Underspecified Relations with a Formal Language of Situation Theory

Roussanka Loukanova
Department of Mathematics, Stockholm University, Stockholm, Sweden

Keywords: Formal Language, Situation Theory, Information, Parametric Information, Partiality, Situations, Restrictions, Memory Variables.

Abstract: The paper is an introduction to a formal language of Situation Theory. The language provides algorithmic processing of situated information. We introduce specialized, restricted variables that are recursively constrained to satisfy type-theoretic conditions by restrictions and algorithmic assignments. The restricted variables designate recursively connected networks of memory locations for 'saving' parametric information that depends on situations and restrictions over objects. The formal definitions introduce richly informative typed language for classification and representation of underspecified, parametric, and partial information that is dependent on situations.

1 INTRODUCTION

Extensive work on Computational Semantics, e.g., by Barwise (Barwise, 1981), has shown that the existing approaches to computational semantics often concentrate on specific tasks, by leaving other important problems in semantics outside their scope. Situation Theory was originally introduced by Barwise (Barwise, 1981) to fill up such gaps by addressing a broad range of semantic phenomena that underlie crucial criteria for an adequate theory of meaning (Loukanova, 2010). As a result of these efforts, Barwise and Perry (Barwise and Perry, 1983) adopted a strategy of developing Situation Theory as a general model theory of information and its the fundamentals, in particular, by modelling relational and partial information, and its dependence on situations. They introduced Situation Semantics as one of the applications of Situation Theory, which provides models of semantic information as a special case of more general kinds of information. For an informal introduction to Situation Theory and Situation Semantics, see (Devlin, 2008). These fundamental ideas of situated information can be modeled by precise mathematics of Situation Theory (Loukanova, 2014) that provides mathematical structures of information as semantic domains of the formal language introduced in this paper. Such mathematical models of information have broad potentials for applications, e.g., to semantics of human languages (Loukanova, 2013c). The key contributions of such approach to Situation Theory is mathematical modelling of information that is situation dependent, partial, enhanced with intrinsic semantic parameters, and structured by types that carry information and information restrictions. This paper complements such mathematical models of Situation Theory with a formal language toward practical applications in advanced technologies.

A detailed review and analyses of existing work on Situation Theory and its applications is in forthcoming work (Loukanova, 2015). Here, we mention some of the key works. E.g., Seligman and Moss (Seligman and Moss, 2011) is a mathematical model theory of Situation Theory, in the spirit of (Loukanova, 2014). On the side of specialized languages for Satiation Theory, there have been developments of languages for programming and medium specifications of versions of Situation Theory, e.g., PROSIT, ASTL, and BABY-SIT (Tin and Akman, 1994; Tin et al., 1995; Tin and Akman, 1996). Limited versions of powerful theories, especially, when targeting specialized domains have been useful approach to applications in many areas, including in human language processing. Specialized languages for Situation Theory have been extensively used in Head-driven Phrase Structure Grammar (HPSG), for semantic representations including semantic underspecification. Current HPSG systems have been using semantic representations with the specification language Minimal Recursion Semantics (MRS), which is situation-theoretical in its nature (Copestake et al., 2005). Situation Semantics inspired other work in linguistics, e.g., for semantics of questions (Ginzburg and Sag, 2000), (Loukanova, 2013b), and for seman-

Loukanova R..
Underspecified Relations with a Formal Language of Situation Theory.
DOI: 10.5220/0005353402980309
In *Proceedings of the International Conference on Agents and Artificial Intelligence* (PUaNLP-2015), pages 298-309
ISBN: 978-989-758-073-4

tics of tense and aspect from cognitive perspective (Van Lambalgen and Hamm, 2004).

This paper is inspired by Moschovakis recursion (Moschovakis, 2006) and our work on its applications to computational semantics and computational syntax-semantics interface for human language (Loukanova, 2011), and its theoretical development (Loukanova, 2013a). Moschovakis recursion (Moschovakis, 2006) is a mathematical theory of a functional, formal language of recursion, which while having a two-sorted type system, is highly elegant from computational perspective. On the other side, our work in this paper is based on relational models of information, i.e., Situation Theory, which is a higher-order model-theory of partial, typed information. The type systems of Situation Theory and its formal language, which is introduced in this paper, are rich and finely-grained with informational content. Situation-theoretic types can have components consisting of partial situations, restricted parameters, and other kinds of typed objects. Higher-order mathematical models of such Situation Theory have been introduced in more precise details recently (Loukanova, 2014). Situation Theory in such lines has already proved useful for applications to computational semantics of human language (Loukanova, 2001; Loukanova, 2002a; Loukanova, 2002b) and more recently in (Loukanova, 2013c).

Situation Theory is model-theory of information, i.e., mathematical structures of mathematical objects. It deserves emphasizing that restricted parameters in such a typed Situation Theory, are not variables — they are specific objects in informational domains, that are organized according to the types of objects in them. situation-theoretical objects are mathematical, set-theoretic objects, some of which can be proper classes, or non-well-founded sets (Aczel, 1988; Rathjen, 2004). In application systems, typically these objects are sets. The important is that these sets, or classes, are not exhaustively given in their fullness. Rather, they are represented by finite number of finite rules for potential generation. Non-well founded phenomena that is inspired by Situation Theory has been studied by (Barwise and Etchemendy, 1995) and (Barwise and Moss, 1996). In this paper, we present a formal language for designating such situation-theoretical objects and the corresponding rules. In model-theoretic Situation Theory, restricted situation-theoretical parameters model either "undeveloped" or "partially known" objects. Such parametric objects are build up recursively from other situation-theoretical objects, including parameters. They are restricted parameters by satisfying various restrictions over their "unknown" ("undeveloped") components. By the language L_{ap}^{ST} introduced in this paper, we provide formal expressions, i.e., L_{ap}^{ST}-terms, including restricted variables, for denoting parametric objects. The L_{ap}^{ST}-terms provide algorithmic patterns for computing the situational objects they denote, which can represent complex and partial information about objects that may include parametric restrictions. Restricted parameters can be specified when more information is added depending on context.

We would like to stress that while Situation Theory and its applications emerged in 80's, mathematics of situation-theoretical models of finely-grained information and their corresponding formal and computational syntax are largely open topics, with newly emerging theoretical developments and applications. Both sides of development, on the one hand, Situation Theory, as a higher-order model theory of typed information, and on the other hand, a formal language with dependent types for it, are new directions of work. In particular, we are targeting a development of a formal language with dependent types for Situation Theory, as a system of Martin-Löf Dependent-Type Theory (MLDT), see (Martin-Löf, 1984; Univalent Foundations Program, 2013). Among the distinctive applications are computational semantics and neuroscience of language. The paper is based on work on several new directions, in particular: (1) type-theory of recursion (a functional approach), (2) relational type-theory of situated, partial, and parametric information (a relational approach), (3) applications of these theories to computational syntax-semantics interfaces in natural and formal languages.

The paper is an introduction to a formal language L_{ap}^{ST}, which has terms that designate objects of Situation Theory. (The subscript in L_{ap}^{ST} indicates that the language uses recursive assignments of specialized variables, representing semantic parameters, with acyclicity.) In particular, we take Situation Theory that is specialized with recursively restricted parameters, as full-standing objects in domains of information. The intended semantic domains of L_{ap}^{ST} are domains of mathematical, situation-theoretical structures of typed objects and information. Objects and information can be partial and parametric. Recursively restricted variables of the formal language L_{ap}^{ST} designate semantic, complex parameters of Situation Theory. We define recursion L_{ap}^{ST}-terms that determine both restrictions over and assignments to the variables. Recursion L_{ap}^{ST}-terms, i.e., terms with restricted variables and recursive assignments, designate networks of allocated slots, which are denoted by the restricted variables, in which abstract, parametric information is saved during computational steps represented by the systems of assignments. The re-

strictions over the variables represent what type of information can be saved in the corresponding memory slots.

In Section 2, we give the formal definitions of the vocabulary and the L_{ap}^{ST}-terms. In Section 3, we demonstrate applications of the formal language L_{ap}^{ST} by expressions with restricted recursion variables. Specialized recursion terms of L_{ap}^{ST} can be used to represent computational patterns. Technically, L_{ap}^{ST}-terms designate parametric objects and algorithms for computing and storing parametric information with restrictions. In this settings, algorithms consist of computational steps that handle situated and parametric information. Assignments of sub-terms to some of the restricted L_{ap}^{ST}-variables provide computational steps for information processing, by saving the outcomes in the memory slots designated by these variables. Assignments of sub-terms to some other restricted variables provide specific instantiations of parametric components. In general, the terms of the language L_{ap}^{ST} designate algorithms for handling relational information. In addition to L_{ap}^{ST}-terms denoting primitive and complex relations, some L_{ap}^{ST}-terms denote functions for designating operations over situated and parametric objects.

The terms of the formal language L_{ap}^{ST} are demonstrated with examples. We give intuitions about the denotational and algorithmic semantics of L_{ap}^{ST}. The denotational semantics provides the objects denoted by L_{ap}^{ST}-terms, while the algorithmic semantics of L_{ap}^{ST}-terms provides the computational patterns (or steps) for processing and saving information in memory networks.

2 BASIC SYNTAX OF L_{ap}^{ST}

2.1 Types of L_{ap}^{ST}

The class Types of L_{ap}^{ST} is defined recursively, by starting with a (relatively small) set of basic types. Note that some of the expressions of the formal language L_{ap}^{ST}, defined later, will be complex types. Types can be a set or a proper class, e.g., a non-well-founded set, depending of further expanding of the definitions of the L_{ap}^{ST}-terms for relevant applications.

Basic (Primitive) Types. The set BTypes can be chosen depending on a specific choice of L_{ap}^{ST} for practical applications. E.g., as a standard for Situation Theory (Loukanova, 2013c; Loukanova, 2014), we take a set BTypes of *basic (primitive) types*:

$$\mathsf{BTypes} = \{\text{IND}, \text{LOC}, \text{REL}, \text{FUN}, \text{POL}, \text{ARGR}, \text{INFON}, \text{SIT}, \text{PROP}, \text{PARAM}, \text{TYPE}, \models\} \quad (1)$$

where IND is the type for individuals; LOC, for space-time locations; REL, for relations, primitive and complex; FUN, for functions, primitive and complex; TYPE, for primitive and complex types; PARAM, for basic and complex parameters; POL, for two polarity objects, e.g., presented by the natural numbers 0 and 1; ARGR, for abstract argument roles, basic and complex; INFON, for situation-theoretical objects that are basic or complex information units; PROP, for abstract objects that are propositions; SIT, for situations; $\models$ is a designated type called "supports" (explained later).

Complex Types. More complex L_{ap}^{ST}-types will be formally defined in the following sections since they will be constructed recursively from other kinds of L_{ap}^{ST}-expressions. I.e., we will define the expressions of L_{ap}^{ST} and among them complex types. For any given expression A and a type T, we use the notation $A : T$ iff A is assigned to the type T, in which case we say "A is of type T". As in MLDT (Univalent Foundations Program, 2013), the construction of new, complex L_{ap}^{ST}-types can be left open-ended, for adding new types depending on needs, by following the L_{ap}^{ST}-rules for complex types, which we will give in the following sections, instead of accumulating all possible types in a class Types. E.g., by following the L_{ap}^{ST}-rules, we can construct the types at stages:

$$\mathsf{Types}_0, \mathsf{Types}_1, \ldots, \mathsf{Types}_n, \ldots \quad (2)$$

so that $\mathsf{Types}_i \subseteq \mathsf{Types}_{i+1}$, for all $i \geq 0$. The process can be left "open", for adding new expressions to the existing levels and new levels. "Closing" the process, by accumulating all Types_i, at once, into a collection Types can lead to a large collection Types, which may be a proper class that is not a set. In what follows, we shall use the notation Types to be either a collection, which can be a proper class, or the set of types Types_i for some stage i. E.g., we will often write $\tau \in \mathsf{Types}$ by considering that Types is the collection of types available at some stage of constructed expressions:

$$\tau \in \mathsf{Types} \iff \tau \in \mathsf{Types}_i, \text{ for some } i \geq 0 \quad (3a)$$

$$\tau : \text{TYPE} \iff \tau \in \mathsf{Types}_i, \text{ for some } i \geq 0 \quad (3b)$$

We will use the usual set-theoretical notations for union of collections, which can be union of proper classes rather than of sets. Technical details of such distinctions and stages involve considerable work by using methods of set theory and are beyond the subject of this paper.

2.2 Vocabulary

The vocabulary of L_{ap}^{ST} consists of pairwise disjoint classes (sets) of objects.

Constants. For each $\tau \in \mathsf{Types}$, L_{ap}^{ST} has a finite (or denumerable, depending on applications) set of constants: $K_\tau = \{ c_0^\tau, c_1^\tau, \ldots, c_{k_\tau}^\tau \}$. Depending on specific applications, the sets of constants can be finite. We allow some of the sets K_τ to be empty. In particular, we take non-empty sets of constants: K_{IND} – for *primitive individuals*, K_{LOC} – for *space-time locations*, K_{REL} – for *primitive relations*, K_{FUN} – for *primitive functions*, $K_{\text{POL}} = \{0,1\}$ – for *polarity values*.

$$K = \bigcup_{\tau \in \mathsf{Types}} K_\tau \tag{4}$$

When the type of the constants is understood, we use K instead of K_τ.

Pure Variables. L_{ap}^{ST} has a set (or a class) of typed *pure variables*,

$$\mathsf{PureVars} = \bigcup_{\tau \in \mathsf{Types}} \mathsf{PureVars}_\tau, \tag{5}$$

where, for each $\tau \in \mathsf{Types}$, $\mathsf{PureVars}_\tau = \{ v_0^\tau, v_1^\tau, \ldots \}$.

Restricted Memory Variables. The formal language L_{ap}^{ST} has a set, which can be a proper class, of typed, *restricted recursion variables*, which we also call *restricted variables* or *(restricted) memory variables*,

$$\mathsf{RestrRecVars} = \bigcup_{\tau \in \mathsf{Types}} \mathsf{RestrRecVars}_\tau, \tag{6}$$

where, $\mathsf{RestrRecVars}_\tau = \{ p_0^\tau, p_1^\tau, \ldots \}$, for each $\tau \in \mathsf{Types}$, i.e., $\tau : \text{TYPE}$. In particular, for each of the basic types, we take a set of basic restricted memory variables, which can be used for saving information and other objects of the associated type:

$$\mathcal{P}_{\text{IND}} = \{ a, b, c, \ldots \}, \quad \mathcal{P}_{\text{LOC}} = \{ l, l_0, l_1, \ldots \}, \tag{7a}$$

$$\mathcal{P}_{\text{REL}} = \{ r_0, r_1, \ldots \}, \quad \mathcal{P}_{\text{FUN}} = \{ f_0, f_1, \ldots \}, \tag{7b}$$

$$\mathcal{P}_{\text{POL}} = \{ p_0, p_1, \ldots \}, \quad \mathcal{P}_{\text{INFON}} = \{ i_0, i_1, \ldots \} \tag{7c}$$

$$\mathcal{P}_{\text{SIT}} = \{ s_0, s_1, \ldots \}. \tag{7d}$$

The sets of the variables of L_{ap}^{ST} are typed, respectively, for each $\tau \in \mathsf{Types}$:

$$\mathsf{Vars}_\tau = \mathsf{PureVars}_\tau \cup \mathsf{RestrRecVars}_\tau, \tag{8a}$$

$$\mathsf{Vars} = \mathsf{PureVars} \cup \mathsf{RestrRecVars} \tag{8b}$$

We use notations without types: Vars, PureVars, RestrRecVars, when the type of the variables is understood. See the discussion on page 3 about (3a)–(3b). Furthermore, we use the following notations, for every $\tau : \text{TYPE}$ and $v \in \mathsf{Vars}$:

$$v^\tau \in \mathsf{Vars} \iff v \in \mathsf{Vars}_\tau \iff v : \tau \tag{9}$$

For semantic reasons, we distinguish between restricted variables, which are syntactic objects, and semantic parameters as semantic objects. Restricted variables are variables in the formal language L_{ap}^{ST}. In addition, they are restricted with types, which can be simple or complex, to insure that the variables can be instantiated only with objects satisfying the restrictions. We should have such distinctions in mind, while their technical details are not in the subject of this paper. Sometimes, when the context makes it clear, the restricted variables are called parameters. Now, we provide an example to demonstrate the ideas behind restricted variables and the distinctions between variables and semantic parameters. The formal definitions of the syntax of such terms is given in the subsequent sections.

Example 2.1. In the following term, (10) the λ-abstractions bind pure variables. The situated term (10) is the type of situations x_s^{SIT} and locations x_l^{LOC}, where an individual denoted by a restricted variable a^{IND} walks. The restrictions over these variables constrain them to be in the respectively typed classes of variables. E.g., the restriction over a requires that $a \in \mathsf{RestrRecVars}_{\mathsf{IND}}$.

$$\lambda x_s^{\text{SIT}}, x_l^{\text{LOC}} (x_s^{\text{SIT}} \models \ll \mathsf{walk}, \mathsf{walker} : a^{\text{IND}}, \mathsf{Loc} : x_l^{\text{LOC}}; \mathsf{Pol} : 1 \gg) \tag{10}$$

The restricted memory variable a^{IND}, which occurs freely in the term (10), can denote only an individual in a given semantics domain, not a location or a situation. The individual denoted by a^{IND} can be specific and known to the interpreter. In addition, Situation Theory (Loukanova, 2013c; Loukanova, 2014) provides genuine semantic parameters that represent unknown or underdeveloped objects. E.g., an interpreter of the term (10) can assign a semantic parameter to the variable a^{IND}, without knowing what that individual is.

Typically, unless otherwise specified either explicitly or by the syntax of the expressions, we shall use letters x, y, z, with or without subscripts, to vary over pure variables of any types, and letters p, q, r, with or without subscripts, to vary over memory variables of any type.

2.3 Argument Roles

In symbolic approaches, the arguments to function and predicate expressions (or their respective denotations) typically are taken as linearly ordered. E.g., A can be a function (or predicate) expression used to denote a function (respectively, a predicate) with n arguments. An instantiation of these arguments by some objects denoted by $a_1, \ldots, a_n$ can be expressed by $A(a_1, \ldots, a_n)$. In fact, strictly speaking,

this means that A has n argument "slots" that are filled up by $a_1, \ldots, a_n$. The linear order makes it clear which slots are filled up by what expressions (or elements). The word "argument" is often overloaded to mean either the argument slots, e.g., the ones resulted in by λ-abstractions in a term like $\lambda x,y,z\ A$, or the objects to which the term applies, e.g., a, b, c in $(\lambda x,y,z\ A)(a,b,c)$. The context makes clear which of these interpretations is the case. In Situation Theory (Loukanova, 2013c; Loukanova, 2014), this context dependent distinction (in fact, ambiguity) is made explicit by the notion of an argument role, which corresponds to "an argument slot". The argument roles represent saturation requirements over functions, predicated, and types. Along this, the introduction of argument roles makes it possible to express complex appropriateness conditions over what objects can fill up these argument roles. Furthermore, using sets of explicit argument roles associated with functions and predicates makes it possible to take away the linear order over them. This corresponds to the relations between objects in nature, which take place in three dimensional space. Usually, the natural relations are not associated with order and do not impose order over the objects related, which can be located in various arrangements and distances in space and time. Mathematics and Computer Science have techniques for unambiguous encoding and distinguishing the argument roles and their respective fillers, by liner orders over them. These techniques meet the needs of expressing relations, functions, and operations, by the means of available writing systems (traditionally on paper) and computer programming. By the new advances of technologies, we have ways of modeling and depicting information in its natural ways — relations between objects taking place in space and time, not always associated with linear orders. Situation Theory reflects on this natural aspect of the structure of relations and operations (processes) that occur in nature and in information transferred in nature. Thus, it distinguishes between arguments roles of relations and operations and the objects filling up these roles. The formal language for Situation Theory, which we introduce here, reflects on this correspondingly. This does not preclude using traditional liner orders to express argument roles, and "suppress" their explicit appearance.

Definition 1 (Argument Roles)**.** 1. We assume a set of *basic argument roles*, which can be associated with the expressions for relations, functions, and types:

$$\mathcal{BA}_{\text{ARGR}} = \{\, \rho_1, \ldots, \rho_m, \ldots \,\}, \text{ for } m \geq 0 \quad (11)$$

2. A finite set of argument roles is assigned to each of the primitive relations and each of the primitive types, by a function *Args*, such that $Dom(Args) = \mathcal{A}_{\text{REL}} \cup B_{\text{TYPE}}$ and $Range(Args) \subseteq \mathcal{P}_{\mathsf{finite}}(\mathcal{BA}_{\text{ARGR}})$, where $\mathcal{P}_{\mathsf{finite}}(\mathcal{BA}_{\text{ARGR}})$ is the set of finite subsets of $\mathcal{BA}_{\text{ARGR}}$.
3. $\mathcal{A}_{\text{ARGR}}$, is a set (or class) of *basic and complex argument roles*,

$$\mathcal{BA}_{\text{ARGR}} \cup \{\, [\xi] \mid \xi \in \mathcal{P}_{\text{IND}} \cup \mathcal{P}_{\text{LOC}} \cup \mathcal{P}_{\text{SIT}} \cup \mathcal{P}_{\text{REL}} \cup \mathcal{P}_{\text{FUN}} \cup \mathcal{P}_{\text{POL}} \,\} \subseteq \mathcal{A}_{\text{ARGR}} \quad (12)$$

Note that the elements of the class $\mathcal{A}_{\text{ARGR}}$ are specialized expressions, i.e., specialized $\text{L}^{\text{ST}}_{\text{ap}}$-terms, which can vary depending on specific applications of $\text{L}^{\text{ST}}_{\text{ap}}$.

Example 2.2. Let smile, read, and give be relation constants. We can associate sets of argument roles with these constants, in a simple way, without any constraints over the arguments, as follows:

$$SArgs(\mathsf{smile}) \equiv \{\, \mathsf{arg}_1 \,\} \quad (13a)$$
$$SArgs(\mathsf{read}) \equiv \{\, \mathsf{arg}_1, \mathsf{arg}_2 \,\} \quad (13b)$$
$$SArgs(\mathsf{give}) \equiv \{\, \mathsf{arg}_1, \mathsf{arg}_2, \mathsf{arg}_3 \,\} \quad (13c)$$
$$SArgs(\gamma) \equiv \{\, \mathsf{arg}_1, \ldots, \mathsf{arg}_n \,\}, \quad (13d)$$

for any relation γ with n arguments ($n \in \mathbb{N}$).

Appropriateness Constraints. Typically, the basic relations, functions, and types are associated with argument roles that have to satisfy constraints for their appropriate filling. We represent such constraints with types.

Definition 2 (Basic Argument Roles with Appropriateness Constraints)**.** A set of argument roles is assigned to each of the primitive relations, and to each of the primitive types, by a function *Args* such that

$$Dom(Args) = (\mathcal{A}_{\text{REL}} \cup \mathcal{A}_{\text{FUN}} \cup B_{\text{TYPE}}) \quad (14a)$$
$$Range(Args) \subseteq \mathcal{P}(\mathcal{A}_{ARG} \times \mathcal{P}(\mathcal{T}_{\text{TYPE}})) \quad (14b)$$

I.e., for every primitive relation, function, and type γ, $\gamma \in \mathcal{A}_{\text{REL}} \cup \mathcal{A}_{\text{FUN}} \cup B_{\text{TYPE}}$, $Args(\gamma)$ is a set of ordered pairs:

$$Args(\gamma) = \{\, \langle \mathsf{arg}_1, T_1 \rangle, \ldots, \langle \mathsf{arg}_n, T_n \rangle \,\} \quad (15)$$

where $n \geq 0$, $\mathsf{arg}_1, \ldots, \mathsf{arg}_n \in \mathcal{A}_{ARG}$ and $T_1, \ldots, T_n$ are sets of types (basic or complex).

Definition 3 (Argument roles with appropriateness constraints)**.**

1. Every relation constant, relation variable, and basic type γ, $\gamma \in \mathcal{A}_{\text{REL}} \cup \mathsf{Vars}_{\text{REL}} \cup B_{\text{TYPE}}$, is associated with a set $Args(\gamma)$, called the set of argument roles of γ, so that:

$$Args(\gamma) \equiv \{\, T_1 : \mathsf{arg}_1, \ldots, T_n : \mathsf{arg}_n \,\} \quad (16)$$

where $n \geq 0$, $\mathsf{arg}_1, \ldots, \mathsf{arg}_n \in \mathcal{A}_{ARG}$, and $T_1, \ldots, T_n$ are sets of types (basic or complex).
The expressions $\mathsf{arg}_1, \ldots, \mathsf{arg}_n$ are called the *argument roles* (or the *argument slots*) of γ. The sets of types $T_1, \ldots, T_n$ are specific for the argument roles of γ and are called the *basic appropriateness constraints of the argument roles* of γ.

2. Every function constant and every function variable γ, $\gamma \in \mathcal{A}_{\text{FUN}} \cup \mathsf{Vars}_{FUN}$, is associated with two sets, $Args(\gamma)$, called the set of argument roles of γ, and $Value(\gamma)$, called the (singleton set of the) value role of γ, so that:

$$Args(\gamma) \equiv \{\, T_1 : \mathsf{arg}_1, \ldots, T_n : \mathsf{arg}_n \,\} \quad (17a)$$
$$Value(\gamma) \equiv \{\, T_{n+1} : \mathsf{arg}_{n+1} \,\} \quad (17b)$$

where $n \geq 0$, $\mathsf{arg}_1, \ldots, \mathsf{arg}_{n+1} \in \mathcal{A}_{ARG}$, and $T_1, \ldots, T_{n+1}$ are sets of types (basic or complex).
The expressions $\mathsf{arg}_1, \ldots, \mathsf{arg}_n$ are called the *argument roles* (or the *argument slots*) of γ. The expression arg_{n+1} is called the *value role* of γ. The sets of types $T_1, \ldots, T_{n+1}$ are specific for the argument roles of γ and are called, respectively, the *basic appropriateness constraints of the arguments* and of the *value* of γ.

2.4 Terms of $\mathrm{L}_{\mathrm{ap}}^{\mathrm{ST}}$

$$\mathsf{Terms}(K) = \bigcup_{\tau \in \mathsf{Types}} \mathsf{Terms}_\tau \quad (18)$$

where the sets Terms_τ are defined recursively as follows.

The typed classes of $\mathrm{L}_{\mathrm{ap}}^{\mathrm{ST}}$-terms, Terms_τ, for $\tau \in \mathsf{Types}$, are defined recursively as follows.

Constants. If $\mathsf{c} \in K_\tau$, then $\mathsf{c} \in \mathsf{Terms}_\tau$, (i.e., every constant of type τ is also a term of type τ) denoted $\mathsf{c} : \tau$. There are no free and no bound occurrences of variables in the term c:

$$\mathsf{FreeVars}(\mathsf{c}) = \varnothing \text{ and } \mathsf{BoundVars}(\mathsf{c}) = \varnothing \quad (19)$$

Variables. If $x \in \mathsf{Vars}_\tau$, then $x : \tau$, i.e., every variable of type τ is also a term of type τ. The only occurrence of the variable x in the term x is free; there are no bound occurrences of variables in x, i.e.:

$$\mathsf{FreeVars}(x) = \{\, x \,\} \text{ and } \mathsf{BoundVars}(x) = \varnothing \quad (20)$$

Infon Terms. For every relation term (basic or complex) $\rho \in \mathsf{Terms}_{\text{REL}}$, associated with argument roles $Args(\rho) = \{\, T_1 : \mathsf{arg}_1, \ldots, T_n : \mathsf{arg}_n \,\}$, and every sequence of terms $\xi_1, \ldots, \xi_n$ such that $\xi_1 \in \mathsf{Terms}_{T_1}$, $\ldots$, $\xi_n \in \mathsf{Terms}_{T_n}$, (i.e., terms $\xi_1, \ldots, \xi_n$ that satisfy the corresponding appropriateness constraints of the argument roles of ρ), $\xi_1 : T_1, \ldots, \xi_n : T_n$, every space-time location term $\tau \in \mathsf{Terms}_{\text{LOC}}$, (i.e., $\tau : \text{LOC}$), and every polarity term $t \in \mathsf{Terms}_{POL}$, ($t : POL$, i.e., $t \in \{0,1\} \cup \mathcal{P}_{POL}$), the expression in (21) is an *infon term*:

$$\begin{aligned} \ll \rho, & T_1 : \mathsf{arg}_1 : \xi_1, \ldots, \\ & T_n : \mathsf{arg}_n : \xi_n, \\ & \text{LOC} : \mathsf{Loc} : \tau, \\ & \text{POL} : \mathsf{Pol} : t \gg \in \mathsf{Terms}_{\text{INFON}} \end{aligned} \quad (21)$$

The expression in (22) is the full infon term, which includes its type association. It is used when the type labeling is relevant.

$$\begin{aligned} \ll \rho, & T_1 : \mathsf{arg}_1 : \xi_1, \ldots, \\ & T_n : \mathsf{arg}_n : \xi_n, \\ & \text{LOC} : \mathsf{Loc} : \tau, \text{ POL} : \mathsf{Pol} : t \gg : \text{INFON} \end{aligned} \quad (22)$$

All free (bound) occurrences of variables in ρ, $\xi_1, \ldots, \xi_n$, τ are also free (bound) in the infon term.

Notation 1. Often, in this paper, we shall use the notation (23a) when the type constraints over the argument roles are irrelevant, or (23b), when in addition, there is an understood order of the arguments.

$$\ll \rho, \mathsf{arg}_1 : \xi_1, \ldots, \mathsf{arg}_n : \xi_n, \mathsf{Loc} : \tau; t \gg \quad (23a)$$
$$\ll \rho, \xi_1, \ldots, \xi_n, \tau; t \gg \quad (23b)$$

Proposition Terms. The expression in (24) is a *basic proposition term*

$$\begin{aligned} (\gamma, & T_1 : \mathsf{arg}_1 : \xi_1, \ldots, T_n : \mathsf{arg}_n : \xi_n, \\ & \text{LOC} : \mathsf{Loc} : \tau, \text{ POL} : \mathsf{Pol} : t) \ : \ \text{PROP} \end{aligned} \quad (24)$$

for every type term $\gamma \in \mathsf{Terms}_{\text{TYPE}}$, associated with argument roles $Args(\gamma) \equiv \{\, T_1 : \mathsf{arg}_1, \ldots, T_n : \mathsf{arg}_n \,\}$, every sequence of terms $\xi_1, \ldots, \xi_n$ such that $\xi_1 \in \mathsf{Terms}_{T_1}, \ldots, \xi_n \in \mathsf{Terms}_{T_n}$ (i.e., terms $\xi_1, \ldots, \xi_n$ that satisfy the corresponding appropriateness constraints of the argument roles of γ, $\xi_1 : T_1, \ldots, \xi_n : T_n$), every space-time location expression $\tau : \text{LOC}$, and every polarity expression $t : \text{POL}$.

Notation 2. The components for a space-time location $\text{LOC} : \mathsf{Loc} : \tau$ and for a polarity $\text{POL} : \mathsf{Pol} : t$ can be omitted, as in the expression in (25a), when they are irrelevant or understood. In such cases, the polarity t is understood to be positive, i.e., $t = 1$. In addition, expressions like that in (25b), omitting their type association to PROP, are used when the type labeling PROP is not relevant.

$$(\gamma, T_1 : \mathsf{arg}_1 : \xi_1, \ldots, T_n : \mathsf{arg}_n : \xi_n,) : \text{PROP} \quad (25a)$$
$$(\gamma, T_1 : \mathsf{arg}_1 : \xi_1, \ldots, T_n : \mathsf{arg}_n : \xi_n) \quad (25b)$$

All the free (bound) occurrences of variables in γ, ξ_1, ..., ξ_n are also free (bound) in the proposition term.

Notation 3. Often, in this paper, we shall use the notational abbreviation (26a) when the type constraints over the argument roles are irrelevant, or (26b) when there is an understood order and type constraints of the arguments.

$$(\gamma, \mathsf{arg}_1 : \xi_1, \ldots, \mathsf{arg}_n : \xi_n) \tag{26a}$$
$$(\gamma, \xi_1, \ldots, \xi_n) \tag{26b}$$

Complex proposition terms are formed from simpler ones by the logic operations $\wedge$, $\vee$, $\rightarrow$, and quantification.

Application Terms. For every function term, basic or complex, $\gamma \in \mathsf{Terms}_{\text{FUN}}$, which comes associated with argument roles $\mathit{Args}(\gamma) \equiv \{ T_1 : \mathsf{arg}_1, \ldots, T_n : \mathsf{arg}_n \}$ and with a value role $\mathit{Value}(\gamma) \equiv \{ T_{n+1} : \mathsf{val} \}$, and for every sequence of terms $\xi_1 \in \mathsf{Terms}_{T_1}$, ..., ξ_n, $\xi_{n+1} \in \mathsf{Terms}_{T_{n+1}}$, i.e., terms $\xi_1, \ldots, \xi_n, \xi_{n+1}$ that satisfy the corresponding appropriateness constraints of the argument and value roles of γ, $\xi_1 : T_1$, ..., $\xi_{n+1} : T_{n+1}$, the expression in (27) is an *application term*:

$$\begin{aligned} &\gamma\{T_1 : \mathsf{arg}_1 : \xi_1, \ldots, \\ &\quad T_n : \mathsf{arg}_n : \xi_n\} \ \in \ \mathsf{Terms}_{T_{n+1}} \end{aligned} \tag{27}$$

Informally said, the term (27) represents application of the function term γ to the argument terms ξ_1, ..., ξ_n that fill up the corresponding argument roles arg_1, ..., arg_n of γ. In addition, the term (27) expresses that arg_1, ..., arg_n satisfy the corresponding appropriateness conditions T_1, ..., T_n, otherwise (27) is not well-formed. The full expression in (28), which includes the type association, is used when the type labeling is relevant.

$$\gamma\{T_1 : \mathsf{arg}_1 : \xi_1, \ldots, T_n : \mathsf{arg}_n : \xi_n\} : T_{n+1} \tag{28}$$

The expression (29) represents the proposition that the value of the function γ for the given argument "fillers" ξ_1, ..., ξ_n has to be ξ_{n+1}.

$$\begin{aligned} &\big(\gamma\{T_1 : \mathsf{arg}_1 : \xi_1, \ldots, \\ &\quad T_n : \mathsf{arg}_n : \xi_n\} = \xi_{n+1}\big) \in \mathsf{Terms}_{\text{PROP}} \end{aligned} \tag{29}$$

All free (bound) occurrences of variables in γ, ξ_1, ..., ξ_n, ξ_{n+1} are also free (bound) in the application term in (27), respectively, in (28) and (29).

λ-abstraction Terms. Case 1: Complex Relations with Complex Arguments. For every infon term $I \in \mathsf{Terms}_{\mathsf{INFON}}$ (basic or complex) and all pure variables $\xi_1, \ldots, \xi_n \in \mathsf{PureVars}$ (which may occur freely in I), the expression $\lambda\{\xi_1, \ldots, \xi_n\} I$ is a *complex-relation term*, i.e.:

$$\lambda\{\xi_1, \ldots, \xi_n\} I \in \mathsf{Terms}_{\text{REL}} \tag{30a}$$
$$\lambda\{\xi_1, \ldots, \xi_n\} I : \text{REL} \tag{30b}$$

The argument roles of $\lambda\{\xi_1, \ldots, \xi_n\} I$, which we denote by $[\xi_1], \ldots, [\xi_n]$, are associated with corresponding *appropriateness constraints* as follows:

$$\begin{aligned} &\mathit{Args}(\lambda\{\xi_1, \ldots, \xi_n\} I) \\ &\equiv \{ T_1 : [\xi_1], \ldots, T_n : [\xi_n] \} \end{aligned} \tag{31}$$

where, for each $i \in \{1, \ldots, n\}$, T_i is the union of all types that are the appropriateness constraints of all the argument roles that occur in I, and such that ξ_i fills up them without being bound. (Note that ξ_i may fill more than one argument role in I.)

Case 2: Complex Types with Complex Arguments. For every proposition term $\theta \in \mathsf{Terms}_{\text{PROP}}$ (basic or complex) and all pure variables $\xi_1, \ldots, \xi_n \in \mathsf{PureVars}$ (which may occur freely in θ, in the interesting cases), the expression $\lambda\{\xi_1, \ldots, \xi_n\} \theta$ is a *complex-type term*, i.e.:

$$\lambda\{\xi_1, \ldots, \xi_n\} \theta \in \mathsf{Terms}_{\text{TYPE}} \tag{32a}$$
$$\lambda\{\xi_1, \ldots, \xi_n\} \theta : \text{TYPE} \tag{32b}$$

The argument roles of $\lambda\{\xi_1, \ldots, \xi_n\}\theta$, which we denote by $[\xi_1], \ldots, [\xi_n]$, are associated with corresponding *appropriateness constraints* to be of the types T_1, ..., T_n, respectively, i.e., $\{ T_1 : [\xi_1], \ldots, T_n : [\xi_n] \}$, as follows:

$$\begin{aligned} &\mathit{Args}(\lambda\{\xi_1, \ldots, \xi_n\} \theta) \\ &\equiv \{ T_1 : [\xi_1], \ldots, T_n : [\xi_n] \} \end{aligned} \tag{33}$$

where, for each $i \in \{1, \ldots, n\}$, T_i is the union of all types that are the appropriateness constraints of all the argument roles that occur in θ and such that ξ_i fills up them, without being bound. (Note that ξ_i may fill more than one argument role in θ.)

Case 3: Complex Functions (Operations) with Complex Arguments. For every term $\varphi \in \mathsf{Terms}_{\tau}$ (basic or complex), where $\tau \in \mathsf{Types}$, $\tau \not\equiv$ INFON, and $\tau \not\equiv$ PROP, and for all pure variables $\xi_1, \ldots, \xi_n \in \mathsf{PureVars}$ (which may occur freely in φ), the expression $\lambda\{\xi_1, \ldots, \xi_n\} \varphi$ is a *complex-function term*:

$$\lambda\{\xi_1, \ldots, \xi_n\} \varphi \in \mathsf{Terms}_{\text{FUN}} \tag{34a}$$
$$\lambda\{\xi_1, \ldots, \xi_n\} \varphi : \text{FUN} \tag{34b}$$

The function term $\lambda\{\xi_1, \ldots, \xi_n\}\varphi$ has a value role, $\mathit{ValueRole}(\varphi) \equiv \{ \tau : \mathsf{val} \}$, and argument roles, denoted by $[\xi_1], \ldots, [\xi_n]$, which are associated with corresponding *appropriateness constraints* as follows:

$$\begin{aligned} &\mathit{Args}(\lambda\{\xi_1, \ldots, \xi_n\} \varphi) \\ &= \{ T_1 : [\xi_1], \ldots, T_n : [\xi_n] \} \end{aligned} \tag{35}$$

where, for each $i \in \{1,\ldots,n\}$, T_i is the union of all types that are the appropriateness constraints of all the argument roles that occur in φ, and such that ξ_i fills up them, without being bound. (Note that ξ_i may fill more than one argument role in φ.)

In the above Cases 1, 2, 3, the newly constructed terms have a common form $\lambda\{\xi_1,\ldots,\xi_n\}\phi$, for a subterm ϕ of the respective type, i.e., infon, type, or neither of these. All the free occurrences of $\xi_1,\ldots,\xi_n$ in ϕ are bound in the new term $\lambda\{\xi_1,\ldots,\xi_n\}\phi$. All other free (bound) occurrences of variables in ϕ are free (bound) in the term $\lambda\{\xi_1,\ldots,\xi_n\}\phi$.

$$\begin{aligned}&\mathsf{FreeVars}\Big(\lambda\{\xi_1,\ldots,\xi_n\}\phi\Big)\\ &= \mathsf{FreeVars}(\phi) - \{\xi_1,\ldots,\xi_n\}\end{aligned} \tag{36}$$

$$\begin{aligned}&\mathsf{BoundVars}\Big(\lambda\{\xi_1,\ldots,\xi_n\}\phi\Big)\\ &= \mathsf{BoundVars}(\phi) \cup \{\xi_1,\ldots,\xi_n\}\end{aligned} \tag{37}$$

Notation 4. The terms $\lambda\{\xi_1,\ldots,\xi_n\}\chi$, from the above Cases 1, 2, 3, are alternatively denoted by

$$[\xi_1,\ldots,\xi_n \mid \chi] \tag{38a}$$

$$[T_1:\xi_1,\ldots,T_n:\xi_n \mid \chi]. \tag{38b}$$

The notation (38b) is used when the type restrictions over the new, complex argument roles $[\xi_1]$, …, $[\xi_n]$ are understood from the context, or are irrelevant.

Restricted Recursion Terms. For any given

- $\mathrm{L}^{\mathrm{ST}}_{\mathrm{ap}}$-terms for unary types $\mathsf{C}_k \in \mathsf{Terms}_{\mathrm{TYPE}}$ (i.e., $\mathsf{C}_k : \mathrm{TYPE}$), with $\mathit{Args}(\mathsf{C}_k) \equiv \{T_k : \mathit{arg}_k\}$ (i.e., C_k has a single argument role arg_k that is restricted to be filled up by expressions of type T_k) and all pairwise different, restricted variables $\mathsf{q}_k \in \mathsf{RestrRecVars}_{T_k}$, for $k = 1,\ldots,m$ ($m \geq 0$), such that

  $$\{(\mathsf{q}_1:\mathsf{C}_1),\ldots,(\mathsf{q}_m:\mathsf{C}_m)\}$$

  is a sequence of type restrictions that satisfies the acyclicity constraint given in Definition 4, and
- $\mathrm{L}^{\mathrm{ST}}_{\mathrm{ap}}$-terms $A_i \in \mathsf{Terms}_{\sigma_i}$ (i.e., $\mathsf{A}_i : \sigma_i$), for $i = 0,\ldots,n$, and pairwise different, restricted variables $\mathsf{p}_i \in \mathsf{RestrRecVars}_{\sigma_i}$, for $i = 1,\ldots,n$ ($n \geq 0$), such that

  $$\{\mathsf{p}_1 := \mathsf{A}_1,\ldots,\mathsf{p}_n := \mathsf{A}_n\}$$

  is a sequence of assignments that satisfies the acyclicity constraint given in Definition 5, the following expression is a *restricted recursion term of type* σ_0:

$$\begin{aligned}&\mathsf{A}_0 \text{ where } \{(\mathsf{q}_1:\mathsf{C}_1),\ldots,(\mathsf{q}_m:\mathsf{C}_m)\}\\ &\qquad\{(\mathsf{p}_1 := \mathsf{A}_1),\ldots,(\mathsf{p}_n := \mathsf{A}_n)\}\\ &: \sigma_0\end{aligned} \tag{39}$$

All free occurrences of $\mathsf{p}_1,\ldots,\mathsf{p}_n$ in A_0, …, A_n are bound in the term $[\mathsf{A}_0 \text{ where } \{\mathsf{p}_1 := \mathsf{A}_1,\ldots,\mathsf{p}_n := \mathsf{A}_n\}]$. All other free (bound) occurrences of variables in $\mathsf{A}_1,\ldots,\mathsf{A}_n$ are free (bound) in the term $[\mathsf{A}_0 \text{ where } \{\mathsf{p}_1 := \mathsf{A}_1,\ldots,\mathsf{p}_n := \mathsf{A}_n\}]$.

$$\begin{aligned}&\mathsf{FreeVars}\Big(\mathsf{A}_0 \text{ where } \{\mathsf{p}_1 := \mathsf{A}_1,\ldots,\mathsf{p}_n := \mathsf{A}_n\}\Big)\\ &= \bigcup \mathsf{p}^n_{i=0}(\mathsf{FreeVars}(\mathsf{A}_i)) - \{\mathsf{p}_1,\ldots,\mathsf{p}_n\}\end{aligned} \tag{40}$$

$$\begin{aligned}&\mathsf{BoundVars}\Big(\mathsf{A}_0 \text{ where } \{\mathsf{p}_1 := \mathsf{A}_1,\ldots,\mathsf{p}_n := \mathsf{A}_n\}\Big)\\ &= \bigcup \mathsf{p}^n_{i=0}(\mathsf{BoundVars}(\mathsf{A}_i)) \cup \{\mathsf{p}_1,\ldots,\mathsf{p}_n\}\end{aligned} \tag{41}$$

Sometimes we enclose the recursion terms in extra brackets to separate them from the surrounding text, for example as in:

$$[\mathsf{A}_0 \text{ where } \{\mathsf{p}_1 := \mathsf{A}_1,\ldots,\mathsf{p}_n := \mathsf{A}_n\} : \sigma_0]$$

In the cases when $m = 0$ or $n = 0$, the respective sequences of restrictions and assignments are empty and can be denoted explicitly by $\{\}$.

Definition 4 (Acyclicity Constraint for Type Restrictions)**.** For any unary types $\mathsf{C}_k : \mathrm{TYPE}$ and $q_k \in \mathsf{RestrRecVars}$ (where q_k is of the type of the argument role of C_k), $k = 0,\ldots,m$ ($m \geq 0$), the sequence of type restrictions

$$\{(\mathsf{q}_1:\mathsf{C}_1),\ldots,(\mathsf{q}_m:\mathsf{C}_m)\}$$

is *acyclic* if and only if there is a ranking function $\mathsf{rank} : \{\mathsf{q}_1,\ldots,\mathsf{q}_m\} \longrightarrow \mathbb{N}$ such that, for all $\mathsf{q}_i,\mathsf{q}_j \in \{\mathsf{q}_1,\ldots,\mathsf{q}_m\}$, if q_j occurs freely in C_i, then $\mathsf{rank}(\mathsf{q}_j) < \mathsf{rank}(\mathsf{q}_i)$. Note that the acyclicity constraint is a proper part of the recursive definition of the $\mathrm{L}^{\mathrm{ST}}_{\mathrm{ap}}$-terms.

Definition 5 (Acyclicity Constraint for Assignments)**.** A sequence of assignments

$$\{\mathsf{p}_1 := \mathsf{A}_1,\ldots,\mathsf{p}_n := \mathsf{A}_n\}$$

is *acyclic* if and only if there is a ranking function $\mathsf{rank} : \{\mathsf{p}_1,\ldots,\mathsf{p}_n\} \longrightarrow \mathbb{N}$ such that, for all memory variables $\mathsf{p}_i,\mathsf{p}_j \in \{\mathsf{p}_1,\ldots,\mathsf{p}_n\}$, if p_j occurs freely in A_i, then $\mathsf{rank}(\mathsf{p}_j) < \mathsf{rank}(\mathsf{p}_i)$. Note that the acyclicity constraint is a proper part of the recursive definition of the $\mathrm{L}^{\mathrm{ST}}_{\mathrm{ap}}$-terms.

2.5 Underspecified Terms for Situated Propositions

Space-time Relations. We assume, that the collection K_{REL} includes relation constants for time precedence $\prec$, time overlapping $\circ$, space overlapping $\diamond$, and time, space, and space-time inclusions between

locations, respectively denoted by $\subseteq_t$, $\subseteq_s$, $\subseteq$. With these relations, we can form specialized infons, e.g.:

$$\ll \circ, \mathsf{l}_1, \mathsf{l}_2; 1 \gg, \quad \text{denoted by } \mathsf{l}_1 \circ \mathsf{l}_2 \tag{42a}$$
$$\ll \diamond, \mathsf{l}_1, \mathsf{l}_2; 1 \gg, \quad \text{denoted by } \mathsf{l}_1 \diamond \mathsf{l}_2 \tag{42b}$$
$$\ll \prec, \mathsf{l}_1, \mathsf{l}_2; 1 \gg, \quad \text{denoted by } \mathsf{l}_1 \prec \mathsf{l}_2 \tag{42c}$$
$$\ll \subseteq_t, \mathsf{l}_1, \mathsf{l}_2; 1 \gg, \quad \text{denoted by } \mathsf{l}_1 \subseteq_t \mathsf{l}_2 \tag{42d}$$
$$\ll \subseteq_s, \mathsf{l}_1, \mathsf{l}_2; 1 \gg, \quad \text{denoted by } \mathsf{l}_1 \subseteq_s \mathsf{l}_2 \tag{42e}$$
$$\ll \subseteq, \mathsf{l}_1, \mathsf{l}_2; 1 \gg \quad \text{denoted by } \mathsf{l}_1 \subseteq \mathsf{l}_2 \tag{42f}$$

These infons are usually given by their abbreviated infix notations, e.g., $\mathsf{l}_1 \circ \mathsf{l}_2$, $\mathsf{l}_1 \prec \mathsf{l}_2$ $\mathsf{l}_1 \subseteq_t \mathsf{l}_2$, $\mathsf{l}_1 \subseteq_s \mathsf{l}_2$, $\mathsf{l}_1 \subseteq \mathsf{l}_2$.

2.6 Situated Proposition Terms

An expression like (43), in order to be well-formed expression of the language $\mathrm{L}_{\mathrm{ap}}^{\mathrm{ST}}$, i.e., a proposition $\mathrm{L}_{\mathrm{ap}}^{\mathrm{ST}}$-term, has to be composed of component sub-expressions (T_1, …, T_n, ξ_1, …, ξ_n, etc.) that are of the corresponding types in the syntax of $\mathrm{L}_{\mathrm{ap}}^{\mathrm{ST}}$.

$$(\gamma, T_1 : \mathsf{arg}_1 : \xi_1, \ldots, T_n : \mathsf{arg}_n : \xi_n, \\ \text{LOC} : \mathit{Loc} : \tau, \text{POL} : \mathit{Pol} : t) : \text{PROP} \tag{43}$$

Intuitively, a proposition term (43) (correspondingly, (24), (25a)–(25b), (26a)–(26b)) expresses the proposition that the objects denoted by ξ_1, …, ξ_n, are (or are not) of the type denoted by γ, in case the polarity $t = 1$ ($t = 0$). Both propositions, with $t = 1$ or $t = 0$, exist, i.e., are defined either positively for $t = 1$, or negatively for $t = 0$, iff $\xi_1, \ldots, \xi_n$ denote objects of the respective types denoted by $T_1, \ldots, T_n$. These type constraints over the argument roles of γ are intrinsic components of the propositions denoted by the expression (43). I.e., the "component" sub-expressions, in the argument structure of (43), constrain the argument roles of $\mathsf{den}(\gamma)$, designated by $\mathsf{arg}_1, \ldots, \mathsf{arg}_n$, to be filled by objects $\mathsf{den}(\xi_1), \ldots, \mathsf{den}(\xi_n)$, for which the corresponding propositions $(\mathsf{den}(\xi_1) : \mathsf{den}(T_1))$, …, $(\mathsf{den}(\xi_n) : \mathsf{den}(T_n))$ hold, i.e., these propositions have to be true. In case any one of these "component" propositions does not hold, either by being false or by being nonsense error, the proposition term (43) (in each case $t = 1$ or $t = 0$) does not denote anything, or denotes an error, assuming that the semantic structure of $\mathrm{L}_{\mathrm{ap}}^{\mathrm{ST}}$ includes such an element error.

We distinguish between the term expressions for propositions and the propositions they denote. We say "the term denoting the proposition that $\xi_1, \ldots, \xi_n$ denote objects of the type denoted by γ", or simply, "the term denoting the proposition that $\xi_1, \ldots, \xi_n$ are of the type γ", when there is no ambiguity between expressions and what they denote.

An important class of propositions consists of the following situated propositions, which use the designated "support" type $\models$. The type $\models$ is associated with two argument roles, one for objects of the type SIT of situations, and the other for objects of the type *INF* of inforns. I.e.:

$$\mathit{Args}(\models) \equiv \{\, \text{SIT} : \mathit{arg}_{\mathit{sit}}, \text{INFON} : \mathit{arg}_{\mathit{infon}} \,\} \tag{44}$$

Definition 6 (Terms for Situated Propositions)**.** Any proposition term of the form (45)

$$(\models,\ \text{SIT} : \mathit{arg}_{\mathit{sit}} : \mathsf{s},\ \text{INFON} : \mathit{arg}_{\mathit{infon}} : \sigma) \tag{45}$$

where $s \in \mathcal{P}_{\text{SIT}}$ and $\sigma \in \mathsf{Terms}_{\text{INFON}}$, is called *a term for situated propositions*. The expression (46) is an abbreviation of the term (45).

$$(\mathsf{s} \models \sigma) \tag{46}$$

Usually, we pronounce terms of the forms (45) and (46) as "the term denoting the proposition that σ holds in the situation s" or "the term denoting the proposition that the situation s supports the infon σ". We distinguish between the term expressions for propositions and the propositions they denote. When the context does not cause ambiguity with respect to this difference, we simply say "the proposition that σ holds in the situation s" or "the proposition that the situation s supports the infon σ".

Example 2.3. Assume that book and read are constants with the associated argument roles given in the corresponding infon sub-terms of the proposition term (47a)–(47b). The term (47a)–(47b) has a single, but complex, conjunctive informational unit with basic infons as parts. The entire term (47a)–(47b) denotes a proposition that in situation s an individual denoted by the pure variable x reads the object denoted by the parameter variable b, in the location denoted by the parameter variable l, and in the same situation s the object b has the property of being a book in a broader location denoted by l_1: $\mathsf{l} \subseteq_t \mathsf{l}_1$ (l is time included in l_1), $\mathsf{l} \subseteq_s \mathsf{l}_1$ (l is space included in l_1).

$$(\mathsf{s} \models \ll \mathsf{book}, \mathsf{arg} : \mathsf{b}, \mathsf{Loc} : \mathsf{l}_1; 1 \gg \tag{47a}$$
$$\wedge \ll \mathsf{read}, \mathsf{reader} : \mathsf{x}, \mathsf{readed} : \mathsf{b}, \mathsf{Loc} : \mathsf{l}; 1 \gg \tag{47b}$$
$$\wedge\ \mathsf{l} \subseteq_t \mathsf{l}_1 \wedge \mathsf{l} \subseteq_s \mathsf{l}_1) \tag{47c}$$

3 APPLICATIONS

This section demonstrates applications of the syntax of the formal language $\mathrm{L}_{\mathrm{ap}}^{\mathrm{ST}}$ for computational representation of relations that include situated and underspecified information. The underspecification can be given via parameters that are partly specified with minimal restrictions depending on specific cases of use.

3.1 Typed Restrictions

In general, underspecified information is represented by typed variables. Usually, information represented by both kinds of variables, pure and memory variables, is partly specified by their type. Their type restriction, which is represented by a type term R, over a variable v, is included as a component $(v:R)$ of the term in which the variable occurs. Sometimes, the type restriction, or a part of the restriction, is given as a superscript over the variable: v^R. A major way to represent underspecified information is to use terms with components that are memory variables (locations) restricted by complex type terms. In addition, a term A can be assigned to a memory variable $p := A$, which can be also restricted $(p:R)$. The term A can itself be underspecified, by having components with restricted memory variables. In the following terms, the λ-abstractions bind pure variables. Assume that I is the following infon term underspecified for the memory locations b, l, while x is pure variable.

$$I \equiv \ll \mathsf{book}, \mathsf{arg}:\mathsf{b}, \mathsf{Loc}:\mathsf{l}; \mathsf{Pol}:1 \gg \wedge \quad (48a)$$

$$\ll \mathsf{read}, \mathsf{reader}:\mathsf{x}, \mathsf{readed}:\mathsf{b}, \quad (48b)$$

$$\mathsf{Loc}:\mathsf{l}; \mathsf{Pol}:1 \gg$$

The type term (49) is the type of individuals that read a particular book $\mathsf{b} \in \mathsf{RestrRecVars}_{\mathrm{IND}}$, in a particular situation $\mathsf{s} \in \mathsf{RestrRecVars}_{\mathrm{SIT}}$, in a particular location $\mathsf{l} \in \mathsf{RestrRecVars}_{\mathrm{LOC}}$. Note that *s* is a memory location (variable) for a situation that is only partly specified by the parametric infons in the term (49).

$$\lambda \mathsf{x}\,(\mathsf{s} \models I) \quad (49)$$

The type term (50) is the type of relations between a situation, a location and an individual, where the individual reads a particular book $\mathsf{b} \in \mathsf{RestrRecVars}_{\mathrm{IND}}$.

$$\lambda \mathsf{x}_s^{\mathrm{SIT}}, \mathsf{l}, \mathsf{x}\,(\mathsf{x}_s^{\mathrm{SIT}} \models I) \quad (50)$$

The sub-term (51b)–(51e) denotes an abstract relation, consisting of informational units, which can be used in proposition terms, e.g., as in (51b)–(51e). The relation denoted by (51b)–(51e) has new, its own argument-roles, denoted by $[x]$, $[y]$, and $[z]$, which are filled up by the objects respectively denoted by *a*, *b*, and *c*.

$$\big(\mathsf{s} \models \quad (51a)$$

$$\ll \lambda \mathsf{x},\mathsf{y},\mathsf{z}\,\big[\ll \mathsf{read\text{-}to}, \mathsf{reader}:\mathsf{x}, \quad (51b)$$

$$\mathsf{readed}:\mathsf{y},$$

$$\mathsf{addressee}:\mathsf{z}, \mathsf{Loc}:\mathsf{l}; 1 \gg \wedge$$

$$\ll \mathsf{book}, \mathsf{arg}:\mathsf{y}, \mathsf{Loc}:\mathsf{l}_1; 1 \gg \wedge \quad (51c)$$

$$\ll \mathsf{listen}, \mathsf{arg}:\mathsf{z}, \mathsf{Loc}:\mathsf{l}_2; 1 \gg \wedge \quad (51d)$$

$$\mathsf{l} \subseteq \mathsf{l}_1 \wedge \mathsf{l}_2 \subseteq_t \mathsf{l}\big], \quad (51e)$$

$$[\mathsf{x}]:\mathsf{a},\ [\mathsf{y}]:\mathsf{b},\ [\mathsf{z}]:\mathsf{c},\ \mathsf{Loc}:\mathsf{l};\ 1 \gg \big) \quad (51f)$$

The proposition term (51a)–(51f) denotes the proposition that in the situation s the individual denoted by a reads the object denoted by b, to the addressee denoted by c, in the location l. The addressee denoted by c listens in the location l_2 that is shorter than l. The object denoted by b is the book in the broader location l_1. The symbols *a*, *b*, *c* can be constants or variables.

3.2 Underspecified Recursion Terms

The sub-term (51b)–(51e) denotes an abstract relation, consisting of informational units, which have an underlying informational structure. We can extract the abstract "pattern" of that structure, by "parameterizing" the specific instances of the relations read-to, book, listen, and their specific arguments. The resulted abstract informational pattern is given by (52b)–(52f) and represents a wide class of relations, among which some are used by human language.

In the following examples, we subsequently give complex terms with different structures, which have sub-terms that have shared components. To save space and avoid repetitions, we introduce auxiliary notations, i.e., abbreviations, for the sub-terms, by using the sign $\equiv$. Note that we take the sign $\equiv$ as a meta-symbol, which is not in the vocabulary of $\mathrm{L}_{\mathrm{ap}}^{\mathrm{ST}}$, to designate syntactic equality, i.e., orthographical equivalence, between $\mathrm{L}_{\mathrm{ap}}^{\mathrm{ST}}$-expressions. E.g., the symbol *P* introduced in (52a)–(52f) is treated as orthographically equal to the entire term in (52b)–(52f), which is a single, complex term. Thus, in (53a)–(53g), the symbol *P* has to be replaced with the term (52b)–(52f), which is a sub-term of the entire recursion term (53a)–(53g).

$$P \equiv \quad (52a)$$

$$\big(\mathsf{s} \models \ll \lambda \mathsf{x},\mathsf{y},\mathsf{z}\,\Big[\quad (52b)$$

$$\ll \mathsf{r}_1, [\mathsf{x}'] : \mathsf{x}, [\mathsf{y}'] : \mathsf{y}, [\mathsf{z}'] : \mathsf{z}; 1 \gg \wedge \quad (52c)$$

$$\ll \mathsf{r}_2, [\mathsf{y}'] : \mathsf{y}; 1 \gg \wedge \quad (52d)$$

$$\ll \mathsf{r}_3, [\mathsf{z}'] : \mathsf{z}; 1 \gg \wedge \mathsf{i}_4 \wedge \mathsf{i}_5\Big], \quad (52e)$$

$$[\mathsf{x}]:\mathsf{a},\ [\mathsf{y}]:\mathsf{b},\ [\mathsf{z}]:\mathsf{c},\ \mathsf{Loc}:\mathsf{l};\ 1 \gg \big) \quad (52f)$$

The abstract informational pattern (52b)–(52f) can be specified, i.e., instantiated, with more specific information, by maintaining the informational pattern in it, by adding where-scope to it and assignment in the where-scope, (53a)–(53g). Note that the entire expression (53a)–(53g) is a single term, i.e., a recursion term with assignments, which, in this case, has no additional restrictions.

$$P \text{ where } \{ \tag{53a}$$
$$r_1 := \lambda x', y', z' \ll \text{read-to}, \text{reader} : x', \tag{53b}$$
$$\text{readed} : y', \text{addressee} : z', \tag{53c}$$
$$\text{Loc} : l; 1 \gg, \tag{53d}$$
$$r_2 := \lambda y' \ll \text{book}, \text{arg} : y', \text{Loc} : l_1; 1 \gg, \tag{53e}$$
$$r_3 := \lambda z' \ll \text{listen}, \text{arg} : z', \text{Loc} : l_2; 1 \gg, \tag{53f}$$
$$i_4 := l \subseteq l_1,\ i_5 := l_2 \subseteq_t l \} \tag{53g}$$

As with the symbol *P* above, we use the same auxiliary notation for the term designated by *T* in (54a)–(54e), and (55a)–(55h). Note that the term *T*, in (54a)–(54e), represents a more general, informational pattern than *P*, in (52b)–(52f), since the memory variables r_2, r_3 in (54a)–(54e) are given the possibility to depend on x, y, z, as for r_1.

$$T \equiv \big(s \models\!\ll \lambda x, y, z \Big[\tag{54a}$$
$$\ll r_1, [x'] : x, [y'] : y, [z'] : z; 1 \gg \wedge \tag{54b}$$
$$\ll r_2, [x'] : x, [y'] : y, [z'] : z; 1 \gg \wedge \tag{54c}$$
$$\ll r_3, [x'] : x, [y'] : y, [z'] : z; 1 \gg \wedge i_4 \wedge i_5 \Big], \tag{54d}$$
$$[x] : a,\ [y] : b,\ [z] : c,\ \text{Loc} : l;\ 1 \gg \big) \tag{54e}$$

The term in (55a)–(55h) has specifying information in its where-scope, with restrictions and assignments. The objects denoted by l, l_1, l_2 have to be of the type LOC, and the objects denoted by a and c of the type of persons in the situation s (according to recursion). The situation denoted by the free parameter variable s is the reading situation by the recursively embedded information in the infons in (55c)–(55e). Note that we treat the parameter variables as memory locations.

$$T \text{ where}$$
$$\{ (l : \text{LOC}), (l_1 : \text{LOC}), (l_2 : \text{LOC}), \tag{55a}$$
$$(a : T), (c : T) \} \tag{55b}$$
$$\{ r_1 := \lambda x', y', z' \ll \text{read-to}, \tag{55c}$$
$$\text{reader} : x', \text{readed} : y',$$
$$\text{addressee} : z', \text{Loc} : l; 1 \gg,$$
$$r_2 := \lambda x', y', z' \ll \text{book}, \tag{55d}$$
$$\text{arg} : y', \text{Loc} : l_1; 1 \gg,$$
$$r_3 := \lambda x', y', z' \ll \text{listen}, \tag{55e}$$
$$\text{arg} : z', \text{Loc} : l_2; 1 \gg, \tag{55f}$$
$$i_4 := l \subseteq l_1,\ i_5 := l_2 \subseteq_t l, \tag{55g}$$
$$T := \lambda u(s \models\!\ll \text{person}, u, l, 1 \gg) \} \tag{55h}$$

In particular, they can be memory variables for memorizing more complex information about corresponding objects.

4 CONCLUSIONS AND FUTURE WORK

The above examples show that different terms, such as infon terms, relation terms, and proposition terms carry different pieces of information, that have components that can be related by complex information structures. The components of the more complex terms carry structured information about typical relations between objects, in space-time locations that can be related. The components also carry information about what objects are appropriate for filling basic and complex argument structures. In realistic, practical systems, these components can be "related" by rich and fine-grained informational structures.

The examples provide arguments for the informational richness and fine-granularity of the formal language L_{ap}^{ST}, that is designed to provide syntax-semantic interface to mathematical structures of Situation Theory (Loukanova, 2001; Loukanova, 2002a; Loukanova, 2002b).

The paper introduces the definitions of the syntax of the formal language L_{ap}^{ST}. The definitions are supplemented with intuitions about its model-theory presented in (Loukanova, 2013c; Loukanova, 2014). In particular, we provide intuitions about its denotational semantics, and what the terms can denote. Section 3.2 provides arguments for information "patterns" with fine-granularity, which are the basis for algorithmic semantics, i.e., for semantics that provides structural information about how the denotations of the terms can be computed.

The formal introduction of the denotational and algorithmic semantics of L_{ap}^{ST} in Situation Theoretical models is not in the scope of this paper. That is more extensive work that is in our plans. Another closely related line of work is development of formal logical calculi and theory of L_{ap}^{ST}. This is an open area of research with potentials for applications in various areas, especially where fine-granular, relational, partial, and parametric information is essential. We foresee its applications in many sub-areas of neuroscience in general, and in particular in neuroscience of language. The target is applications to advanced technologies involving intelligent systems, including in the areas of Artificial Intelligence.

REFERENCES

Aczel, P. (1988). *Non-well-founded Sets*. Number 14 in CSLI Lecture Notes. CSLI Publications, Stanford, California.

Barwise, J. (1981). Scenes and other situations. *Journal of Philosophy*, 78:369–397.

Barwise, J. and Etchemendy, J. (1995). *The Liar: An Essay on Truth and Circularity*. Oxford University Press.

Barwise, J. and Moss, L. (1996). *Vicious Circles*. CSLI Publications, Stanford, California.

Barwise, J. and Perry, J. (1983). *Situations and Attitudes*. Cambridge, MA:MIT press. Republished as (Barwise and Perry, 1999).

Barwise, J. and Perry, J. (1999). *Situations and Attitudes*. The Hume Series. CSLI Publications, Stanford, California.

Copestake, A., Flickinger, D., Pollard, C., and Sag, I. (2005). Minimal recursion semantics: an introduction. *Research on Language and Computation*, 3:281–332.

Devlin, K. (2008). Situation theory and situation semantics. In Gabbay, D. and Woods, J., editors, *Handbook of the History of Logic*, volume 7, pages 601–664. Elsevier.

Ginzburg, J. and Sag, I. A. (2000). *Interrogative Investigations: The Form, Meaning, and Use of English Interrogatives*. CSLI Publications, Stanford, California.

Loukanova, R. (2001). Russellian and Strawsonian Definite Descriptions in Situation Semantics. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing*, volume 2004 of *Lecture Notes in Computer Science*, pages 69–79. Springer Berlin / Heidelberg.

Loukanova, R. (2002a). Generalized Quantification in Situation Semantics. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing*, volume 2276 of *Lecture Notes in Computer Science*, pages 46–57. Springer Berlin / Heidelberg.

Loukanova, R. (2002b). Quantification and Intensionality in Situation Semantics. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing*, volume 2276 of *Lecture Notes in Computer Science*, pages 32–45. Springer Berlin / Heidelberg.

Loukanova, R. (2010). Computational Syntax-Semantics Interface. In Bel-Enguix, G. and Jiménez-López, M. D., editors, *Language as a Complex System: Interdisciplinary Approaches*, pages 111–150. Cambridge Scholars Publishing.

Loukanova, R. (2011). Syntax-Semantics Interface for Lexical Inflection with the Language of Acyclic Recursion. In Bel-Enguix, G., Dahl, V., and Jiménez-López, M. D., editors, *Biology, Computation and Linguistics — New Interdisciplinary Paradigms*, volume 228 of *Frontiers in Artificial Intelligence and Applications*, pages 215–236. IOS Press, Amsterdam; Berlin; Tokyo; Washington, DC.

Loukanova, R. (2013a). Algorithmic Granularity with Constraints. In Imamura, K., Usui, S., Shirao, T., Kasamatsu, T., Schwabe, L., and Zhong, N., editors, *Brain and Health Informatics*, volume 8211 of *Lecture Notes in Computer Science*, pages 399–408. Springer International Publishing.

Loukanova, R. (2013b). Algorithmic Semantics for Processing Pronominal Verbal Phrases. In Larsen, H. L., Martin-Bautista, M. J., Vila, M. A., Andreasen, T., and Christiansen, H., editors, *Flexible Query Answering Systems*, volume 8132 of *Lecture Notes in Computer Science*, pages 164–175. Springer Berlin Heidelberg.

Loukanova, R. (2013c). Situated Agents in Linguistic Contexts. In Filipe, J. and Fred, A., editors, *Proceedings of the 5th International Conference on Agents and Artificial Intelligence*, volume 1, pages 494–503, Barcelona, Spain. SciTePress — Science and Technology Publications.

Loukanova, R. (2014). Situation Theory, Situated Information, and Situated Agents. *Transactions on Computational Collective Intelligence (TCCI) Journal*, TCCI XVII 2014, LNCS 8790.

Loukanova, R. (2015). Higher-order Theory of Recursion and Situation Theory — present and future potentials. Forthcoming.

Martin-Löf, P. (1984). *Intuitionistic Type Theory*. Bibliopolis, Napoli.

Moschovakis, Y. N. (2006). A logical calculus of meaning and synonymy. *Linguistics and Philosophy*, 29:27–89.

Rathjen, M. (2004). Predicativity, circularity, and anti-foundation. In Link, G., editor, *One hundred years of Russelll's paradox (De Gruyter Series in Logic and Its Applications)*, volume 6, pages 191–219. Walter de Gruyter, Berlin, New York.

Seligman, J. and Moss, L. S. (2011). Situation Theory. In van Benthem, J. and ter Meulen, A., editors, *Handbook of Logic and Language*, pages 253–329. Elsevier, Amsterdam.

Tin, E. and Akman, V. (1994). BABY-SIT: Towards a situation-theoretic computational environment. *Current Issues in Mathematical Linguistics, North-Holland Linguistic Series*, 56:299–308.

Tin, E. and Akman, V. (1996). Information-oriented computation with BABY-SIT. In Seligman, J. and Westerståhl, D., editors, *Logic, Language and Computation, Volume 1*, number 58 in CSLI Lecture Notes, pages 19–34. CSLI Publications, Stanford.

Tin, E., Akman, V., and Ersan, M. (1995). Towards Situation-oriented Programming Languages. *SIGPLAN Notices*, 30(1):27–36.

Univalent Foundations Program, T. (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study.

Van Lambalgen, M. and Hamm, F. (2004). *The Proper Treatment Of Events*. Explorations in Semantics. Wiley-Blackwell, Oxford.