," *History and Philosophy of Logic*, vol. 36, no. 2, pp. 97–113, 2015.

[9] D. H. H. Ingalls, *Materials for the Study of Navya-Nyāya Logic* (Harvard Oriental Series 40), Cambridge, Massachusetts: Harvard University Press, 1951. ISBN-13: 978-8120803848.

[10] B. K. Matilal, "Double Negation in Navya-Nyāya," in *Sanskrit and Indian Studies. Essays in Honour of Daniel H.H. Ingalls*, B. K. Matilal and J. M. Masson, Eds. Dordrecht/Boston/London: D. Reidel Publishing Company, 1980, pp. 1–10. ISBN-13: 978-94-009-8943-6.

[11] B. K. Matilal, *Logic, Language and Reality*, Delhi: Motilal Banarsidass, 1990. ISBN: 81-208-0008-7.

[12] E. Mendelson, *Introduction to Mathematical Logic*, London: Chapman & Hall, 1997. ISBN: 0-412-80830-7.

[13] T. Wada, *The Analytical Method of Navya-Nyāya*, (Gonda Indological Studies XIV), Groningen: Egbert Forsten, 2007. ISBN: 90-6980-153-1.

Eberhard Guhe

Received October 3, 2016

Fudan University, Shanghai
E–mail: eberhard@guhe.de

Computer Science Journal of Moldova, vol.**24**, no.3(72), 2016

About Applications of Distances on Monoids of Strings

Mitrofan Choban, Ivan Budanaev

Dedicated to Professor, Corresponding Member of the Academy of Science of Moldova Constantin Gaindric on the occasion of his seventy-fifth anniversary

Abstract

In this article we show that there are invariant distances on the monoid $L(A)$ of all strings closely related to Levenshtein's distance. We will use a distinct definition of the distance on $L(A)$, based on the Markov - Graev method, proposed by him for free groups. As result we will show that for any quasimetric d on alphabet A in union with the empty string there exists a maximal invariant extension d^* on the free monoid $L(A)$. This new approach allows the introduction of parallel and semiparallel decompositions of two strings. In virtue of Theorem 3.1, they offer various applications of distances on monoids of strings in solving problems from distinct scientific fields. The discussion covers topics in fuzzy strings, string pattern search, DNA sequence matching etc.

Keywords: String pattern matching, parallel decomposition, semiparallel decomposition, free monoid, invariant distance, quasimetric, Levenshtein distance, Hamming distance, proper similarity.

1 Introduction

The dynamic transition of our technological civilization to digital processing and data transmission systems created many problems in the design of modern systems in computer science and telecommunications. Providing robustness and noise immunity is one of the most important and difficult tasks in data transmission, recording, playback, and storage. The distance between information plays a paramount role in mathematics, computer science, and other interdisciplinary research areas. The first among many scientists in the field, who presented the theoretical solutions to error detection and error correction problems, were C. Shannon, R. Hamming, and V. Levenshtein (see [11],[12],[18]). We begin this section with introductions into the field, focusing mainly on abstract monoid of strings $L(A)$.

A monoid is a semigroup with an identity element. Fix a non-empty set A. The set A is called an *alphabet*. Let $L(A)$ be the set of all finite strings $a_1a_2\dots a_n$ with $a_1, a_2, \dots, a_n \in A$. Let ε be the empty string. Consider the strings $a_1a_2\dots a_n$ such that $a_i = \varepsilon$ for some $i \leq n$. If $a_i \neq \varepsilon$, for any $i \leq n$ or $n = 1$ and $a_1 = \varepsilon$, the string $a_1a_2\dots a_n$ is called a *canonical string*. The set

$$Sup(a_1a_2\dots a_n) = \{a_1, a_2, \dots, a_n\} \cap A$$

is the support of the string $a_1a_2\dots a_n$ and

$$l(a_1\dots a_n) = |Sup(a_1\dots a_n)|$$

is the length of the string $a_1a_2\dots a_n$. For two strings $a_1\dots a_n$ and $b_1\dots b_m$, their product(concatenation) is $a_1\dots a_nb_1\dots b_m$. If $n \geq 2, i < n$ and $a_i = \varepsilon$, then the strings $a_1\dots a_n$ and $a_1\dots a_{i-1}a_{i+1}\dots a_n$ are considered equivalent. In this case any string is equivalent to one unique canonical string. We identify the equivalent strings. In this case $L(A)$ becomes a monoid with identity ε. Let $Sup(a,b) = Sup(a) \cup Sup(b) \cup \{\varepsilon\}$, and $Sup(a,a) = Sup(a) \cup \{\varepsilon\}$.

It is well known that any subset $L \subset L(A)$ is an abstract language over the alphabet A.

2 Distances on spaces

2.1 Definitions

Let A be a non-empty set and $d : X \times X \to \mathbb{R}$ be a mapping such that for all $x, y \in X$ we have:

(i_m) $d(x, y) \geq 0$;

(ii_m) $d(x, x) = 0$.

Then (X, d) is called a *pseudo-distance space* and d is called a *pseudo-distance* on X. In addition,

(iii_m) $d(x, y) + d(y, x) = 0$ if and only if $x = y$,

then (X, d) is called a *distance space* and d is called a *distance* on X. Furthermore,

(iv_m) $d(x, y) = 0$ if and only if $x = y$,

then (X, d) is called a *strong distance space* and d is called a *strong distance* on X.

General problems in distance spaces were studied by different authors (see [1], [3], [4], [8], [15]). The notion of a distance space is more general than the notion of *o*-metric spaces in sense of A. V. Arhangelskii [1] and S. I. Nedev [15]. A distance d is an *o*-metric if from $d(x, y) = 0$ it follows that $x = y$, i.e. d is a strong distance.

Let X be a non-empty set and d be a pseudo-distance on X. Then:

- (X, d) is called a *pseudo-symmetric space* and d is called a *pseudo-symmetric* on X if for all $x, y \in X$

$$(v_m) d(x, y) = d(y, x);$$

- (X, d) is called a *symmetric space* and d is called a *symmetric* on X if d is a distance and a pseudo-symmetric simultaneously;
- (X, d) is called a *pseudo-quasimetric space* and d is called a *pseudo-quasimetric* on X if for all $x, y, z \in X$

$$(vi_m) d(x, z) \leq d(x, y) + d(y, z);$$

- (X, d) is called a *quasimetric space* and d is called a *quasimetric* on X if d is a distance and a pseudo-quasimetric simultaneously;
- (X, d) is called a *pseudo-metric space* and d is called a *pseudo-metric* if d is a pseudo-symmetric and a pseudo-quasimetric simultaneously;
- (X, d) is called a *metric space* and d is called a *metric* if d is both symmetric and quasimetric;

- a distance d is called discrete if $d(x,y) \in \omega = \{0,1,2,\ldots\}$ for all $x,y \in X$.

Let G be a semigroup and d be a pseudo-distance on G. The pseudo-distance d is called:

- Left (respectively, right) invariant if $d(xa,xb) \leq d(a,b)$ (respectively, $d(ax,bx) \leq d(a,b)$) for all $x,a,b \in G$;
- Invariant if it is both left and right invariant.

A distance d on a semigroup G is called *stable* if $d(xy,uv) \leq d(x,u) + d(y,v)$ for all $x,y,u,v \in G$.

Proposition 1. *Let d be a pseudo-quasimetric on a semigroup G. The next assertions are equivalent:*

1. *d is invariant,*
2. *d is stable.*

2.2 Extension of pseudo-quasimetrics on free monoids

Fix an alphabet A and let $\bar{A} = A \cup \{\varepsilon\}$. We assume that $\varepsilon \in \bar{A} \subseteq L(A)$ and ε is the identity of the monoid $L(A)$. Let ρ be a pseudo-quasimetric on the set $\bar{A}$ and $Q(\rho)$ be the set of all stable pseudo-quasimetrics d on $L(A)$ for which $d(x,y) \leq \rho(x,y)$ for all $x,y \in \bar{A}$. The set $Q(\rho)$ is non-empty since it contains the trivial pseudo-quasimetric $d(x,y) = 0$ for all $x,y \in L(A)$. For all $a,b \in L(A)$ let $\hat{\rho}(a,b) = sup\{d(a,b) : d \in Q(\rho)\}$. We say that $\hat{\rho}$ is the maximal stable extension of ρ on $L(A)$.

The following properties are proved in [5].

Property 2.1. $\hat{\rho} \in Q(\rho)$.

For any $r > 0$ let $d_r(a,a) = 0$ and $d_r(a,b) = r$ for all distinct points $a,b \in L(A)$. Then d_r is an invariant metric on $L(A)$.

Property 2.2. *Let $r > 0$ and $\rho(x,y) \geq r$ for all distinct points $x,y \in A$. Then $\hat{\rho}$ is a quasimetric on $L(A)$, $d_r \in Q(\rho)$, and $\hat{\rho}(a,b) = r$ for all distinct points $a,b \in L(A)$.*

For any $a, b \in L(A)$ let

$$\bar{\rho}(a,b) = inf\{\Sigma\{\rho(x_i, y_i) : i \leq n\}\},$$

where $n \in \mathbb{N} = \{1, 2, \dots\}$, $x_1, y_1, x_2, y_2, \dots, x_n, y_n \in \bar{A}$, $a = x_1 x_2 \dots x_n, b = y_1 y_2 \dots y_n$. Let

$$\rho^*(a,b) = inf\{\bar{\rho}(a, z_1) + \dots + \bar{\rho}(z_i, z_{i+1}) + \dots + \bar{\rho}(z_n, b)\},$$

where $n \in \mathbb{N}, z_1, z_2, \dots, z_n \in L(A)$.

Property 2.3. *$\bar{\rho}$ is a pseudo-distance on $L(A)$ and $\bar{\rho}(x,y) \leq \rho(x,y)$ for all $x, y \in \bar{A}$.*

Property 2.4. *$\bar{\rho}(x,y) = \rho(x,y)$ for all $x, y \in X$.*

Property 2.5. *The pseudo-distance $\bar{\rho}$ is invariant on $L(A)$.*

Property 2.6. *The pseudo-distance ρ^* is a stable pseudo-quasimetric on $L(A)$ and $\rho^* \in Q(\rho)$.*

Property 2.7. *If ρ is a quasimetric on X, then $\bar{\rho}$ is a distance on $L(A)$.*

Property 2.8. *Let $a, b \in L(A)$ be two distinct points in $L(A)$ and $r(a,b) = min\{\rho(x,y) : x \in Sup(a,a), y \in Sup(b,b), x \neq y\}$. Then*

$$\hat{\rho}(a,b) = \rho^*(a,b) \geq r(a,b).$$

The following properties follow from Property 2.8.

Property 2.9. *If ρ is a quasimetric on $\bar{A}$, then ρ^* and $\hat{\rho}$ are quasimetrics on $L(A)$.*

Property 2.10. *If ρ is a strong quasimetric on $\bar{A}$, then ρ^* and $\hat{\rho}$ are strong quasimetrics on $L(A)$.*

Property 2.11. *Let ρ be a pseudo-quasimetric on $\bar{A}$, Y be a subspace of $\bar{A}$, and $\varepsilon \in \bar{Y}$. Let $M(Y) = L(Y)$ be the submonoid of the monoid $L(A)$ generated by the set Y, and by d_Y be the extension $\hat{\rho}|Y$ on $M(Y)$ of the pseudo-quasimetric ρ_Y on Y, where $\rho_Y(y,z) = \rho(y,z)$ for all $y, z \in \bar{Y}$. Then*

1. *$d_Y(a,b) = \hat{\rho}(a,b)$ for all $a, b \in M(Y)$,*
2. *If ρ is a (strong) quasimetric on Y, then $\hat{\rho}$ is a (strong) quasimetric on $M(Y)$,*
3. *If ρ is a metric on Y, then $\hat{\rho}$ is a metric on $M(Y)$,*
4. *If $a, b \in L(A)$ are distinct points and ρ is a quasimetric on $Sup(a,b)$, then $\hat{\rho}(a,b) + \hat{\rho}(b,a) > 0$,*
5. *If $a, b \in L(A)$ are distinct points and ρ is a strong quasimetric on $Sup(a,b)$, then $\hat{\rho}(a,b) > 0$ and $\hat{\rho}(b,a) > 0$,*
6. *For any $a, b \in L(A)$ there are $n \in \mathbb{N}$, $x_1, x_2, \ldots, x_n \in Sup(a,a)$ and $y_1, y_2, \ldots, y_n \in Sup(b,b)$ such that $a = x_1x_2 \cdots x_n$, $b = y_1y_2 \cdots y_n$ ρ, $n \leq l(a) + l(b)$ and $\bar{\rho}(a,b) = \Sigma\{\rho(x_i, y_i) : i \leq n\}$,*
7. *$\hat{\rho} = \bar{\rho} = \rho^*$.*

Property 2.12. *For any $a{=}a_1a_2 \ldots a_n$ we put $a^{-1}{=}a_n \ldots a_2a_1$. Then $\rho^*(a,b){=}\rho^*(a^{-1}, b^{-1})$ and $(ab)^{-1}{=}b^{-1}a^{-1}$ for all $a, b \in L(A)$.*

Remark 2.1. *The method of extensions of distances for free groups, used by us, was proposed by A. A. Markov [13] and M. I. Graev [9]. For free universal algebras it was extended in [3], for free groups and varieties of groups it was examined in [6], [17].*

2.3 Discrete distances on $L(A)$

Fix an alphabet A and $\bar{A}{=}A \cup \{\varepsilon\}$. Consider on A some linear ordering for which $\varepsilon < x$ for any $x \in A$. On $\bar{A}$ consider the following distances ρ_l, ρ_r, ρ_s, where $\rho_l(x,x) = \rho_r(x,x) = 0$ for any $x \in \bar{A}$; if $x, y \in \bar{A}$ and $x < y$, then $\rho_l(x,y) = 1, \rho_l(y,x) = 0$, $\rho_r(x,y) = 0, \rho_r(y,x) = 1$, $\rho_s(x,y) = \rho_l(x,y) + \rho_r(x,y)$. By construction, ρ_l and ρ_r are quasimetrics and ρ_s is a metric on $\bar{A}$. Then ${\rho_l}^*(x,y)$ and ${\rho_r}^*(x,y)$ are invariant discrete quasimetrics on $L(A)$ and ${\rho_s}^*$ is a discrete invariant metric on $L(A)$.

Theorem 2.1. *Let ρ be a quasimetric on $\bar{A}$, and $\rho(a,\varepsilon) = \rho(b,\varepsilon)$ for all $a, b \in A$. Then $\rho^*(ac, bc) = \rho^*(ca, cb) = \rho^*(a,b)$ for all $a, b, c \in L(A)$.*

Corollary 2.1. *If $\rho^* = \rho_s^*$, then $\rho^*(ac, bc) = \rho^*(ca, cb) = \rho^*(a,b)$ for all $a, b, c \in L(A)$.*

3 Parallel decompositions of two strings

The longest common substring and pattern matching in two or more strings is a well known class of problems. For any two strings $a, b \in L(A)$ we find the decompositions of the form $a = v_1u_1v_2u_2 \cdots v_ku_kv_{k+1}$ and $b = w_1u_1w_2u_2 \cdots w_ku_kw_{k+1}$, which can be represented as $a = a_1a_2 \cdots a_n$, $b = b_1b_2 \cdots b_n$ with the following properties:

- some a_i and b_j may be empty strings, i.e. $a_i = \varepsilon$, $b_j = \varepsilon$;
- if $a_i = \varepsilon$, then $b_i \neq \varepsilon$ and if $b_j = \varepsilon$, then $a_j \neq \varepsilon$;
- if $u_1 = \varepsilon$, then $a = v_1$ and $b = w_1$;
- if $u_1 \neq \varepsilon$, then there is a sequence $1 \leq i_1 \leq j_1 < i_2 \leq j_2 < \cdots < i_k \leq j_k \leq n$ such that:
  - $u_1 = a_{i_1} \cdots a_{j_1} = b_{i_1} \cdots b_{j_1}$, $u_2 = a_{i_2} \cdots a_{j_2} = b_{i_2} \cdots b_{j_2}$, $u_k = a_{i_k} \cdots a_{j_k} = b_{i_k} \cdots b_{j_k}$;
  - if $v_1 = w_1 = \varepsilon$, then $i_1 = 1$;
  - if $v_{k+1} = w_{k+1} = \varepsilon$, then $j_k = n$;
  - if $k \geq 2$, then for any $i \in \{2, \cdots, k\}$ we have $v_i \neq \varepsilon$ or $w_i \neq \varepsilon$.

In this case

$$l(u_1) + l(u_2) + \cdots + l(u_k) = |\{i : a_i = b_i\}|.$$

The above decomposition forms are called *parallel decompositions* of strings a and b. For any parallel decompositions $a = v_1u_1 \cdots v_ku_kv_{k+1}$ and $b = w_1u_1 \cdots w_ku_kw_{k+1}$ the number

$$E(v_1u_1 \cdots v_ku_kv_{k+1}, w_1u_1 \cdots w_ku_kw_{k+1}) = \sum_{i \leq k+1} \{\max\{l(v_i), l(w_i)\}\}$$

is called the efficiency of the given parallel decompositions. The number $E(a, b)$ is equal to the minimum of the efficiencies of all parallel decompositions of the strings a, b and is called the *common efficiency of the strings a,b*. It is obvious that $E(a, b)$ is well determined. We say that the parallel decompositions $a = v_1u_1v_2u_2 \cdots v_ku_kv_{k+1}$ and $b = w_1u_1w_2u_2 \cdots w_ku_kw_{k+1}$ are optimal if

$$E(v_1u_1v_2u_2 \cdots v_ku_kv_{k+1}, w_1u_1w_2u_2 \cdots w_ku_kw_{k+1}) = E(a, b).$$

These types of parallel decompositions are associated with the problem of approximate string matching [14]. If the decompositions $a = v_1u_1 \cdots v_ku_kv_{k+1}$ and $b = w_1u_1 \cdots w_ku_kw_{k+1}$ are optimal and $k \geq 2$, then we may consider that $u_i \neq \varepsilon$ for any $i \leq k$.

Any parallel decompositions $a = a_1a_2 \cdots a_n = v_1u_1 \cdots v_ku_kv_{k+1}$ and $b = b_1b_2 \cdots b_n = w_1u_1 \cdots w_ku_kw_{k+1}$ generate a common subsequence $u_1u_2 \cdots u_k$. The number

$$m(a_1a_2 \cdots a_n, b_1b_2 \cdots b_n) = l(u_1) + l(u_2) + \cdots + l(u_k)$$

is the *measure of similarity* of the decompositions [2], [16]. There are parallel decompositions $a = v_1u_1v_2u_2 \cdots v_ku_kv_{k+1}$ and $b = w_1u_1w_2u_2 \cdots w_ku_kw_{k+1}$ for which the measure of similarity is maximal. The maximum value of the measure of similarity of all decompositions is denoted by $m^*(a, b)$. The maximum value of the measure of similarity of all optimal decompositions is denoted by $m^\omega(a, b)$. We can note that $m^\omega(a, b) \leq m^*(a, b)$. For any two parallel decompositions $a = a_1a_2 \cdots a_n$ and $b = b_1b_2 \cdots b_n$ as in [16], we define the *penalty factor* as

$$p(a_1a_2 \cdots a_n, b_1b_2 \cdots b_n) = |\{i \leq n : a_i = \varepsilon\}| + |\{j \leq n : b_j = \varepsilon\}|$$

and

$$\begin{aligned} &M(a_1a_2 \cdots a_n, b_1b_2 \cdots b_n) \\ &\quad = m(a_1a_2 \cdots a_n, b_1b_2 \cdots b_n) - p(a_1a_2 \cdots a_n, b_1b_2 \cdots b_n) \end{aligned}$$

as the *measure of proper similarity.* The number

$$d_H(a_1a_2 \cdots a_n, b_1b_2 \cdots b_n) = |\{i \leq n : a_i \neq b_i\}|$$

is the Hamming distance between decompositions and it is another type of penalty. We have that

$$p(a_1 \cdots a_n, b_1 \cdots b_n) \leq d_H(a_1 \cdots a_n, b_1 \cdots b_n).$$

Theorem 3.1. *Let a and b be two non-empty strings, $a = a_1a_2 \cdots a_n$ and $b = b_1b_2 \cdots b_n$ be the initial optimal decompositions, and $a = a'_1a'_2 \cdots a'_q$ and $b = b'_1b'_2 \cdots b'_q$ be the second decompositions, which are arbitrary. Denote by*

$$\begin{aligned} m_0 &= m(a_1a_2 \cdots a_n, b_1b_2 \cdots b_n), & m_1 &= m(a'_1a'_2 \cdots a'_n, b'_1b'_2 \cdots b'_q), \\ p_0 &= p(a_1a_2 \cdots a_n, b_1b_2 \cdots b_n), & p_1 &= p(a'_1a'_2 \cdots a'_n, b'_1b'_2 \cdots b'_q), \\ r_0 &= d_H(a_1a_2 \cdots a_n, b_1b_2 \cdots b_n), & r_1 &= d_H(a'_1a'_2 \cdots a'_n, b'_1b'_2 \cdots b'_q), \\ M_0 &= m_0 - p_0, & M_1 &= m_1 - p_1. \end{aligned}$$

The following assertions are true

1. *If $m_1 \geq m_0$, then $M_0 \geq M_1$ and $p_1 - p_2 = 2(m_1 - m_0) + 2(r_1 - r_0)$,*
2. *If $m_1 \geq m_0$ and the second decompositions are non-optimal, then $M_0 > M_1$,*
3. *If $m_1 = m_0$ and the second decompositions are optimal, then $p_0 = p_1$ and $M_0 = M_1$,*
4. *If $m_1 \leq m_0$ and the second decompositions are non-optimal, then $m_1 - r_1 < m_0 - r_0$.*

Proof. Firstly, we prove the following claims:

Claim 1. *If $m_1 > m_0$, then $M_0 > M_1$ and $p_1 - p_2 = 2(m_1 - m_0) + 2(r_1 - r_0)$.*

Assume that $M_0 \leq M_1$. Hence,

$$m_0 - p_0 \leq m_1 - p_1, p_0 \leq r_0, p_1 \leq r_1, n = m_0 + r_0, q = m_1 + r_1.$$

Moreover, $l(a) + l(b) = 2n - p_0 = 2q - p_1$. Since $m_0 < m_1$, $r_0 \leq r_1$ and $m_0 = n - r_0 < q - r_1 = m_1$, we obtain that $n < q$. From $l(a) + l(b) = 2n - p_0 = 2q - p_1$ it follows that $p_0 < p_1$.

Let $m_1 = m_0 + \delta_0$ and $p_1 = p_0 + \delta_1$, with $\delta_0 > 0$ and $\delta_1 > 0$. Then, from assumptions, we have that $m_0 - p_0 \leq m_1 - p_1 = m_0 + \delta_0 - p_0 - \delta_1 = (m_0 - p_0) + (\delta_0 - \delta_1)$. Hence

$$\delta_1 \leq \delta_0. \tag{1}$$

On the other hand, $q = m_1 + r_1 = m_0 + \delta_0 + r_1 = n - r_0 + \delta_0 + r_1$ and $q = (n + \delta_0) + (r_1 - r_0)$. Since $p_1 = 2q - l(a) - l(b)$ and $p_0 = 2n - l(a) - l(b)$, after substitutions, we obtain that $p_1 + l(a) + l(b) = p_0 + l(a) + l(b) + 2\delta_0 + 2(r_1 - r_0)$, or $p_0 + \delta_1 = p_0 + 2\delta_0 + 2(r_1 - r_0)$, or

$$\delta_1 = 2\delta_0 + 2(r_1 - r_0). \tag{2}$$

From (2), $\delta_1 > \delta_0$, a contradiction with inequality (1). Hence $M_0 > M_1$ provided that $m_1 > m_0$. From (2) it follows that $p_1 - p_0 = 2(m_1 - m_0) + 2(r_1 - r_0)$, provided that $m_1 > m_0$. The claim is proved.

Claim 2. *If $m_1 = m_0$, then $M_0 \geq M_1$ and $p_1 - p_2 = 2(r_1 - r_0)$.*

We have that $n = m_0 + r_0$ and $q = m_0 + r_1$. Since $r_0 \leq r_1$, we have that $n \leq q$. Assume that $M_0 < M_1$. Then $m_0 - p_0 < m_0 - p_1$, $p_1 = 2q - l(a) - l(b)$ and $p_0 = 2n - l(a) - l(b)$. Hence $m_0 - 2n + l(a) + l(b) < m_0 - 2q + l(a) + l(b)$, or $-2n < -2q$ and $n > q$, a contradiction.

From Claims 1 and 2, Assertions 1-3 of the Theorem 3.1 follow immediately. Since $r_1 > r_0$, from $m_1 \leq m_0$ it follows that $m_1 - r_1 < m_0 - r_0$. Assertion 4 and Theorem 3.1 are proved. □

Remark 3.1. *From Assertions 1 and 3 of Theorem 3.1 it follows that on the class of all optimal decompositions of two strings:*

- *The maximal measure of proper similarity is attained on the optimal parallel decomposition with minimal penalties (minimal measure of similarity),*

- *The minimal measure of proper similarity is attained on the optimal parallel decomposition with maximal penalties (maximal measure of similarity).*

For any two non-empty strings there are parallel decompositions with maximal measure of similarity and optimal decompositions on which the measure of similarity is minimal.

The following example shows that there are some exotic non-optimal parallel decompositions $a = a'_1 a'_2 \cdots a'_q$ and $b = b'_1 b'_2 \cdots b'_q$, such that for optimal decompositions $a = a_1 a_2 \cdots a_n$ and $b = b_1 b_2 \cdots b_n$ we have $m_1 < m_0$, $p_1 < p_0$, and $M_1 > M_0$.

Example 3.1. *Let*

$$\begin{matrix} A & A & A & A & C & C & C \\ C & C & C & B & B & B & B \end{matrix}$$

be trivial optimal decompositions of strings a, b, and

$$\begin{matrix} A & A & A & A \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{matrix} \begin{pmatrix} C & C & C \\ C & C & C \end{pmatrix} \begin{matrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ B & B & B & B \end{matrix}$$

be their non-optimal decompositions. Then

$$m_1 = 3, r_1 = 8, p_1 = 8,$$

$$m_0 = 0, r_0 = 7, p_0 = 0.$$

In this example we have that $-5 = m_1 - r_1 > m_0 - r_0 = -7$ and $-5 = m_1 - p_1 = M_1 < M_0 = m_0 - p_0 = 0$.

Example 3.2. *Let*

$$\begin{matrix} A & B & C & D \\ C & D & E & F \end{matrix} \begin{pmatrix} E \\ E \end{pmatrix} \begin{matrix} F \\ D \end{matrix}$$

be trivial non-optimal decompositions of strings a, b and

$$\begin{matrix} A & B \\ \varepsilon & \varepsilon \end{matrix} \begin{pmatrix} C & D & E & F \\ C & D & E & F \end{pmatrix} \begin{matrix} \varepsilon & \varepsilon \\ E & D \end{matrix}$$

be their optimal decompositions. Then

$$m_1 = 1, r_1 = 5, p_1 = 0,$$

$$m_0 = 4, r_0 = 4, p_0 = 4.$$

We have that $m_1 - p_1 = M_1 > M_0 = m_0 - p_0$, and $m_1 - r_1 < m_0 - r_0$.

The above examples show that Theorem 3.1 cannot be improved in the case of $m_1 < m_0$.

Decompositions with minimal penalty and maximal proper similarity are of significant interest. Moreover, if we solve the problem of text editing and correction, the optimal decompositions are more favorable. Therefore, the optimal decompositions are the best parallel decompositions and we may solve the string match problems only on class of optimal decompositions.

Remark 3.2. *The optimal decompositions:*

- *describe the proper similarity of two strings,*
- *permit to obtain long common sub-sequences,*
- *permit to calculate the distance between strings,*
- *permit to appreciate changeability of information over time.*

4 Relations to Hamming and Levenshtein Distances

If $a, b \in L(a, b)$ and $a = a_1 a_2 \cdots a_n, b = b_1 b_2 \cdots b_m$ are the canonical decompositions, then for $m \leq n$ the number

$$d_H(a, b) = d_H(b, a) = |\{i \leq m : a_i \neq b_i\}| + n - m$$

is called the *Hamming distance* [11] between strings a and b.

The *Levenshtein distance* [12] between two strings $a = a_1 a_2 \cdots a_n$ and $b = b_1 b_2 \cdots b_m$ is defined as the minimum number of insertions, deletions, and substitutions required to transform one string to the other. A formal definition of Levenshtein's distance $d_L(a, b)$ is given by the following formula:

$$d_L(a_1 \cdots a_i, b_1 \cdots b_j) = \begin{cases} i, & \text{if j=0,} \\ j, & \text{if i=0,} \\ \min \begin{cases} d_L(a_1 \cdots a_{i-1}, b_1 \cdots b_j) + 1 \\ d_L(a_1 \cdots a_i, b_1 \cdots b_{j-1}) + 1 \\ d_L(a_1 \cdots a_{i-1}, b_1 \cdots b_{j-1}) + 1_{(a_i \neq b_j)}, \end{cases} \end{cases}$$

where $1_{(a_i \neq b_j)}$ equals to 0 if $a_i = b_j$ and to 1 otherwise.

Theorem 4.1. *$d_L(a,b) = \rho^*(a,b) \leq d_H(a,b)$ for any $a,b \in L(A)$.*

Proof. To prove the equality $d_L(a,b) = \rho^*(a,b)$, we will first prove that $d_L(a,b) \leq \rho^*(a,b)$, and then that $d_L(a,b) \geq \rho^*(a,b)$.

We begin with the observation that the parallel decompositions of two strings a, b allow more transparent evaluation of the Levenshtein distance $d_L(a,b)$. If $a = v_1u_1v_2u_2 \cdots v_n$ and $b = w_1u_1w_2u_2 \cdots w_n$ are optimal parallel decompostions, then for transformation of b to a it is sufficient to transform any w_i to v_i. The cost of transformation of w_i to v_i is $\leq \max\{l(w_i), l(v_i)\}$. Hence $d_L(a,b) \leq \rho^*(a,b)$.

The proof of the inequality $d_L(a,b) \geq \rho^*(a,b)$ is based on the Levenshtein distance formula, as well as the construction of the transformation of string a to string b. We observe that the Levenshtein distance is calculated recursively using the *memoization* matrix and *dynamic programming* technique [7, pp. 359–378]. A small snapshot of the memoization matrix calculation is presented below.

Table 1. Construction of memoization matrix for Levenshtein distance

Diag	Above
Left	min(Above + delete, Left + insert, Diag + $1_{a_i \neq b_j}$)

Distance d_L calculated on subtrings $a_1 \cdots a_i$ of string a and substring $b_1 \cdots b_j$ of string b is equal to the minimum of the following values:

- $d_L(a_1 \cdots a_{i-1}, b_1 \cdots b_j) + 1,$ (1)
- $d_L(a_1 \cdots a_i, b_1 \cdots b_{j-1}) + 1,$ (2)
- $d_L(a_1 \cdots a_{i-1}, b_1 \cdots b_{j-1}) + 1_{a_i \neq b_j}.$ (3)

Remark : the operation (1) is the delete operation, (2) is the insert operation, and (3) is the substitution operation.

Once all of the above values are calculated and the memoization matrix is filled, the distance is given by the value in the cell on the n^{th} row and m^{th} column.

The construction of the transformation of string a into string b is based on the values of the memoization matrix. At each point of the construction process, we will execute operations on both strings a and b, and obtain another pair of strings a' and b' equivalent to the initial pair a and b. We use the top-down analysis approach to describe the transformation process step by step. The process below starts with $i = n, j = m$, $p = 0$, $q = 0$ and both a', b' as empty strings:

- if when calculating $d_L(a_1 \cdots a_i, b_1 \cdots b_j)$ we used operation (1), then we deleted a character from string a at position i, which is equivalent to inserting the ε character in string b at the corresponding position. In this case, in the building process of a' and b', we put $p := p+1$, $v'_p = \{a_i\}$,$w'_p = \{\varepsilon\}$, $a' := v'_p \cup a'$, $b' := w'_p \cup b'$. Next, we proceed to calculate $d_L(a_1 \cdots a_{i-1}, b_1 \cdots b_j)$.

- if when calculating $d_L(a_1 \cdots a_i, b_1 \cdots b_j)$ we used operation (2), then we inserted the ε character in string a at position i. In this case, in the building process of a' and b', we put $p := p + 1$, $v'_p = \{\varepsilon\}$,$w'_p = \{b_j\}$, $a' := v'_p \cup a'$, $b' := w'_p \cup b'$. Next, we proceed to calculate $d_L(a_1 \cdots a_i, b_1 \cdots b_{j-1})$.

- if when calculating $d_L(a_1 \cdots a_i, b_1 \cdots b_j)$ we used operation (3), then we either substituted the character at position i of string a with the character at position j of string b, or we did not make any change in case if $a_i = b_j$. If $a_i = b_j$, we put $q =: q + 1$,$u'_q = \{a_i\}$, $a' := u'_q \cup a'$, $b' := u'_q \cup b'$. If $a_i \neq b_j$, we put $p =: p+1$, $v'_p = \{a_i\}$, $w'_p = \{b_j\}$, $a' := v'_p \cup a'$, $b' := w'_p \cup b'$. Next, we proceed to calculate $d_L(a_1 \cdots a_{i-1}, b_1 \cdots b_{j-1})$.

According to the above steps, we observe that string a' is equivalent to string a, and string b' is equivalent to b by construction. But, we also have that the decomposition $a' = v'_p u'_q v'_{p-1} u'_{q-1} \cdots u'_1 v'_1$ and $a' = w'_p u'_q w'_{p-1} u'_{q-1} \cdots u'_1 w'_1$ obtained from the above construction process, represent a parallel decomposition of strings a and b. Thus, we have that $d_L(a,b) = E(a,b) \geq \rho^*(a,b)$. This completes the proof of the equality $d_L(a,b) = \rho^*(a,b)$.

We will now prove the second part of the theorem, namely that $\rho^*(a,b) \leq d_H(a,b)$. Let $d_H(a,b) < max\{l(a), l(b)\} = n$, where $n = l(a) \geq l(b) = m$. Then $a = a_1a_2\cdots a_n, b = b_1b_2\cdots b_m$, $a_i \neq \varepsilon$ for any $i \leq n$, and or $m = 1$ and $b_1 = \varepsilon$, or $b_j \neq \varepsilon$ for any $j \leq m$. In this case $d_H(a,b) = n - |\{i \leq m : a_i = b_i\}|$ and we have the representations $a = (a_1)(a_2)\cdots(a_m)(a_{m+1}\cdots a_n)$ and $b = (b_1)(b_2)\cdots(b_m)(\varepsilon)$ which generate two parallel decompositions α, β with $E(\alpha,\beta) = d_H(a,b)$. Therefore $\rho^*(a,b) \leq E(\alpha,\beta) = d_H(a,b)$. The proof is complete.

□

Corollary 4.1. *Distance d_L is strictly invariant, i.e. $d_L(ac,bc) = d_L(ca,cb) = d_L(a,b)$ for any $a,b,c \in L(A)$.*

Remark 4.1. *The Hamming distance d_H is not invariant.*

Example 4.1. *Let $n = m + p$ and strings $a = (01)^n, b = (10)^m$, $c = (01)^p$. We obtain the following distance values for the above strings:*

$$d_L(a,b) = 2p, \rho^*(a,b) = 2p, d_H(a,b) = 2n,$$
$$d_L(ac,bc) = 2p, \rho^*(ac,bc) = 2p, d_H(ac,bc) = 2n.$$

Remark 4.2. *If $l(a) = l(b)$, then $d_H(ac,bc) = d_H(a,b)$ for any $a,b,c \in L(A)$. Additionally, the following equality always holds:*

$$d_H(ca,cb) = d_H(a,b).$$

5 Applications

First and foremost let us look at how we can apply the results of this article in information distance problems such as string search, text correction, and pattern matching. We have presented one such example in the previous section – the edit distance.

We also mentioned the problem of DNA/RNA sequence alignment, which goes back as early as 1970 [16]. Other bioinformatic applications of the distance ρ^* include phylogenetic analysis, whole genome phylogeny, and detection of acceptable mutations.

We begin this section with the pseudo-codes of two algorithms: distance calculation and decompositions alignment.

The first algorithm describes how to calculate the distance between two strings a and b. The approach is based on dynamic programming and it has a complexity of $O(mn)$, where m and n are the lengths of a and b.

Algorithm 1.

```
Description: Computes the metric $\rho^*$ on strings a and b.
Input: Strings $a, b \in L(A)$
Output: Value of $\rho^*(a, b)$
Initialisation: $m := l(a)$, $n := l(b)$, $D[m, n] := 0$
Pseudocode:
for i := 0 to m D[i,0] := i;
for j := 0 to n D[0,j] := j;
for j := 1 to n do
        for i := 1 to m do
              if a[i]= b[j] then
                        D[i,j] := D[i-1,j-1]
              else
                        D[i,j] := min(D[i-1,j] + 1,
                        min(D[i,j-1] + 1, D[i-1,j-1] + 1));
return D[m,n];
```

The algorithm that follows constructs the optimal parallel decompositions of strings a and b that give the value of distance ρ^*. This algorithm uses the memoization matrix $D[m, n]$ calculated in the previous algorithm. The idea is to traverse from the bottom right cell $D[m, n]$ to the top left cell $D[0, 0]$ and at each step to evaluate whether the minimal distance was obtained by replacement, deletion or insertion. The algorithm uses recursive *backtracking* to reconstruct all decompositions of strings a and b. We modified the classical version of the pseudo-code to print only the most optimal decomposition, instead of printing all possible paths.

Algorithm 2.

```
Description: Constructs optimal parallel decompositions
of strings $a$ and $b$.
Input: $n, m$ - current indexes in matrix $D$
        $a_r, b_r$ - recontructed decompositions
Output: Optimal parallel decompositions of strings $a$, $b$
Initialisation: Read $D[m, n]$ from Algorithm 1
Pseudocode:
if (n=0) and (m=0) then return $a_r, b_r$
if ((n>0)and(m>0)) and((D[n,m]=D[n-1,m-1]+$c_{dist}$)
        or ((D[n,m]=D[n-1,m-1]) and ($c_{dist}$=0)))
                then recOPD(n-1, m-1, $a_r + a[n]$, $b_r + b[m]$)
        else
        if (n>0) and (D[n,m]=D[n-1,m] +$cost_r$)
                then recOPD(n-1, m, $a_r$+a[n], $b_r$+$\varepsilon$)
        else
        if (m>0) and (D[n,m]=D[n,m-1] +$cost_i$)
                then recOPD(n, m-1, $a_r$+$\varepsilon$, $b_r$+b[m])
```

In the worst case scenario its complexity is $O(m+n)$ (this happens when we separately traverse the matrix horizontally and vertically). This result is achieved with the help of prioritizing the direction of analysis when traversing the matrix. We first look to the north-west and only afterwards to the northern and western cell values. We stop the reconstruction process once the algorithm reaches the cell at $D[0, 0]$. The reasoning behind this decision is to find the most optimal decomposition among all possible decompositions of strings a and b. The example that follows is a good illustration of this approach.

Example 5.1. *Let's investigate the example where $a = industry$ and $b = interest$. In this case we have $\rho^*(a, b) = 6$. The possible decompositions of strings a and b are as follows:*

industry	*in$\varepsilon\varepsilon$dustry*	*inεdεustry*	*ind$\varepsilon\varepsilon$ustry*	*inεduεstry*
interest	*interest$\varepsilon\varepsilon$*	*interest$\varepsilon\varepsilon$*	*interest$\varepsilon\varepsilon$*	*interest$\varepsilon\varepsilon$*

The first pair of parallel string decompositions is the optimal one as it has minimal string length. Another good example of two strings decomposition into their building blocks u_i, v_j, and w_j is illustrated below.

Example 5.2. *Consider the alphabet* $\bar{A} = \{\varepsilon, X, Y, Z, W\}$ *and two strings* $a = XXYYWZYX$ *and* $b = YXXWZWXY$*. For this example we obtain that* $\rho^*(a, b) = 5$ *as well as the following optimal decomposition:*

$$\begin{array}{c} \varepsilon \\ Y \end{array} \begin{pmatrix} X & X \\ X & X \end{pmatrix} \begin{array}{cc} Y & Y \\ W & Z \end{array} \begin{pmatrix} W \\ W \end{pmatrix} \begin{array}{c} Z \\ X \end{array} \begin{pmatrix} Y \\ Y \end{pmatrix} \begin{array}{c} X \\ \varepsilon \end{array}$$

Lets look at results in detection of the mutational events. We extend the parallel decompositions and present the construction of the *semiparallel decompositions.* We take into consideration the ordering $\preceq$ and the corresponding distance ρ^*_l. From this point of view, for any two strings $a, b \in L(A)$ we find the decompositions of the form $a = v_1u_1v_2u_2 \cdots v_ku_kv_{k+1}$ and $b = w_1u'_1w_2u'_2 \cdots w_ku'_kw_{k+1}$, where

- u_i, u'_i are canonical substrings of the strings a and b and u_i, u'_i may be empty strings;
- v_j is a substring of a and v_j may be an empty string;
- w_j is a substring of b and w_j may be an empty string;
- $\rho^*_l(u_i, u'_i) = 0$ for all $i \leq k$.

Like in the case with parallel decompositions, the semiparallel decompositions are optimal if

$$\rho^*_l(a, b) = \Sigma\{\rho(v_i, w_i) : i \leq k + 1\}.$$

This given interpretation of the metric and string decompositions can be used in the study of the minimum number of acceptable and unacceptable (when metric ρ^*_r is used) *mutational events* required to convert one sequence to another.

To illustrate the application of the semiparallel decomposition let us partition the strings from the previous example.

Example 5.3. *Let $a = XXYYWZYX$ and $b = YXXWZWXY$, with the alphabet $\bar{A} = \{\varepsilon, X, Y, Z, W\}$, on which we consider the classic ordering $\preceq$, meaning that $\rho_l^*(z_i, z_j) = 0$ for all $z_i, z_j \in \bar{A}$, where $z_i \preceq z_j$. This time we obtain that $\rho_l^*(a, b) = 3$, as well as the following optimal decomposition:*

$$\begin{pmatrix} X & X \\ Y & X \end{pmatrix} \begin{matrix} Y \\ X \end{matrix} \begin{pmatrix} Y \\ W \end{pmatrix} \begin{matrix} W \\ Z \end{matrix} \begin{pmatrix} Z \\ W \end{pmatrix} \begin{matrix} Y \\ X \end{matrix} \begin{pmatrix} X \\ Y \end{pmatrix}$$

For semiparallel decompositions we can define measure of similarity, penalty, and proper similarity.

Remark 5.1. *Our algorithms are effective for any quasimetric on $\bar{A}$. Some authors consider the possibility to define the generalized Levenshtein metric with distinct values $\rho(a, b)$ and $\rho(b, a)$. It is necessary to require that $\rho(a, b)$ is a quasimetric. In other cases we may obtain some confusions as will be seen from the next example.*

Example 5.4. *Let $A = \{a, b\}, \bar{A} = \{\varepsilon, a, b\}$. The following table defines the distance ρ on $\bar{A}$:*

0	*0*	*1*	ε
1	*0*	*0*	a
0	*1*	*0*	b
ε	a	b	$y \diagdown x$

In this example we have $0 = \rho(a, b) + \rho(b, \varepsilon) < \rho(a, \varepsilon) = 1$ and:

1. for $u = aba, v = ba$ we get $\bar{\rho}(u, v) = \bar{\rho}(v, u) = 0$,

2. for $u = a, v = b$ we get $\bar{\rho}(u, v) = \bar{\rho}(v, u) = 0$, when $\rho(v, u) = 1$.

Example 5.5. *Let us examine the example from [16] in the context of the results achieved. We have strings $a = AJCJNRCKCRBP$ and $b = ABCNJROCLCRPM$ for which there are eight pairs of optimal decompositions. We present two of them, the shortest and the longest:*

$$\begin{pmatrix} A \\ A \end{pmatrix} \begin{matrix} J \\ B \end{matrix} \begin{pmatrix} C \\ C \end{pmatrix} \begin{matrix} \varepsilon \\ N \end{matrix} \begin{pmatrix} J \\ J \end{pmatrix} \begin{matrix} N & R \\ R & O \end{matrix} \begin{pmatrix} C \\ C \end{pmatrix} \begin{matrix} K \\ L \end{matrix} \begin{pmatrix} C & R \\ C & R \end{pmatrix} \begin{matrix} B & P \\ P & M \end{matrix}$$

$$\begin{pmatrix} A \\ A \end{pmatrix} \begin{smallmatrix} J \\ B \end{smallmatrix} \begin{pmatrix} C \\ C \end{pmatrix} \begin{smallmatrix} J \\ \varepsilon \end{smallmatrix} \begin{pmatrix} N \\ N \end{pmatrix} \begin{smallmatrix} \varepsilon \\ J \end{smallmatrix} \begin{pmatrix} R \\ R \end{pmatrix} \begin{smallmatrix} \varepsilon \\ O \end{smallmatrix} \begin{pmatrix} C \\ C \end{pmatrix} \begin{smallmatrix} K \\ L \end{smallmatrix} \begin{pmatrix} C & R \\ C & R \end{pmatrix} \begin{smallmatrix} B \\ \varepsilon \end{smallmatrix} \begin{pmatrix} P \\ P \end{pmatrix} \begin{smallmatrix} \varepsilon \\ M \end{smallmatrix}$$

For the first pair we have $\rho^* = 7$, $m = 6$, $p = 1$, and $M = 5$. For the second pair we have $\rho^* = 7$, $m = 8$, $p = 5$, and $M = 3$. Our algorithms allow us to calculate all optimal decompositions with distinct measure of similarity. Authors from [16] prefer the second pair of decomposition since it has maximal possible measure of similarity. We consider more preferable the first pair, which has the maximal proper similarity.

6 Conlusion

We showed that there are invariant distances on $L(A)$ closely related to Levenshtein's distance, which help us solve various problems in mathematics, computer science, and bioinformatics. The results can be applied in different areas such as data correction of signals transmitted over channels with noise, finding matching DNA sequence after mutations, text searching with possible typing errors, and estimation of dialect pronunciations proximity [8], [14]. For construction of the matching sequence we propose the method of optimal decompositions of strings, priority of which is confirmed by Theorem 3.1. Our distances of ρ^* type can be defined for distinct values $\rho(a, b)$ of strings a,b, in general, and for $\rho(a, b) \neq \rho(b, a)$. In such a case, the metric can be used in solving the stable marriage problem [10].

References

[1] A. V. Arhangel'skii, "Mappings and spaces," *Uspekhi Mat. Nauk,* vol. 21, no. 4, pp. 133–184, 1966. [in Russian] (English translation: *Russian Math. Surveys,* vol. 21, no. 4, PP. 115–162, 1966).

[2] V. B. Barahnin, V. A. Nehaeva, and A. M. Fedotov, "Prescription of the similarity measure for clustering text documents," *Vestnik Novosib. Gos. Univ., Ser.: Informacionnye tehnologii*, vol. 1, pp. 3–9, 2008. [in Russian]

[3] M. M. Choban, "The theory of stable metrics," *Math. Balkanica,* vol. 2, pp. 357–373, 1988.

[4] M. M. Choban, "Some topics in topological algebra," *Topol. Appl.,* vol. 54, pp. 183–202, 1993.

[5] M. M. Choban and I. A. Budanaev, "Distances on Monoids of Strings and Their Applications," In *Conference on Mathematical Foundations of Informatics: Proceedings MFOI2016, July 25-29, 2016, Chisinau, Republic of Moldova*, Chişinău, Institute of Mathematics and Computer Science, pp. 144–159, 2016. ISBN: 978–9975–4237–4–8

[6] M. M. Choban and L. L. Chiriac, "On free groups in classes of groups with topologies," *Bul. Acad. Ştiinţe Repub. Moldova, Matematica,* no. 2-3, pp. 61–79, 2013.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms.* (3rd ed.), MIT Press and McGraw-Hill, 2009. ISBN: 0–262–03384–4.

[8] M. M. Deza and E. Deza, *Encyclopedia of Distances*, Berlin: Springer, 2009. ISBN: 978-3-642-00233-5; e-ISBN: 978-3-642-00234-2; DOI 10.1007/978-3-642-00234-2.

[9] M. I. Graev, "Free topological groups," *Izv. Akad. Nauk SSSR Ser. Mat.,* vol. 12, no. 3, pp. 279–324, 1948. [in Russian] (English translation: *Amer. Math. Soc. Transl.* (1), vol. 8, pp. 305–364, 1962).

[10] D. Gusfield and R. W. Irving, *The Stable Marriage Problem: Structure and Algorithms*, Cambridge, MIT Press, 1989. ISBN: 9780262515528.

[11] R. W. Hamming, "Error Detecting and Error Correcting Codes," *The Bell System Technical Journal*, vol. 29, no 2, pp. 147–160, 1952.

[12] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *DAN SSSR,* vol. 163, no 4, pp. 845–848, 1965. [in Russian] (English translation: *Soviet Physics – Doklady*, vol. 10, no. 8, pp. 707–710, 1966).

[13] A. A. Markov, "On free topological groups," *Izv. Akad. Nauk. SSSP, Ser. Matem.*, vol. 9, no. 1, pp. 3–64, 1945. [in Russian]

(English translation: *Amer. Math. Soc. Transl.* (1), vol 8, no. 1, pp. 195–272, 1962).

[14] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys,* vol. 33, no. 1, pp. 31–88, 2001.

[15] S. I. Nedev, "o-metrizable spaces," *Trudy Moskov. Mat.Ob-va,* vol. 24, pp. 213–247, 1974. [in Russian] (English translation: *Trans. Moscow Math. Soc.*, vol. 24, pp. 213–247, 1974).

[16] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology,* vol. 48, no 3, pp. 443–453, 1970.

[17] S. Romaguera, M. Sanchis and M. Tkachenko, "Free paratopological groups," *Topology Proceed.*, vol. 27, no 2, pp. 613–640, 2003.

[18] C. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal,* vol. 27, pp. 379–423, pp. 623–656, 1948.

Mitrofan Choban, Ivan Budanaev, Received September 22, 2016

Mitrofan Choban
Professor, Doctor of Science,
Academician of the Academy of Science of Moldova
Tiraspol State University, Republic of Moldova
str. Iablochkin 5, Chisinau, Moldova
Phone: +373 22 754906
E–mail: `mmchoban@gmail.com`

Ivan Budanaev
Doctoral School of Mathematics and Information Science
Institute of Mathematics and Computer Sciences of ASM
Tiraspol State University, Republic of Moldova
str. Academiei, 3/2, MD-2028, Chisinau, Moldova
Phone:+373 60926999
E–mail: `ivan.budanaev@gmail.com`

Computer Science Journal of Moldova, vol.**24**, no.3(72), 2016

Insertion Modeling and Its Applications

Alexander Letichevsky, Oleksandr Letychevskyi
Vladimir Peschanenko

Abstract

The paper relates to the theoretical and practical aspects of insertion modeling. Insertion modeling is a theory of agents and environments interaction where an environment is considered as agent with a special insertion function. The main notions of insertion modeling are presented. Insertion Modeling System is described as a tool for development of different kinds of insertion machines. The research and industrial applications of Insertion Modeling System are presented.

Keywords: process algebra, insertion modeling, formal models, verification.

1 Introduction

Insertion modeling is an approach for research of distributed multiagent systems and for development of tools for verification of its models. The first papers about insertion modeling were published about 20 years ago [1], [2]. A model of the agents and environments interaction which helps the insertion function notion was presented in these papers.

The main sources of insertion modeling are in a model of interacting control and operating automata, which were found by V.M. Glushkov [3], [4] for the computers description. An algebraic abstraction of this model has been studied in the theory of discrete transformers and has provided some important results on the problem of equivalence of programs, their equivalent transformation and optimization. Macroconveyor models of parallel computing [5] are even closer to the model of interaction between agents and environments. In these models processes corresponding to parallel processors can be regarded as agents interacting in distributed environment data structures. In recent years the insertion simulation becomes a tool for development applications of verification of systems requirements and specifications of distributed interacting systems [6]–[10].

Another source of insertion modeling is a general theory of interacting information processes, which was created in previous century and is the basis for modern research in this area. It includes CCS (Calculus of Communicated Processes) [11], [12] and π-calculus of R. Milner [13], CSP (Communicated Sequential Processes) of T. Hoar [14], ACP (Algebra of Communicated Processes) [15] and many other different branches of these basic theories. A quite complete review of the classical theory of processes is represented in the handbook on algebra processes [16], which was published in 2001.

The second section is defined by the algebra of behaviors and the bisimulation equivalence of transition systems. The third section introduces the concepts of environment and agents features. The fourth section is devoted to the Insertion modeling system. The fifth section deals with the application of insertion modeling, and finally discusses the possibilities for further development and possible new applications.

2 Behavior algebras

2.1 Transition System

A common approach for describing the dynamics of systems in modern computer science is the notion of transition system, which is defined by sets of states and transitions. Usually this notion is enriched by the additional structures, the most important of which are the transition labelling (labelled transition system introduced by Park [8] to describe the behavior of automata on infinite words). The basic notion in the insertion modeling is an attribute transition system [10], which is defined as follows:

$$< S, A, U, T, \varphi > \tag{1}$$

where S is a set of states, A is a set of actions, which are used for marking the transition, U is a set of labeled attributes, which are used for marking the states, T is transition relation: $T \subseteq S \times A \times S \cup S \times S$, which consists of labeled transitions $s \xrightarrow{a} s'$ and not labeled transitions $s \rightarrow s'$.

Function $\varphi : S \rightarrow U$ is a function of labeling states. U could be defined as a set $U = D^R$ of mapping of a set R of attributes in a set of data D (a range of values of attributes) or as a $U = \left(D_\xi^{R_\xi}\right)_{\xi \in \Xi}$, where Ξ is a set of data types. A formula of some logic language $L(R)$ is used for symbolic modeling as attributes labels $U \subseteq L(R)$, where R is a set of attributes or a set of attributes with types $R = (R_\xi)_\xi$. It could be interpreted by first order language, which could be expanded by some temporal logic modality. States labeling is considered as some equivalence for symbolic case.

Transition system can also be configured by highlighting some specific sets of states from the set of states S. Among them there are the most important set of initial states S_0, a set of termination states S_Δ and a set of non-defined states $S_\perp$. The last one is used in the theory to determine the relationship of approximation and to build infinite systems in form of finite limits.

As in the theory of automata states the transition systems are considered as some equivalence. In the branch of different equivalences which are considered in the [19] the most important are the trace and bisimulation equivalence (strongest and weakest respectively).

For simplicity, we consider only the system with no hidden transitions. History of operation of attribute transition system is defined as a finite or infinite sequence $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$ of transitions, and a trace corresponding to this history is defined as a sequence $\varphi(s_1) \xrightarrow{a_1} \varphi(s_2) \xrightarrow{a_2} \ldots$

The trace is called maximal if it can't be continued. Let $L(s)$ be the set of all maximal traces which are started in a state s. The states s and s' are called *trace equivalent*, if $L(s) = L(s')$.

Bisimulation equivalence is weaker than trace and defined by thinner manner. A binary relation R on the set of states of the system 1 is called a *relation of bisimulation*, if for every pair (s, s') of its states the following rules are true:

- $(s, s') \in R \Rightarrow \varphi(s) = \varphi(s')$,
- $(s, s') \in R \wedge s \xrightarrow{a} t \Rightarrow \exists t' \Big((t, t') \in R \wedge s' \xrightarrow{a} t'\Big)$,
- $(s, s') \in R \wedge s' \xrightarrow{a} t' \Rightarrow \exists t \Big((t, t') \in R \wedge s \xrightarrow{a} t\Big)$.

The states s and s' of the system 1 are called *bisimulation equivalent* if a bisimulation relation R exists, such as $(s, s') \in R$.

Equivalence of systems is usually defined in terms of their equivalence of states. For example, for the initial systems two systems are declared to be equivalent, if the initial state of each of them is equivalent to the initial state of another. The difference between the trace and bisimulation equivalence occurs only in the case of non-deterministic systems. A labeled system is called deterministic if

$$s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'' \Rightarrow \big(s', s''\big) \in R.$$

Two deterministic systems are bisimulation equivalent if and only if they are trace equivalent.

2.2 Behavior algebra

In contrast to the trace equivalence for which the invariant of equivalence (a set of traces) is given together with the definition, the invariant of bisimulation equivalence is not so obvious. In insertion modeling as invariants (generally infinite) expressions or system of equations in algebra behavior are used. A behavior algebra is arranged simply. It is a two-sorted algebra $< U, A >$, the first component U is a set of behaviors, and the second A is a set of actions. The signature of the behavior algebra consists of two operations, one relation and three constants. The first operation $a.u$ is called *prefixing*. Its arguments are action a and behavior u. The result is a new behavior. The second operation is the operation of a non-deterministic choice of $u + v$. This is a binary operation defined in the set of behaviors. It is commutative, associative and idempotent. The behavior algebras constants are the successful termination Δ, undefined behavior $\perp$ and the deadlock behavior 0, which is a neutral element of non-deterministic choice. On the set of behaviors a binary relation of approximation $\subseteq$ is defined, which is a relation of a partial order with the smallest element Δ. Prefixing and non-deterministic choice operation are monotonous and continuous with respect to this relation. The main role is played by a full behavior algebra $F(A)$, which contains all limits of directed sets and, therefore, a theorem on the minimal fixed point is applied. The exact structure algebra $F(A)$ (for any, including infinite number of actions) is presented in [17].

In the full algebra of behavior each element has the following representation:

$$u = \sum_{i \in I} a_i.u_i + \varepsilon_u,$$

which is uniquely defined (up to commutativity and associativity), if all $a_i.u_i$ are different.

With each state s of transition system a behavior $beh(s) = u_s$ of system S is associated as the lowest component of the system of equations

$$u_s = \sum_{s \xrightarrow{a} t} a.u_t + \varepsilon_s,$$

where $\varepsilon_s = 0, \Delta, \perp, \Delta + \perp$ depends on the conditions $s \notin S_\Delta \cup S_\perp, s \in S_\Delta \setminus S_\perp, s \in S_\perp \setminus S_\Delta, s \in S_\Delta \setminus S_\perp$, respectively. The main theorem, which characterizes a bisimulation equivalence claims that *two states are bisimulation equivalent if and only if they have equal behavior.* Other approaches to the characterization of a bisimulation equivalence can be found in [20].

3 Agents and Environments

Agent is a transition system, which defines a state up to bisimulation equivalence.

Environment is an agent that has an insertion function. In additional environments there is $< E, C, A, Ins >$, where E is a set of states of an environment, C is a set of actions which could be inserted into an environment, $Ins : E \times F(a) \rightarrow E$ is an insertion function. Since the states transition systems are considered as bisimulation equivalence, they can be identified with the behavior and talk about continuity of an insertion function. The main requirement for the environment is a continuity of an insertion function. This assumption implies a number of useful effects. For example, the fact that an insertion function can be set with the help of systems of rewriting rules as the minimal fixed point of the system of functional equations. A result $Ins(e, u)$ of agents insertion, which is in a state u, is defined as $e\,[u]$. Assuming $e\,[u, v] = (e\,[u])\,[v]$ we get the opportunity to talk about the combination of agents that are inserted in an environment and to consider the state of an environment of the form $e\,[u_1, u_2, \ldots]$. Taking into account that an environment is an agent, it can be inserted in a top level environment, considering the multi-level environments like $e\left[e_1\,[u_{11}, u_{12}, \ldots]_{E_1}, e_2\,[u_{21}, u_{22}, \ldots]_{E_2}, \ldots\right]$, where definition $e\,[u_1, u_2, \ldots]_E$ clearly shows environment E, which belongs to the state e. The behavior u of initialized agent defines relation $[u] : E \rightarrow T$, which is defined by the relation $[u]\,(e) = e\,[u]$ and an insertional equivalence of agents $\sim_E$ relative to environment E, which is defined by relation $u \sim_E v \Longleftrightarrow [u] = [v]$. This equivalence is usually weaker than bisimulation and plays main role in the applications, because a transformation of algorithms and software implementations of the agents which live in some environment should be executed as transformation which saves insertional equivalence.

In [17] some classification of the insertion functions and the obtained results on a reduction of the complex class of functions to the simple ones are presented.

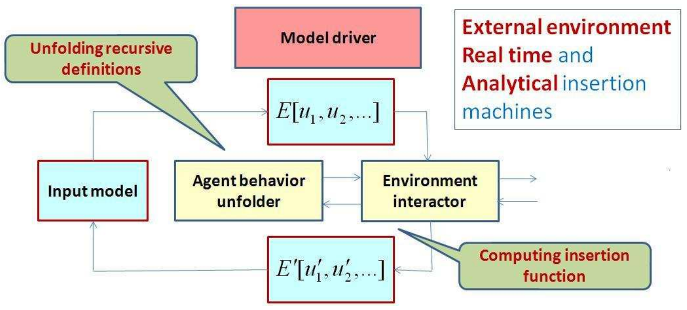


Figure 1. Architecture of Insertion Machine

4 Insertion Modeling System

Insertion modeling system [21] is an environment for the development of insertion machines and performing experiments with them. The notion of insertion machine was used as a tool for programming with some special class of insertion functions. Later this notion was extended for wider area of applications, different levels of abstraction, and multilevel structures.

Insertion model of a system represents this system as a composition of environment and agents inserted into it. Contrariwise the whole system as an agent can be inserted into another environment. In this case we talk about internal and external environment of a system. Agents inserted into the internal environment of a system themselves can be environments with respect to their internal agents. In this case we talk about multilevel structure of agent or environment and about high level and low level environments.

The general architecture of insertion machine is represented in Figure 1.

The main component of insertion machine is model driver, the component which controls the machine movement along the behavior tree of a model. The state of a model is represented as a text in the input language of insertion machine and is considered as an algebraic expression. The input language includes the recursive definitions of agent behaviors, the notation for insertion function, and possibly some compositions for environment states. Before computing insertion function the state of a system must be reduced to the form $E\,[u_1, u_2, ...]$. This functionality is performed by the module called agent behavior unfolder. To make the movement, the state of environment must be reduced to the normal form

$$\sum_{i\in I} a_i.E_i + \varepsilon,$$

where a_i are actions, E_i are environment states, ε is a termination constant. This functionality is performed by the module environment interactor. It computes the insertion function calling if it is necessary the agent behavior unfolder. If the infinite set I of indices in the normal form is allowed, then the weak normal form $a.F + G$ is used, where G is arbitrary expression of input language.

Two kinds of insertion machines are considered: *real time* or *interactive* and *analytical* insertion machines. The first ones exist in the real or virtual environment, interacting with it in the real or virtual time. Analytical machines are intended for model analyses, investigation of its properties, solving problems etc. The drivers for two kinds of machines correspondingly are also divided into interactive and analytical drivers.

Interactive driver after normalizing the state of environment must select exactly one alternative and perform the action specified as a prefix of this alternative. Insertion machine with interactive driver operates as an agent inserted into external environment with insertion function defining the laws of functioning of this environment.

Analytical insertion machine as opposed to interactive one can consider different variants of making decision about performed actions, returning to choice points (as in logic programming) and consider different paths in the behavior tree of a model. The model of a system can include the model of external environment of this system, and the driver performance depends on the goals of insertion machine. In the general case analytical machine solves the problems by search of

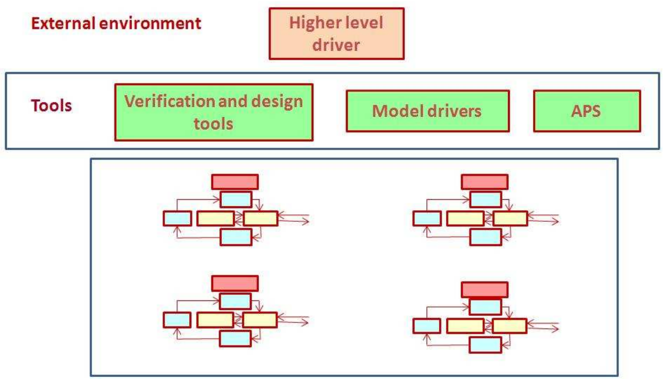


Figure 2. Architecture of Insertion Modeling System

states, having the corresponding properties (goal states) or states in which given safety properties are violated. The external environment for insertion machine can be represented by a user who interacts with insertion machine, sets problems, and controls the activity of insertion machine.

Analytical machine enriched by logic and deductive tools is used for generating traces of symbolic models of systems. The state of symbolic model is represented by means of properties of the values of attributes rather than their concrete values.

General architecture of insertion modeling system is represented in Figure 2.

High level model driver provides the interface between the system and external environment including the users of the system. Design tools based on Algebraic Programming system *APS* [21] are used for the development of insertion machines and model drivers for different application domains and modeling technologies. Verification tools are used for the verification of insertion machines, proving their properties statically or dynamically. Dynamic verification uses generating symbolic model traces by means of special kinds of analytical model drivers and deductive components.

The repository of insertion machines collects already developed machines and their components which can be used for the development of new machines as their components or templates for starting. Special library of $APLAN$ functions supports the development and design in new projects. The $C++$ library for IMS supports $APLAN$ compilers and efficient implementation of insertion machines. Deductive system provides the possibility of verification of insertion models [22].

5 Applications

Based on the ideas of insertion modeling the Verification of Requirement Specification (VRS) system was developed by researchers from V.M. Glushkov Institute of Cybernetics of National Academy of Science of Ukraine.

The language of basic protocols is implemented in VRS, which supports the usage of numerical attributes and symbolic types, arrays, lists and functional data types. The deductive system provides proof of the identities in the theory of the first order logic, which is the integration of theories of real and integer linear inequalities, free uninterpreted function symbols and theory query. Symbolic modeling in the VRS is based on satisfiability checking and predicate transformer functions [23].

Proving Programming System is a new and modern system that is designed to maintain a high level of training of qualified specialists in programming. This system is created based on the Insertion Modeling system and Algebraic Programming System which was developed at the V.M. Glushkov Institute of Cybernetics of NAS of Ukraine with the participation of authors from Kherson State University. This system implements Floyds algorithm of proving partial correctness of annotated programs [24].

Insertion Modeling system was successfully used for implementation of theory for building of invariants of the models [25] and loops in software [26], for the set of school computer algebra systems[27], for interleaving reduction in symbolic insertion models [28].

6 Conclusion

In this paper the main notions of insertion modeling are given. Insertion modeling theory is one of the most general theories of process algebra. Its main difference consists in the fact that an environment is considered as an agent with insertion function. Insertion Modeling System was developed for supporting this theory in practice and is used for developing industrial and research insertion machines.

In the nearest future we are planning to use such theory and system for research of models which came from law and economics.

References

[1] D.R. Gilbert and A.A. Letichevsky, "A universal interpreter for nondeterministic concurrent programming languages," In *Fifth Compulog network area meeting on language design and semantic analysis methods*, M. Gabbrielli, Ed. September, 1996. [Online]. Available: http://trove.nla.gov.au/work/7032623.

[2] A. Letichevsky and D. Gilbert, "Interaction of agents and environments," *Recent trends in Algebraic Development technique, LNCS*, vol. 1827, pp. 311–328, September, 2000.

[3] V.M. Glushkov, "The automata theory and design issues digital machines structures," *Cybernetics*, vol. 1, pp. 3–11, 1965. (in Russian)

[4] V. M. Glushkov and A. A. Letichevsky, "Theory of algorithms and descrete processors," *Advances in Information Systems Science (J. T. Tou, Ed.)*, vol. 1, pp. 1–58, Plenum Press, 1969.

[5] M. Kwan, "The design of the ICE encryption algorithm," In *The 4th International Workshop, Fast Software Encryption - FSE '97 Proc. LNCS*, vol. 1267, pp. 69–82, 1997.

[6] Y.V. Kapitonova and A.A. Letichevsky, *The mathematical theory of digital systems*, Moscow, Science, 1988, 295 p. (in Russian)

[7] A. Letichevsky, J. Kapitonova, A. Letichevsky Jr., V. Volkov, S. Baranov, V.Kotlyarov and T. Weigert, “Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications,” In *ISSRE 2004, WITUL (Workshop on Integrated reliability with Telecommunications and UML Languages)*, Rennes, November, 2005, pp. 117–132.

[8] S. Baranov, C. Jervis, V. Kotlyarov, A. Letichevsky and T. Weigert, “Leveraging UML to deliver correct telecom applications in UML for Real,” In *Design of Embedded Real-Time Systems*, L.Lavagno, G. Martin and B. Selic, Eds. Kluwer Academic Publishers, 2003, pp. 323–342.

[9] J. Kapitonova, A. Letichevsky, V. Volkov and T. Weigert, “Validation of Embedded Systems,” In *The Embedded Systems Handbook*, R. Zurawski, Ed. CRC Press, Miami, 2005, pp.6-1–6-26.

[10] A.A.Letichevsky, J.V.Kapitonova, V.A.Volkov, A.A.Letichevsky, jr., S.N.Baranov, V.P.Kotlyarov and T.Weigert, “System Specification with Basic Protocols,” *Cybernetics and System Analyses*, vol. 4, 2005, pp. 3–21.

[11] R. Milner, *A Calculus of Communicating Systems*, (LNCS), vol. 92, 1980.

[12] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[13] R. Milner, “The polyadic π-calculus: a tutorial,” Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, Tech. Rep. ECSLFCS91180, 1991.

[14] C. A. R. Hoare, “Communicating Sequential Processes,” *Prentice Hall*, 1985.

[15] J. A. Bergstra and J. W. Klop, “Process algebra for synchronous communications,” *Information and Control*, vol. 60 (1/3), pp. 109–137, 1984.

[16] J. A. Bergstra, A. Ponce and S. A. Smolka, Eds. *Handbook of Process Algebra*, North- Holland, 2001.

[17] A.Letichevsky, "Algebra of behavior transformations and its applications," *Structural theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry*, V.B.Kudryavtsev and I.G.Rosenberg, Eds. vol. 207, pp. 241–272, 2005.

[18] D. Park, "Concurrency and automata on infinite sequences," *LNCS*, vol. 104, 1981, pp. 167–183.

[19] R. J. Glabbeek, *The linear time-branching time spectrum the semantics of concrete, sequential processes*, (Handbook of Process Algebra), J. A. Bergstra, A. Ponce and S. A. Smolka, Eds. North-Holland, 2001.

[20] M. Roggenbach, M. Majster-Cederbaum, "Towards a unified view of bisimulation: a comparative study," *TCS*, vol. 238, pp. 1–130, 2000.

[21] A.A. Letichevsky, O.A.Letychevskyi and V.S. Peschanenko, "Insertion Modeling System," *LNCS*, vol. 7162, pp. 262–274, 2011.

[22] A.A. Letichevsky, O.A. Letychevskyi and V.S. Peschanenko, "Algebraic Programming System APS and Insertion Modeling System IMS, 2016. [Online]. Available: http://www.apsystems.org.ua.

[23] A. Letichevsky, O. Letychevskyi, V. Peschanenko and T. Weigert, "Insertion Modeling and Symbolic Verification of Large Systems," *Lecture Notes in Computer Science*, vol. 9369, pp. 3–18, 2015.

[24] O. Letychevskyi, M. Morokhovets and V. Peschanenko, "System of Provable Programming," *Control Systems and Computers*, vol. 6, pp. 64-71, 2012. (in Russian)

[25] A. Letichevsky, A. Godlevsky, A. Guba, A. Kolchin, O. Letichevkyi and V. Peschanenko, "Usage of Invariants for Sym-

bolic Verification of Requirements," *Risc-Linz report series*, vol. 13-06, pp. 124–124, 2013.

[26] M.S. Lvov, "A Method of Proving the Invariance of Linear Inequalities for Linear Loops," *Cybernetics and Systems Analysis*, vol. 50, no. 4, pp. 643–648, 2014.

[27] *National Projects of Kherson State University*, 2016. [Online]. Available: http://www.kspu.edu/About/DepartmentAndServices/DSAICI/internationalprojects/NationalProjects.aspx?lang=en.

[28] A. Letichevsky, O. Letychevskyi and V. Peschanenko, "An Interleaving Reduction for Reachability Checking in Symbolic Modeling," *In: Proc. 11-th Int. Conf. ICTERI 2015, Lviv, Ukraine*, Ermolayev, V. et al., Eds. May, 2015, pp. 338–353. Available: http://ceur-ws.org/Vol-1356/paper_74.pdf.

Alexander Letichevsky, Oleksandr Letychevskyi,
Vladimir Peschanenko

Received October 13, 2016

Alexander Letichevsky
V.M. Glushkov Institute of Cybernetics of National Academy of Sciences of Ukraine
40 Glushkov ave., Kyiv, Ukraine, 03187
Phone: +38 (044) 526-00-58
E–mail: aaletichevsky78@gmail.com

Oleksandr Letychevskyi
V.M. Glushkov Institute of Cybernetics of National Academy of Sciences of Ukraine
40 Glushkov ave., Kyiv, Ukraine, 03187
Phone:+38 (044) 526-00-58
E–mail: lit@iss.org.ua

Vladimir Peschanenko
Kherson State University of Ukraine
27, 40 rokiv Zhovtnya St., Kherson, Ukraine 73000
Phone: +38 (0552) 32-67-68
E–mail: vladim@ksu.ks.ua

Computer Science Journal of Moldova, vol.**24**, no.3(72), 2016

Proving Properties of Programs on Hierarchical Nominative Data*

Ievgen Ivanov, Mykola Nikitchenko,
Volodymyr G. Skobelev

Abstract

In the paper we develop methods for proving properties of programs on hierarchical nominative data on the basis of the composition-nominative approach. In accordance with this approach, the semantics of a program is a function on nominative data constructed from basic operations using compositions (operations on functions) which represent programming language constructs. Nominative data can be considered as a class of abstract data models which is able to represent many concrete types of structured and semistructured data that appear in programming. Thus, proofs of properties of programs depend on proofs of properties of compositions and basic operations on nominative data.

To simplify the parts of such proofs that deal with program compositions we propose to represent compositions of programs on nominative data using effective definitional schemes of H. Friedman. This permits us to consider proofs in data algebras (which are simpler to derive, automate, etc.) instead of proofs in program algebras. In particular, we demonstrate that the properties of programs related to structural transformations of data can be reduced to the data level. The obtained results can be used in software development and verification.

Keywords: Programming language semantics, algorithmic algebras, nominative data, composition, Friedman scheme.

* This work was supported in part by the project "Development of logic-algorithmic methods for investigation of formal models of natural languages" of Taras Shevchenko National University of Kyiv, Ukraine, Ref. Nr. 0116U004780

1 Introduction

The importance of the problem of elaborating the theory of programming and connecting it with software development practice was recognized by many researchers. In particular, it was mentioned as one of the grand challenges in computing by T. Hoare in his influential paper [1]. More generally, one may argue that development of tools and methods of program analysis that can make sure that it has the desired runtime properties before the program is run (e.g. model checking, verification against a formal specification using logical methods and automatic theorem provers, etc.) is a very important research topic.

In this paper we consider one aspect of the mentioned problem that is concerned with simplification of the process of proving properties of programs which operate on complex data structures (e.g. records, multidimensional arrays, trees, etc.). The types of properties we consider are the properties which can be described by special predicates on input and output data of a program: if i is an input data, o is the output of a program on the input i, then the property can be formulated as $P(i, o)$, where P is a predicate such that its truth domain is a transitive binary relation. Another kind of property which we consider is monotonicity of a program as a function from input data to output data with respect to some preorder relation on data (or, in particular, equivalence relation).

The ability to check such properties is useful in many cases. For example, in terms of such properties one can formulate the statement of partial correctness of a cyclic program in Floyd-Hoare logic [2], [3], the statement about correctness of implementation of a procedure/function/method with respect to its specification in contract programming [4], the statement about preservation of the structure or content of the data by a program (if a program is intended to perform a certain transformation of its input like optimization, translation, compilation, etc.).

Usually a proof of such a property of a program can be done by induction on the program structure and it ultimately reduces to a number of proofs of properties of similar type for the basic operations on data and for programming constructs (compositions – such as sequential execution, branching, cycle, etc.).

However, the complexity of such a proof can be lowered, if one is able to reduce the proofs of properties of different compositions to proofs of properties of operations on data. In this paper we propose a way of achieving such a reduction by representing compositions using effective definitional schemes of H. Friedman. The achieved reduction permits us to consider proofs in data algebras (which are simpler to derive, automate, etc.) instead of proofs in program algebras. Using this approach we demonstrate that the properties of programs related to structural transformations of data can be simplified by reducing them to the data level.

An informal description of how our approach works is given below. Consider the following version of the greatest common divisor computation algorithm (GCD algorithm).

Input: x, y (integer), local variables: a, b (integer)

```
a:=x; b:=y;
while a ≠ b do begin
if a>b do a:=a-b;
if b>a do b:=b-a;
end
```

The program has a state which can be represented as an association between variable names a, b, x, y and integer values e.g. its initial state can be $d = [x \mapsto 10, y \mapsto 5, a \mapsto 10, b \mapsto 5]$. Such an association can be formalized as a nominative data. We will denote the value associated with a name x in a data d as $d(x)$. The program state changes during execution, however, all operations which change the state (assignments $a := a - b$, $b := b - a$) leave the value $gcd(a, b)$ unchanged. Let us define as P the input-output relation of the program, i.e. the set of pairs (d_i, d_o) of nominative data such that if d_i is the initial state of the program, then d_o is the final state of the program. Let us denote as $\leqslant$ the following binary relation: if d_1, d_2 are nominative data which give values to the names a, b, x, y, then $d_1 \leqslant d_2$ if and only if

$$d_1(x) = d_2(x) \wedge d_1(y) = d_2(y) \wedge gcd(d_1(a), d_1(b)) = gcd(d_2(a), d_2(b)).$$

Obviously, $\leqslant$ is a transitive relation. If we can show $(d_i, d_o) \in P$ implies that $d_i \leqslant d_o$, then we can easily conclude that the program is partially correct, i.e. if the program terminates, then $a = b = gcd(x, y)$, so it indeed computes the greatest common divisor of x and y. Thus for proving the property of partial correctness of this program it is sufficient to show that its input-output mapping (P) is an increasing function in the sense of some transitive relation. Now our observation is that to prove that P defines an $\leqslant$-increasing function, it may be sufficient to check that all basic transformations of data (e.g. assignments) which appear in the program's source code are $\leqslant$-increasing without analyzing how these basic transformations are composed using various programming constructs like the conditional operator (if) and cycle operator (while).

The benefit of this approach to proving partial correctness (or other properties) of a program is that when this approach is applicable, it gives its user the ability to reuse the proof of a program's property when a program undergoes various changes/improvements/optimizations that do not change the set of basic operations (transformations) of data which appear in it. For example, if we modify the above mentioned version of the GCD algorithm in the following way:

Input: x, y (integer), local variables: a, b (integer)

```
a:=x; b:=y;
while a ≠ b do begin
while a>b do a:=a-b;
while b>a do b:=b-a;
end
```

then the modified algorithm still consists of the same basic operations as the original one, so it is still partially correct.

To describe formally our approach we need a formal model of complex data structures used in programming and of programs that operate on such data. We choose the formal models of data, programs and programming constructs provided by the composition-nominative approach [5]. In accordance with this approach, the denotational semantics of a program is a function on nominative data [5] (a class of abstract models of data which is able to represent many concrete types of structured and semistructured data that appear in programming) constructed from basic operations on nominative data using compositions (operations on functions) which represent programming language constructs. The model of nominative data is particularly suitable for our purposes since, it was demonstrated [11] that nominative data with complex names and/or values can adequately represent many data structures used in programming practice. For example, the data representable in JSON (JavaScript Object Notation) data-interchange format, which is very popular in web development, can be naturally modeled using nominative data.

Besides data formalization, we will need the formalizations of common programming language constructs in terms of operations on programs on nominative data. As such formalizations we will use the operations of the Associative Nominative Glushkov Algorithmic Algebra (ANGAA) introduced in [11] which is a rich, but tractable generalization of Glushkov algorithmic algebras [14] to programs on complex data structures.

We give the necessary preliminaries about the composition-nominative approach in the next section.

We will use the following notation:

- $f : A \to B$ denotes a total function from a set A to B;
- $f : A \tilde{\to} B$ denotes a partial function from a set A to B;
- $f(x) \downarrow$ means that a partial function f is defined on a value x;
- $f(x) \uparrow$ means that a partial function f is undefined on a value x;
- $\cong$ denotes the strong equality: $f(x) \cong g(x)$ means that either f and g are both defined on x and have the same value on x, or f and g are both undefined on x.

2 Composition-Nominative Approach

The composition-nominative approach [5] aims to propose a mathematical basis for development of formal methods of analysis and synthesis of software systems. According to this approach, program models are specified as *composition-nominative systems* (CNS) which consist of simpler systems: composition, description, and denotation systems.

Composition system defines semantic aspects of programs, description system defines syntactical aspects, and denotation system specifies meanings of descriptions. Semantics of programs are defined as partial functions over a class of data processed by programs and means of construction of complex programs from simpler programs (e.g. branching, cycle, etc.) are defined as n-ary operations (called compositions) over functions over data. A composition system can be specified as two algebras: data algebra and function algebra. Syntactically programs are represented as terms in the function algebra. The corresponding term algebra defines a descriptive system and the ordinary procedure of term interpretation gives a denotation system.

Data on which programs operate are modeled as *nominative data* [5]. Such data are special kinds of associations between names and values. There are several types of nominative data [8], [9], [10]. Among them the simplest type is the class of nominative sets, where a nominative set is a partial function from a set of abstract names to a set of abstract values [5], [8]. Nominative sets are frequently used in denotational semantics for formalizing program state [17]. In the general case, nominative data are classified in accordance with the following parameters:

- *values* can be simple (unstructured) or complex (structured),
- *names* can be simple (unstructured) or complex (structured).

Here “complex values” mean that the values corresponding to names in a nominative data can be nominative data themselves. Complex (structured) names are understood as strings consisting of simple (unstructured) names. The possible values of the mentioned parameters give four types of nominative data which are denoted as follows:
TND_{SS} – nominative data with simple names and simple values,
TND_{CS} – nominative data with complex names and simple values,
TND_{SC} – nominative data with simple names and complex values,
TND_{CC} – nominative data with complex names and complex values.

The formal definitions of the mentioned types of nominative data are given below.

- For any fixed sets of names V and values A, the class of data of the type TND_{SS} over V and A is defined as $D_0(V, A) = V \overset{n}{\to} A$, where $V \overset{n}{\to} A$ denotes the set of partial functions from V to A which have a finite graph. The elements of this class are denoted using notation $[v_1 \mapsto a_1, ..., v_n \mapsto a_n]$, where $v_i \in V$ are names and $a_i \in A$ are the corresponding values. For example, data $d = [u \mapsto 1, v \mapsto 2]$ belongs to $D_0(V, A)$, where $u, v \in V$ are distinct elements and $\{1, 2\} \subseteq A$, $dom(d) = \{u, v\}$ and $d(u) = 1$, $d(v) = 2$.

- For any fixed sets of names V and values A, the class of data of the type TND_{SC} over V and A is $D_1(V, A) = ND(V, A)$, where

  – $ND(V, A) = \bigcup_{k \geq 0} ND_k(V, A)$,

  – $ND_0(V, A) = A \cup \{\emptyset\}$,

  – $ND_{k+1}(V, A) = A \cup \left(V \overset{n}{\longrightarrow} ND_k(V, A)\right), \quad k \geq 0.$

  Here, we denote by $\emptyset$ the empty nominative data, i.e. a function with an empty graph (this notation is also used for the empty set).

  Data of type TND_{SC} are hierarchically constructed. An example of such data is $[u \mapsto 1, v \mapsto [w \mapsto 2]]$, where $u, v, w \in V$, $1, 2 \in A$. Such data can be represented by oriented trees (of varying arity) with arcs labelled by names and with leafs labelled by elements from A or $\emptyset$.

  A *path* is a nonempty finite sequence $(v_1, v_2, ..., v_k)$, $v_1, ..., v_k \in V$.

  For a given data d, a *value of a path* $(v_1, v_2, ..., v_k)$ in d is defined by the expression $d(v_1, v_2, ..., v_k) \cong (...((d(v_1))(v_2))...(v_k))$.

  We say that a path $(v_1, v_2, ..., v_k)$ is *a path in a data* $d \in ND(V, A)$, if a value of $(v_1, v_2, ..., v_k)$ in d is defined, i.e. $d(v_1, v_2, ..., v_k) \downarrow$ (a path in data corresponds to a path from the root to a node in an oriented tree). A *terminal path* in a data $d \in ND(V, A)$ is a path in d such that its value belongs to $A \cup \{\emptyset\}$. The least k such that $d \in ND_k(V, A)$ is the *rank* of a data d.

- For any fixed sets of names V and values A, the class of data of the type TND_{CS} over V and A is defined as $D_2(V, A) = NDVS(V, A)$, where $NDVS(V, A)$ is the set of all elements of $A \cup (V^+ \xrightarrow{n} A)$ such that either $d \in A$, or $d \in V^+ \xrightarrow{n} A$ and all strings from $dom(d)$ are pairwise incomparable in the sense of the prefix relation (*principle of unambiguous associative naming*). An example of such data is $[uv \mapsto 1, uw \mapsto 2, w \mapsto 3]$, $u, v, w \in V$. Such data have *complex names* i.e. names that are strings.

- For any fixed sets of names V and values A, the class of data of the type TND_{CC} over V and A is defined as $D_3(V, A) = NDVC(V, A)$, where $NDVC(V, A)$ is the class of all data $d \in ND(V^+, A)$ such that for any two paths $(u_1, u_2, ..., u_k)$ and $(v_1, v_2, ..., v_l)$ in d, neither of which is a prefix of another, the words $u_1u_2...u_k$ and $v_1v_2...v_l$ are incomparable in the sense of the prefix relation (*principle of unambiguous associative naming*). Such data is also called *complex-named data* [10]. An example of such data is $[uv \mapsto 1, w \mapsto [uw \mapsto \emptyset]]$, $u, v, w \in V$.

3 Basic Operations on Nominative Data

The basic operations on nominative data are the operations of
– *denaming* (taking the value of a name),
– *naming* (assigning a new value to a name),
– *overlapping.*

Let us define these operations for data of the most interesting and complex type TND_{CC}.

Let V and A be fixed sets of names and basic values respectively.

Definition 1 (Denaming)**.** *The (associative) denaming is an operation $v \Rightarrow_a$ with a parameter $v \in V^+$ defined by induction on the length of v:*

- *if $v \in V$, then* $v \Rightarrow_a (d) \cong \begin{cases} d(v), & \text{if } d(v)\downarrow; \\ d/v, & \text{if } d(v)\uparrow \text{ and } d/v \neq \emptyset; \\ \text{undefined}, & \text{if } d(v)\uparrow \text{ and } d/v = \emptyset, \end{cases}$
where $d/u = [v_1 \mapsto d(z) \mid d(z)\downarrow, z = uv_1, v_1 \in V^+]$;

- *if $v \in V^+ \backslash V$, then $v \Rightarrow_a (d) \cong v_2 \Rightarrow_a (v_1 \Rightarrow_a (d))$, where v_1 is the first symbol of v and v_2 is the suffix, i.e. v_1, v_2 are (unique) words such that $v = v_1v_2$ and $v_1 \in V$.*

The following examples illustrate this operation:

- $u \Rightarrow_a ([u \mapsto 1, v \mapsto 2]) = 1$;
- $(uv) \Rightarrow_a ([u \mapsto [vw \mapsto 1, u \mapsto 2]]) = [w \mapsto 1]$.

This operation has the following property (*associativity*) [10]:

$$u \Rightarrow_a (d) \cong u_n \Rightarrow_a (u_{n-1} \Rightarrow_a (\dots u_1 \Rightarrow_a (d)\dots))$$

for all complex names $u, u_1, u_2, ..., u_n \in V^+$ such that $u = u_1u_2...u_n$.

Definition 2 (Naming)**.** *Naming is an unary operation $\Rightarrow v$ with a parameter $v \in V^+$ such that $\Rightarrow v(d) = [v \mapsto d]$.*

Overlapping is a kind of updating operation which updates the values of names in its first argument with the values of names in its second argument. For different types of nominative data different overlapping operations can be considered. We will define two kinds of overlapping: global and local overlapping. Global (associative or structural) overlapping ∇_a updates several values in the first argument while the local one ∇_a^v (with a parameter name v) updates only one value which is associated with the name v.

Global overlapping can be used, e.g. for formalizing procedures calls, while the local overlapping can be used as a formalization of the assignment operator in programming languages.

Definition 3 (Global overlapping)**.** *For nominative data of the type TND_{CC}, global overlapping is a binary operation ∇_a defined inductively by the rank of the first argument as follows.*

Let $NDVC_k(V, A) = NDVC(V, A) \cap ND_k(V^+, A)$ be the data from the set $NDVC(V, A)$, the rank of which is $\leq k$.

Induction base of the definition. If $d_1 \in NDVC_0(V, A)$, then

$$d_1 \nabla_a d_2 \cong \begin{cases} d_2, & \text{if } d_1 = \emptyset \text{ and } d_2 \in NDVC(V, A) \backslash A; \\ \text{undefined}, & \text{if } d_1 \in A \text{ or } d_2 \in A. \end{cases}$$

Induction step of the definition. Assume that the value $d_1\nabla_a d_2$ is already defined for all d_1, d_2 such that $d_1 \in NDVC_k(V,A)$. Let

$$d_1 \in NDVC_{k+1}(V,A)\backslash NDVC_k(V,A).$$

Then $d_1\nabla_a d_2 = d$, where d is defined for each name $u \in V^+$ as follows:

1) $d(u) = d_2(u)$, if $u \in dom(d_2)$ and u does not have a proper prefix which belongs to $dom(d_1)$;

2) $d(u) = d_1(u)\nabla_a(d_2/u)$, if $d_1(u)$ is defined and does not belong to A and u is a proper prefix of some element of $dom(d_2)$, where $d_2/u = [v_1 \mapsto d_2(v) \mid d_2(v)\downarrow, v = uv_1, v_1 \in V^+]$;

3) $d(u) = d_2/u$, if $d_1(u)$ is defined and belongs to A and u is a proper prefix of some element of $dom(d_2)$;

4) $d(u) = d_1(u)$, if $d_1(u)$ is defined and u is not comparable (in the sense of the prefix relation) with any element of $dom(d_2)$;

5) $d(u)\uparrow$, otherwise.

The global overlapping has the following properties [10]:

- $[u \mapsto d_1]\nabla_a[v \mapsto d_2] = [u \mapsto d_1, v \mapsto d_2], \quad u, v \in V, \quad u \neq v$;
- $[uv \mapsto d_1]\nabla_a[u \mapsto d_2] = [u \mapsto d_2]$, $u, v \in V^+$, i.e. the value under a name u in the second argument overwrites the values under names in the first argument which are extensions of u;
- $[u \mapsto d_1]\nabla_a[uv \mapsto d_2] = [u \mapsto (d_1\nabla_a[v \mapsto d_2])]$, if $u, v \in V^+$, $d_1 \notin A$, i.e. the value under a name uv in the second argument modifies values under prefixes of uv in the first argument.

Definition 4 (Local overlapping)**.** *For nominative data of the type TND_{CC} local overlapping is a binary operation ∇_a^v with a parameter $v \in V^+$ defined as follows: $d_1\nabla_a^v d_2 \cong d_1\nabla_a(\Rightarrow v(d_2))$.*

Definition 5. *Name checking predicate $u!$ on $NDVC(V,A)$ with a parameter $u \in V^+$ is defined as follows:*

$u!(d) = T$, if $u \Rightarrow_a (d)\downarrow$; $u!(d) = F$, if $u \Rightarrow_a (d)\uparrow$.

Definition 6. *Emptiness checking predicate $IsEmpty$ on $NDVC(V,A)$ is defined as follows:*

$IsEmpty(d) = T$, if $d = \emptyset$; $IsEmpty(d) = F$, if $d \neq \emptyset$.

Using the basic operations on nominative data, we can define an algebraic structure of nominative data.

Definition 7. *An algebraic structure of nominative data of the type TND_{CC} is defined as follows:*

$$NDAS_{CC}(V,A) = (NDVC(V,A); \emptyset, \{v \Rightarrow_a\}_{v \in V^+},$$
$$\{\Rightarrow v\}_{v\in V^+}, \{\nabla_a^v\}_{v\in V^+}, \{v!\}_{v\in V^+}, IsEmpty),$$

where $\emptyset$ is a constant – the empty nominative data.

Note that $NDAS_{CC}(V,A)$ is a structure without equality.

One can extend $NDAS_{CC}(V,A)$ with additional unary predicates and operations on nominative data.

Definition 8. *Let $k, l \in \mathbb{N} \cup \{0\}$, $p_1, p_2, ..., p_k$ be partial predicates on $NDVC(V,A)$ and $f_1, ..., f_l$ be partial functions of the type $NDVC(V,A) \tilde{\to} NDVC(V,A)$. An extended algebraic structure of nominative data of the type TND_{CC} with additional unary predicates $p_1, ...p_k$ and operations $f_1, ..., f_l$ is defined as follows:*

$$NDAS_{CC}(V,A; p_1, ..., p_k; f_1, ..., f_l) = (NDVC(V,A);$$
$$\emptyset, \{v \Rightarrow_a\}_{v\in V^+}, \{\Rightarrow v\}_{v\in V^+}, \{\nabla_a^v\}_{v\in V^+}, \{v!\}_{v\in V^+}, IsEmpty,$$
$$p_1, ..., p_k, f_1, ..., f_l),$$

where $\emptyset$ is a constant – the empty nominative data.

Definition 9. *A path in $d \in NDVC(V,A)$ is a nonempty sequence $(v_1, v_2, ..., v_n)$ of words from V^+ such that the value $((d(v_1))(v_2)...)(v_n)$ is defined. This value $((d(v_1))(v_2)...)(v_n)$ is called the value of the path $(v_1, v_2, ..., v_n)$ in d. A path is called a terminal path in d, if its value in d belongs to $A \cup \{\emptyset\}$.*

Definition 10. *1) $d_1 \in NDVC(V,A)$ is nominatively included in $d_2 \in NDVC(V,A)$, if either $d_1, d_2 \in A$ and $d_1 = d_2$, or $d_1, d_2 \notin A$ and for each terminal path $(v_1, v_2, ..., v_n)$ in d_1 there is a terminal path $(v'_1, v'_2, ..., v'_m)$ in d_2 such that $v_1v_2...v_n = v'_1v'_2...v'_m$ and the values of $(v_1, v_2, ..., v_n)$ in d_1 and $(v'_1, v'_2, ..., v'_m)$ in d_2 coincide.*

2) d_1, d_2 are nominative equivalent ($d_1 \approx d_2$), if d_1 is nominatively included in d_2 and d_2 is nominatively included in d_1.

4 Associative Nominative Glushkov Algorithmic Algebra

Programs on nominative data can be formalized as functions from nominative data (input data) to nominative data (output data) which can be constructed from the operations of the algebraic structure $NDAS_{CC}(V, A)$ using compositions which represent programming language constructs, e.g. sequential execution, branching, cycle, etc.

The set of such programs together with compositions forms an algorithmic algebra similar to, e.g. Glushkov algorithmic algebras [14].

In [11] the authors of this paper proposed a generalization of Glushkov algorithmic algebras to algebras of functions and predicates over nominative data of the type TND_{CC} (i.e. data with complex names and complex values) in order to obtain a rich, but tractable formal language for specifying and reasoning about programs. This generalization is called an Associative Nominative Glushkov Algorithmic Algebra (ANGAA).

Let V and A be fixed sets of basic names and values. Denote
$Pr_{CC}(V, A) = NDVC(V, A) \tilde{\to} \{T, F\}$,
$Fn_{CC}(V, A) = NDVC(V, A) \tilde{\to} NDVC(V, A)$.

We will assume that T and F do not belong to $NDVC(V, A)$.

We will call the elements of $Pr_{CC}(V, A)$ *(partial nominative) predicates* and the elements of $Fn_{CC}(V, A)$ *(partial binominative) functions*.

Let us denote by $\bar{U}$ the set of all tuples $(u_1, u_2, ..., u_n)$, $n \geq 1$ of complex names from V^+ such that whenever $i \neq j$, u_i and u_j are incomparable in the sense of the prefix relation.

- Sequential composition of functions (denoted using the infix notation) $\bullet : Fn(V, A) \times Fn(V, A) \to Fn(V, A)$ is defined as follows: for all $f, g \in Fn(V, A)$ and data d: $(f \bullet g)(d) \cong g(f(d))$.

- Prediction composition [14] $\cdot : Fn(V, A) \times Pr(V, A) \to Pr(V, A)$ is defined as follows: for all $f \in Fn(V, A)$, $p \in Pr(V, A)$, and data d: $(f \cdot p)(d) \cong p(f(d))$.

- Assignment composition $Asg^u : Fn(V, A) \to Fn(V, A)$ with a parameter $u \in V^+$ is defined as follows: for each $f \in Fn(V, A)$ and data d, $(As^u(f))(d) \cong d\nabla_a^u f(d)$.

- The composition of superposition into a function

$$S_F^{u_1,u_2,\ldots,u_n} : Fn(V, A) \times (Fn(V, A))^n \to Fn(V, A)$$

  with parameters $n \geq 1$ and $u_1, ..., u_n \in V^+$ such that $(u_1, ..., u_n) \in \bar{U}$ is defined as follows:

$$S_F^{u_1,\ldots,u_n}(f, f_1, ..., f_n)(d) \cong f(...(d\nabla_a^{u_1} f_1(d))...\nabla_a^{u_n} f_n(d))...).$$

  We will also use the following notation for this composition: for each tuple $\bar{u} = (u_1, u_2, ..., u_n) \in \bar{U}$, $S_F^{\bar{u}}$ denotes $S_F^{u_1,u_2,\ldots,u_n}$.

- The composition of superposition into a predicate

$$S_P^{u_1,u_2,\ldots,u_n} : Pr(V, A) \times (Fn(V, A))^n \to Pr(V, A)$$

  with parameters $n \geq 1$ and $u_1, ..., u_n \in V^+$ such that $(u_1, ..., u_n) \in \bar{U}$ is defined as follows:

$$S_P^{u_1,\ldots,u_n}(p, f_1, ..., f_n)(d) \cong p(...(d\nabla_a^{u_1} f_1(d))...\nabla_a^{u_n} f_n(d))...).$$

  We will also use the following notation for this composition: for each tuple $\bar{u} = (u_1, u_2, ..., u_n) \in \bar{U}$, $S_P^{\bar{u}}$ denotes $S_P^{u_1,u_2,\ldots,u_n}$.

- Branching composition $IF : Pr(V, A) \times Fn(V, A) \times Fn(V, A) \to Fn(V, A)$ is defined as follows: for each $p \in Pr(V, A)$, $f, g \in Fn(V, A)$:

  $IF(p, f, g)(d) \cong f(d)$, if $p(d) \downarrow= T$.

  $IF(p, f, g)(d) \cong g(d)$, if $p(d) \downarrow= F$.

  $IF(p, f, g)(d)$ undefined, if $p(d) \uparrow$.

- Cycle composition $WH : Pr(V, A) \times Fn(V, A) \to Fn(V, A)$ is defined as follows: for each $p \in Pr(V, A)$, $f \in Fn(V, A)$, and d:

$WH(p,f)(d)\downarrow= f^{(n)}(d)$, if there exists $n \geq 0$ such that $(f^{(i)} \cdot p)(d)\downarrow= T$ for all $i \in \{0,1,...,n-1\}$ and $(f^{(n)} \cdot p)(d)\downarrow= F$, where $f^{(n)}$ is a n-times sequential composition of f with itself ($f^{(0)}$ is the identity function), and $WH(p,f)(d)$ is undefined otherwise.

- Negation $\neg : Pr(V,A) \to Pr(V,A)$ is a composition such that for each $p \in Pr(V,A)$ and data d: $(\neg p)(d) \cong T$, if $p(d)\downarrow= F$; $(\neg p)(d) \cong F$, if $p(d)\downarrow= T$; $(\neg p)(d)$ is undefined, if $p(d)\uparrow$.

- Disjunction $\vee : Pr(V,A) \times Pr(V,A) \to Pr(V,A)$ is a composition defined as follows: for each $p_1, p_2 \in Pr(V,A)$ and data d:

$$(p_1 \vee p_2)(d) \cong \begin{cases} T, & \text{if } p_1(d)\downarrow= T \text{ or } p_2(d)\downarrow= T; \\ F, & \text{if } p_1(d)\downarrow= F \text{ and } p_2(d)\downarrow= F; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

- Identity composition $Id : Fn(V,A) \to Fn(V,A)$ is defined as follows: $Id(f) = f$ for all $f \in Fn(V,A)$.

- True constant predicate (null-ary composition) $True \in Pr(V,A)$ is defined as follows: $True(d)\downarrow= T$ for all data d.

- Bottom function (null-ary composition) $\perp_F \in Fn(V,A)$ is defined as follows: $\perp_F (d)\uparrow$ for all data d.

- Bottom predicate (null-ary composition) $\perp_P \in Pr(V,A)$ is defined as follows: $\perp_P (d)\uparrow$ for all data d.

- Name checking predicate (null-ary composition) with a parameter $u \in V^+$: $u!(d) = T$, if $u \Rightarrow_a (d)\downarrow$; $u!(d) = F$, if $u \Rightarrow_a (d)\uparrow$.

- Empty constant function (null-ary composition): $Empty(d) = \emptyset$.

- Emptiness checking predicate (null-ary composition):
  $IsEmpty(d) = T$, if $d = \emptyset$; $IsEmpty(d) = F$, if $d \neq \emptyset$.

Our generalization of Glushkov algorithmic algebras to an algebra of programs on hierarchical data is defined below.

Definition 11. *An Associative Nominative Glushkov Algorithmic Algebra (ANGAA) is a two-sorted algebra*

$$NGA^a_{CC}(V,A) = (Pr_{CC}(V,A), Fn_{CC}(V,A); \bullet, IF, WH, \cdot, \{Asg^u\}_{u\in V^+},$$
$$\{S_F^{\bar{u}}\}_{\bar{u}\in\bar{U}}, \{S_P^{\bar{u}}\}_{\bar{u}\in\bar{U}}, \vee, \neg, Id, True, \bot_F, \bot_P, \{u!\}_{u\in V^+},$$
$$Empty, IsEmpty)$$

One can further extend ANGAA by adding constant symbols which denote certain fixed predicates from $Pr_{CC}(V,A)$ and/or functions from $Fn_{CC}(V,A)$ to its signature.

Definition 12. *Let $k,l \in \mathbb{N}\cup\{0\}$, $p_1,p_2,...,p_k \in Pr_{CC}(V,A)$ and $f_1,f_2,...,f_l \in Fn_{CC}(V,A)$.*

An extended Associative Nominative Glushkov Algorithmic Algebra (eANGAA) with predicate constants $p_1,...,p_k$ and function constants $f_1,...,f_l$ is a two-sorted algebra

$$NGA^a_{CC}(V,A;p_1,...,p_k;f_1,...f_l) =$$
$$(Pr_{CC}(V,A), Fn_{CC}(V,A); \bullet, IF, WH, \cdot, \{Asg^u\}_{u\in V^+},$$
$$\{S_F^{\bar{u}}\}_{\bar{u}\in\bar{U}}, \{S_P^{\bar{u}}\}_{\bar{u}\in\bar{U}}, \vee, \neg, Id, True, \bot_F, \bot_P, \{u!\}_{u\in V^+}, Empty, IsEmpty,$$
$$p_1,p_2,...,p_k,f_1,f_2,...f_l).$$

5 Effective Definitional Schemes and Generalization of eds Definability

The generalized recursion theory as proposed by H. Friedman [15] and subsequently developed in [16] investigates generalized notions of computability on objects of algebraic structures. In this context in [15] H. Friedman defined the notion of a generalized Turing algorithm and the equivalent notion of an effective definitional scheme (eds) [15]. Basically, eds are definitions by infinite cases which have a recursive enumerable structure. They can be used to give a very general definition of a computable function; in fact it was argued [16] that for reasonable definitions of computable functions over algebraic structures computable functions need to be eds definable.

Such a definition of a computable function can be described as follows [16]. Consider a language L with finitely many constant, relation, operation symbols interpreted in an algebraic structure M with some domain, constants, relations and operations. Then a function f (on the domain of M) is eds definable, if there is a set S of conditions of the form $\varphi_i(v, v_1, ..., v_n) \to t_i(v, v_1, ..., v_n)$ (eds) consisting of terms $t_i(v, v_1, ..., v_n)$ in L and basic semialgebraic conditions (i.e. finite conjunctions of atomic formulas and their negations) $\varphi_i(v, v_1, ..., v_n)$ in L, where $v, v_1, ..., v_n$ are formal variable names, such that S is effective (recursively enumerable as a set of strings) and there exist elements $a_1, ..., a_n$ of the domain of M (parameters of the definition) such that for each i: $f(x) = t_i(x, a_1, ..., a_n)$, if $\varphi_i(x, a_1, ..., a_n)$.

It is known [16, Theorem 2] that in a suitable formalization, programs expressible in imperative programming languages with variables ranging over M and assignments, stacks of values from the domain of M with the operations *Push* and *Pop,* conditional operators (*If-Then-Else*), and jumps (*Goto*) define eds definable functions.

However, as it is, the notion of eds definability has a limited applicability to semantics of programming languages, since it tells only what are computable functions from M (or more generally, M^n) to M or M^m, where the elements of M are considered as unstructured data.

In contrast, in the context of semantics of programming languages [17], [2], [3], it is more important to describe computability of the steps taken by the program during execution, which are usually transformations of structured program states to structured program states.

Below we generalize eds definability of functions on a structure M to eds definability of transformations of program execution states for programs operating on complex data structures (e.g. multidimensional arrays, lists, trees and tree-like structures, etc.) over M.

Let $V = \{v_1, v_2, ..., v_m\}$ be a fixed finite set of basic names and A be a fixed set of basic values.

Let $p_1, ..., p_k \in Pr_{CC}(V, A)$, $f_1, ..., f_l \in Fn_{CC}(V, A)$ be fixed finite sequences of predicates and functions. If $x_1, ..., x_n$ are variable names, denote by $T_{x_1,x_2,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$ the set of all terms in $NDAS_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$ in $x_1, ..., x_n$, and by $\Phi_{x_1,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$ the set of all basic semalgebraic conditions, i.e. formulas which have a form of a finite conjunction of atomic formulas in $NDAS_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$ or their negations (note that equality is not allowed).

For each term t in $T_{x_1,x_2,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$ or formula φ in $\Phi_{x_1,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$, denote by $[t]$ and $[\varphi]$ their standard interpretations (i.e. the corresponding partial function and predicate on tuples of elements of $NDVC(V, A)$).

Definition 13. *A function $f \in Fn_{CC}(V, A)$ is eds definable with respect to $p_1, ..., p_k$ and $f_1, ..., f_l$, if there exists a natural number n, data $d_1, d_2, ..., d_n \in NDVC(V, A)$, and a finite or countable set S of pairs of the form*

$\{(\varphi_i(x, x_1, x_2, ..., x_n), t_i(x, x_1, ..., x_n)) \mid i \in I\}$

(I is a set of indices $I = \mathbb{N}$ or $I = \{1, 2, ..., K\}$ for some natural K), where $\varphi_i(x, x_1, ..., x_n) \in \Phi_{x,x_1,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$ and $t_i(x, x_1, ..., x_n) \in T_{x,x_1,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$ and $x, x_1, ..., x_n$ are different variable names, such that

1) the set of all strings of the form $\varphi(x, x_1, ..., x_n) \to t(x, x_1, ..., x_n)$ for $(\varphi(x, x_1, ..., x_n), t(x, x_1, ..., x_n)) \in S$ in the alphabet $\{v_1, v_2, ..., v_m, \boldsymbol{,}, (,), x, x_1, ..., x_n, \emptyset, \Rightarrow, a, !, IsEmpty, \neg, \wedge, p_1, ..., p_k, f_1, ..., f_l\}$ is recursively enumerable (it is assumed that symbols with sub/superscripts ∇_a^v in terms are represented as ∇av).

2) For each $d \in NDVC(V, A)$ and $i \in I$, if $[\varphi_i](d, d_1, ..., d_n) \downarrow = T$, then $f(d) \cong [t_i](d, d_1, ..., d_n)$.

3) For each $d \in NDVC(V, A)$, if $[\varphi_i](d, d_1, ..., d_n) \uparrow$ for all $i \in I$, then $f(d) \uparrow$.

Note that the problem of checking whether a given set of the form $\{(\varphi_i(x, x_1, x_2, ..., x_n), t_i(x, x_1, ..., x_n)) \mid i \in I\}$ defines an eds definable function as in Definition 13 may be algorithmically undecidable. However, this does not have any implications on applicability of the notion of eds definable functions to the problems considered in this paper.

Definition 14. *A predicate $p \in Pr_{CC}(V,A)$ is eds definable with respect to $p_1,...,p_k$ and $f_1,...,f_l$, if there exists a natural number n, data $d_1,d_2,...,d_n \in NDVC(V,A)$, and a finite or countable set S of pairs of the form*

$\{(\varphi_i(x,x_1,x_2,...,x_n),b_i) \mid i \in I\}$

(I is a set of indices $I = \mathbb{N}$ or $I = \{1,2,...,K\}$ for a natural K, b_i is a Boolean value), where $\varphi_i(x,x_1,...,x_n) \in \Phi_{x,x_1,...,x_n}(V;p_1,...,p_k;f_1,...,f_l)$ and $b_i \in \{T,F\}$ and $x,x_1,...,x_n$ are different variable names, such that

1) the set of all strings of the form $\varphi(x,x_1,...,x_n) \to b$ for $(\varphi(x,x_1,...,x_n),b) \in S$ in the alphabet $\{v_1,v_2,...,v_m,\textbf{,},(,),x,x_1,...,x_n,$ $\emptyset,\Rightarrow,a,!,IsEmpty,\neg,\wedge,p_1,...,p_k,f_1,...,f_l\}$ is recursively enumerable (it is assumed that symbols with superscripts ∇_a^v in terms are represented as ∇av).

2) For each $d \in NDVC(V,A)$ and $i \in I$, if $[\varphi_i](d,d_1,...,d_n) \downarrow= T$, then $p(d) \cong b_i$.

3) For each $d \in NDVC(V,A)$, if $[\varphi_i](d,d_1,...,d_n) \uparrow$ for all $i \in I$, then $p(d) \uparrow$.

6 Main results

Let us introduce the following notation.

- $PrEds_{CC}(V,A;p_1,...,p_k;f_1,...,f_l)$ is the set of all predicates in $Pr_{CC}(V,A)$ which are eds definable with respect to $p_1,...,p_k$ and $f_1,...,f_l$.
- $FnEds_{CC}(V,A;p_1,...,p_k;f_1,...,f_l)$ is the set of all functions in $Fn_{CC}(V,A)$ which are eds definable with respect to $p_1,...,p_k$ and $f_1,...,f_l$.

The following theorem shows that all programs of eANGAA with predicate constants $p_1,...,p_k$ and function constants $f_1,...,f_l$ are eds definable with respect to $p_1,...,p_k$ and $f_1,...,f_l$.

Theorem 1 (eds definability of programs of eANGAA). *The sets $PrEds_{CC}(V,A;p_1,...,p_k;f_1,...,f_l)$, $FnEds_{CC}(V,A;p_1,...,p_k;f_1,...,f_l)$ form a subalgebra of $NGA^a_{CC}(V,A;p_1,...,p_k;f_1,...,f_l)$.*

Proof. (Sketch)

It is easy to check that all null-ary compositions of the algebra $NGA^a_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$ are eds definable by Definition 13 and Definition 14.

The fact that all unary and binary compositions of the algebra $NGA^a_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$ (sequential composition, branching, cycle, etc.) preserve eds definability can be proven similarly to [16]. This implies the statement of the theorem. □

For any transitive binary relation $\leqslant$ on $NDVC(V, A)$ let us denote:

- $PrM_{CC}(V, A, \leqslant)$ is the set of all $p \in Pr_{CC}(V, A)$ such that for all d_1, d_2, if $p(d_1)\downarrow$ and $d_1 \leqslant d_2$, then $p(d_2)\downarrow= p(d_1)$. The elements of $PrM_{CC}(V, A, \leqslant)$ are called $\leqslant$ -equitone predicates.

- $FnI_{CC}(V, A, \leqslant)$ is the set of all $f \in Fn_{CC}(V, A)$ such that for each d, if $f(d)\downarrow$, then $d \leqslant f(d)$. The elements of $FnI_{CC}(V, A, \leqslant)$ are called $\leqslant$-increasing functions.

- $FnM_{CC}(V, A, \leqslant)$ is the set of all $f \in Fn_{CC}(V, A)$ such that for each d_1, d_2, if $f(d_1)\downarrow$ and $d_1 \leqslant d_2$, then $f(d_2)\downarrow$ and $f(d_1) \leqslant f(d_2)$. The elements of $FnM_{CC}(V, A, \leqslant)$ are called $\leqslant$-monotone functions.

- $PrM^n_{CC}(V, A, \leqslant)$ is the set of all $p \in Pr^n_{CC}(V, A)$ such that for all $d_1, d_2, ..., d_n, d'_1, d'_2, ..., d'_n$, if $p(d_1, d_2, ..., d_n)\downarrow$ and $d_1 \leqslant d'_1$, $d_2 \leqslant d'_2$, ..., $d_n \leqslant d'_n$, then $p(d'_1, d'_2, ..., d'_n)\downarrow= p(d_1, d_2, ..., d_n)$. The elements of $FnM^n_{CC}(V, A, \leqslant)$ are called $\leqslant$-equitone n-ary predicates.

- $FnI^n_{CC}(V, A, \leqslant)$ is the set of all $f \in Fn^n_{CC}(V, A)$ such that for each $d_1, d_2, ..., d_n$, if $f(d_1, d_2, ..., d_n)\downarrow$, then $d_i \leqslant f(d_1, d_2, ..., d_n)$ for each $i = 1, 2, ..., n$. The elements of $FnM_{CC}(V, A, \leqslant)$ are called $\leqslant$-increasing n-ary functions.

- $FnM^n_{CC}(V, A, \leqslant)$ is the set of all $f \in Fn^n_{CC}(V, A)$ such that for each $d_1, d_2, ..., d_n, d'_1, d'_2, ..., d'_n$, if $f(d_1, d_2, ..., d_n)\downarrow$ and $d_1 \leqslant d'_1$, $d_2 \leqslant d'_2$, ..., $d_n \leqslant d'_n$, then $f(d'_1, d'_2, ..., d'_n)\downarrow$ and $f(d_1, d_2, ..., d_n) \leqslant$

$f(d'_1, d'_2, ..., d'_n)$. The elements of $FnM^n_{CC}(V, A, \leqslant)$ are called $\leqslant$-monotone n-ary functions.

Lemma 1. *Let* $\leqslant$ *be a transitive binary relation on* $NDVC(V, A)$. *Assume that:*

$f_1, f_2, ..., f_l, \Rightarrow u, u \Rightarrow_a, \nabla^u_a \in FnI_{CC}(V, A, \leqslant)$ *for each* $u \in V^+$.

Then for each $n \in \mathbb{N}$, *distinct variable names* $x_1, ..., x_n$, *and a term* $t(x_1, ..., x_n) \in T_{x_1,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$ *we have*

$[t] \in FnI^n_{CC}(V, A, \leqslant)$.

Proof. The proof can be straightforwardly done by induction on the structure of the term $t(x_1, ..., x_n)$ of the algebraic structure of nominative data $NDAS_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$ (note that the base of the induction follows from the assumptions). □

Lemma 2. *Let* $\leqslant$ *be a transitive binary relation on* $NDVC(V, A)$. *Assume that:*

$f_1, f_2, ..., f_l, \Rightarrow u, u \Rightarrow_a, \nabla^u_a \in FnI_{CC}(V, A, \leqslant)$ *for each* $u \in V^+$.

$p_1, p_2, ..., p_k, IsEmpty \in PrM_{CC}(V, A, \leqslant), u!$, *for each* $u \in V^+$.

Then for each $n \in \mathbb{N}$, *distinct variable names* $x_1, ..., x_n$, *a term* $t(x_1, ..., x_n) \in T_{x_1,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$, *and a formula* $\varphi(x_1, ..., x_n) \in \Phi_{x_1,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$ *we have*

$[t] \in FnI^n_{CC}(V, A, \leqslant)$ *and* $[\varphi] \in PrM^n_{CC}(V, A, \leqslant)$.

Proof. The proof can be straightforwardly done by induction on the structure of the term $t(x_1, ..., x_n)$ and the formula $\varphi(x_1, ..., x_n)$ of the algebraic structure of nominative data $NDAS_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$ (note that the base of the induction follows from the assumptions). □

Lemma 3. *Let* $\leqslant$ *be a preorder* $NDVC(V, A)$.
Assume that:

$f_1, f_2, ..., f_l, \Rightarrow u, u \Rightarrow_a, \nabla^u_a, u! \in FnM_{CC}(V, A, \leqslant)$ *for each* $u \in V^+$ *and* $p_1, p_2, ..., p_k, IsEmpty \in PrM_{CC}(V, A, \leqslant)$.

Then for each $n \in \mathbb{N}$, *distinct variable names* $x_1, ..., x_n$, *and a term* $t(x_1, ..., x_n) \in T_{x_1,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$, *and a formula* $\varphi(x_1, ..., x_n) \in \Phi_{x_1,...,x_n}(V; p_1, ..., p_k; f_1, ..., f_l)$ *we have*

$[t] \in FnM^n_{CC}(V, A, \leqslant)$ *and* $[\varphi] \in PrM^n_{CC}(V, A, \leqslant)$.

Proof. The proof can be straightforwardly done by induction on the structure of the term $t(x_1, ..., x_n)$ and the formula $\varphi(x_1, ..., x_n)$ of the algebraic structure of nominative data $NDAS_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$. $\square$

Theorem 2. *(1) If $\leqslant$ is a transitive relation on $NDVC(V, A)$ and the conditions of Lemma 1 hold, then*

$FnEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l) \subseteq FnI_{CC}(V, A, \leqslant)$.

(2) If $\leqslant$ is a transitive relation on $NDVC(V, A)$ and the conditions of Lemma 2 hold, then

$PrEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l) \subseteq PrM_{CC}(V, A, \leqslant)$ *and*

$FnEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l) \subseteq FnI_{CC}(V, A, \leqslant)$.

(3) If $\leqslant$ is a preorder on $NDVC(V, A)$ and the conditions of Lemma 3 hold, then

$PrEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l) \subseteq PrM_{CC}(V, A, \leqslant)$ *and*

$FnEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l) \subseteq FnM_{CC}(V, A, \leqslant)$.

Proof. (1) Let us show that $FnEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l) \subseteq FnI_{CC}(V, A, \leqslant)$. Let $f \in FnEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$, $d \in NDVC(V, A)$, and $f(d) \downarrow$. Then using notations of Definition 13 we can say that there exists i such that $[\varphi_i](d, d_1, ..., d_n) \downarrow= T$ and $[t_i](d, d_1, ..., d_n) \downarrow= f(d)$. By Lemma 2, we have $[t_i] \in FnI_{CC}^{n+1}(V, A, \leqslant)$, so $d \leqslant [t_i](d, d_1, ..., d_n) = f(d)$. Since d is arbitrary, we have $f \in FnI_{CC}(V, A, \leqslant)$.

(2) Let us show that

$PrEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l) \subseteq PrM_{CC}(V, A, \leqslant)$.

Let $p \in PrEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$, $d', d'' \in NDVC(V, A)$ and $d' \leqslant d''$. Assume that $p(d') \downarrow$. Then using notations of Definition 14 we can say that there exists i such that $p(d') = b_i$ and $[\varphi_i](d', d_1, ..., d_n) \downarrow= T$. By Lemma 2, $[\varphi_i] \in PrM_{CC}^{n+1}(V, A, \leqslant)$, so $[\varphi_i](d'', d_1, ..., d_n) \downarrow= T$. Then $p(d'') \downarrow= b_i = p(d')$. Since d is arbitrary, we have $p \in PrM_{CC}(V, A, \leqslant)$.

That $FnEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l) \subseteq FnI_{CC}(V, A, \leqslant)$ can be shown similarly to the case (1) above.

(3) Let us show that

$PrEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l) \subseteq PrM_{CC}(V, A, \leqslant)$.

Let $p \in PrEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$, $d', d'' \in NDVC(V, A)$ and $d' \leqslant d''$. Assume that $p(d') \downarrow$. Then using notations of Definition 14 we can say that there exists i such that $p(d') = b_i$ and $[\varphi_i](d', d_1, ..., d_n) \downarrow= T$. By Lemma 3, $[\varphi_i] \in PrM^{n+1}_{CC}(V, A, \leqslant)$, so $[\varphi_i](d'', d_1, ..., d_n) \downarrow= T$. Then $p(d'') \downarrow= b_i = p(d')$. Since d is arbitrary, we have $p \in PrM_{CC}(V, A, \leqslant)$.

Let us show that

$FnEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l) \subseteq FnM_{CC}(V, A, \leqslant)$.

Let $f \in FnEds_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$, $d, d' \in NDVC(V, A)$, $d \leqslant d'$, and $f(d) \downarrow$. Then using notations of Definition 13 we can say that there exists i such that $[\varphi_i](d, d_1, ..., d_n) \downarrow= T$ and $[t_i](d, d_1, ..., d_n) \downarrow= f(d)$. By Lemma 3, $[t_i] \in FnM^{n+1}_{CC}(V, A, \leqslant)$ and $[\varphi_i] \in PrM^{n+1}_{CC}(V, A, \leqslant)$, so $[t_i](d', d_1, ..., d_n) \downarrow$ and $f(d) = [t_i](d, d_1, ..., d_n) \leqslant [t_i](d', d_1, ..., d_n)$. Moreover, $[\varphi_i](d', d_1, ..., d_n) \downarrow= T$, so $f(d') \downarrow= [t_i](d', d_1, ..., d_n)$ and $f(d) \leqslant f(d')$. Since d is arbitrary, we have $f \in FnM_{CC}(V, A, \leqslant)$.

□

Corollary 1. *Under the conditions of Lemma 1 or Lemma 2, all unary functions (programs) expressible in $NGA^a_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$ belong to $FnI_{CC}(V, A, \leqslant)$.*

Proof. Follows immediately from Theorem 1 and Theorem 2. □

Corollary 2. *Under the conditions of Lemma 3, all unary functions (programs) expressible in $NGA^a_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$ belong to $FnM_{CC}(V, A, \leqslant)$.*

Proof. Follows immediately from Theorem 1 and Theorem 2. □

Corollary 1 implies that in order to show that a program expressible in $NGA^a_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$ has the property $(d, f(d)) \in\leqslant$ which expresses the fact that the input and output of f belong to a transitive relation $\leqslant$, it is sufficient to check several properties of basic operations on nominative data with respect to $\leqslant$ which are formulated in Lemma 1. It is not necessary to prove preservation of this property by the compositions of $NGA^a_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$, since this preservation follows automatically from eds definability of all programs expressible in $NGA^a_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$.

It is easy to see that the results similar to corollaries from Theorem 2 hold not only for eANGAA, but for a reduct of ANGAA [18], i.e. an algebra with narrower carriers and/or sets of operations (in particular, constants) and predicates:

1) if all function constants of a reduct of eANGAA are $\leqslant$-increasing, then all unary functions (programs) expressible in this reduct are $\leqslant$-increasing.

2) if in a reduct of eANGAA all function constants are $\leqslant$-monotone and all predicate constants are $\leqslant$-equitone, then all unary functions (programs) expressible in this reduct are $\leqslant$-monotone.

Using this observation, e.g., we can show partial correctness of the GCD program mentioned in the introduction by considering a reduct of eANGAA of functions and predicates over $NDVC(\{x, y, a, b\}, \mathbb{Z})$ which has as function constants the functions which perform assignment operations which appear in this program: $f_1(d) = d\nabla_a[a \mapsto d(x)]$, $f_2(d) = d\nabla_a[b \mapsto d(y)]$, $f_3(d) = d\nabla_a[a \mapsto d(a)-d(b)]$, $f_4(d) = d\nabla_a[b \mapsto d(b) - d(a)]$ and showing that they are $\leqslant$-increasing.

Corollary 2 implies that in order to show that a program expressible in $NGA^a_{CC}(V, A; p_1, ..., p_k; f_1, ..., f_l)$ is monotone with respect to some preorder on data, it is sufficient to check several properties of basic operations on nominative data with respect to $\leqslant$ which are formulated in Lemma 3.

An example of application of the obtained results is given below. In [11] the authors considered a special property of programs which is called *nominative stability* [11], [8], [9], [10]. This property is a formalization of the idea of stability of program semantics when the data structures used in the program are changed to equivalent in the sense of information content and supported operations.

It can be illustrated by the following feature of the Pascal programming language: the two-dimensional array definitions `var A: array [1..n, 1..m] of real` and `var A:array [1..n] of array [1..m] of real` are equivalent and both the `A[i,j]` and `A[i][j]` syntax can be used to access the array elements regardless of the form of its definition (it should be noted that the languages like C++ and Java do not have this feature). This implies that one can safely swap two-dimensional array definitions in a program without changing the rest of the text of the program while preserving program semantics.

Nominative stability is defined using the *nominative equivalence* relation on nominative data of the type TND_{CC}. This relation is a formalization of the idea that data are equivalent, if they have essentially the same information content, but may have different hierarchical naming structure. For example, the following data are nominatively equivalent: $[v_1 \mapsto [v_2 \mapsto [v_3 \mapsto 1]]]$ and $[v_1 v_2 v_3 \mapsto 1]$, as they differ only in the naming hierarchy, but contain the same basic names and values. A function on nominative data is nominative stable, if on nominative equivalent data it gives nominative equivalent results.

Formally,

Definition 15. *1) Data $d_1 \in NDVC(V,A)$ is nominatively included in $d_2 \in NDVC(V,A)$, if either $d_1, d_2 \in A$ and $d_1 = d_2$, or $d_1, d_2 \notin A$ and for each terminal path $(v_1, v_2, ..., v_n)$ in d_1 there is a terminal path $(v'_1, v'_2, ..., v'_m)$ in d_2 such that $v_1 v_2 ... v_n = v'_1 v'_2 ... v'_m$ and the values of $(v_1, v_2, ..., v_n)$ in d_1 and $(v'_1, v'_2, ..., v'_m)$ in d_2 coincide.*

2) Data d_1, d_2 are nominative equivalent ($d_1 \approx d_2$), if d_1 is nominatively included in d_2 and d_2 is nominatively included in d_1.

3) The elements of $FnM_{CC}(V, A, \approx)$ are called nominative stable functions (programs).

Using Corollary 2 formulated above we can easily show that all programs expressible in $NGA^a_{CC}(V,A)$ (i.e. ANGAA without additional predicates and functions) are nominative stable.

In [11] it was shown that $\approx$ is an equivalence on $NDVC(V,A)$ (and thus is a preorder) and the functions $\Rightarrow u$, $u \Rightarrow_a$, ∇^u_a are nominative stable. It is trivial to check by the definition that $u!$ is $\approx$-equitone binary predicate on $NDVC(V,A)$ and $IsEmpty$ is $\approx$-monotone (i.e. nominative stable). Then by Corollary 2, all functions expressible in $NGA^{a}_{CC}(V,A)$ are nominative stable.

7 Conclusions

We have investigated methods of proving properties of programs on hierarchical nominative data on the basis of the composition-nominative approach. The proofs of properties of programs depend on proofs of properties of compositions and basic operations on data. The complexity of such proofs can be lowered, if one is able to reduce the proofs of properties of various compositions to proofs of properties of operations on data. We have proposed a way of achieving such a reduction by representing compositions using effective definitional schemes of H. Friedman. The achieved reduction permits us to consider proofs in data algebras (which are simpler to derive, automate, etc.) instead of proofs in program algebras. Using this approach we have demonstrated that the properties of programs related to structural transformations of data can be reduced to the data level. The obtained results can be used in software development and verification.

References

[1] T. Hoare, "The verifying compiler: A grand challenge for computing research," *Journal of the ACM*, vol. 50, no. 1, pp 63–69, 2003.

[2] R. W. Floyd, "Assigning meanings to programs," in *Proceedings of Symposium on Applied Mathematics*, vol. 19, J.T. Schwartz, Ed. A.M.S., 1967, pp. 19–32.

[3] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[4] B. Meyer, “Design by Contract,” Interactive Software Engineering Inc., Technical Report TR-EI-12/CO, 1986.

[5] N. S. Nikitchenko, “A composition-nominative approach to program semantics,” Technical University of Denmark, Technical report IT-TR 1998-020, 1998.

[6] M. S. Nikitchenko and S. S. Shkilniak, *Mathematical logic and theory of algorithms*, Publishing house of Taras Shevchenko National University of Kyiv, 2008, 528 p. ISBN: 966-439-007-0. (in Ukrainian)

[7] A. Kryvolap, M. Nikitchenko and W. Schreiner, “Extending Floyd-Hoare logic for partial pre- and postconditions,” in *Information and Communication Technologies in Education, Research, and Industrial Applications*, vol. 412 of Communications in Computer and Information Science, Springer International Publishing, 2013, pp. 355–378.

[8] M. Nikitchenko and Ie. Ivanov, “Programming with nominative data,” in *Proceedings of CSE'2010 International Scientific Conference on Computer Science and Engineering*, September 20-22, 2010, Kosice, Slovakia, 2010, pp. 30–39.

[9] M. S. Nikitchenko and Ie. Ivanov, “Stability and monotonicity of programs with respect to structure transformations of data,” *Problems in programming*, no. 2–3, pp. 58–67, 2010. (in Ukrainian)

[10] M. S. Nikitchenko and Ie. Ivanov, “Composition-nominative languages of programs with associative denaming,” *Bulletin of Lviv University, Ser. Appl. Math. Inform.*, no. 16, pp. 124–139, 2010. (in Ukrainian)

[11] V. G. Skobelev, M. Nikitchenko and Ie. Ivanov, “On algebraic properties of nominative data and functions,” in *Information and Communication Technologies in Education, Research, and Industrial Applications*, vol. 469, Communications in Computer and In-

formation Science, Springer International Publishing, 2014, pp. 117–138.

[12] M. S. Nikitchenko and V. G. Tymofieiev, "Satisfiability in composition-nominative logics," *Central European Journal of Computer Science*, vol. 2, no. 3, pp. 194–213, 2012.

[13] V. G. Skobelev, Ie. Ivanov and M. Nikitchenko, "Set-theoretic analysis of nominative data," *Computer Science Journal of Moldova*, vol. 23, no. 3(69), pp. 270–288, 2015.

[14] V. M. Glushkov, "Automata theory and formal transformations of microprograms," *Cybernetics*, no. 5, pp. 1–10, 1965. (in Russian)

[15] H. Friedman, "Algorithmic procedures, generalized Turing algorithms, and elementary recursion theory," in *LOGIC COLLOQUIUM '69*, Studies in Logic and the Foundations of Mathematics, Elsevier, vol. 61, pp. 361–389, 1971.

[16] H. Friedman and R. Mansfield, "Algorithmic Procedures," *Trans. Amer. Math. Soc.*, vol. 332, pp. 297–312, 1992.

[17] H. R. Nielson and F. Nielson, *Semantics with applications: a formal introduction*, Wiley Professional Computing, John Wiley & Sons, Inc., New York, 1992, 252 p., ISBN: 0-471-92980-8.

[18] P. Burmeister, "Lecture notes on universal algebra. Many-sorted partial algebras," 2002. [Online]. Available: `http://www.mathematik.tu-darmstadt.de/Math-Net/Lehrveranstaltungen/Lehrmaterial/SS2002/AllgemeineAlgebra/download/LNPartAlg.pdf`

Ievgen Ivanov, Mykola Nikitchenko,
Volodymyr G. Skobelev

Received October 6, 2016

Ievgen Ivanov
Taras Shevchenko National University of Kyiv
64/13 Volodymyrska Street, 01601 Kyiv, Ukraine
Phone: +380442590519
E–mail: `ivanov.eugen@gmail.com`

Mykola Nikitchenko
Taras Shevchenko National University of Kyiv
64/13 Volodymyrska Street, 01601 Kyiv, Ukraine
Phone: +380442590519
E–mail: `nikitchenko@unicyb.kiev.ua`

Volodymyr G. Skobelev
V.M. Glushkov Institute of Cybernetics of NAS of Ukraine
40 Glushkova ave., Kyiv, Ukraine, 03187
Phone: +380634318605
E–mail: `skobelevvg@mail.ru`

Computer Science Journal of Moldova, vol.**24**, no.3(72), 2016

Natural Language Processing versus Logic. Pros and cons on the dispute whether logic is useful in the computational interpretation of language

Dan Cristea

Abstract

In this essay I express some personal opinions regarding the influence that logic has on modern approaches to process natural language in artificial systems. I start by presenting some successful linguistic formalisms that were originated in logic, arguing why logic is important in conveying the meaning of language expression. Then, I counterbalance the argumentation with a number of examples where logic is impuissant to mirror language usage, finally supporting a rather temperate opinion about the usefulness of logic to formalise low level linguistic processes and about the limits of language formalisation.

Keywords: natural language processing (NLP), logic theories in NLP, statistical approaches, symbolic versus statistical approaches in NLP.

1 Introduction

It seems that we, human beings, motored by the need to understand the reality among us (a condition for survival), make use of shallower or deeper cognitive processes in our efforts to assign meanings to messages we receive through language. This cognitive behaviour resembles, in some cases, logical formalisms (what is logic if not an expression of thinking?). This is just as we have in our brains a symbolic machinery capable to help us make inferences and offer solutions to complex puzzles that the language encodes. In other cases, however, logic is of no use: language manifests a totally weird behaviour.

The need for symbolic approaches, as opposed to statistical ones, is in itself an expression of the feelings researchers have that language can be expressed as a system of rules, which would make its messages to be "computable". This paradigm is similar to the one of mathematical logic, since there too, based on a very clear notation of a number of basic ingredients and of defined ways in which they can be grouped together, truth values of sentences (i.e. syntactical constructions) can be deduced.

The debate here, is not if logical, i.e. symbolic, formalisms are useful, but to what extend can they be applied to explain a range of linguistic phenomena. Then, for the whole rest of linguistic phenomena for which logic fails to offer support, ought we instead resort to statistical or neural-based solutions? Or part of our manifestations triggered by language escapes from any formalisation, being it based on logic, statistics or neural grounds? And finally, what is the range of practical applications of language which put at their base logical solutions?

In this essay I express some personal opinions regarding the influence that logic has on modern approaches to processing natural language.

2 When do we need logic to decipher language?

Language ought to be logical, or, else, the communication based on it would be impossible (imagine that the inscription of messages would be made in a randomizing system of signs...). In most of cases, people are able to transmit their intentions correctly to the intended receivers. So, when we say *two horses* we mean something of the kind: $\exists\, S = \{x|horse(x)\}^{\wedge} card(S) = 2$, where $card(S)$ means the cardinal of the set S, $horse(x)$ is a qualifying function asserting that x, its argument, has a semantic property that is shared in the common knowledge of the living inhabitants of this world, that this property is currently denominated in English by the word *horse*, and that the pragmatic context in which the expression is uttered clearly separates the different meanings that this English word may encode. In the same time, uttering this noun phrase, we are also aware that the listener possesses an equivalent decoding mechanism that enables her to coagulate an equivalent meaning. So, it seems that in extremely many situations of the real life, when we use language, we implicitly resort to mathematical logic to express our messages. In fact, although we don't really do that, it is just like doing that, i.e. just like somebody above us, listening to what we are saying, quickly encodes our saying in a logical expression that could unambiguously be "read" or decoded by our partner in the conversation.

The challenge to describe language in a logical system of notation that would allow non-ambiguous representations of its lexical elements and coherent composition/decomposition has preoccupied modern linguistics for a long time already. In this section I will make a very quick survey of only some of these approaches.

2.1 Generative Lexicon and Qualia Structures

This kind of attitude towards language brings forward Generative Lexicon (GL) [6], a theory that intends to build lexical and semantic resources capable of expressing in computational terms (which is another name for logic) the rich lexical variety of the language (any language, in principle), including its capacity to combine meanings of lexical items through grammar and, to a certain extend, through pragmatics. GL tries to describe the semantic flexibility shown by words in combination with others. To account for diverse interpretations that words can display when placed in combinations with others, GL associates a hidden event in the lexicon description of nouns, adjectives and adverbs. Originated in the Aristotelian concept of *aitia* (explanation), with reinterpretations added by Moravcsik [4], further developments of GL [5, 7] introduce Qualia structures, describing roles according to which the meaning of words can be decomposed on four different coordinates:

- *Formal* (F): encoding taxonomic information about the lexical item (the *is-a* relation); □
- *Constitutive* (C): encoding information on the parts and constitution of an object (*part-of* or *made-of* relation); □
- *Telic* (T): encoding information on purpose and function (the *used-for* or *functions-as* relation); □
- *Agentive* (A): encoding information about the origin of the object (the *created-by* relation). □

Qualia are formally represented as typed feature structures. For instance, the one in Fig. 1 can account for combinations such as: *large car* (F = vehicle), *broken car* (C = motor), *speedy car* (T = drive), *Italian car* (A = made in Italy). Also, adding arguments to roles, Qualia structures can deal with metonymy, as in: *the car from behind honked*: T = drive (human, vehicle). Such lexicon representations actually encode in a logical form part of an ontology of lexicalised concepts, out of which "understanding" can be computed.

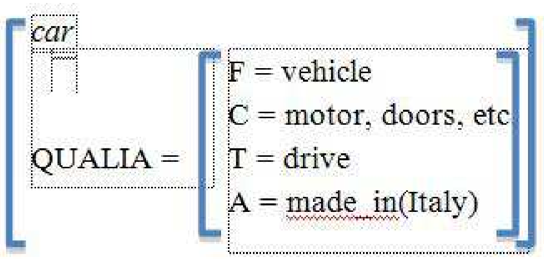


Figure 1. A Qualia structure for the lexical concept car

Qualia structures are recognised to have various shortcomings. Pustejosky and Jezek [7], for instance, recognise the rather limited "ability [of the formalism] to take on an indefinite variety of possible senses depending on the other words they combine with". He gives the example of the verb *like* and wonders whether it has two different meanings in "He likes my sister" and "He likes vanilla ice cream", and if so, how is this difference to be represented in decompositional terms?

2.2 Discourse coherence

But logic is needed at upper levels of language interpretation as well. One example is the need to consolidate meanings of sequences of utterances in discourse. Compare, for instance:

Maria dropped the egg from her hand. (1.1)
She cleaned the floor. (1.2)

with:

Maria dropped the feather from her hand. (2.1)
She cleaned the floor. (2.2)

While sequence (1) is perfectly coherent, sequence (2) apparently has no meaning, although the utterances of each sequence convey unambiguous meanings. It is clear that an inferential chain of deductions, triggered by common sense (or ontological) knowledge, link the two utterances in (1), as opposed to (2), where the connection is much harder to establish. The meaning in (1) is built out of a reasoning sequence showing a temporal occurrence of events that could be schematized as follows:

(1.1) ⇒ drops(AG:Maria, OB:egg) ⇒ falls(REC:egg) ⇒
touches(AG:egg, OB:X) (3)
Qualia.Formal(egg) = container; Qualia.Constitutive(egg) =
{eggshell(fragile),liquid} (4)
(3), (4) ⇒ breaks(REC:eggshell) ⇒ leaks(REC:liquid, ON:X) ⇒
perceives(AG:Maria, OB:dirty(X)) (5)
(1.2) ⇒ cleans(AG:Maria, OB:floor) ⇒
perceives(AG:Maria, OB:dirty(floor)) (6)

The equality of (5) and (6), when X equals floor, closes the inference chain, proving the high degree of coherence of the sequence (1). A longer inference chain (if any), implying volitional searches in a space of possibilities fuelled by imagination, as opposed to the first case, in which the inferences are common-sense, natural, spontaneous, could, in the mind of an intrigued reader, possibly link the utterances in (2), thus showing a much lower degree of coherence.

2.3 Textual Entailment

In the fight to decipher the meaning expressed in language, two contrary phenomena have to be faced: variability and ambiguity. Variability of language means that the same meaning can be verbalized in different surface forms. Ambiguity means that one surface form can be interpreted as having different meanings. A number of NLP applications that deal with the variability of language trying to reduce the distance between form and meaning are: Information Retrieval (IE), Textual Entailment (TE) and Question Answering.

In TE, it is said that **text *t* entails hypothesis *h* ($t \Rightarrow h$) if humans reading *t* will infer that *h* is most likely true**. So, textual entailment is a directional relation between two texts. In practical applications of TE (including competitions[1]) t could be complemented with external knowledge in order for h to be entailed, but h cannot be entailed only by the knowledge itself (for instance, by searching on the web).

Here is an example of a true entailment (from RTE data):

t: *... a shootout at the Guadalajara airport in May, 1993, that killed Cardinal Juan Jesus Posadas Ocampo and six others.*
h: *Cardinal Juan Jesus Posadas Ocampo died in 1993.* (7)

and of a false one (same source):

t: *Regan attended a ceremony in Washington to commemorate the landings in Normandy.*
h: *Washington is located in Normandy.* (8)

One of the methods used to measure the similarity between t and h does syntactic matching or transformations at the syntactic level. To

[1]For instance, the EU FP-6 Funded PASCAL Network of Excellence 2004-7: Recognizing Textual Entailment (RTE) Challenges.

see the complexity of such an attempt, I will examine in some detail an example. Suppose t is:

Philanthropic Golding Inc. came into existence in January 2004. (9.1)
One year after its foundation the company declared bankruptcy. (9.2)

and h:

Philanthropic Golding Inc. bankrupted in January 2005. (10)

One way to check the validity of such an entailment, is to launch a pipeline of processes, at the end of which the sentences of both t and h are expressed in a symbolic form that allows close comparison. Applied to (9.1) the pipeline produces the following successive results (simplified)[2]:

Step1: tokenisation (not shown), part-of-speech tagging (not shown), chunking noun phrases and clashing multi-word expressions.
<NP id="n1">*Philanthropic Golding Inc.*</NP>
<MWE id="m1">*came into existence*</MWE>

Step 2: recognition of entity mentions, of time expressions and resolution of anaphora.
<COREF-LIST id="ent1" TYPE = "ENTITY" REF-LIST="n1" />[3]
<TIMEX3 tid="t1" type="DATE" value="2004-01">*January 2004*
</TIMEX3>

Step 3: functional dependency parsing; in Figure 2 we show a Universal Dependency (UD) coding [3].

Step 4: time analysis, in which EVENT and TLINK elements, formalizing the events and their temporal relations, are generated[4].
<EVENT eid="ev1" VB="m1" AG="ent1"/>
<TLINK eventID="ev1" relatedToTime="t1" relType="BEGINS"/>

[2]Here we use an XML coding, but a representation that uses RTF tuples or another notation convension can also be employed.

[3]A COREF-LIST with only one member signals the first mention of an ENTITY or EVENT, according to TYPE.

[4]To simplify notations, MAKEINSTANCE and SIGNAL elements are ignored and EVENT elements are complemented with roles.

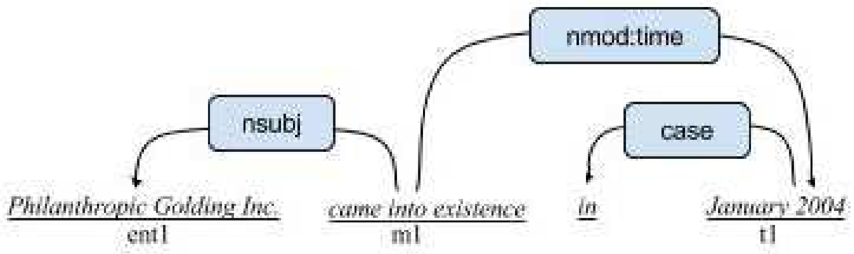


Figure 2. UD parsing of sentence (9.1)

Step 5: generation of equivalent structures (transformations) and the application of the closure tool, which computes the transitive closure of temporal relations; the transformation here concerns the equivalence of the expressions: *X comes into existence* and *UNKNOWN founds X*. This rule triggers the element:

<EVENT eid="ev2" VB="found" AG="UNKNOWN" OB="ent1"/>

and its correspondent time link:

<TLINK eventID="ev2" relatedToTime="t1" relType="BEGINS"/>

And now the pipeline applied to (9.2):

Step 1:
<NP id="n3"><NP id="n2">*its*</NP>*foundation*</NP>
<NP id="n4">*the company*</NP>
<NP id="n5">*bankruptcy*</NP>
Step 2:
<TIMEX3 tid="t2" type="DURATION" value="P1Y">*one year*
</TIMEX3>
<COREF-LIST id="ent1" TYPE="ENTITY" REF-LIST="n1 n2 n4" />
<COREF-LIST id="eve1" TYPE="EVENT" REF-LIST="ev1 ev2 n3" />
<COREF-LIST id="eve2" TYPE="EVENT" REF-LIST="n5" />
Step 3:

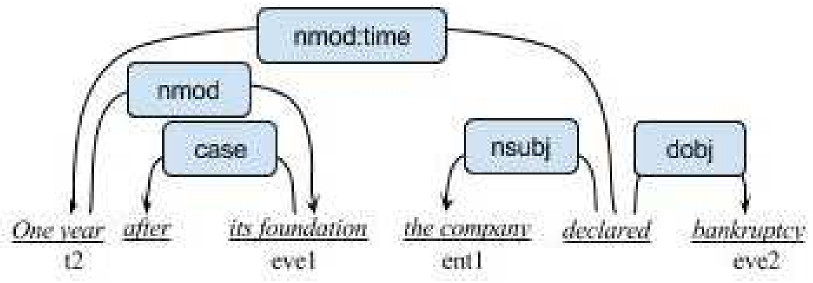


Figure 3. UD parsing of sentence (9.2)

Step 4:
<EVENT eid=”e3” POS=”VERB” CLASS=”REPORTING” AG=”ent1” OB=”eve2”>*declared*</EVENT>
<EVENT eid=”e4” POS=”NOUN” CLASS=”OCCURRENCE” AG= ”ent1”> *bankruptcy*</EVENT>
<TLINK eventID=”e3” relatedToTime=”t2” relType=”AFTER”/>
<SLINK eventID=”e3” subordinatedEvent=”e4” relType= ”FACTIVE”/>

Step 5: the closure tool produces:
<TIMEX3 tid=”t4” type=”DATE” value=”2005-01-xx” />
<TLINK eventID=”e4” relatedToTime=”t4” relType= ”DURING”/> (11)

A similar processing pipeline applied to (10) should yield:
<EVENT eid=”e5” POS=”VERB” AG=”ent1”>*bankrupted* </EVENT>
<TIMEX3 tid=”t5” type=”DATE” value=”2005-01-xx”> *January 2005* </TIMEX3>
<TLINK eventID=”e5” relatedToTime=”t5” relType= ”DURING”/> (12)

And the equivalence of (11) and (12) proves the entailment.

2.4 Other NLP formalisms rooted on logic

Prolog, the programming language of logic, has inspired much work on NLP. In syntax, this means to express a grammar as a set of statements in a logic formalism (e.g. Horn clauses), and to use a theorem prover (e.g. resolution) in order to parse or generate sentences. Recently used in information extraction, SHERLOCK [8] is a system able to learn Horn clauses in a large-scale, domain independent manner, from Web texts. The learned rules can then be used to fuel a first-order reasoning system, as HOLMES, described by Schoenmackers *et al.* [9], which infers answers from tuples.

3 And when logic is of no use?

Languages have specific ways to express linguistic phenomena. Some of them seem to escape any logical explanations.

3.1 Double negation

In propositional logic the double negation is equivalent to an affirmation. However, applied to language, this rule does not always hold. In connection to this phenomenon, Falaus [2] inventories two main types of languages. In Double Negation languages, among which standard varieties of Germanic and Scandinavian, two negative elements cancel each other out resulting in a positive reading, as in (13) below:

Paul didn't see nobody. = *Paul saw somebody.* (13)

However, in Negative Concord languages, among which Romanian and Italian, multiple occurrences of negation are interpreted as one semantic negation, as in (14):

Paul n-a văzut pe nimeni. = *Paul didn't see anybody.* (14)

The sentence can be paraphrased as "It is not the case that there is an individual x, such that Paul saw x."

3.2 Linear position

Romanian is known to be ambivalent with respect to the position of quality adjectives around the nouns they modify: they may occur pre-nominally as well as post-nominally. Cornilescu [1] notices that certain associations of noun+adjective versus adjective+noun makes a difference of interpretation, as here:

femeia singură => *the woman alone*
singura femeie => *the only woman* (15)

I believe that the following examples display similar behaviour. Suppose somebody has two cars, one bought some time ago and one recently bought, and the one recently bought belongs to an old brand while the one bough in the past belongs to a newer brand. Then:

maşina lui cea veche refers to *his old brand car*, while
vechea lui maşină refers to *the car owned by him for a long time* (16)

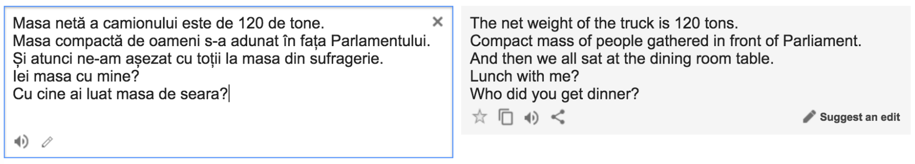
However, in the following associations the sense does not change:

domnişoara frumoasă, frumoasa domnişoară => *the beautiful young lady*
cartea interesantă, interesanta carte => *the interesting book* (17)

Also the positional ambivalence does not apply to any adjective. Certain modifiers make sense only when situated in the pre-position with respect to the modified noun. For example, *biet* (*poor*, *pitiful*) is not accepted unless it precedes the noun: *biet om* (*poor man*), but not: *om biet*. Vulchanova [10] explains this for Balkan languages: these associations seem to contradict the usual intersection-based composition. In general, if X is an adjective and Y – a noun, then XY (or YX) means the set of objects Y that have the property X, or the intersection between the set of objects having the property X and the set of objects Y. As such, *poor men* should be taken as the intersection between the set of things which are poor and the set of men, but *bieţii oameni* means something different than the subset of the set of men which are poor, it means a subset of the set of men which are in a pitiful/miserable state.

3.3 Contexts and the mist of pragmatics

It is a truism that the context determines the meaning of words, and the previous section showed some examples. By “context” here I mean both textual (i.e. positional) and not textual (for instance, temporal).



Masa netă a camionului este de 120 de tone.
Masa compactă de oameni s-a adunat în fața Parlamentului.
Și atunci ne-am așezat cu toții la masa din sufragerie.
Iei masa cu mine?
Cu cine ai luat masa de seara?

The net weight of the truck is 120 tons.
Compact mass of people gathered in front of Parliament.
And then we all sat at the dining room table.
Lunch with me?
Who did you get dinner?

Figure 4. Google Translate solutions for difference occurrences of the Romanian word “masa”

The left side of Fig. 4 shows different contexts of occurrence of the Romanian word “masa”. As with any other poli-semantic words, its sense is fixed by the context. Google Translate applies statistics to disambiguate and, as can be seen, remarkably well. For the time being, I cannot imagine a workable logical solution to this issue, and if this would ever be achieved, what would be the cost of the supportive lexico-semantic resources?

But it is also clear that words induce different reactions in humans, depending of their culture, the moment of the utterance and any other pragmatic conjuncture. A notorious example is the wood language. I wonder how would a logical approach detect the humorous effect that is conveyed by phrases such as the following:

“Obiectivele majore, de însemnătate cu adevărat istorică, pe care Partidul Comunist Român le-a încredinţat frontului culturii româneşti în perioada făuririi societăţii socialiste multilateral dezvoltate angajează – deschis şi plenar…”[5] (approximately: *The major objectives,*

[5]Florian Georgescu (1982) The History Museum – dynamic factor in implementing the P.C.R. policy of patriotic education of the masses, of formation of socialist consciousness, in the Scientific Session: Permanency, unity and progress in the history of the Romanian people. Romanian Communist Party at the 60th anniversary, VI. www.mnir.ro

of truly historic significance, which the Romanian Communist Party has entrusted to the frontline of the Romanian culture in the making of the multilaterally developed socialist society – openly and fully...) (18)

"Pus astfel în lumină, ancorat în sinergia faptelor, recursul la universalitate nu eludează meandrele concretului."[6] (approximately: *Thus put in light, anchored in the synergy of its facts, the appeal to universality does not circumvent the meanders of the concrete.*) (19)

"Să luptăm pentru propăşirea neamului şi aducerea României pe cele mai înalte culmi de civilizaţie multilateral dezvoltată." (approximately: *Let's fight to thrive our stirps and bring Romania on the highest peaks of multilaterally developed civilization.*) (20)

3.4 Style in literature

Humans perceive co-occurrence of words as producing very suggestive images. Confronted with the extraordinary diversity of suggestion that words can convey, logic seems to me faint, forceless, impuissant. How could poetical expressions, such as the following:

"constelaţia ochilor mei" (*the constellation of my eyes*), "atingi cu auzul" (approximately: *your hearing touches*), "nisipuri de fiară" (*beast sands*) – Nichita Stănescu, *Autoportret în timp de veghe* (Auto portrait during watch time) (21)

be encoded in logical constructions? Or how could emotions incurred in sentences like the following one be seized in logical expressions?:

"It's enough for me to be sure that you and I exist at this moment." –Garcia Marquez: *One Hundred Years of Solitude* (22)

I agree that an effort to formalise in logical terms a metonymic sense of an expression, as in this magnificent sequence of simple words:

"lipindu-se de răcoarea tocului uşii" (approximately: *sticking to the chil of the door frame*) — Garcia Marquez: *One Hundred Years of Solitude* (23)

[6]attributed to Ion Iliescu

in which a touched object is replaced with a sensation that the agent borrows from that object is a challenging task. The same happens when looking for logical equivalent of metaphorical language, as here:

"Te mângâi cu degetele muiate în amintiri." (*I caress you with my fingers dipped in memories.*). (24)

4 And the solution is? (instead of conclusions)

The last examples I have given address the philosophical question of whether it is worth looking for a formalisation of language able to encode all its extremely large diversity of expressing power. Supposing logic proves to be successful in representing some language aspects, there should be a limit where the ambition to express natural language in logical form has to stop because it reaches an insurmountable limit.

Some people, including me, think that since humans use language all the time, but only rarely make explicit use of logic in their lives, the domain of NLP should not necessarily be dependent on logical formalisms. Their lack of confidence in logic as a universal machinery for processing language comes from observations of the inability of logic to support exhaustively the processing infrastructures of language. In their opinions, there are aspects of language production and understanding for which something else than logic should be used in order to explain and reproduce on the machine these human performances. On the other hand, these people agree that logic is necessary for acquiring certain types of representations. What they don't believe is that each sentence, or each sequence of sentences, should be transformed into a theorem that necessitates a proof.

It has to be clear by now that I am not talking here about reasoning, as the one needed to make volitional connections, to find explicit links, without which understanding would be impossible, as the ones exemplified in sections 2.2 and 2.3, where, clearly, logic is on the first plan, but about the primary processes that enable the use of language as a communication channel, therefore that allow cognition based on language.

However, it can be said that the whole domain of formal linguistics is inspired by logical formalisms. Indeed, why representing NL sentences as trees? Because, doing this, we intrinsically incorporate decisions about their ambiguities, this way preparing the path towards semantic representations and reasoning. A logical system applied to language means to interpret signs of the language, i.e. words, in their surface variation, as dictated by morphology, then their sequences, as dictated by syntax, and their meanings, as dictated by semantics, with the goal of arriving to an overall formal and unambiguous representation. Above the sentence boundaries, the sentential representations would be combined in discourse trees (sometimes graphs), on which rhetorical deductions could be made and inter-sentential links, as those implied by anaphorae, would be fulfilled, or extra-textual connections, as those evoked by named entities in the cultural background of the receiver of the message, would be activated. If sufficiently sensitive antennae oriented towards the external world would also be available, pragmatic contexts could make subtle revisions to these representations.

But is this enough? Suppose the day D has come when all computational theories now evolving in the field of language with the aim to decipher and represent language **in symbolic form** would reach a successful finalization, and a sufficiently rich collection of accompanying resources, necessary to support with data these formalizations, would be acquired. Are we done with the interpretation of language in that day? Can we install this tremendous computational machinery on a high performance computer or on an whatever network of cloud interconnected devices and say: from now on, whatever text we read (we, the humans), this Goliath super-computer can also read and it will get similar reactions (of course, inventoried in a very rich annotation language)?

First, let's notice that many successful NLP applications already exist, that are so dumb in a real "interpretation" of the language that would horripilate a "bad" classical linguist. Among them: machine translation. The Google Translate machine, exemplified in Fig. 4, does not "understand" a iota, in the classical sense. Put to represent the meaning of those sentences, it will be incapable. And yet, it deals so well with those sentences: the result is equivalent to that obtained by a human being graduated in Romanian-English translation.

In these approaches, of a purely statistical nature, the performance to translate any source language text into any other target language can be obtained by "compiling" a very large collection of parallel documents and a very large collection of target language documents, out of which huge tables of figures, called language models, are extracted. This computer performance in fact copies the human ability to learn a language by practising it, instead of using grammar books and drill exercises that formally incorporate the language competence.

Then, going a little bit further, we may think that a similar statistical apparatus could be used in a dialog system, that would support a human-machine dialogue, resembling intelligence (as in a Turing's test). Again, a machine, statistically trained to answer questions, could arrive to a similar level of performance as a humanly incorporated call-centre operator. Also, close to this, many models of now-a-days chat-bots make use of purely statistical solutions.

However, we should not exaggerate with congratulations and compliments. Until now, statistical solutions proved to behave well in applications where of interest is the conveying of meaning more than the stylistic expression, the clear message more than the poetical language that adorns the message with subliminal adds, in general there where language refinements that imply more than pure transmission of information, those that touch the domain of literature and art, are not involved. But, although still far away from any acceptable solutions, challenged to approach this side of language, my conviction is that statistical methods (and neural) have more chances than rule-based ones.

A vivid area of research in NLP is called Sentiment Analysis. The type of applications belonging to this domain addresses the interests that big commercial companies have to interpret opinions from their clients involving their products, in order to improve them or to estimate future trends. Big data methods put to work on text files are being employed successfully here. Still, a sentiment means much more than the mere and overtly expression of a taste or inclination. For instance, actual systems would perhaps categorise as positive a sentence like *I love you.* and as neutral one that says *I see and I smell everything around us differently since I met you.*

I am confident that machines will arrive to interpret texts, in the sense of extracting the information contained in them (if I would not believe in this success, why bothering to remain in the field?...). Moreover, more and more complex applications that put the interpretation of language at their very base will be on the market. However, I am rather reserved on the usefulness of pure logical approaches in practical NLP settings, one important reason for this being the difficulty of fuelling these systems with the amount of resources that are needed to support the complex reasoning processes. On the contrary, mixed approaches, which maculate the purity of logical approaches with statistics and/or neural models, have a much greater chance of being successful. But my optimism dilutes significantly if the border of semantic content interpretation is overpassed, and we will dig our feet on touching more subtle aspects of language, those that involve emotion sourced in language and interpretation of artistic style, therefore those that address the genuine and inspired juxtaposition of words.

References

[1] A. Cornilescu, "On the Linearization of Adjectives in Romanian," in *Romance Languages and Linguistic Theory, 2006*, D. Torck and L. Wetzles, Eds. Amsterdam: John Benjamins Publishing, 2009, pp. 45–68.

[2] A. Falaus, “Romanian N-words as Negative Quantifiers,” in *Working Papers in Linguistics, Proceedings of the 31st Annual Penn Linguistics Colloquium*, (University of Pennsylvania), vol. 14, no. 1, 2008, pp. 119–134.

[3] M.-C. de Marneffe and C. D. Manning. *Stanford typed dependencies manual.* (2008). [Online]. Available: http://nlp.stanford.edu/software/dependencies_manual.pdf

[4] J. M. Moravcsik, "Aitia as Generative Factor in Aristotle's Philosophy," *Dialogue*, vol.14, no. 4, pp. 622–638, 1975. DOI: 10.1017/S001221730002655X.

[5] J. Pustejovsky, "The Generative Lexicon," *Computational Linguistics*, vol. 17, no. 4, pp. 409–441, 1991.

[6] J. Pustejovsky, *The Generative Lexicon.* Cambridge, MA: MIT Press, 1995, 312 p. ISBN: 9780262161589.

[7] J. Pustejovsky and E. Jezek. *Integrating Generative Lexicon and Lexical Semantic Resources*, LREC Tutorials, May 23 (2016). [Online]. Available: http://lrec2016.lrec-conf.org/media/filer_public/2016/05/10/tutorialmaterial _pustejovsky.pdf.

[8] S. Schoenmackers, J. Davis, O. Etzioni and D. Weld, "Learning First-Order Horn Clauses from Web Text," *Proceedings of EMNLP '10, Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, (Cambridge, Massachusetts), Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 1088–1098.

[9] S. Schoenmackers, O. Etzioni and D. Weld, "Scaling Textual Inference to the Web," *Procs. of EMNLP '08, Proceedings of the Conference on Empirical Methods in Natural Language Processing*, (Honolulu, Hawaii), Stroudsburg, PA, USA: Association for Computational Linguistics, 2008, pp. 79–88.

[10] M.-D. Vulchanova, "Modification in the Balkan nominal expression: An account of the (A)NA – AN(A) order contract," in *From NP to DP. Volume 1: The syntax and semantics of noun phrases*, Martine Coene and Yves D'hulst, Eds. Amsterdam/Philadelphia: John Benjamin Publishing Company, 2003, pp. 91–118.

Received June 15, 2016

Dan Cristea

"Alexandru Ioan Cuza" University of Iaşi,
Faculty of Computer Science and
Romanian Academy, Institute for Computer Science,
E–mail: `dcristea@info.uaic.ro`

Computer Science Journal of Moldova, vol.**24**, no.3(72), 2016

Executable choreographies applied in OPERANDO

Sînică Alboaie, Lenuta Alboaie,
Mircea-Florin Vaida, Cristina Olariu

Abstract

The objective of this paper is to present the software architecture used for the OPERANDO privacy platform, funded by the European Union in a Horizon 2020 project. For integration, OPERANDO is using SwarmESB, an open source Enterprise Service Bus (ESB) based on executable choreographies. In this paper we are presenting the concept of service transformations, presented as a bridge between the world of REST web services and the world of services implemented with executable choreographies. These transformations are improving the heterogeneity aspects when we are analysing SwarmESB as a distributed system. Five types of transformations that have been analysed and implemented as open source software have been integrated. This proposal is shaped around a common language capable of expressing all these five transformation types we have identified working for OPERANDO. Therefore, the Domain Specific Language proposed, renders the essential elements for transformations among functions, web services and executable choreographies. This unification will trigger a quantitative effect on the productivity of the teams creating or integrating web services in a federated service bus environment which is a key architectural component in the future Internet-of-Things and cloud systems.

Keywords: middleware · architectures · DSL · executable choreographies · web service transformations.

1 Introduction

The OPERANDO's [1] architecture presented in this article focuses on the usage of an Enterprise Service Bus (ESB) [2] based on the open source research project SwarmESB [3]. The main goal of the OPERANDO project is to integrate and extend the existing privacy techniques to create a platform that will be used by independent organisations called Privacy Service Providers (PSPs) to ensure policies compliance regarding privacy laws and regulations. OPERANDO should ensure comprehensive user privacy enforcement in the form of a dedicated online service, called “Privacy Authority”. The OPERANDO platform supports flexible and viable business models, including targeting of individual market segments such as public administration, social networks and Internet of Things. We are approaching the concept of service transformations, presented as a bridge between the world of REST web services and the world of services implemented with executable choreographies. Web services can be seen as working on a request/response communication pattern. Executable choreographies [4] can be intuitively seen as arbitrary complex workflows that get executed in systems belonging to multiple organisations or authorities. Executable choreographies are implemented in SwarmESB using the swarm communication idea [5]. Therefore, SwarmESB is a research and engineering effort to implement and adapt ideas specific to the mobile calculus theory. While theoretical research on mobile code [6] and on systems for asynchronous calculus have existed for many years, SwarmESB is a practical approach that can be appealing for the specialists used to program in mainstream languages Java, C#, Java Script and who will not easily switch to research programming languages (actor inspired languages[7], pi calculus[8] et al.).

In SwarmESB, messages have a long time identity during multiple communication events and during complex communication processes. Groups of related messages called swarms change their state after each communication event. In actor model inspired approaches, a message does not have identity or an associated behaviour. Identity, state changes or behaviours are associated only with the message receivers (actors). Associating state, mobile code and behaviour with its own messages is the main difference between swarm communication and the actor model. In the actual implementation message, queues are used and the mobile code is securely deployed on the processing nodes but swarm communication hides all the details of the code migration message queues. Executable choreographies are scripts that get executed in multiple processing nodes which may belong to multiple organisations. Swarm communication environments can be easily integrated with web services by manually exposing remote endpoints in JavaScript functions. In OPERANDO project, we have decided to automate this process by creating methods of describing web service and providing multiple types of "transformations" between web services and executable choreographies. Choreographies can implement a larger number of communication patterns compared with web services. However, we are currently living in a world of web services and since OPERANDO is a complex project that uses existing components and technologies, we have found mandatory to automate the integration of the web services.

2 ESB middleware's based on choreography – concepts overview

An important concern in Service Oriented Architecture (SOA) [9] is to extract the business processes from the application code and orchestrate the business process grounded on services. When multiple organisations are involved in the same business process, we talk about choreography. When business processes spread over multiple organisations, governance, security and privacy aspects become suddenly critical and have a big influence on the business and technology choices. In OPERANDO, we have chosen to use executable choreography, a concept emerging from our previous research [4, 5, 13], [10]. Executable choreographies propose the existence of a business process description that is aware of the location aspects (which is the organisation). It also unifies short living processes as ESB routing and long living business processes (implemented as an extension to the routing). Executable choreographies are technical descriptions of business agreements among multiple organisations and should be treated as such.

One of the most popular integration methods is the nightly batch processing [11]. However, batch-processing integration strategies are prone to errors caused by multiple data changes on shared resources and are bound to cause delays in information retrieval. An ESB can eliminate many latency problems by providing real-time throughput of the data flows among applications and organisations. This real-time flow of data requires support for data transformations [12]. From the development process point of view, an ESB can be seen as the foundation of a SOA architecture that may enable an agile style of working. Agile main goal of reducing waste is accomplished by lowering the need of complex ad-hoc architectures. The development team can understand the big picture from an early stage and actively contribute to defining the services scope and detailed requirements.

Executable choreographies that should be executed by multiple organisations will be manually or automatically verified and approved each time they get updated. Any ESB allows parallel development of integrated services, reducing the need of stubs or fake service implementation during development. The missing services can be simulated within the integration scripts (e.g. executable choreographies). The integration scripts as executable artefacts of the short/long living processes may be independently developed by each team which implements different services. Different versions of the choreographies can be merged at any time, usually without requiring any changes in the service implementation.

The typical ESB roles include connectivity, routing, transformations and various methods to represent short or long living business processes (integrations, orchestration or choreographies) [13].

Connectivity is the basic feature for any Service Bus. An ESB reduces the configuration efforts because the producers will send information only towards BUS and do not have to be aware of consumers.

Routing: beside connectivity, if integration is a subject of interest, the necessity to route the messages in an efficient way becomes apparent. A service consumer only receives that piece of information that should be handled. Typically, routing can take multiple approaches:

- The "pipe" pattern: a single event triggers a sequence of processing steps, each performing a specific function.
- The "content based router" pattern: the message content is used to take decisions about the receivers
- The "message dispatcher" pattern: a message is sent to a list of services
- The "scatter gather" pattern: a request is sent to a number of service providers but all the responses get aggregated into a single response message

In case of SwarmESB based choreographies, all these patterns and many others can be achieved in explicit declarative and imperative code (see Figure 1).

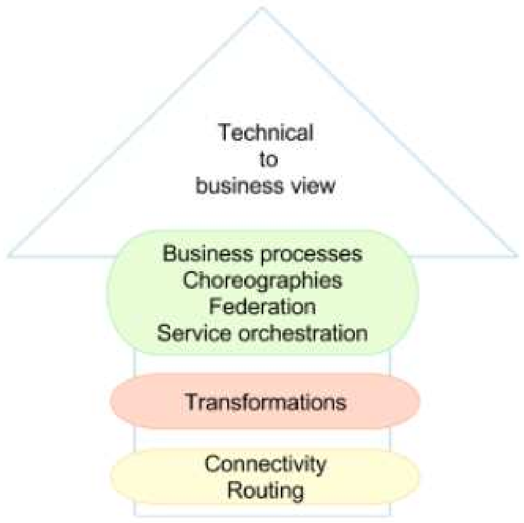


Figure 1. The main roles of an ESB

Transformation: integrated service and applications do not have the same data formats and the ESB is a good place to handle the transformations among these formats. The transformation services that are specialized in the needs of individual applications plugged into the bus can be located anywhere and accessible everywhere on the bus. The transformation can be implemented in the form of adapter nodes or can be implicit in the scripts describing the routing. In this paper, we present the transformation layer implemented in SwarmESB. The proposed methods are able to automate integration with web services, expose web services and perform complex data transformations related to integration or privacy concerns.

Business processes and Service Orchestration concepts are unifying concerns that can be explained meaningfully to the final user (map in scripts or descriptions specific user stories or use cases). They also provide useful abstractions for software analysts, software architects and developers. There are two main types of business processes: long living processes and short living processes. Long living processes are abstracting business concerns that take a long time to be executed (they have a persistent state stored in databases). Until the end of their execution, long living processes are prepared to receive various human inputs or special events in their execution environment (time events, changes in data structures, creation of new objects, etc.).

Human intervention in business processes is usually described by the "workflow" concept. It is quite tempting to use workflow and business process concept as synonyms. This is justified and it is acceptable because, in execution, any manual intervention from a dedicated operator is almost identical to non-human change. In both cases, we are talking about a set of events and changes in databases or in data structures.

A key point is that workflows follow the opposite paradigm of state-based approach rather than a flow-based one like Business Process Execution Language (BPEL) orchestrators. In some cases, the workflow approach is better adapted to long-lived processes, without being restricted from sitting on top of orchestrated services. Hence, workflow servers are usefully complemented by "straight" orchestrators and we may find solutions that are deploying two business process-oriented servers. Additionally, in many ESBs, short living processes are represented by the routing mechanism and in some others by the BPEL type of orchestration. Unfortunately, this approach exposes the developers to too many different languages or approaches when describing short living processes (integration processes). In OPERANDO, SwarmESB choice avoids the complexity and the redundancy of effort and resources caused by the usage of three quite different process description languages. The usage of orchestration concept is discussed in multiple contexts and sometimes with different meanings. We can talk about orchestrations in the context of provisioning in the virtualized deployment environments (in dynamic data-center use cases) and in Service Oriented Architectures. In both cases, orchestration is about aligning the business request with the applications, data, and infrastructure. It defines the policies and service levels through automated workflows, provisioning, and change management. From the data-center or deployment management perspective, orchestration creates an application-aligned infrastructure that can be scaled up or down based on the needs of the applications. For simplicity, we will call this kind of orchestration, *orchestration for deployments and provisioning.*

A somewhat different usage of the orchestration concept is related to the process of coordinating an exchange of information through interactions of web services. We will call this kind of orchestration service *orchestration*. Advanced Service Oriented Architectures could try to decouple the orchestration layer from the service layer in the form of the service orchestration or service choreography. Systems like ESB or integration Platform as a Service (iPaaS) are typically deployed and fine-tuned in order to perform this role. For dynamic data-center use cases, the orchestration is typically related and closely connected to monitoring infrastructure and to the management of the virtualisation solutions. As it will be explained below, the choreography concept and especially the executable choreography is a technique that offers an alternative implementation for the service orchestration. The final results of the service orchestration and of the choreography may look identical (some services are mixed together) but from the point of view of performance, scalability, security and privacy, the decentralised way of choreography brings important benefits. An ESB is a strategic component in any complex system as it succeeds in reducing coupling between solution's components. Reduced coupling enables parallel work to be performed by multiple teams that use separate tools, processes and even platforms/technologies (Java, C#, PHP, node.js etc.). An ESB enables an SOA that is an alternative to the client server model. An ESB promotes agility and flexibility regarding communication between applications and subsystems (see Figure 2).

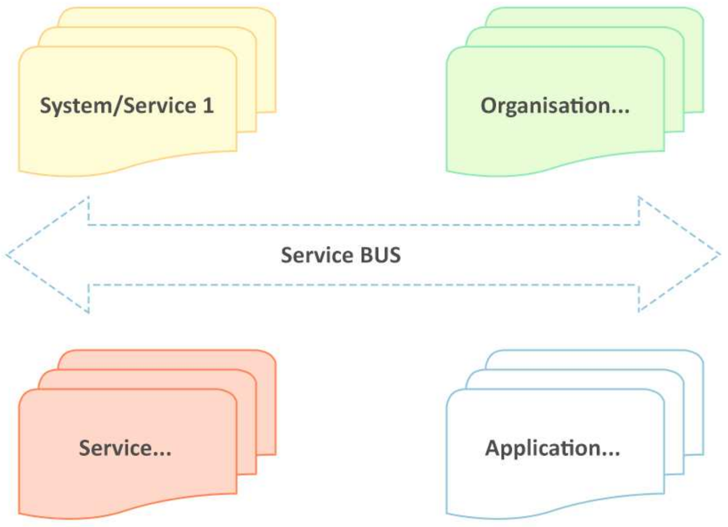


Figure 2. The generic architecture for ESB based systems

The purpose of the integration bus is to provide a flexible method to compose services and components, ensure security and scalability of the system and to allow development towards a federated system between multiple ESBs. Enterprise Service Bus systems must be seen as an architectural pattern. An ESB offers a standard way of integration between applications, services or other kinds of integration objects. An ESB mediates between service providers and service consumers. Integration of loosely coupled services within or across organizations can be obtained.

The SwarmESB current architecture starts from the premises that we are supporting the federation of services among multiple organisations. This perspective implies a technology capable of executing business processes among multiple organisations (choreography) (see Figure 3). Any usage of centralised message queues or centralised Business Process Management (BPM) engines will not be sufficient because of the security and privacy issues raised by centralisation. SwarmESB uses a script based on the routing method that circumvents these privacy concerns.

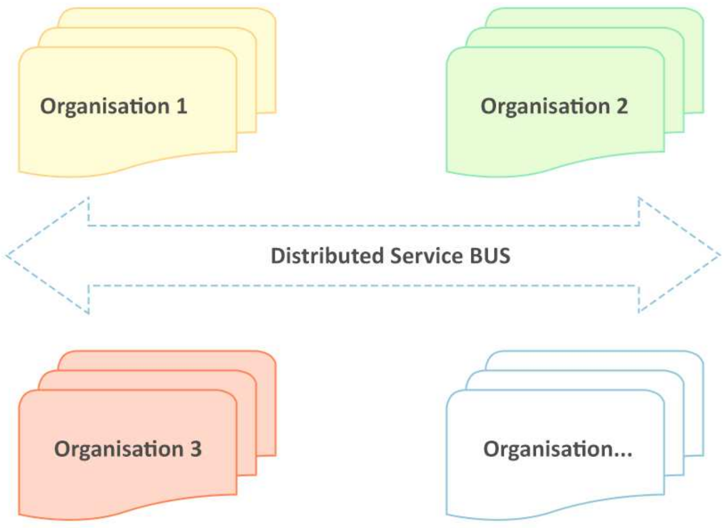


Figure 3. The architecture for choreography based integration offering federation

3 Web Service transformation language proposal

To enable complex communication between the distributed bus provided by SwarmESB and the external world, we have analysed the types of transformation that we have to create in order to enable inbound and outbound usage of web services. A typical integration case is the need to call existing web services inside executable choreography scripts. Another case is the requirement of a new or existing application to communicate with the ESB using web services. These capabilities were not available by default in SwarmESB and as workaround, we used to create custom code for each case. Beyond these two cases, our research for OPERANDO has shown that other three types of transformations exist. The current implementation can be found in the TransRest open source project [14]. The resulted five types of transformations are presented in Table 1.

All five types of transformations may be described in a common language called Swarm to web service Transformation (SwarmTL). For syntax description, we used Backus-Naur Form notation. SwarmTL DSL is an internal DSL (Domain Specific Language) so all JavaScript syntactic and semantic rules should be considered. By using an internal DSL we can benefit from existing tools for debugging, Integrated Development Environments and programming expertise, therefore we reduce adoption risks for this new technology.

SwarmTL language is presented in Table 2.

In order to get an intuitive image about the syntax of the transformations we are exemplifying a SF transformation that takes a remote REST web service from http://localhost:3000 and exposes a set of functions with the name of the blocks (e.g. baseUrl or createEntity).

```
{
baseUrl: 'http://localhost:3000',
getEntity: {
method:'get',
params: ['entity', 'token'],
```

Table 1. Types of transformations

Name	**Description**
Service to Functions transformations (SF)	This transformation can translate a REST service into functions usable in a processing node (e.g. Swarm ESB adapter) and from choreographies. Intuitively, this transformation is just a quick method to generate some functions that asynchronously call remote web services. This simple transformation allows documenting the web service and it also permits a uniform working style inside the SwarmESB based project in which the adapters are plain JavaScript functions.
Choreography to Service transformations (CS)	This transformation exposes a swarm workflow (choreography) as a REST web service. Since the same based systems are real time systems that allow push notification and multiple results for a call, this transformation offers a bridge to the applications that are designed to work in an ask/request method promoted by REST services. The CS transformation allows that existing services to be refactored to use SwarmESB and allows the reuse of the existing skills and tools.
Function to Service transformations (FS)	The FS transformation exposes functions as REST web APIs. This type of transformation is very useful for testing and mocking web services but also for the creation of REST web services with very little code. As we see below, the transformation language hides all the wiring usually required to create web services. This transformation will work together with CS and I transformations allowing to expose an enriched set of services.
Service to Choreography transformations (SC)	This transformation can change a REST Service into a workflow/choreography (swarm description/script) based on an existing template. This kind of transformation is complex and requires metaprogramming capabilities from the choreography implementation. This transformation has not been implemented yet in SwarmESB. The SF transformation allows manual creation of new choreography based on existing web services so basically the SC transformations should be manually programmed.
Interceptor transformations (I)	This kind of transformation can be seen as a combination between SC and CS transformations. An Interceptor transformation can be seen as a smart proxy between some arbitrary REST APIs and an exposed REST APIs. The benefit will be that the transformation can intercept every call and can enrich each call with some arbitrary logic that will be hosted in a swarm workflow description.

Table 2. SwarmTL language

<transformation>	:==	*“{“ <properties> ”,” <blockList> ”}”*
<properties>	:==	*”” | <property> | <property> <opt-comma> <properties>*
<blockList>	:==	*<block> | <block> <opt-comma> <blockList>*
<block>	:==	*<blockName> <opt-whitespace> “:” <opt-whitespace> “{“ <blockPropertyList> “}”*
blockPropertyList	:==	*”” | <blockProperty> | <blockProperty><opt-comma> <blockPropertyList>*
<blockProperty>	:==	*<mandatoryProperty> | <specificProperty>*
<property>	:==	*<globalKey> <equal> <value>*
<mandatoryProperty>	:==	*<mandatoryKey> <equal> <value>*
<specificProperty>	:==	*<specificKey> <equal> <value>*
<mandatoryKey>	:==	*”method” | ”params” | ”path”*
<globalKey>	:==	*”baseUrl” | ”port” | ”swarm”*
<specificKey>	:==	*”code” | ”phase”*
<value>	:==	*jsString | jsAnonymousFunction | jsArray*
<opt-comma>	:==	*<opt-whitespace> ”,” <opt-whitespace> | ””*
<equal>	:==	*<opt-whitespace> ”=” <opt-whitespace>*
<opt-whitespace>	:==	*” ” <opt-whitespace> | ””*

```
path:'/$entity/$token'
},
createEntity: {
method: 'put',
params: ['entityId', 'token', '__body'],
path : '/?id=$entityId&token=$token'
}
}
```

Any transformation is composed of global properties and a list of transformation blocks. The global properties are basically *key value* assignments. Each block is composed of a list of properties known as 'block' properties. A set of properties is present in all the transformations (and are called mandatory properties) but the others are optional or transformation specific. The mandatory properties are "method", "params" and "path". The values for "method" are "get", "post", "put", "delete" corresponding to the HTTP verbs. The "path" parameter specifies the part of the URL that is used to route the request to the actual implementation. The path value is a string that consists of fixed strings and "parameters". All parameters are prefixed by an "$" character that enables the url parse to determine the place of the corresponding values in the actual urls. The values for "params" property are a JavaScript array of string denoting the parameters names. The actual usage of the parameter depends on the type of the transformation. These parameters should appear as strings in the url prefixed by a "$". To terminate a parameter placeholder and to begin a new string or a new parameter, the "/" character should be used. As we can see, this scheme is similar to the ones used in the other routing web engines. A similar naming scheme for routing is used to connect node.js framework but instead of "$" they use ":".

Additionally, we support variables that are not part of the URL, specifically the "__body" parameter that will contain the content of the POST and PUT requests. All the names of variables prefixed with "__" are reserved to be used with the parameters of the POST and PUT body content. In the global section, a set of attributes can be used. ”baseURL” key means the base url of the rest services. The ”node” means the group (or the node type for the processing nodes) on which the transformation will be executed. Other specific properties are specific to particular transformation types as we can see in Table 3.

Tests and code demonstrating the transformations can be found in the TransREST open source project [14].

4 Web service transformations applied in OPERANDO

OPERANDO system is built around a Shared Bus that supports federation and advanced transformation capable of integrating internal and third party web services and functionalities.

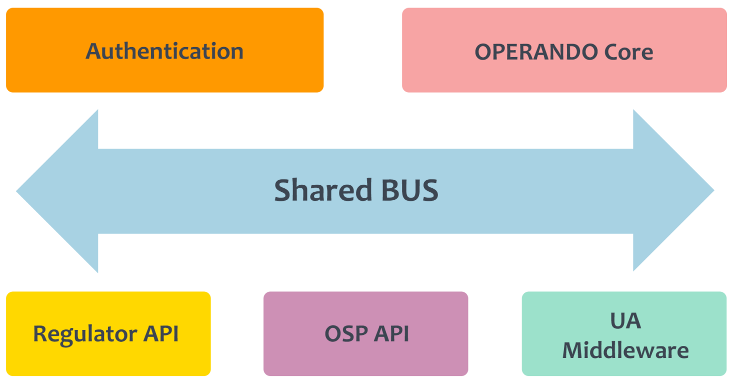


Figure 4. OPERANDO architecture. The high-level view diagram

The major components or layers in the OPERANDO architecture consist of:

- Authentication layer: a set of services and components responsible with the authentication and monitoring of all the business

Table 3. Swarm Transformation Language property names

Property	**Transformations**	**Semantic description**	**Possible value**
baseUrl	CS,SC,SF,I	Global property that specifies the base url for a remote service,	A remote URL
swarm	I	Global property that specifies the name of a swarm used in I transformations to actually call the remote REST service.	String
template	SC	Global property that specifies the name of a swarm used as template in SC transformations	A swarm name
method	CS,FS,SC,SF,I	A block property that specifies the HTTP method used for routing in local and remote services	GET | POST | PUT | DELETE
path	CS,FS,SC,SF,I	A block property that specifies the path in the url for remote services or for the local router	specially formatted string
params	CS,FS,SC,SF,I	A block property having as value an array with the name of the parameters used in the choreography constructors, of the generated functions in all transformations	jsArray: JavaScript array with strings
phase	CS	A block property specifying the phase name that is transformed as a service in CS transformations	String
Code	FS	A block property used by FS transformations to specify the actual implementation of the service. The value is just a plain JavaScript function returning a value asynchronously.	*jsAnonymous-Function:* anonymous function
Result-Phase	CS	A block property used by CS transformations to specify the phase name of the result.	String

processes involving OPERANDO;

- OPERANDO Core services: a collection of complex services, techniques and algorithms that offer functionalities to OSPs such as secure data vaults, anonymization, data mining, etc.;
- REGULATOR API: a collection of web services offered to legal authorities (regulators) to monitor and control OPERANDO's features regarding privacy laws and regulations;
- Online Service Providers APIs (OSP APIs) refer to a set of extensible APIs that can be integrated and transformed by the OPERANDO to be made available for use in applications developed by third party developers called OSPs;
- UA Middleware (User Agent Middleware): a collection of services and workflows used by the OPERANDO client side components.

For OPERANDO we have found three generic use cases where we may use web service transformations:

a) composition of multiple services from the OPERANDO's internal services (OPERANDO Core in Figure 4). For this use case, we use SF transformations to translate external web services into JavaScript functions. These web services are external from the point of view of the bus but are internal for OPERANDO. These functions are exposed to choreographies and used by processing nodes that are called adapter nodes in SwarmESB [6]. With this type of transformation, we can automatically integrate multiple services developed in various languages and make them accessible to the bus without writing any code. In SwarmTL only the declarative descriptions are required and it reduces risks of bugs when using lower level libraries to do REST remote calls.

b) exposition of a single service from Core that will be directly exposed almost unchanged. In this case, the existing web services are enriched by adding only a layer of authentication or by filtering the data within a logical layer responsible with transparent data transformation consisting in real-time anonymization. For this use case we can use an

I transformation that can enrich existing web services while exposing web services to the external environment.

c) creation of custom made web services that have to fit with the need of particular OSP APIs and UA Middleware.

For this use case, we make combinations of FS, SF and CS transformations. SF transformations are capable of exposing various Web Services (implemented with various technologies and by different partners) to the Shared Bus. FS and CS transformations are capable of exposing web services towards outside parties (OSPs, clients, legal regulators) by translating custom made functions and SwarmESB choreographies in web services.

For OPERANDO project, we have analysed the short and medium term quantitative and qualitative effects of the web service transformations. By unifying a set of 5 complementary operations between functions, web services and choreographies we have managed to reduce the quantity of conventions that a programmer has to gasp. An obvious quantitative effect is the reduction of the number of code lines required to create a web service or to use existing web services in choreographies. The reduction in the number of code lines correlates with the reduction in the number of bugs as it is commonly accepted [15]

We constantly evolve SwarmESB in area of building better, generic error handling mechanisms. Our perspective is that every step that increases the use of these generic mechanisms instead of relying on custom code – created by the programmers using lower level libraries – is very important for the reduction of the programming costs and can increase the maintainability of the resulted systems.

5 Conclusions

ESBs created around the concept of executable choreographies and other classical ESBs that are using orchestration engines for web services may have similar purposes. However, as it has been demonstrated in the previous research [4] executable choreographies are designed to provide federation concepts and better privacy ensuring capabilities in complex solutions involving multiple organisations. Executable choreographies do not have a direct correspondent in the web service world and in this paper we have presented five types of web service transformations that enable a bridge between REST web services programming environments and the executable choreography environments. Providing real time messaging [5], the swarm communication pattern can be seen as a generalization for request/response case of the http communication. Likewise, web service transformations are a general case for the more well-known concept of data transformation [12]. The service transformations can be used to implement the well-known concept of data transformations but can also be used for other integration purposes that typically do not belong to data transformation. The most widely used description languages for web services do not annotate data for privacy concerns. Therefore, it makes sense to extend the descriptions used for web service transformations in order to add support for automated checks or automated anonymization of the choreographies. We have already allocated research efforts in this direction. Nevertheless, the ubiquity of web services encouraged our efforts to extend the executable choreographies with deeper support for web services and this has turned out to be an opportunity to create technologies that provide qualitative improvements for programmers' productivity.

6 Acknowledgements

This work was partially supported by the European Commission under the Horizon 2020 Programme (H2020), as part of the OPERANDO project (Grant Agreement no. 653704). Also this work is partially supported by POC-A1-A1.2.3-G-2015 program, as part of the PrivateSky project (P_40_371/13/01.09.2016).

References

[1] "OPERANDO," [Online]. Available: http://cordis.europa.eu/project/rcn/194891_en.html.

[2] D.A. Chappell, *Enterprise Service Bus: Theory in Practice,* O'Reilly Media, 2004, 276 p.

[3] "SwarmESB open source project," [Online]. Available: http://github.com/salboaie/SwarmESB.

[4] Sinica Alboaie, Lenuta Alboaie and Andrei Panu, "Levels of Privacy for e-Health systems in the cloud era," in *24th International Conference on Information Systems Development*, (Harbin, China), August 25-27, 2015, pp. 243–253.

[5] Lenuta Alboaie, Sinica Alboaie and Andrei Panu, "Swarm Communication-A Messaging Pattern Proposal for Dynamic Scalability in Cloud," in *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on IEEE*, 2013, pp. 1930–1937.

[6] Antonio Carzaniga, Pietro Picco Gian and Giovanni Vigna, "Designing distributed applications with mobile code paradigms," in *Proceedings of the 19th international conference on Software engineering*, (Boston, Massachusetts, USA – May 17 - 23, 1997), New York, NY, USA: ACM, 1997, pp. 22–32. ISBN:0-89791-914-9. DOI: 10.1145/253228.253236.

[7] Gul A. Agha, "Actors: A model of concurrent computation in distributed systems," Massachusetts Inst of Tech Cambridge Artificial Intelligence LAB, Rep. No. AI-TR-844, 1985.

[8] Robin Milner, *Communicating and mobile systems: the pi calculus*, Cambridge university press, 1999, 176 p. ISBN-10: 0521643201. ISBN-13: 978-0521643207.

[9] Thomas Erl, *Service-oriented architecture: concepts, technology, and design*, Pearson Education India, 2005, 600 p. ISBN-10: 0133858588. ISBN-13: 9780133858587.

[10] Florin-C. Pop, Marcel Cremene, Mircea-F. Vaida and Michel Riveill, “Natural language service composition with request disambiguation,” in *ICSOC 2010, Lecture Notes in Computer Science, volume: 6470*, pp. 670–677.

[11] *JSR-000352 Batch Applications for the Java^{TM} Platform*, 2014.

[12] Alfredo Cuzzocrea, ”A framework for modeling and supporting data transformation services over data and knowledge grids with real-time bound constraints,” *Concurrency and Computation: Practice and Experience*, vol. 23, no. 5, pp. 436–457, 2011.

[13] Lenuta Alboaie, Sinica Alboaie and Tudor Barbu, “Extending Swarm Communication to Unify Choreography and Long-lived Processes,” in *23rd International Conference on Information Systems Development (ISD 2014)*, 2014, pp. 375–382.

[14] [TransRest] implementation: http://github.com/salboaie/transrest.

[15] Steve McConnell, *Code complete*, Pearson Education, 2004, 914 p.

Sînică Alboaie, Lenuta Alboaie,
Mircea-Florin Vaida, Cristina Olariu

Received October 2, 2016

Sînică Alboaie[1,2]
[1]Technical University of Cluj-Napoca, Gh. Baritiu Street, 26-28,
Cluj-Napoca, Romania
[2]RomSoft Srl, Iasi, Romania, Research Department
E–mail: salboaie@gmail.com

Lenuta Alboaie
Faculty of Computer Science of the University “Al. I Cuza” of Iasi, Romania
E–mail: adria@info.uaic.ro

Mircea-Florin Vaida
Technical University of Cluj-Napoca, Communication Department,
Gh. Baritiu Street, 26-28, Cluj-Napoca, Romania
E–mail: mircea.vaida@com.utcluj.ro

Cristina Olariu
Faculty of Computer Science of the University ”Al. I Cuza” of Iasi, Romania
E–mail: cristina21olariu@gmail.com