INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Team LIS

Logical Information Systems

Rennes

THEME SYM

Table of contents

1. Team

Head of the team
Olivier Ridoux [Professor, Université Rennes 1, habilité(e)]

Administrative assistant
Lydie Mabil [TR Inria]

Université Rennes 1 personnel
Sébastien Ferré [Assistant Professor]
Annie Foret [Assistant Professor]

Insa personnel
Mireille Ducassé [Professor, habilité(e)]

PhD students
Benjamin Sigonneau [MENRT grant, since Oct. 2003]
Olivier Bedel [Région Bretagne grant, since Oct. 2005]
Peggy Cellier [MENRT grant, since Oct. 2005]

Associate members
Erwan Quesseveur [Assistant Professor, Université Rennes 2]
François Le Prince [President of ALKANTE and Associate Professor, Université Rennes 2]

2. Overall Objectives

2.1. Overview

The LIS team aims at developing *formal* methods for handling complex data sets in a *flexible* and *precise* way. “Flexible” means that the content determines the shape of the container. Very often, it is the opposite that is observed; e.g., the tree-like shape of a hierarchical file system enforces the tree-like shape of software packages. “Precise” means that any subset of the data set can be easily characterized. Again, it is the opposite that is often observed; e.g., in a hierarchical file system only sub-trees can be easily characterized. More and more information is available on the Web, and more and more information can be stored on a single machine. However, whereas the related low-level technology is developing, and performance is increasing, little is done for organizing the ever-growing amount of information. Therefore, the LIS team addresses the issues of organizing and querying information in general. The solutions are to be both formal and practical. Operational issues such as index technologies are important, but we are convinced that their scope is too limited to solve the crucial issues.

At a formal level, $queries$ and $answers$ are two key notions. It is nowadays standard to consider queries as logical formulas and answers as special models of queries. Computing the preferred model of a query in some context is conceptually easy, and it warrants flexibility. However, the opposite is not that easy in general; given a subset of the data, how can we compute a query of which it is a model? Given two different subsets of the data, how can we compute a query that explains the difference? Knowing this would warrant precision. The LIS team proved that *formal concept analysis* (FCA [38]) is a powerful framework for analyzing $\langle query, answer \rangle$ pairs. *Formal concepts* formalize the association between a query and its answers. Formal concepts are structured into a lattice which provides navigation links between concepts.

However, standard FCA cannot deal with queries considered as logical formulas (recall that this is the key for flexibility). Therefore a variant of FCA for logical description has been developed [6] altogether with the generic notion of *Logical information system* (LIS) that provided a reconstruction of all information system operations based on logical concept analysis. In particular, some data-mining operations are native in LIS [6], [3], [4].

The mottoes of the LIS research are:

1. *Never enforce* a priori *structure to information.* E.g., do not use hierarchical structures. Enforcing *a priori* structure causes the *tyranny of the dominant decomposition* [47]. For instance, the usual class-based organisation of source code makes highly visible the connections between methods of the same class, but masks the possible connections between methods in different classes.
   Instead, consider pieces of information as a bulk. Structure should emerge *a posteriori* from the contents or the point of view. As a consequence, updating the contents may change the structure: we accept it.
2. *Consider every possible rational classification, and permit changes at any time.* Here, rational means that what makes a piece of information belong or not to a class depends on the very piece of information, not on other pieces. The concept lattice induced by FCA is precisely a means to grasp all possible rational classifications.
3. *Rare events are as important as the frequent ones.* One cannot say *a priori* if a piece of information is interesting because it presents a frequent pattern, or because it presents a rare pattern.
   So, rare events must not be masked by statistical artefacts. Statistics is not forbidden, but it is only a complement of a symbolic logic approach.
4. *Queries should be possible answers.*
   In usual information systems (say relational databases or Web browsers) there is a strict dichotomy between queries (they are intensional expressions), and answers (they are strictly extensional expressions, i.e. sets of things). We contend that a good answer must be a mix of extensional and intensional answers. E.g. the good answer to “I would like to buy a book” is seldom the whole catalog of the bookshop; it is more relevant to answer such a query with other queries, like “Is this for a child” or “Do you prefer novels of documents”.
   Note that hierarchical file systems already do that. Queries (i.e., *filepaths*) yield answers that contain other queries (i.e., *sub-directories*). One of the LIS achievements is a formalization of this behaviour that does not rely on an *a priori* hierarchical structure.

Our research is intended to be *vertical* in the sense that all aspects of information systems are of interest: design, implementation, and applications.

On the implementation side, the LIS team develops systems that present the LIS abstraction either at the file system level [10], [9] or at the user level.

On the application side, the LIS team explores the application of LIS to *Geographical information systems* (GIS). The intuition here is that the traditional layered organization of information in GIS suffers a rigid structure of thematic layers. Moreover, GIS applications usually cope with highly heterogeneous information and large amount of data; this makes them an interesting challenge for LIS. The team also works on a data-mining interpretation of bug tracking. In this case, the intuition is that pieces of information relevant to software engineering, e.g. programs, specifications or tests, can be explored very systematically by a LIS. More generally, applications to software engineering are important for the team.

2.2. Key Issues

In its current state, LIS raises several questions that we wish to answer.

- The file system implementation of LIS can handle around 1 million elementary pieces of information. *How can it handle more? Can we reach 100 million in the next few years?*
- The LIS formalism is generic w.r.t. the logic used for describing pieces of information.
  What are the appropriate logics for the application fields that we have chosen? (GIS and error localization) Do we need a brand new logic for every application, or is there something that different applications can share?

- Genericity of LIS w.r.t. logic opens the door for creating ad hoc logics for describing pieces of information of an application. We already have proposed the framework of *logic functors* for helping a user build safely ad hoc logics. Logic functors are certified logic components that can be composed to form certified implementations of a logic.
  What are the useful logic functors? How can we be sure that a toolbox of logic functors is complete for a given purpose?
  Can the idea of certified composition be applied to another domain? Given a domain *foo*, *foo* functors would be certified *foo* components that can be assembled to form certified implementations of *foo* systems.
  Is it possible to certify other properties than meta-logical properties? E.g. is it possible to characterize complexity, or other non-functional properties like security?
- A family of non-commutative logics has developed over the years in the domain of computational linguistics, e.g. Lambek logic, pregroups. As for LIS, a great amount of creativity is expected for extending this family with ad hoc logics that would tackle fine-grained linguistics phenomena.
  Is it possible to build up an implementation of these logics using logic functors?
  Some LIS applications deal with objects that are sequential by nature (say, texts).
  Can these non-commutative logics primarily developed for computational linguistics help in LIS applications?
- Hierarchical file systems have a preferred metaphor which is the tree.
  What is the proper metaphor for LIS?
  The tree is also the graphical metaphor of hierarchical file systems.
  What is the graphical metaphor for LIS?
  Knowing this is crucial for the acceptance of LIS in end-user applications.
- Geographical information systems also suffer the *tyranny of the dominant decomposition*. Here, the dominant decomposition is in rigid thematic layers that inherit from plastic sheets of ancient map design. These layers are omnipresent in the design and interface of GIS applications.
  How can LIS abstract these layers, and still display layers when needed?
  Mining geographical information is difficult because of the layers and because it must cope with complex spatial relations.
  What is the proper modeling of these relations that will permit efficient LIS operations, including data-mining?
- Up to now, mining execution traces for bug tracking has used poor trace representations and ad hoc algorithms.
  How can the theoretical and practical framework of LIS help benefit from the wide range of information of program development environments?

3. Scientific Foundations

3.1. Logics for Information Systems

Keywords: *Syntax, interpretation, semantics, subsumption.*

Syntax Definition of the well-formed statements of a language. Statements are finite.

Interpretation Complete description of a world. Interpretations can be arbitrary mathematical constructs, and so can be infinite. Interpretations are models of statements, namely the worlds in which the statement is true. Statements are features of interpretations, namely the statements that are true in the world.

Semantics A binary relation between syntactic statements and interpretations.

Subsumption A relation which states that a property is more specific than another property.

Logic is the core of Logical Information Systems. However, this does not say everything because every particular usage of logic is also a point of view on logic. For instance, logic in Logic Programing is not the same as in Description Logic. This section describes the point of view on logic from information systems.

Logic is a wide domain that is concerned with formal representation and reasoning. The point of view on logic in logical information systems can be characterized by two things. Firstly, we are interested in the individual description of objects (e.g., files, pictures, program functions or methods), so that we need to represent concrete domains and data structures. This entails two levels of statements: (1) statements about objects, and (2) statements about the world (e.g., ontologies and *subsumption*). Subsumption helps to decide when an object is an answer to a query. Secondly, we need automated reasoning facilities as the subsumption must be decided between any object and a query in information retrieval. This forces us to only consider decidable logics, unless consistency or completeness are weakened.

3.1.1. Properties of a Logic

A characteristic of logic is the ability to derive new statements from known statements. Such a derivation is valid w.r.t. semantics only if every model of the known statements is also a model of the new statements. This ability opens the room for *reasoning*, i.e. the production of valid statements by working at the syntactical level only. Reasoning is formalized by *inference systems* (e.g., axioms and rules). An inference system is *consistent* if it produces only valid statements; it is *complete* if it produces all valid statements. Reasoning is *decidable* if a consistent and complete inference system can be made an algorithm.

3.1.2. Examples of Logics for Information Systems

Proposition logic is a possible logic for an information system, but it needs a lot of encoding for handling structured information. Instead, non-standard logics have been defined for some structured domains.

A large family of logics that comes into our scope is the family of Description Logics (DL) [32], [33], which have been widely studied, implemented, and applied in knowledge and information management. Moreover, their semantic structure is especially well-suited to be used in a LIS. The semantics of proposition logic is often exposed in terms of truth values and truth tables. In the opposite, the semantics of description logic is defined in terms of sets of objects that are close to answers to a query. DL are, therefore, of a special interest for the LIS team.

Another family of interest is *categorial grammars*. Many substructural logics come into this scope, among which non-commutative linear logic or Lambek Calculus [43] that handle various concatenation principles (or ordered conjunction) in categorial grammars where logic is used both for attaching formulas to objects and for parsing seen as deduction.

At an empirical level, the categorial approach comes very close to the LIS approach. Categorial grammars correspond to LIS contents, because they both attach formulas to objects, and sentence types correspond to queries. The difference is that the answer to a LIS query is an unordered set, whereas a sentence generated

by a categorial grammar is an ordered sequence. We expect a cross-fertilization of both theories in the future, especially in the LIS applications where the objects are naturally ordered.

3.2. Concept Analysis

Keywords: *Objects, concept, context, descriptors, extension, instance, intension, property.*

Objects A set of distinguished individuals.

Descriptors A set of distinguished properties.

Context A set of objects associated with descriptors.

Instance An object is an *instance* of a descriptor if it is associated with it in a given context.

Property A descriptor is a *property* of an object if it is associated with it in a given context.

Extension The *extension* of a collection of descriptors is the set of their common instances. Extent is a synonym.

Intension The *intension* of a collection of objects is the set of their common properties. Intent is a synonym.

Concept Given a context, and extensions and intensions taken from it, a *concept* is a pair (E, I) of an extension E and an intention I that are mutually complete; i.e., I is the intention of the extension, and E is the extension of the intention.

3.2.1. Formal Concept Analysis

Formal Concept Analysis (FCA) is part of the mathematical branch of applied lattice theory [31], [35]. It can be seen as a reformulation by Wille of Galois lattices [28] that emphasizes lattices as conceptual hierarchies [48]. The mathematical foundations of FCA have been extensively studied by Ganter and Wille [38].

FCA mainly aims at the automatic construction of *concepts* and their classification according to a generalization ordering, given a flat representation of data. The adjective *formal* means that concepts are given a mathematical definition, which reflects the usual philosophical meaning of a “concept”. The basic notions of FCA are those of *formal context*, and *formal concept*.

A *formal context* is a binary relation between a set of objects, and a set of attributes. Through this relation attributes can be seen as properties of objects, and reciprocally, objects can be seen as instances of attributes. This is a very general settings that applies to various domains such as data analysis, information retrieval, data-mining or machine learning. In all these domains, the objects of interest are described by sets of attributes, and the objective is to relate in some way sets of objects and sets of attributes. In information retrieval a set of attributes is a query, whose answers is a set of objects. In machine learning a set of objects is a set of positive examples, whose characterization is a set of attributes.

A *formal concept* is the association of a set of objects, the *extent*, and a set of attributes, the *intent*. This comes close to the classical definition of concept in philosophy, but in FCA the relationship between extent and intent is formally defined. The extent must be the set of instances shared by all attributes of the intent; and the intent must be the set of properties shared by all objects in the extent.

The fundamental theorem of FCA says that the set of all concepts forms a complete lattice when they are ordered according to the set inclusion on extents (or intents). This is called the *concept lattice*, and it can be computed automatically from the formal context. The concept lattice is the structure that is implicit in any formal context. It contains all the information contained in the formal context; the latter can be rebuilt from the former. In data analysis, the concept lattice permits a flexible classification of data (where a concept is a class), because concepts are not organized as a strict hierarchy. In information retrieval and data-mining it is used as a search space for answers.

3.2.2. Logical Concept Analysis

In Formal Concept Analysis (FCA) object properties are restricted to Boolean attributes. In many applications there is a need for richer properties, where properties are not independent. For instance, if a book has been published in 2000, it can be given the property `year = 2000`, and has then the implicit properties `year in 1990..2000` and `year in 2000..2010`. This means that properties are statements about objects that can be subject to reasoning, exactly like logical statements. Other examples of useful properties are strings and string patterns, spatial descriptions for locating objects, or patterns over the programming type of functions and methods.

FCA has been extended by other authors to handle multi-valued contexts [38], but this extension takes the form of a preprocessing stage that results in a standard formal context, and forgets all logical relations between properties. Moreover it is limited in practice to valued attributes with finite domains of attributes. In 2000 we proposed a logical generalization of FCA, named Logical Concept Analysis (LCA) [6], [3], that is the abstraction of FCA w.r.t. object descriptions and concept intents. This makes LCA an abstract component, and makes FCA the composition of LCA with a logic component. LCA makes the theory of concept analysis easily reusable in various applications.

For good composability of LCA and logics, they must agree on the specification of logics. What LCA needs from a logic is:

- a language of formulas (or statements), L, for the representation of object descriptions and concept intents,
- a procedure, $\sqsubseteq$, for deciding the subsumption between 2 formulas; $\sqsubseteq$ means “is subsumed by”, “is more specific than”, “entails”,
- a procedure, $\sqcup$, for computing the least common subsumer of 2 formulas; it is a kind of logical disjunction,
- a formula, $\bot$, that is the most specific according to subsumption (logical contradiction).

This specification provides everything required to extend fundamental results of FCA to LCA (formal context, extent, intent, concept, complete lattice of concepts). For information retrieval and the expression of queries, it is useful to add, to this specification, operations such as logical conjunction, and logical tautology (the most general formula).

Any formal context defines a logic whose subsumption relation is isomorphic to the concept lattice that is derived from the formal context. An interesting result is that the *contextualized logic* (the logic defined by the logical context) is a refinement or extension of the logic used by LCA. Everything true in the logic is also true in the contextualized logic (because it is *eternal truth*); and everything true only in the contextualized logic says something that is true in the context, but not in general (because it is *instant truth*). Thus, contextualized logic forms the basis for data-mining and machine learning tasks, whose aim is to discover outstanding regularities in a given context [6], [4].

3.3. Logical Querying, Navigation, and Data-mining

Keywords: *Querying*, *data-mining*, *navigation*.

Querying The process that takes a query (e.g., a logical formula), and returns the collection of objects that satisfy the query (e.g., the extent of the query).

Navigation The process of moving from place to place, where each place indicates objects they contain (i.e. *local objects*) and other places where it is possible to move (i.e. *neighbouring places*).

Data-mining The process of extracting outstanding regularities from data (e.g., a context) hoping to discover new and useful knowledge.

In most information systems, querying and navigation are two disconnected means for information retrieval. With querying, users formulate queries which belong to more or less complex querying languages, from simple words as in Google to highly structured languages like SQL. The system returns a set of answers to the query. This permits expressive search criteria over large amounts of data, but lacks interactivity because the dialogue is only one-way. If the answers are not satisfying, users have to imagine new queries and formulate them, which requires *a priori* knowledge of both querying language and data. With navigation, users move from place to place following links. The most common systems are folder hierarchies (e.g., file systems, bookmarks, emails), and hypertext. As opposed to querying, navigation provides interactivity by making suggestions at each step, but offers limited expressivity because navigation structures are rigid. In a hierarchy, selection criteria are presented in a fixed order. For instance, if pictures are classified first by date, then by type, one cannot easily find all landscape pictures.

The need for combining querying and navigation has already been recognized. Most proposals, however, are unsatisfying. Indeed, either querying and navigation cannot be mixed freely in a same search, or consistency of querying is not maintained. An example of the former is SFS [39], once a querying step is done, there is no more navigation. An example of the latter is HAC [41], some query answers may not satisfy the query. A proposal based on FCA has not these drawbacks [40], and we have generalized it to work within LCA, which allows us to use logical formulas for object description and queries [3], [6]. Logic brings expressivity in querying, and concept analysis brings the concept lattice as a navigation structure (i.e., navigation places are formal concepts). The advantages of this navigation structure is that (1) it is automatically derived from data, the logical context (see motto 1), (2) it is complete as navigation alone makes it possible to reach any object (see motto 3), and (3) it is flexible because selection criteria can be chosen in any order, thus allowing user to express their preferences (see motto 2). Querying and navigation can be freely mixed (see motto 4) in a same search because every logical formula points to a formal concept, and every formal concept is labelled by a logical formula. Put concretely, this means that a user can at each step of his search: either modify by hand the current query and reach a new place, or follow a suggested link that will modify the current query and reach a new place.

The critical operation is the computation of navigation links, which correspond to edges in the concept lattice. Indeed, the worst-case time complexity for computing the concept lattice is exponential in the number of objects, which makes it intractable in most interesting cases. We demonstrated both in theory and practice that this computation is not necessary. A key feature of LIS is that its semantics is expressed in terms of LCA, though it is not required to actually build the concept lattice. This is opposed to most (all?) previous proposals for using LCA in information retrieval.

The concept lattice upon which our navigation is based is also a rich structure for data-mining and machine learning [42]. Here again, we have combined existing techniques with logic [6], [4], and applied them to the automatic classification of emails [36], and the prediction of the function of proteins from their sequence [4].

3.4. Genericity and Components

Keywords: *Abstraction*, *component*, *composability*, *reusability*.

Abstraction a mechanism and practice to reduce and factor out details so that one can focus on few concepts at a time.

Reusability the likelihood a segment of structured code can be used again to add new functionalities with slight or no modification. Reusable code reduces implementation time, it increases the likelihood that prior testing and use has eliminated bugs and it localizes code modifications when a change in implementation is required.

Composability a system design principle that deals with the inter-relationships of components. A highly composable system provides recombinant components that can be selected and assembled in various combinations to satisfy specific user requirements.

Component an object written to a specification. It does not matter what the specification is as long as the object adheres to the specification. It is only by adhering to the specification that the object becomes a component and gains features like reusability, composability, and abstraction.

The application scope of Logical Information Systems is very large (see Section 4.), and we do not expect that one design (e.g., one logic) will fit all possible applications. That is why we emphasize genericity, and we use the plural in "logical information systems". The need for genericity is not limited to theoretical results and design, but extends to the concrete implementation of LIS.

Genericity requires programming facilities for *abstraction*, *composability*, and *reusability* of *software components*.

In LIS, abstraction is of the upper importance in the design of logical concept analysis; LCA is an abstraction of FCA. It is also at the heart of the *logic functor* framework and of its implementation; a logic functor is an abstraction of a logic (see Section 3.5). Reusability and composability are the expected outcomes of this framework. It is expected to make things easier to the designer of a LIS application. Composability is also at the heart of the very notion of formal context, and thus at the heart of concept analysis. Indeed, the flat structure of formal concepts makes it trivial to extend a context or merge two contexts, and the burden of giving a structure to the context is left to the construction of the concept lattice.

A generic implementation of LIS can be seen as a central component that is parameterized by several application-dependent components: at least a logic, and a transducer for importing data. These parameter components can be linked at compilation time (plugins). The central component as well as parameter components can themselves be the result of the composition of smaller components.

3.5. Logic Functors

Keywords: *Logics*, *composability*, *genericity*.

The genericity of LCA and LIS w.r.t. logic implies that for every new application a logic has to be found for describing objects in a logic context. Either a suitable logic is already known, or it must be created. Creating a logic requires designing a syntax, a semantics, algorithms for subsumption and other procedures, and proving that these algorithms are correct w.r.t. semantics. This definitely requires logic expertise and programming skills, especially for the subsumption procedure that is a theorem prover for which consistency and completeness must be proven. However, application developers and logic experts are likely to be different persons in most cases. Moreover, creating new logics from scratch for each application is unsatisfying w.r.t. reusability as these logics certainly share common parts. For instance, many applications need propositional reasoning, only changing the notion of what is a propositional variable.

We introduced high-level logic components, named *logic functors* [5], [3], in order to make the creation of a new logic the mere composition of abstract and reusable components. All logics share a common specification that contains all useful procedures (e.g., subsumption); logic functors are functions from logics to logics, implemented as parameterized modules. Some logic functors take no parameter, and provide stand-alone but reusable logics: this is the case of concrete domains such as integers or strings. Other logic functors take one or several logics as parameters. For instance the functor `Prop(X)` is propositional logic abstracted over its atoms. This makes it possible to replace atoms in propositional logic by the formulas of another logic (e.g., valued attributes, terms from a taxonomy).

Logics are built by applying logic functors to sub-logics, which can themselves be defined as a composition of logic functors. For instance, the propositional logic where atoms are replaced by integer-valued attributes (and allowing for integer intervals) can be defined by the expression

```
L = Prop(Set(Prod(Atom,Interval(Int)))).
```

This results in a concrete software component `L` that is fully equipped with implementations of the logic specification procedures. This component can then be composed itself with LCA or a LIS system.

3.6. Categorial Grammars

Keywords: *Categorial grammar, identification in the limit.*

Categorial grammars are used for natural language modeling and processing; they mainly handle syntactic aspects, but Lambek variants also have a close link with semantics and Lambda-calculus. Formally, a *categorial grammar* is a structure $G = (\Sigma, I, S)$ where: Σ is a finite alphabet (the words in the sentences); I is a function that maps a finite set of types to each element of Σ (the possible categories of each word, a lexicon); S is the *main type* associated to correct sentences. A *k-valued categorial grammar* is a categorial grammar where, for every word $a \in \Sigma$, $I(a)$ has at most k elements. A *rigid categorial grammar* is a 1-valued categorial grammar.

Each variant of categorial grammar formalism is also determined by a derivability relation on types $\vdash$ (which can be seen as a subcase of *linear logic* deduction in the case of Lambek grammars). Given a categorial grammar $G = \langle \Sigma, I, S \rangle$, a sentence w on the alphabet Σ belongs to the language of G whenever the words in w can be assigned by I a sequence of types that derive (according to $\vdash$) the distinguished type S.

A simplified example is $G_1 = (\Sigma_1, I_1, S)$ with $\Sigma_1 = \{John, Mary, likes\}$ $I_1 = \{John \mapsto \{N\}$, $Mary \mapsto \{N\}$, $likes \mapsto \{N \backslash (S/N)\}\}$ the sentence "John likes Mary" belongs to the language of G_1 because $N, N \backslash (S/N), N \vdash S$ due to successive applications of the two elimination rules : $X, X \backslash Y \vdash Y$ and $Y, Y/X \vdash Y$. Type constructors $/$ and $\backslash$ can be seen as oriented logic implications, the elimination rules are analogues of the "Modus Ponens" logic rule. An interesting issue is how the underlying rules or logics may compose (this is the design of logic functors) to deal with more fine-grained linguistic phenomenon.

Since they are lexicalized, such grammar formalisms seem well-adapted to automatic acquisition or completion perspectives. Such studies are performed in particular in Gold's paradigm.

Identification in the limit in the model of Gold consists in defining an algorithm on a finite set of (possibly structured) sentences that converges to obtain a grammar in the class that generates the examples. Let $\mathcal{G}$ be a class of grammars that we wish to learn from positive examples; let $\mathcal{L}(G)$ denote the language associated with a grammar G; a *learning algorithm* is a function φ from finite sets of (structured) strings to $\mathcal{G}$, such that for any $G \in \mathcal{G}$ and $\langle e_i \rangle_{i \in \mathbb{N}}$ any enumeration of $\mathcal{L}(G)$, there exists a grammar $G' \in \mathcal{G}$ such that $\mathcal{L}(G') = \mathcal{L}(G)$ and $n_0 \in \mathbb{N}$ such that $\forall n > n_0$ $\varphi(\{e_0, ..., e_n\}) = G'$.

4. Application Domains

4.1. Geographical Information Systems

Participants: Olivier Bedel, Olivier Ridoux, Sébastien Ferré, Erwan Quesseveur, François Le Prince.

Geographical Information Systems (GIS) is an important, fast developing domain of Information technology, and it is almost absent from INRIA projects. It is especially important for local communities (e.g. region and city councils).

Geographical information systems [44] handle information that are localized in space (*geolocalized*). GIS form a fast-developing area which incorporates various technologies such as web, databases, or imaging. One characteristic of GIS is their organization as *layers*. This is inherited from the plastic sheets that where used until recently for drawing maps. A layer represents the road system, another the fluvial system, another the relief, etc. This is another instance of the tyranny of the dominant decomposition, and is not satisfactory: to which layer belong bridges, into which layer can we represent a multimodal network? Moreover, mining GIS is known to be difficult for the same reason; the layer structure makes inter layer relationships difficult to discover.

The first advantage of applying LIS to GIS is to allow cross-layer navigation. Another advantage is to permit a logical handling of scales. In current GIS systems, scales are treated as different layers, and it is difficult to keep the consistency between all layers that describe the same object. Another advantage that we have observed in a preliminary work is that LIS helps cleaning a data-base. This was not expected, and opens an interesting research area.

4.2. Mining Software Repositories

Participants: Peggy Cellier, Mireille Ducassé, Olivier Ridoux, Benjamin Sigonneau.

There exist numerous repositories related to the development of software: for example source versions generated by control systems, archived communications between project personnel, defect tracking systems, component libraries and execution traces. They are used to help manage the progress of software projects. Software practitioners and researchers are beginning to recognize the potential benefit of mining this information to support the maintenance of software systems, improve software design/reuse, and empirically validate novel ideas and techniques.

Logical information systems seem particularly adapted to mine these repositories. Indeed, the repositories contain heterogeneous and incomplete information. Their size is too large to be directly handled by human beings and it is still manageable by the current implementations of LIS. The LIS team currently focuses on component retrieval and execution traces cross-checking.

5. Software

5.1. LISFS

Participants: Yoann Padioleau, Olivier Ridoux [contact point].

The main objective of a LIS is not to go *faster*; it is to go *easier*. This must be evaluated by experimenting the use of LIS in various contexts. However, the price for an easier usage must not be too high compared to more classical storage means such as a file system. At the same time we want to promote a *file system level implementation of LIS* so that every application that uses the file system interface could use a LIS.

LISFS is a file system that implements LIS under the Linux file system interface [10], [9][22]. It uses the whole usual file system technology (e.g. caching, journaling) to offer reactivity, safety, robustness, etc. At the same time, it is not intimately attached to Linux technology because it is programmed at the user level, and it uses the FuseFS bridge to redirect file system calls to the user level.

LISFS manages a set of objects described with attributes that may be valued. The attributes of an object form a logic conjunction, disjunction and negation can be expressed in queries. More sophisticated logical descriptions can be embedded in the attributed values. For instance, `title:contains "logic"` is an attribute whose value is `contains "logic"` and refers to a logic of string pattern matching. Objects can be created and deleted (e.g. shell commands `touch` f and `rm` f); their attributes can be changed anytime (e.g. shell command `mv` $f1$ $f2$; and new attributes can be created anytime (e.g. shell command `mkdir` a).

LISFS response time grows with the number of objects, the number of attributes, and the complexity of their values. We have proved, under hypotheses which are met in practice, that LISFS response time to queries is linear with the number of objects. However, this is not enough in practice, and the ideal complexity is amortized constant time. This is what we try to achieve through the use of file system and database technologies like caches, indexes, and journals. In its current state LISFS can manage up to 1 000 000 objects$\times$attributes with affordable response time for queries. However, the response time for updates is not yet as good as we wish, and this is a track for improvement in the future. Similarly, 100 000 objects is good enough for a personal computer, but is not enough for some professional usage; this is also something we wish to improve.

An important service of LISFS is to permit navigation, querying and updates inside files. This is the part-of-file service, PofFS [46][9]. The idea is that a file is considered as a composition of subparts; the subparts are to the file as files are to a mount point. This is a way to overcome artificial constraints that are often imposed by applications. For instance, it is often the case that methods of the same class must be textual neighbours in a source file. Sometimes what is desired is to see together all methods with the same role, say print. PofFS permits that. In fact, PofFS is just the right thing to do in many applications where the goal is not to find one answer but to display together all answers. This is the case of GIS applications, for instance.

5.2. Camelis

Participant: Sébastien Ferré.

Camelis is a stand-alone application that allows to store, retrieve and update objects through a graphical interface. Its main purpose is to experiment with the LIS paradigm. In particular, it has been very useful for refining the query-answer principle in special circumstances (e.g. when there are many answers, or when there are few answers). It is currently used as a personal storage device for handling photos, music, bibliographical references, etc, up to thousands of objects. It implements as closely as possible the LIS paradigm. It is generic w.r.t. logics, and is compatible with our library of logic functors, LogFun (see Section 5.3). It is available on Linux and Windows, and comes with a user manual.

5.3. LogFun

Participant: Sébastien Ferré.

The formal definition of a LIS is generic with respect to the logic used for object descriptions and for queries. The counterpart is that it is up to the user to design and implement a logic solver to plug in a LIS. This is too demanding on the average user, and we have developed a framework of *logic functors* that permits to build *certified* logic solvers (see Section 3.5).

LogFun is a library of *logic functors* and a *logic composer*. A user defines a logic using the logic functors, and produces a certified software implementation of the logic (i.e., parser, printer, prover) by applying the logic composer to the definition. For instance, using a functor *Interval* for reasoning on intervals (e.g. $x \in [2,5] \Longrightarrow x \in [0,10]$), and a functor *Prop* for propositional reasoning (e.g. $a \wedge b \Longrightarrow a$), a user can define logic *Prop(Interval)*. In this logic, a theorem like $x \in [2,5] \vee x \in [7,9] \Longrightarrow x \in [0,10]$ can be proven. Note that $[2,5] \cup [7,9]$ *is not* an interval, so that *Prop(Interval)* is an actual extension over *Interval*.

What the logic composer does when building logic *Prop(Interval)* is to compose the solver of *Interval* and the generic solver of *Prop*, and build a solver for *Prop(Interval)*. It also type-checks *Prop(Interval)* to produce its certificate using the certificates of *Interval* and *Prop*. In this example, the certificate says that *Prop(Interval)* is complete: everything that could be deduced from the meaning of *Prop(Interval)* can be proved by its solver. In other circumstances, the certificate indicates that the logic defined by the user is incomplete, w.r.t. the semantics and solvers that come with the functors. In this case, the certificate also indicates what hypotheses are missing for completeness; this may help the user to define a more complete variant of its logic.

Logic functors offer basic bricks and a building rule to safely design new logics. For instance, in a recent application of LIS to geographical information system, a basic reasoning capability on locations was needed. The designer of the application, not a LIS or LogFun author, could build a relevant ad hoc logic safely and rapidly.

6. New Results

6.1. Logic Functors

Keywords: *Logic functors, completeness, consistency, epistemic knowledge, strings.*

Participants: Sébastien Ferré, Olivier Ridoux.

An important issue of logic functors is the validity of composed logics, e.g. the consistency and completeness of the subsumption decision procedure w.r.t. its syntax and semantics. Not all compositions are valid, and their verification *a priori* requires tedious proofs. We have managed to encapsulate these proofs locally to each logic functor for a number of logic properties (e.g., consistency and completeness), and to automate their composition in parallel to the composition of procedure algorithms, so that every composed logic is accompanied by an automatic diagnostic of its properties. This is an important result because it puts all the hard work in the design of logic functors (made by logic experts), and not in their composition (made by application developers) [26].

An aspect of logic that is relevant to logical information system is *epistemic reasoning*. We chose to use the logic AIK ("All I Know") [45], as it is the only monotonous one for epistemic reasoning. AIK allows for the application of the "closed world assumption" (everything that can not be proven true is considered as false), and can be adapted to handle the distinctions between negation and opposition (e.g., "hot" is the opposite of cold, "not cold" is the negation), and between certainty and possibility [21]. We have abstracted the epistemic reasoning of AIK as a logic functor so that it can be applied to about any logic, instead of propositional logic only [26].

We have also designed a functor for a logic of strings and sub-strings, where the subsumption relation corresponds to the relation "contains" on strings. It allows for the automatic and efficient extraction of maximal repeated sub-strings, which can be use as a relevant vocabulary for navigation in LIS [20].

6.2. GEOLIS: a LIS for Geographical Data

Keywords: *geographical information system*.

Participants: Olivier Bedel, Olivier Ridoux, Sébastien Ferré, Erwan Quesseveur.

We study how a LIS could serve as the storage of a GIS tool. A recent trend in the design of GIS tools is to separate the interface from the database. This yields a natural separation of concerns in which LIS will be used as the database, whereas standard GIS servers can be used as interfaces. Still, these interfaces must be slightly extended so that they can benefit from all LIS navigation facilities. We have achieved first experiments with actual GIS data collected in parts by our colleagues at Université de Rennes 2 on Sahelian rodents. The data result from over 50 years of observations. They are therefore very heterogeneous, and it was a challenge in itself to cope with them.

The first merit of LIS in this experiment is that it played the role of a data integrator:

- abnormal data could easily be spotted because they created spurious concepts,
- a taxonomy could be created to integrate attributes of similar meanings but different presentations (unit, reference, etc).

The second merit was that it permitted to integrate usual GIS navigation and LIS navigation. This experiment uses in a fundamental way the PofFS service of LISFS to navigate inside a file that contains the data. Note that the data format and the map of the interface are completely standard. Specific parts are on the one hand the transducer for giving a logical description to geographical features, and on the other hand a navigation tree in the interface for displaying the answers of LISFS and support logical navigation [16], [17].

6.3. Software Component Retrieval

Keywords: *reusability*, *software components*, *type isomorphism*.

Participants: Benjamin Sigonneau, Olivier Ridoux.

Software reuse is a promising practice for controlling the cost and quality of software production. However, the first step to reusing a component is to locate it. The criteria for choosing a component, however, are numerous. For instance, commercial-off-the-shelf (COTS) are described by many properties: e.g., cost, compliance to standards, (non-)functional properties. There is no reason to prioritize these properties. Therefore, we have started studying how various component properties could be used at the same time to search a component repository. Some properties are completely informal, like words used in component comments. Other are more formal, and even related to the program semantics, like types. The most prominent piece of work on type-based search is the theory of type isomorphisms [34]. Thus, we use types as a logic for LIS. The neat result is that a user may query for a component by specifying a type (e.g. the expected result type), and LIS will answer with other type characteristics (e.g. an accepted input type). This solves a problem that the theory of type isomorphism does not solve well; how to answer an approximate answer. Moreover, LIS accepts that

type criteria are mixed with other criteria. So, it is readily possible that a question based on a type criterion is answered by questions based on other criteria (e.g. visibility modifiers) [24], [15].

6.4. Mining software repositories for bug localization

Participants: Peggy Cellier, Mireille Ducassé, Sébastien Ferré, Olivier Ridoux.

When a failed execution indicates that a program contains at least an error, one has to locate and fix the error(s). Once errors have been roughly localized, competent programmers usually have no problem to finish the localization and fix the program. Hence, the problem that we are addressing is the coarse-grain localization of bugs, when programmers are facing large programs with long executions. The size of the data to investigate is already a problem, a further difficulty is the multiple sources of heterogeneous data. For example, the programmer can investigate the history of the program, in order to know who has made changes and when. He can use the structure of the source code in order to understand the static dependencies. He can also trace executions in order to have information about the behaviors of the program. Tracing can be made at many levels of abstraction. It can therefore generate information of many different types.

Analyzing huge amounts of heterogeneous data is almost impossible with semantics-based tools. We conjecture that LIS can significantly contribute to automatically crosscheck information of different types. LIS, however, lacked statistical reasoning capabilities. We have defined an extension of LIS which can find association rules [6]. The state of the art techniques consider attributes of a context as a non-ordered set. In many contexts, however, taxonomies are present. In our case, for example, parse trees and call graphs contain interesting structured information. We have, therefore, extended the framework so that it can benefit from contexts with taxonomies and remove redundancies from the resulting rules [19], [18]. Using the new extension, we are currently setting up experiments to draw correlations between program properties and the fact that executions fail.

6.5. Log Analysis

Participant: Mireille Ducassé.

A log analysis activity existed in the Lande team, prior to the creation of the LIS team. It addresses the analysis of logs generated by softwares, mainly for security purposes.

In the framework of a CRE contract with France Telecom R&D and the Dream project (cf. Section 7.1), we are working on two types of alarm logs: logs from a Virtual Private Network server that generates many alarms due to normal and abnormal connections and alarm logs generated by France-Telecom routers when they detect suspicious flows.

In VPN (Virtual Private Network) abnormal and normal connections are mixed in a unique log. Network experts cannot read the whole log. Thus, we proposed a method to abstract the log into transactions (sets of alarms). Alarm attributes were extracted by matching patterns from a set of attribute signatures. Next, the alarms are built by correlating attributes values. For an expert, reading the extracted transactions is then a lot easier than reading the whole log [25].

A second aspect of our work is devoted to the extraction of knowledge from a sequence of DDoS records [27]. Distributed Denial of Service (DDoS) attacks aim at overwhelming a target server with a huge volume of useless traffic from distributed and coordinated attack sources. A DDoS record indicates a suspicious Internet flow from sources to targets. France-Telecom uses DDoS records to detect DDoS attacks but the current method is not accurate enough. Thus, we developed a method for visualizing DDoS records in order to extract knowledge. From the visualization, several patterns were extracted that will serve as a basis for building an Internet flow model which to help detecting DDoS attacks.

A third aspect is related to the improvement of diagnosis in intrusion detection. A thesis has been defended at the Insa of Rennes about the study and application of a combination of anomaly and misuse detection methods [11]. Using the two intrusion detection methods allows to take advantage of their qualification skills. We use a serial combination of anomaly detection followed by misuse detection. Such a combination provides

a diagnostic where all events are qualified, lowering the amount of false positives and allowing unknown attacks to be discovered. The formalism of intrusion detection diagnostic presented justifies the use of such a combination and allows to reason about other possible combination. The anomaly model construction is based on known and well-tried statistical methods. Selection of events is based on profiles that are built and presented in the thesis. Misuse detection uses signatures combination, their severity level and the event context to diagnose an attack. The thesis has been co-supervised by Mireille Ducassé, Ludovic Mé from Supelec Rennes and Hervé Debar from France Telecom R&D Caen.

6.6. Categorial Grammar Acquisition and Structured Data

Participants: Annie Foret, Denis Béchet [LINA, Nantes].

We describe results on learning categorial grammars from positive examples in the model of Gold ; in [29], extended in [13], we study possible structures for data, close to parsing derivations, and their impact for learning.

We focus on Lambek categorial grammars and some related formalisms used for natural language modelling and processing, with a special interest in syntax and automatic grammar acquisition issues. An important application would be to implement categorial grammar acquisition algorithms and incorporate them into usual natural language tools such as taggers and parsers so as to enhance their overall coverage, reliability, modularity or efficiency.

Functor-argument structures (written FA) are usual syntactical decompositions of sentences in sub-components distinguishing the functional parts from the argument parts defined in the case of classical categorial grammars also known as AB-grammars. In the case of non-associative type-logical grammars, we propose [13] a similar notion that we call *generalized functor-argument structures* and we show that these structures capture the essence of non-associative Lambek calculus (NL) without product.

We show that (i) rigid and k-valued non-associative Lambek (NL without product) grammars are learnable (in the sense of Gold's model) from generalized functor-argument structured sentences.

We also define subclasses of k-valued grammars in terms of arity. We first show that (ii) for each k and each bound on arity the class of FA-arity bounded k-valued NL languages of FA structures is finite and (iii) that FA-arity bounded k-valued NL grammars are learnable both from strings and from FA structures as a corollary. Result (i) is obtained from (ii); this learnability result (i) is interesting and surprising when compared to other results: in fact we also show that (iv) this class has infinite elasticity. Moreover, these classes are very close to classes like rigid associative Lambek grammars learned from natural deduction structured sentences (that are different and much richer than FA or generalized FA) or to k-valued non-associative Lambek grammars unlearnable from strings [37][2] or even from bracketed strings [30]. Thus, the class of k-valued non-associative Lambek grammars learned from generalized functor-argument sentences is at the frontier between learnable and unlearnable classes of languages.

The availability of FA structures, is not yet guaranteed in practice. Some experiments have been carried out for English in the Categorial Combinatory Grammar formalism by Hockenmaier and Steedman. For French, experiments of categorial grammar acquisition, have been performed by a Master student on the French TreeBank of Paris 7 (A. Abeille) ; this work consisted in two phases : (i) an algorithm translates the French Treebank (a fragment , in XML format) developed at Paris 7 into annotated functor-argument structures, binary trees close to categorial grammar derivations ; (ii) an adaptation of Buszkowski's RG-learning algorithm has been proposed and implemented, it runs on the annotated functor-argument data obtained from the treebank fragment, to automatically construct the grammar lexicon (in the AB grammar formalism). These preliminary results, co-supervised with Denis Béchet, have been presented at a National Conference [23].

An article [12], in collaboration with past members of ARC Gracq, also reviews and discusses various aspects of categorial grammar acquisition, including theoretical limitation results and algorithmic aspects, such as those mentioned above.

6.7. Mining Protein Sequences for Function Prediction

Participants: Sébastien Ferré, Ross D. King [University of Wales, Aberystwyth].

We have designed a novel algorithm for the discovery of biological sequence motifs. Our motivation is the prediction of gene function. We seek to discover motifs, and combinations of motifs, in the secondary structure of proteins, for application to the understanding and prediction of functional classes. The motifs found by our algorithm allow both flexible length structural elements and flexible length gaps, and can be of arbitrary length. The algorithm is based on neither top-down nor bottom-up search, but rather is dichotomic. It is also “anytime”, so that fixed termination of the search is not necessary.

We have applied our algorithm to Yeast sequence data to discover rules predicting function classes from secondary structure. These resultant rules are informative, consistent with known biology, and a contribution to scientific knowledge. Surprisingly, the rules also demonstrate that secondary structure prediction algorithms are effective for membrane proteins; and suggest that the association between secondary structure and function is stronger in membrane proteins than globular ones. We demonstrate that our algorithm can successfully predict gene function directly from predicted secondary structure, e.g. we correctly predict the gene YGL124c to be involved in the functional class ”cytoplasmic and nuclear degradation” [14].

7. Contracts and Grants with Industry

7.1. CURAR: using chronicles for network security

Participant: Mireille Ducassé.

This contract, CRE no 171978 (External Research Contract), is a focused collaboration between the Dream project (namely Marie-Odile Cordier, René Quiniou, Alexandre Vautier), the Lande (and now LIS) team and France Telecom R & D on the problem of detecting specific network attacks. Dream is the project leader for IRISA. This study began in November 2004 and is planned to last three years. The first objective is to evaluate the use of chronicles, patterns of temporally constrained events, for representing and detecting attack scenarios on telecommunication networks. The second objective is to learn or discover automatically such attack scenarios from network logs, either generated by a simulation process or really observed on active networks. See Section 6.5 for the results of 2006.

8. Other Grants and Activities

8.1. National Collaborations

- The LIS team has a contract with Région Bretagne in collaboration with the laboratory RESO of the University of Rennes 2, for the funding of O. Bedel’s PhD.
- Annie Foret is
  - external collaborator of LINA (research lab. Nantes), in TALN team (Natural Language Processing).
  - member of a project in “Maison des sciences de l’homme” Lille (MSH) on “Apprentissage naturel et artificiel de langages naturels et artificiels”
  - INRIA ARC Mosaique member, on “ modèles syntaxiques de haut niveau” (2006,2007)
  - member of “Agence Universitaire de la Francophonie” (AUF) , LTT network on “Lexicologie, terminologie et traduction”

9. Dissemination

9.1. Awards

Yoann Padioleau, a former member of our team, received the System Research prize for his PhD on *logic file systems*, and its related implementation, LISFS (see Section 5.1). This PhD was supervised by Olivier Ridoux, and defended in February 2005 at the University of Rennes 1. The System Research prize is awarded every two years by the ASF, French branch of the association ACM SIGOPS that groups together researchers, academics and industrialists in the field of operating systems and distributed systems. The prize money of 5000 euros, sponsored by an industrialist, goes to the best French thesis on system. France Telecom sponsored it this year, following Gemplus and Microsoft the previous two issues.

> *"In a deep overhaul of a well-threaded theme, Yoann Padioleau brings fresh ideas to the domain of file systems. He proposes a brand new vision of our hard drive. Today, the user is forced to store his documents, pictures, movies, emails, meetings, ..., in a strict hierarchical way. For retrieving a document, often lost from sight, the only available tool is global search. The proposal of Logic File System is to permit the user to express logically a combination of hierarchical patterns and of research patterns. Every document is endowed with multiple attributes: for instance, the date, location, names of persons of interest are possible attributes of a picture. The picture is stored, then retrieved, using any combination of the attributes. This neat approach lays on a strong theoretical basis, and has undergone an experimental validation. Performance and usability have been measured using well-chosen and reproducible metrics. Finally, Logic File System can be downloaded freely. The results have been published in the international conference USENIX."* – ASF
>
> *"This work presents a real breakthrough in the way information can be organised and stored, using logical relations instead of purely hierarchical ones, like the sub-directories in classic file-management systems. The user can thus combine hierarchical and feature elements: for a photo for instance, its date, the place where it was taken, or the names of the people on the picture, which helps to file and retrieve documents in a much more efficient way."* – France Telecom

9.2. Involvement in the Scientific Community

- Olivier Ridoux has been a member of the program committee of ICCS'2006 (Int. Conf. on Conceptual Structures). He is in the editorial board of Interstice (http://interstice.info).
- Mireille Ducassé has been invited at the university of Melbourne by NICTA, National ICT Australia, from August 2nd to August 24th 2006, to work as an expert on the G12 project on Constraint Programming url : . She has fostered the specification of a trace schema for the new constraint features of Mercury, the functional and logic programming language used to build the G12 project. While in Melbourne she gave a presentation at NICTA's *big seminar series* on "Automatic debugging: mining for bugs".
  Mireille Ducassé has served in the program committees of PADL'06 (International Symposium on Practical Aspects of Declarative Languages), Charleston, South Carolina, USA ; TAIC'06 (Testing Academic & Industrial Conference), Windsor, UK. and WLPE'06 (Workshop on Logic-based Programming Environments), Seattle, USA. She will serve in the program committees of JFPC'07 (Journées Francophones de Programmation par Contraintes), Rocquencourt, France et TAIC'07 (Testing Academic & Industrial Conference), Windsor, UK.
- Sébastien Ferré has been a member of the program committees of ICFCA'07 (Int. Conf. Formal Concept Analysis), and CLA'06 (Concept Lattices and their Applications).

- Annie Foret has been a program committee member of *Formal Grammar* 2006 International Conference, and of *CAP* 2006 French Conference on Learning (Conférence francophone sur l'apprentissage automatique). She has also been a member in the following Phd Committee : "Acquisition de grammaires lexicalisées pour les langues naturelles." ("Acquisition of lexicalized grammars for natural languages") by Erwan Moreau, Nantes 2006.

9.3. Teaching

- Olivier Ridoux is the head of IFSIC (Institut de Formation Supérieure en Informatique et Communication - the Computing Science department at University of Rennes 1).
  Olivier Ridoux teaches compilation, logic and constraint programming, as well as software engineering at the Master level of IFSIC. He also teaches an introduction to computability and complexity at the Licence level.
- Mireille Ducassé is an elected member of the board of directors of the Insa of Rennes since June 2006. She is the chair of the committee in charge of the employment of teaching and research staff in computer science at the Insa of Rennes ("commission de spécialistes"). She is a member of two other such committees: at the University of Rennes 1 and the University of "La Réunion". She is responsible of the student exchange program for the computer science department of the Insa of Rennes.
  Mireille Ducassé has been invited to chair the committee assessing the final internship of computer science students of the Ecole Spéciale Militaire de Saint-Cyr, in January 2006. At Insa, she teaches compilation and formal methods for software engineering (with the "B formal method") at Master 1 level of Insa. She leads an exercise of participatory design based on the work of Wendy Mackay from the "In Situ" project of Inria Futurs. She has started this year to contribute to a course on risk analysis at Licence 2 level.
- Sébastien Ferré teaches programming in various languages (Objective Caml, Scheme, Mathematica, Java), and algorithmics of graphs and sequences. The former is in the form of an initiation to programming (L1 Physics-Chemistry, L2 Miage, M1 Bioinformatics). The latter concerns M1 students. He is also co-responsible of the 1st year of a master in bioinformatics.
- Annie Foret teaches university courses including formal logic, functional programming, and databases.

10. Bibliography

Major publications by the team in recent years

[1] D. BECHET, A. DIKOVSKY, A. FORET. *Dependency Structure Grammars*, in "Int. Conf. Logical Aspects of Computational Linguistics (LACL)", LNAI 3492, 2005.

[2] D. BECHET, A. FORET. *k-valued Non-Associative Lambek Categorial Grammars are not Learnable from Strings*, in "Annual Meeting of the Association for Computational Linguistics (ACL)", 2003.

[3] S. FERRÉ. *Systèmes d'information logiques : un paradigme logico-contextuel pour interroger, naviguer et apprendre*, Awarded by SPECIF (Society of French Professors in Computer Science) in 2003 as the second best PhD, Thèse d'université, Université de Rennes 1, October 2002.

[4] S. FERRÉ, R. D. KING. *A dichotomic search algorithm for mining and learning in domain-specific logics*, in "Fundamenta Informaticae – Special Issue on Advances in Mining Graphs, Trees and Sequences", vol. 66, nº 1-2, 2005, p. 1–32.

[5] S. FERRÉ, O. RIDOUX. *A Framework for Developing Embeddable Customized Logics*, in "Int. Work. Logic-based Program Synthesis and Transformation", A. PETTOROSSI (editor). , LNCS 2372, Springer, 2002, p. 191–215.

[6] S. FERRÉ, O. RIDOUX. *An Introduction to Logical Information Systems*, in "Information Processing & Management", vol. 40, nº 3, 2004, p. 383–419.

[7] E. JAHIER, M. DUCASSÉ. *Generic Program Monitoring by Trace Analysis*, in "Theory and Practice of Logic Programming", vol. 2, nº 4-5, July-September 2002, p. 611-643.

[8] L. LANGEVINE, P. DERANSART, M. DUCASSÉ. *A Generic Trace Schema for the Portability of CP(FD) Debugging Tools*, in "Recent advances in Constraint Programming", J. VANCZA, K. APT, F. FAGES, F. ROSSI, P. SZEREDI (editors). , Springer-Verlag, Lecture Notes in Artificial Intelligence 3010, 2004.

[9] Y. PADIOLEAU. *Logic File System, un système de fichier basé sur la logique*, Awarded by ASF (ACM SIGOPS France) in 2006 as the best french PhD in operating systems, Thèse d'université, Université de Rennes 1, February 2005.

[10] Y. PADIOLEAU, O. RIDOUX. *A Logic File System*, in "Usenix Annual Technical Conference", 2003.

Year Publications

Doctoral dissertations and Habilitation theses

[11] E. TOMBINI. *Amélioration du diagnostic en détection d'intrusions : étude et application d'une combinaison de méthodes de détection d'intrusions comportementale et par scénarios*, M. Ducassé, H. Debar et L. Mé, directeurs de thèse, Ph. D. Thesis, INSA de Rennes, septembre 2006.

Articles in refereed journals and book chapters

[12] D. BECHET, R. BONATO, A. DIKOVSKY, A. FORET, Y. L. NIR, E. MOREAU, C. RETORE, I. TELLIER. *Modèles algorithmiques de l'acquisition de la syntaxe : concepts et méthodes, résultats et problèmes*, in "Journal Recherches linguistiques de Vincennes", 2006.

[13] D. BECHET, A. FORET. *k-Valued Non-Associative Lambek Grammars are learnable from Generalized Functor-Argument Structures*, in "Journal of Theoretical Computer Science", vol. 355, nº 2, 2006.

[14] S. FERRÉ, R. D. KING. *Finding Motifs in Protein Secondary Structure for Use in Function Prediction*, in "Journal of Computational Biology", vol. 13, nº 3, 2006, p. 719–731.

[15] B. SIGONNEAU, O. RIDOUX. *Indexation multiple et automatisée de composants logiciels*, in "Technique et Science Informatiques", vol. 25, nº 1, 2006.

Publications in Conferences and Workshops

[16] O. BEDEL, S. FERRÉ, O. RIDOUX, E. QUESSEVEUR. *GEOLIS: A Logical Information System for Geographical Data*, in "Int. Conf. Spatial Analysis and GEOmatics (SAGEO)", ISBN 2-9526014-1-0, 2006.

[17] O. BEDEL, O. RIDOUX, E. QUESSEVEUR. *Combining Logical Information System and OpenGIS Tools for Geographical Data Exploration*, in "Int. Conf. Free and OpenSource Sofware for Geoinformatics", september 2006, http://www.foss4g2006.org/contributionDisplay.py?contribId=131&sessionId=51&confId=1.

[18] P. CELLIER, S. FERRÉ, O. RIDOUX, M. DUCASSÉ. *A Parameterized Algorithm for Exploring Concept Lattices*, in "Int. Conf. Formal Concept Analysis", S. KUZNETSOV, S. SCHMIDT (editors). , LNAI 4390, To appear, Springer, feb .

[19] P. CELLIER, S. FERRÉ, O. RIDOUX, M. DUCASSÉ. *An Algorithm to Find Frequent Concepts of a Formal Context with Taxonomy*, in "Int. Conf. Concept Lattices and Their Applications", S. B. YAHIA, E. M. NGUIFO (editors). , ISBN 978-9973-61-481-0, Faculté des Sciences de Tunis, 2006, p. 243–248.

[20] S. FERRÉ. *The Efficient Computation of Complete and Concise Substring Scales with Suffix Trees*, in "Int. Conf. Formal Concept Analysis", S. KUZNETSOV, S. SCHMIDT (editors). , LNAI 4390, To appear, Springer, feb .

[21] S. FERRÉ. *Negation, Opposition, and Possibility in Logical Concept Analysis.*, in "Int. Conf. Formal Concept Analysis", R. MISSAOUI, J. SCHMID (editors). , LNCS 3874, Springer, 2006, p. 130-145.

[22] Y. PADIOLEAU, B. SIGONNEAU, O. RIDOUX. *LISFS: a logical information system as a file system (demo)*, in "Int. Conf. Software Engineering", L. J. OSTERWEIL, H. D. ROMBACH, M. L. SOFFA (editors). , ACM, 2006, p. 803-806.

[23] E. POUPARD, D. BECHET, A. FORET. *Categorial Grammar Acquisition from a French Treebank*, in "Actes de la Conférence d'Apprentissage (CAP)", (Poster), 2006.

[24] B. SIGONNEAU, O. RIDOUX. *Software Engineering Applications of Logic File System – Application to Automated Multi-Criteria Indexation of Software Components*, in "ICSE Work. Mining Software Repositories", 2006.

[25] A. VAUTIER, M.-O. CORDIER, M. DUCASSÉ, R. QUINIOU. *Agrégation d'alarmes faiblement structurées*, in "Actes de l'atelier "Fouille de données temporelles" associé aux 6es Journées Francophones "Extraction et de Gestion des Connaissances"", Janvier 2006, http://www.irisa.fr/dream/Pages_Pros/Alexandre.Vautier/publis/egc_06_vpn.pdf.

Internal Reports

[26] S. FERRÉ, O. RIDOUX. *Logic Functors: A Toolbox of Components for Building Customized and Embeddable Logics*, Research Report, n° RR-5871, INRIA, March 2006, http://www.inria.fr/rrrt/rr-5871.html.

[27] A. VAUTIER, M.-O. CORDIER, M. DUCASSÉ, R. QUINIOU. *Visualization of Internet Flow Records*, CURAR deliverable - CRE France Telecom R & D, Publication Interne, n° 1828, IRISA, november 2006.

References in notes

[28] M. BARBUT, B. MONJARDET. *Ordre et classification — Algèbre et combinatoire (2 tomes)*, Hachette, Paris, 1970.

[29] D. BECHET, A. FORET. *k-Valued Non-Associative Lambek Grammars are Learnable from Function-Argument Structures*, in "Workshop Logic, Language, Information and Computation (WoLLIC), volume 85, Electronic Notes in Theoretical Computer Science", July 2003.

[30] D. BECHET, A. FORET. *On Intermediate Structures for Non-Associative Lambek Grammars and Learnability*, in "Proceedings of the CG 2004 Conference : Categorial Grammars "An efficient tool for Natural Language Processing"", June 2004.

[31] G. BIRKHOFF. *Lattice Theory*, American Mathematical Society, 1940.

[32] R. J. BRACHMAN. *On the Epistemological Status of Semantic Nets*, in "Associative Networks: Representation of Knowledge and Use of Knowledge by Examples, New York", N. V. FINDLER (editor). , Academic Press, 1979.

[33] D. CALVANESE, M. LENZERINI, D. NARDI. *Description Logics for Conceptual Data Modeling*, in "Logics for Databases and Information Systems", J. CHOMICKI, G. SAAKE (editors). , Kluwer, 1998, p. 229-263.

[34] R. D. COSMO. *Isomorphisms of Types: from λ-calculus to information retrieval and language design*, Progress in theoretical computer science, Birkhäuser, 1995.

[35] B. A. DAVEY, H. A. PRIESTLEY. *Introduction to Lattices and Order*, Cambridge University Press, 1990.

[36] S. FERRÉ, O. RIDOUX. *The Use of Associative Concepts in the Incremental Building of a Logical Context*, in "Int. Conf. Conceptual Structures", G. A. U. PRISS (editor). , LNCS 2393, Springer, 2002, p. 299–313.

[37] A. FORET, Y. LE NIR. *Lambek rigid grammars are not learnable from strings*, in "COLING'2002, 19th International Conference on Computational Linguistics, Taipei, Taiwan", 2002.

[38] B. GANTER, R. WILLE. *Formal Concept Analysis — Mathematical Foundations*, Springer, 1999.

[39] D. K. GIFFORD, P. JOUVELOT, M. A. SHELDON, J. W. J. O'TOOLE. *Semantic file systems*, in "13th ACM Symposium on Operating Systems Principles", ACM SIGOPS, 1991, p. 16–25.

[40] R. GODIN, R. MISSAOUI, A. APRIL. *Experimental Comparison of Navigation in a Galois Lattice with Conventional Information Retrieval Methods*, in "International Journal of Man-Machine Studies", vol. 38, n° 5, 1993, p. 747–767.

[41] B. GOPAL, U. MANBER. *Integrating Content-Based Access Mechanisms with Hierarchical File Systems*, in "third symposium on Operating Systems Design and Implementation", USENIX Association, 1999, p. 265–278.

[42] S. O. KUZNETSOV. *Machine Learning and Formal Concept Analysis.*, in "Int. Conf. Formal Concept Analysis", P. W. EKLUND (editor). , LNCS 2961, Springer, 2004, p. 287-312.

[43] J. LAMBEK. *The Mathematics of Sentence Structure*, in "American Mathematical Monthly", vol. 65, 1958, p. 154-170.

[44] R. LAURINI, D. THOMPSON. *Fundamentals of Spatial Information Systems*, Elsevier, Academic Press Limited, 1992.

[45] H. LEVESQUE. *All I know: a study in autoepistemic logic*, in "Artificial Intelligence", vol. 42, n° 2, March 1990.

[46] Y. PADIOLEAU, O. RIDOUX. *A Parts-of-File File System*, in "USENIX Annual Technical Conference, General Track (Short Paper)", 2005, http://www.usenix.org/events/usenix05/tech/general/padioleau.html.

[47] P. TARR, H. OSSHER, W. HARRISON, S. SUTTON. *N Degrees of Separation: Multi-Dimentional Separation of Concerns*, in "ICSE", IEEE Computer Society, 1999, p. 107–119.

[48] R. WILLE. *Ordered Sets*, chap. Restructuring lattice theory: an approach based on hierarchies of concepts, Reidel, 1982, p. 445-470.