INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Team LIS

Logical Information Systems

Rennes

THEME SYM

Activity Report

2007

Table of contents

1. Team **1**
2. Overall Objectives **1**
2.1. Overview 1
2.2. Key Issues 2
3. Scientific Foundations **3**
3.1. Logics for Information Systems 3
3.1.1. Properties of a Logic 4
3.1.2. Examples of Logics for Information Systems 4
3.2. Concept Analysis 5
3.2.1. Formal Concept Analysis 5
3.2.2. Logical Concept Analysis 5
3.3. Logical Querying, Navigation, and Data-mining 6
3.4. Genericity and Components 7
3.5. Logic Functors 8
3.6. Categorial Grammars 8
4. Application Domains **9**
4.1. Geographical Information Systems 9
4.2. Mining Software Repositories 9
5. Software **10**
5.1. LISFS 10
5.2. GEOLIS 10
5.3. Camelis 11
5.4. LogFun 11
6. New Results **11**
6.1. Dynamic Taxonomies: from Taxonomies to Logics 11
6.2. CAMELIS: Organizing and Browsing a Personal Photo Collection with a Logical Information System 12
6.3. The Efficient Computation of Complete and Concise Substring Scales with Suffix Trees 12
6.4. A Parameterized Algorithm to Explore Formal Contexts with a Taxonomy 12
6.5. Formal Concept analysis enhances Fault Localization in Software 12
6.6. Observational Semantics of Prolog 13
6.7. Tracer Driver and Dynamic Analyses of CLP(FD) 13
6.8. Using polyhedral abstractions in constraint-based testing 13
6.9. GEOLIS: A Logical Information System for Geographical Data 14
6.10. Pregroup Calculus as a Logical Functor 14
6.11. Fully Lexicalized Pregroup Grammars 14
6.12. Learnability of Categorial Grammars 15
7. Other Grants and Activities **15**
7.1. International Collaborations 15
7.2. National Collaborations 15
8. Dissemination **16**
8.1. Scientific Popularization: a short film to introduce logical information systems 16
8.2. Involvement in the Scientific Community 16
8.3. Teaching 16
9. Bibliography **17**

1. Team

Head of the team
Olivier Ridoux [Professor, Université Rennes 1, HdR]

Administrative assistant
Lydie Mabil [TR Inria]

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

Insa personnel
Mireille Ducassé [Professor, HdR]
Véronique Abily [Research Engineer (1/5 of the time)]

PhD students
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]

Visitors
Daniela Bargelli [Professor, MacGill University, Montreal]

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 [39]) 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 [9] 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 [9], [5], [6].

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* [49]. 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 [13], [12] 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) [35], [36], 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 [45] 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 [34], [37]. It can be seen as a reformulation by Wille of Galois lattices [33] that emphasizes lattices as conceptual hierarchies [50]. The mathematical foundations of FCA have been extensively studied by Ganter and Wille [39].

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 [39], 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) [9], [5], 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, $\perp$, 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 [9], [6].

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 [40], once a querying step is done, there is no more navigation. An example of the latter is HAC [42], some query answers may not satisfy the query. A proposal based on FCA has not these drawbacks [41], and we have generalized it to work within LCA, which allows us to use logical formulas for object description and queries [5], [9]. 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 [44]. Here again, we have combined existing techniques with logic [9], [6], and applied them to the automatic classification of emails [38], and the prediction of the function of proteins from their sequence [6].

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, 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* [8], [5], 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 $\Sigma_1 = \{John, Mary, likes\}$ $I_1 = \{John \mapsto \{N\}$, $Mary \mapsto \{N\}$, $likes \mapsto \{N \diagdown (S/N)\}\}$ the sentence "John likes Mary" belongs to the language of G_1 because $N, N \diagdown (S/N), N \vdash S$ due to successive applications of the two elimination rules : $X, X \diagdown Y \vdash Y$ and $Y, Y/X \vdash Y$. Type constructors $/$ and $\diagdown$ 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 $\forall n > n_0$ $\phi(\{e_0, ..., e_n\}) = G'$.

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 [46] 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.

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 [13], [12]. 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 [47][12]. 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. GEOLIS

Participant: Olivier Bedel.

GEOLIS is a prototype combining a Logical Information System (LIS) and webmapping tools for geographical data exploration. GEOLIS takes the form of a web application. Server-side, GEOLIS relies on LISFS to organize of the data and on the webmapping engine MapServer to produce a map representation of data selection. Client-side, the GEOLIS user interface provides three components: 1) a query box similar to web search engines querying interfaces, 2) a map area, and 3) a navigation tree gathering navigation links. Navigation links can be followed to reduce (resp. enlarge) the current selection of data visible on the map by refining (resp. generalizing) the current query written in the query box. Online demo of GEOLIS works with a partial dataset concerning rodents distribution in Soudano-Sahelian Africa. It aims at helping geographs in the research of factors impacting on rodents distribution.

5.3. 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 5.4). It is available on Linux and Windows, and comes with a user manual.

5.4. 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. Dynamic Taxonomies: from Taxonomies to Logics

Participants: Sébastien Ferré, Olivier Ridoux.

Like logical information systems, dynamic taxonomies [48] have been proposed as a solution for combining querying and navigation, offering both expressivity and interactivity. Navigation is based on the filtering of a multidimensional taxonomy w.r.t. query answers, which helps users to focus their search. We show that properties that are commonly used only in queries can be integrated in taxonomies, and hence in navigation, by the use of logics [31]. Hand-designed taxonomies and concrete domains (e.g., dates, strings) can be combined so as to form complex taxonomies. For instance, valued attributes can be handled, and different roles between documents and locations can be distinguished.

6.2. CAMELIS: Organizing and Browsing a Personal Photo Collection with a Logical Information System

Participant: Sébastien Ferré.

Since the arrival of digital cameras, many people are faced to the challenge of organizing and retrieving the overwhelming flow of photos their life produces. Most people put no metadata on their photos, and we believe this is because existing tools make a very limited use of them. We present a tool, Camelis, that offers users with an organization of photos that is dynamically computed from the metadata, making worthwhile the effort to produce it. Camelis is designed along the lines of Logical Information Systems (LIS), which are founded on logical concept analysis. Hence, (1) an expressive language can be used to describe photos and query the collection, (2) manual and automatic metadata can be smoothly integrated, and (3) expressive querying and flexible navigation can be mixed in a same search and in any order. This presentation is illustrated by experiences on a real collection of more than 5000 photos [29].

6.3. The Efficient Computation of Complete and Concise Substring Scales with Suffix Trees

Participant: Sébastien Ferré.

Strings are an important part of most real application multi-valued contexts. In the framework of formal concept analysis, their conceptual treatment requires the definition of *substring scales*, i.e., sets of relevant substrings, so as to form informative concepts. However these scales are either defined by hand, or derived in a context-unaware manner (e.g., all words occuring in string values). We present an efficient algorithm based on suffix trees that produces complete and concise substring scales [30]. Completeness ensures that every possible concept is formed, like when considering the scale of all substrings. Conciseness ensures the number of scale attributes (substrings) is less than the cumulated size of all string values. This algorithm is integrated in Camelis, and illustrated on the set of all ICCS paper titles.

6.4. A Parameterized Algorithm to Explore Formal Contexts with a Taxonomy

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

Formal Concept Analysis (FCA) is a natural framework to learn from examples. Indeed, learning from examples results in sets of frequent concepts whose extent contains mostly these examples. In terms of association rules, the above learning strategy can be seen as searching the premises of rules where the consequence is set. In its most classical setting, FCA considers attributes as a non-ordered set. When attributes of the context are partially ordered to form a taxonomy, Conceptual Scaling allows the taxonomy to be taken into account by producing a context completed with all attributes deduced from the taxonomy. The drawback, however, is that concept intents contain redundant information.

We have proposed a parameterized algorithm, to learn rules in the presence of a taxonomy. It works on a non-completed context. The taxonomy is taken into account during the computation so as to remove all redundancies from intents. Simply changing one of its operations, this parameterized algorithm can compute various kinds of concept-based rules. Two instantiations of the parameterized algorithm have been proposed to learn rules as well as to compute the set of frequent concepts [25], [19].

6.5. Formal Concept analysis enhances Fault Localization in Software

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

The current trend in debugging and testing is to cross-check information collected during several executions. Jones et al. [43], for example, propose to use the instruction coverage of passing and failing runs in order to visualize suspicious statements. The implicit underlying technique is to search for *association rules* which indicate that executing a particular source line will cause the whole execution to fail. This technique, however, has limitations, for instance at least one statement must be considered as faulty .

We have proposed a process that combines association rules and formal concept lattices to give a relevant way to navigate into the source code of faulty programs. We consider more expressive association rules where several lines imply failure, it allows to alleviate some limitations of the Jones *et al*'s method. Formal Concept Analysis (FCA) is used to analyze the resulting numerous rules in order to improve the readability of the information contained in the rules. Casting the fault localization problem in the FCA framework helps analyze existing approaches, as well as alleviate some of their limitations. [24].

6.6. Observational Semantics of Prolog

Keywords: *Debugging*, *Execution Trace semantics*, *Programming Environment*, *Prolog*, *Software Engineering*.

Participant: Mireille Ducassé.

This work specifies an observational semantics of Prolog and gives an original presentation of the Byrd's box model. The approach accounts for the semantics of Prolog tracers independently of a particular implementation. Traces are, in general, considered as rather obscure and difficult to use. The proposed formal presentation of a trace constitutes a simple and pedagogical approach for teaching Prolog or for implementing Prolog tracers. It constitutes a form of declarative specification for the tracers. Our approach highlights the qualities of the box model which made its success, but also its drawbacks and limits. As a matter of fact, the presented semantics is only one example to illustrate general problems relating to tracers and observing processes. Observing processes know, from observed processes, only their traces. The issue is then to be able to reconstitute by the sole analysis of the trace the main part of the observed process, and if possible, without any loss of information[28].

This activity is done in collaboration with Pierre Deransart from the INRIA Rocquencourt team Contraintes, and Gérard Ferrand from the University of Orléans.

6.7. Tracer Driver and Dynamic Analyses of CLP(FD)

Keywords: *Constraint Logic Programming*, *Debugging*, *Execution Monitoring*, *Execution Trace Analysis*, *Execution Tracing*, *Execution Visualization*, *Programming Environment*, *Software Engineering*.

Participant: Mireille Ducassé.

Tracers provide users with useful information about program executions. In this article, we propose a "tracer driver". From a single tracer, it provides a powerful front-end enabling multiple dynamic analysis tools to be easily implemented, while limiting the overhead of the trace generation. The relevant execution events are specified by flexible event patterns and a large variety of trace data can be given either systematically or "on demand". The proposed tracer driver has been designed in the context of constraint logic programming; experiments have been made within GNU-Prolog. Execution views provided by existing tools have been easily emulated with a negligible overhead. Experimental measures show that the flexibility and power of the described architecture lead to good performance.

The tracer driver overhead is inversely proportional to the average time between two traced events. Whereas the principles of the tracer driver are independent of the traced programming language, it is best suited for high-level languages, such as constraint logic programming, where each traced execution event encompasses numerous low-level execution steps. Furthermore, constraint logic programming is especially hard to debug. The current environments do not provide all the useful dynamic analysis tools. They can significantly benefit from our tracer driver which enables dynamic analyses to be integrated at a very low cost[20].

This activity is done in collaboration with Ludovic Langevine from Mission Critical IT, Belgium.

6.8. Using polyhedral abstractions in constraint-based testing

Keywords: *Constraint-based automatic test data generation*, *constraint propagation*, *linear relaxation*.

Participant: Mireille Ducassé.

Constraint-Based Testing (CBT) is the process of generating test cases against a testing objective by using constraint solving techniques. In CBT, testing objectives are given under the form of properties to be satisfied by program's input/output. Whenever the program or the properties contain disjunctions or multiplications between variables, CBT faces the problem of solving non-linear constraint systems. Currently, existing CBT tools tackle this problem by exploiting a finite-domains constraint solver. But, solving a non-linear constraint system over finite domains is NP_hard and CBT tools fail to handle properly most properties to be tested. In this work, we introduced a CBT approach where a finite domain constraint solver is enhanced by Dynamic Linear Relaxations (DLRs). DLRs are based on linear abstractions derived during the constraint solving process. They dramatically increase the solving capabilities of the solver in the presence of non-linear constraints without compromising the completeness or soundness of the overall CBT process. We implemented DLRs within the CBT tool TAUPO that generates test data for programs written in C and got initial results over a few (academic) C programs [27]. Extending this approach to handle loops that depend on input variables led us to combine narrowing techniques within constraint combinators by defining a new constraint language inspired from the imperative programming style [26].

This activity is done in collaboration with Tristan Denmat and Arnaud Gotlieb from the IRISA Lande team.

6.9. GEOLIS: A Logical Information System for Geographical Data

Participants: Olivier Bedel, Sébastien Ferré, Olivier Ridoux.

Today, the thematic layer is still the prevailling structure in geomatics for handling geographical information. However, the layer model is rigid: it implies partitionning geographical data in predefined categories and using the same description schema for all elements of a layer. Futhermore, Geographical Information Systems (GIS) rely exclusively on querying for geographical information retrieval. Using Logical Information Systems (LIS) paradigm for information management and retrieval, we propose a more flexible organisation of vectorial geographical data at a thiner level since it is centered on the geographical object. Our data model allows to consider every collections of geographical objects that share a common description. Geographical objects descriptions mix spatial and non-spatial properties that are handled by specialized logics. Especially, a spatial logic has been designed to test the inclusion of the different kinds of geometrical description (i.e. polygon, line and point) and to reason on derived properties such as the area or the length [21]. Our navigation model allows to freely combine querying and navigation on geographical data. More particularly, the navigation model relies on three different views over the geographical data: 1) the current selection is described intentionnaly by the current query, 2) its extension is represented graphically on the geographical map, and 3) the navigation tree gathers the properties describing objects of the current selection. These properties also serve as navigation links to refine or generalize the current query. The data and the navigation models have been implemented in the GEOLIS prototype, which has been used to lead experiments on a real dataset [17].

6.10. Pregroup Calculus as a Logical Functor

Participants: Annie Foret, Sébastien Ferré.

The concept of pregroup was introduced by Lambek for natural language analysis, with a close link to non-commutative linear logic. Pregroup grammars are a context-free grammar formalism introduced as a simplification of Lambek calculus. We reformulate the pregroup calculus so as to extend it by composition with other logics and calculii.The cut elimination property and the decidabilityproperty of the sequent calculus proposed in the article are shown. Properties of composed calculii are also discussed [32].

6.11. Fully Lexicalized Pregroup Grammars

Participant: Annie Foret.

The pregroup formalism is interesting for several reasons: the syntactical properties of words are specified by a set of types like the other type-based grammar formalisms ; as a logical model, compositionality is easy ; a polytime parsing algorithm exists. However, this formalism is not completely lexicalized because each pregroup grammar is based on the free pregroup built from a set of primitive types together with a partial order, and this order is not lexical information. In fact, only the pregroup grammars that are based on primitive types with an order that is equality can be seen as fully lexicalized. We show here how we can transform, using a morphism on types, a particular pregroup grammar into another pregroup grammar that uses the equality as the order on primitive types. This transformation is at most quadratic in size (linear for a fixed set of primitive types), it preserves the parse structures of sentences and the number of types assigned to a word [23].

6.12. Learnability of Categorial Grammars

Participant: Annie Foret.

In [16] the Learnability of Pregroup Grammars is investigated. This paper focuses on the learnability by positive examples in the sense of Gold of Pregroup Grammars. In a first part, Pregroup Grammars are presented and a new parsing strategy is proposed. Then, theoretical learnability and non-learnability results for subclasses of Pregroup Grammars are proved. In the last two parts, we focus on learning Pregroup Grammars from a special kind of input called feature-tagged examples. A learning algorithm based on the parsing strategy presented in the first part is given. Its validity is proved and its properties are examplified.

In [18], recent results on the acquistion of the syntax of natural languages are presented, from the point of view of the theory of grammatical inference. Given a class of possible grammars, the objective is to identify, from a set of positive examples, a grammar in the class which produces the examples. The Gold model formalises the learning process and gives stringent criteria of its success: when does there exist an algorithm producing a target grammar ? what kind of structure should the examples have (strings of words, strings of tagged words, trees) ? >From a theoretical point of view, our results establish the learnability or the unlearnability of various classes of categorial grammars. From a practical perspective, these results enable the extraction of syntactic information from real data. Finally, we discuss the interest of this approach for modelling child language acquisition and for automated induction of grammars from corpora.

7. Other Grants and Activities

7.1. International Collaborations

Daniela Bargelli, from McGill Canada, has been invited for three months in the LIS project. The collaboration involves the pregroup formalism and the development of actual pregroup grammars for some choosen natural languages, French in particular. D. Bargelli and A. Foret have organized a meeting around Lambek formalisms the day after the TALN'07 conference in Toulouse (details can be found here).

7.2. 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”

8. Dissemination

8.1. Scientific Popularization: a short film to introduce logical information systems

Participants: Olivier Bedel, Peggy Cellier, Sébastien Ferré.

In the framework of the popular science short-film festival of Rennes (TCM 2007), the members of the team LIS have written and directed a short-film of 4 min. It briefly describes the mechanism of logical information systems in a pictorial and amusing manner. The short film is entitled "A la recherche des photos insolites..." (Seeking the funny pictures...) and has been shown at the festival and the science fest. The short-film has also been used to present the research area of the team to several groups of students.

8.2. Involvement in the Scientific Community

- Olivier Ridoux has been a member of the program committee of ICCS'07 (Int. Conf. on Conceptual Structures). He is in the editorial board of Interstices. He also has been a member in the HDR committee of Laure Bertille on “Quality awareness in data management and mining”, in the PhD committee of Julien Pley on “Protocoles d'accord pour la gestion d'une grille de calcul dynamique”, and of Gwenaelle Marquet on “Réalisation d'une ontologie du glioblastome par intégration d'ontologies biologiques et médicales”.
- Mireille Ducassé has served in the program committees of JFPC'07 (Journées Francophones de Programmation par Contraintes), Rocquencourt, France; TAIC'07 (Testing Academic & Industrial Conference), Windsor, UK and PADLE'08 (International Symposium on Practical Aspects of Declarative Languages), San Francisco, USA.
- Sébastien Ferré has been a member of the program committees of ICFCA'07 (Int. Conf. Formal Concept Analysis), and CLA'07 (Concept Lattices and their Applications). He also served as an external reviewer of the International Journal of Foundations in Computer Science (IJFCS), and the DEXA Workshop on Advances in Conceptual Knowledge Engineering (ACKE).
- 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.

8.3. 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, which led to the publication of a book [15].

  He also published in Interstices a survey on the 2007 TIPE[1] about “Variability, limit, and stability in Computer Science”.

[1]TIPE (Travaux d'Initiative Personelle Encadrés) are proposed to undergraduate students as an introduction to research. TIPE subjects are defined nationally every year. The 2007 subject was “Variability, limit, and stability”.

- Mireille Ducassé is the head of the computer science department of the INSA of Rennes since March 2007. She is an elected member of the board of directors of the Insa of Rennes. She has been 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”) until May 2007. She is a member of two other such committees: at the University of Rennes 1 and the University of “La Réunion”. She has been responsible of the student exchange program for the computer science department of the Insa of Rennes until October 2007. Since November 2007 she is an elected member of the “Conseil National des Université (CNU) 27e section”.

  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 contributes 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.
- Peggy Cellier teaches Java, databases and data mining at the INSA of Rennes.

9. Bibliography

Major publications by the team in recent years

[1] 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.

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

[3] 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, n^{o} 2, 2006.

[4] 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.

[5] 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.

[6] 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, n^{o} 1-2, 2005, p. 1–32.

[7] S. FERRÉ, R. D. KING. *Finding Motifs in Protein Secondary Structure for Use in Function Prediction*, in "Journal of Computational Biology", vol. 13, n^{o} 3, 2006, p. 719–731.

[8] 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.

[9] S. FERRÉ, O. RIDOUX. *An Introduction to Logical Information Systems*, in "Information Processing & Management", vol. 40, n^o 3, 2004, p. 383–419.

[10] E. JAHIER, M. DUCASSÉ. *Generic Program Monitoring by Trace Analysis*, in "Theory and Practice of Logic Programming", vol. 2, n^o 4-5, July-September 2002, p. 611-643.

[11] 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.

[12] 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.

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

[14] B. SIGONNEAU, O. RIDOUX. *Indexation multiple et automatisée de composants logiciels*, in "Technique et Science Informatiques", vol. 25, n^o 1, 2006.

Year Publications

Books and Monographs

[15] O. RIDOUX, G. LESVENTES. *Calculateurs, calculs, calculabilité*, Sciences Sup, To appear, Dunod, 2008.

Articles in refereed journals and book chapters

[16] D. BECHET, A. FORET, I. TELLIER. *Learnability of Pregroup Grammars*, in "Studia Logica Journal", 2007.

[17] O. BEDEL, S. FERRÉ, O. RIDOUX, E. QUESSEVEUR. *GEOLIS: A Logical Information System for Geographical Data*, in "Revue Internationale de Géomatique", Accepted for publication, 2008.

[18] D. BÉCHET, R. BONATO, A. DIKOVSKY, A. FORET, Y. L. NIR, E. MOREAU, C. RETORÉ, I. TELLIER. *Modèles algorithmiques de l'acquisition de la syntaxe : concepts et méthodes, résultats et problèmes"*, in "Recherches linguistiques de Vincennes", Vol. 37, Presses Universitaires de Vincennes, 2007.

[19] P. CELLIER, S. FERRÉ, O. RIDOUX, M. DUCASSÉ. *A Parameterized Algorithm to Explore Formal Contexts with a Taxonomy*, in "Int. J. Foundations of Computer Science (IJFCS)", Accepted for publication, 2008.

[20] L. LANGEVINE, M. DUCASSÉ. *Design and Implementation of a Tracer Driver: Easy and Efficient Dynamic Analyses of Constraint Logic Programs*, in "Theory and Practice of Logic Programming, Cambridge University Press", Accepted for publication, 2008.

Publications in Conferences and Workshops

[21] O. BEDEL, S. FERRÉ, O. RIDOUX. *Exploring a Geographical Dataset with GEOLIS*, in "DEXA Work. Advances in Conceptual Knowledge Engineering (ACKE)", IEEE Computer Society, 2007, p. 540–544.

[22] O. BEDEL, S. FERRÉ, O. RIDOUX. *Handling Spatial Relations in Logical Concept Analysis To Explore Geographical Data*, in "Int. Conf. Formal Concept Analysis", R. MEDINA, S. OBIEDKOV (editors), Accepted for publication, 2008, p. 241–257.

[23] D. BÉCHET, A. FORET. *Fully Lexicalized Pregroup Grammars*, in "Proceedings of WOLLIC 2007", vol. LNCS 4576, Springer, 2007, p. 12–25.

[24] P. CELLIER, M. DUCASSÉ, S. FERRÉ, O. RIDOUX. *Formal Concept analysis enhances Fault Localization in Software*, in "Int. Conf. Formal Concept Analysis", R. MEDINA, S. OBIEDKOV (editors), Accepted for publication, Springer, 2008.

[25] 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, Springer, 2007.

[26] T. DENMAT, A. GOTLIEB, M. DUCASSÉ. *An Abstract Interpretation-based Combinator for Modelling While Loops in Constraint Programming*, in "Int. Conf. Principles and Practice of Constraint Programming (CP)", C. BESSIÈRE (editor), LNCS 4741, Springer-Verlag, September 2007.

[27] T. DENMAT, A. GOTLIEB, M. DUCASSÉ. *Improving Constraint-Based Testing with Dynamic Linear Relaxations*, in "Int. Symp. Software Reliability Engineering (ISSRE)", K. GOSEVA-POPSTOJANOVA, P. RUNESON (editors), IEEE Press, November 2007.

[28] P. DERANSART, M. DUCASSÉ, G. FERRAND. *Une sémantique observationnelle du modèle des boîtes pour la résolution de programmes logiques*, in "Actes de Troisièmes Journées Francophones de Programmation par Contraintes", F. FAGES (editor), HAL : http://hal.inria.fr/JFPC07, 2007.

[29] S. FERRÉ. *CAMELIS: Organizing and Browsing a Personal Photo Collection with a Logical Information System*, in "Int. Conf. Concept Lattices and Their Applications", J. DIATTA, P. EKLUND, M. LIQUIÈRE (editors), 2007, p. 112–123.

[30] 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, Springer, 2007.

[31] S. FERRÉ, O. RIDOUX. *Logical Information Systems: from Taxonomies to Logics*, in "DEXA Work. Dynamic Taxonomies and Faceted Search (FIND)", IEEE Computer Society, 2007, p. 212–216.

[32] A. FORET. *Pregroup Calculus as a Logical Functor*, in "Proceedings of WOLLIC 2007", vol. LNCS 4576, Springer, 2007.

References in notes

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

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

[35] 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.

[36] 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.

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

[38] 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.

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

[40] 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.

[41] 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^{o} 5, 1993, p. 747–767.

[42] 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.

[43] J. A. JONES, M. HARROLD, J. T. STASKO. *Visualization of test information to assist fault localization*, in "Int. Conf. Software Engineering", ACM, 2002, p. 467-477.

[44] 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.

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

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

[47] 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.

[48] G. M. SACCO. *Dynamic Taxonomies: A Model for Large Information Bases*, in "IEEE Transactions Knowledge and Data Engineering", vol. 12, n^{o} 3, 2000, p. 468–479.

[49] 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.

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