UMR IRISA

Project-Team LIS

Logical Information Systems

Rennes

Activity Report

2008

1 Team

Head of the team
Olivier Ridoux, Professor, Université Rennes 1

Administrative assistant
Lydie Mabil, INRIA

Université Rennes 1 personnel
Annie Foret, Assistant Professor
Sébastien Ferré, Assistant Professor
Peggy Cellier, MENRT grant until Sep. 2008, ATER since Oct. 2008

Insa personnel
Mireille Ducassé, Professor
Véronique Abily, Research Enginner (part time: 20%)

PhD students
Olivier Bedel, Région Bretagne grant, since Oct. 2005
Pierre Allard, Région Bretagne grant, since Oct. 2008

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

Visitors
Daniela Bargelli, Professor, MacGill University, Montreal, a two-week period in June

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 [GW99]) 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 [5] 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 [5, 6, 3].

The mottoes of the LIS research are:

1. *Never impose* a priori *a structure on information.* E.g., do not use hierarchical structures. Imposing *a priori* a structure causes the *tyranny of the dominant decomposition*[TOHS99]. 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.

[GW99] B. Ganter, R. Wille, *Formal Concept Analysis — Mathematical Foundations*, Springer, 1999.

[TOHS99] P. Tarr, H. Ossher, W. Harrison, S. Sutton, "N Degrees of Separation: Multi-Dimentional Separation of Concerns", *in: ICSE*, IEEE Computer Society, p. 107–119, 1999.

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 or 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 [8, 9] or at the user level [6].

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 studies the following key issues:

- 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?

- The file system implementation of LIS can handle around 1 million elementary pieces of information, which corresponds approximately to a full homedir with 10 to 20 thousands files. This is rather small compared to relational database capabilities, but already large compared to other approaches based on formal concept analysis.

  How can it handle more? Can we reach 100 million in the next few years?

3 Scientific Foundations

3.1 Logics for Information Systems

Keywords: Syntax, interpretation, semantics, subsumption.

Glossary :

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 Logics. 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.

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 realized by an algorithm.

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) [Bra79,CLN98], 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. To the contrary, 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[Lam58] 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, descriptors, context, instance, property, extension, intension, concept.

Glossary :

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.

[Bra79] R. J. Brachman, "On the Epistemological Status of Semantic Nets", *in: Associative Networks: Representation of Knowledge and Use of Knowledge by Examples*, N. V. Findler (editor), Academic Press, New York, 1979.

[CLN98] 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, p. 229–263, 1998.

[Lam58] J. Lambek, "The Mathematics of Sentence Structure", *American Mathematical Monthly 65*, 1958, p. 154–170.

Formal Concept Analysis Formal Concept Analysis (FCA) is part of the mathematical branch of applied lattice theory [Bir40,DP90]. It can be seen as a reformulation by Wille of Galois lattices [BM70] that emphasizes lattices as conceptual hierarchies [Wil82]. The mathematical foundations of FCA have been extensively studied by Ganter and Wille [GW99].

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.

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

[Bir40] G. Birkhoff, *Lattice Theory*, American Mathematical Society, 1940.

[DP90] B. A. Davey, H. A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, 1990.

[BM70] M. Barbut, B. Monjardet, *Ordre et classification — Algèbre et combinatoire (2 tomes)*, Hachette, Paris, 1970.

[Wil82] R. Wille, *Ordered Sets*, Reidel, 1982, ch. Restructuring lattice theory: an approach based on hierarchies of concepts, p. 445–470.

[GW99] B. Ganter, R. Wille, *Formal Concept Analysis — Mathematical Foundations*, Springer, 1999.

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 [GW99], 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) [5, 6], 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 [5, 3].

3.3 Logical Querying, Navigation, and Data-mining

Keywords: Querying, navigation, data-mining.

Glossary :

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 [GJSO91], once a querying step is done, there is no more navigation. An example of the latter is HAC [GM99], some query answers may not satisfy the query. A proposal based on FCA has not these drawbacks [GMA93], and we have generalized it to work within LCA, which allows us to use logical formulas for object description and queries [6, 5]. 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

[GJSO91] 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, p. 16–25, 1991.

[GM99] B. Gopal, U. Manber, "Integrating Content-Based Access Mechanisms with Hierarchical File Systems", *in: third symposium on Operating Systems Design and Implementation*, USENIX Association, p. 265–278, 1999.

[GMA93] R. Godin, R. Missaoui, A. April, "Experimental Comparison of Navigation in a Galois Lattice with Conventional Information Retrieval Methods", *International Journal of Man-Machine Studies 38*, 5, 1993, p. 747–767.

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 [Kuz04]. Here again, we have combined existing techniques with logic [5, 3], and applied them to the automatic classification of emails [FR02], and the prediction of the function of proteins from their sequence [3].

3.4 Genericity and Components

Keywords: Abstraction, reusability, composability, component.

Glossary :

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 a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

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

[Kuz04] S. O. Kuznetsov, "Machine Learning and Formal Concept Analysis.", *in: Int. Conf. Formal Concept Analysis*, P. W. Eklund (editor), *LNCS 2961*, Springer, p. 287–312, 2004.

[FR02] 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, D. Corbett (editor), *LNCS 2393*, Springer, p. 299–313, 2002.

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, genericity, composability.

The genericity 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* [4, 6], 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. Rigidity is a useful constraint to get learnable subclasses of grammars (and related algorithms).

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 $\Sigma_1 = \{John, Mary, likes\}$ $I_1 = \{John \mapsto \{N\}$, $Mary \mapsto \{N\}$, $likes \mapsto \{N \setminus (S/N)\}\}$ 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 : $X, X \setminus Y \vdash Y$ and $Y, Y/X \vdash Y$. Type constructors / and $\setminus$ 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 $\forall n > n_0$ $\phi(\{e_0, \ldots, e_n\}) = G'$.

4 Application Domains

4.1 Geographical Information Systems

Participants: Olivier BedelOlivier RidouxSébastien FerréErwan QuesseveurFrançois 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 [LT92] handle information that are localized in space

[LT92] R. Laurini, D. Thompson, *Fundamentals of Spatial Information Systems*, Elsevier, Academic Press Limited, 1992.

(*geolocalized*). GIS form a 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. Another characteristic of GIS is an intensive usage of topological relations (toochs, overlaps, etc) and geographical relations (North, upstream, etc). Logic offers a rich language for expressing these relations and combining them.

4.2 Mining Software Repositories

Participants: Peggy CellierMireille DucasséOlivier 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 PadioleauOlivier 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 [8, 9]. 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×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 [PR05][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.

In 2008, LISFS has been extended with relations for linking objects together. This has been applied to spatial relations between geographic features (distance and topology) [10].

5.2 GEOLIS

Participants: 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

[PR05] 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`.

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. GEOLIS is not yet distributed, but online demos are available. A demo is 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

Participants: 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 tens of 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

Participants: 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. $x \in [2,5] \implies x \in [0,10]$), and a functor *Prop* for propositional reasoning (e.g. $a \wedge b \implies a$), a user can define logic *Prop(Interval)*. In this logic, a theorem like $x \in [2,5] \vee x \in [7,9] \implies x \in [0,10]$ 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.

5.5 Odalisque

Participants: Pierre AllardSébastien Ferré [contact point].

Odalisque is an implementation of LIS for the browsing of OWL-DL ontologies [8]. These ontologies come from the semantic web domain, that aims to enable machines to better understand the contents of documents on the web. Odalisque delegates all logical representation and reasoning to standard reasoners (e.g., Pellet), and does so through the Java API Jena. Odalisque provides a user interface that is similar and consistent with the interface of Camelis. In addition, it provides the ability to navigate through relations between objects, thus allowing the construction of complex queries through navigation only.

5.6 Typed grammars

Participants: Denis Béchet [LINA-Nantes] Annie Foret [contact point].

A Pregroup ToolBox is under development on the gforge Inria (https://gforge.inria.fr/projects/pregroup/) as a collaborative work with LINA. It includes a generic pregroup parser (LINA) and grammar lexicon definitions and manipulation tools based on XML. An interface with Camelis has been developped (from Camelis to the Pregroup XML format, and the other way round). It has been used to define and experiment grammar prototypes for different natural languages.

6 New Results

6.1 Fair(er) and (almost) serene committee meetings with Logical and Formal Concept Analysis

Participants: Mireille DucasséSébastien Ferré.

In academia, many decisions are taken in committee, for example to hire people or to allocate resources. Genuine people often leave such meetings quite frustrated. Indeed, it is intrinsically hard to make multi-criteria decisions, selection criteria are hard to express and the global picture is too large for participants to embrace it fully. We describe a recruiting process where logical concept analysis and formal concept analysis are used to address the above problems [13]. We do not pretend to totally eliminate the arbitrary side of the decision. We claim, however, that, thanks to concept analysis, genuine people have the possibility to 1) be fair with the candidates, 2) make a decision adapted to the circumstances, 3) smoothly express the rationales of decisions, 4) be consistent in their judgements during the whole meeting, 5)

vote (or be arbitrary) only when all possibilities for consensus have been exhausted, and 6) make sure that the result, in general a total order, is consistent with the partial orders resulting from the multiple criteria.

6.2 Organizing and browsing a collection of documents with Camelis

Participants: Sébastien Ferré.

Since the arrival of digital cameras, many people are faced with the challenge of organizing and browsing the overwhelming flood of photos their life produces. The same is true for all sorts of documents, e.g. emails, audio files. Existing systems either let users fill query boxes without any assistance, or drive them through rigid navigation structures (e.g., hierarchies); or they do not let users put annotations on their documents, even when this would support the organization and retrieval of any documents on customized criteria. We present [6] a tool, Camelis, that offers users with an organization that is dynamically computed from documents and their annotations. 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 annotations 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 on a real collection of more than 5,000 photos.

In the domain of Dynamic Taxonomies and faceted search, we show that Camelis extends the navigational capabilities. In addition to the zoom-in navigation mode (e.g., replacing Europe by France in the query), we present other navigation modes for less directed and more exploratory browsing of a document collection [14]. The presented navigation modes are zoom-out (e.g., replacing France by Europe), shift (e.g., replacing France by Spain), pivot (e.g., switching location and time), and querying by examples. These modes all correspond to query transformations, and make use of boolean operators. The current focus of the search is always clearly specified by a query, and complex boolean queries can be constructed through navigation only, i.e. by successive selection of navigation links.

6.3 GEOLIS: a logical information system for geographical data

Participants: Olivier BedelSébastien FerréOlivier 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. 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 [4].

6.4 Handling spatial relations in Logical Concept Analysis to explore geographical data

Participants: Olivier BedelSébastien FerréOlivier Ridoux.

Because of the expansion of geo-positioning tools and the democratization of geographical information, the amount of geo-localized data that is available around the world keeps increasing. So, the ability to efficiently retrieve informations in function of their geographical facet is an important issue. In addition to individual properties such as position and shape, spatial relations between objects are an important criteria for selecting and reaching objects of interest: e.g., given a set of touristic points, selecting those having a nearby hotel or reaching the nearby hotels. We propose Logical Concept Analysis (LCA) and its handling of relations for representing and reasoning on various kinds of spatial relations [10]: e.g., Euclidean distance, topological relations. Furthermore, we present an original way of navigating in geolocalized data, and compare the benefits of our approach with traditional Geographical Information Systems (GIS).

6.5 Dynamic Taxonomies for the Semantic Web

Participants: Pierre AllardSébastien Ferré.

The semantic web aims at enabling the web to understand and answer the requests from people and machines. It relies on several standards for representing and reasoning about web contents. Among them, the Web Ontology Language (OWL) is used to define ontologies, i.e. knowledge bases, and is formalized with description logics. We demonstrate how the benefits of dynamic taxonomies and logical information systems can be transposed to browse OWL DL ontologies [8]. We only assume the ontology has an assertional part, i.e. defines objects and not only concepts. The existence of relations between objects in OWL leads us to define new navigation modes for crossing these relations. A prototype, Odalisque, has been developed on top of well-known tools for the semantic web.

This is a result of the master thesis of Pierre Allard, supervised by Sébastien Ferré.

6.6 Optional and iterated types for Pregroup Grammars

Participants: Annie Foret.

Pregroup grammars are a context-free grammar formalism which may be used to describe the syntax of natural languages. However, this formalism is not able to easily define types corresponding to optional or iterated arguments like an optional complement of a verb or a sequence of its adverbial modifiers. This paper introduces two constructions that make up for this deficiency [9]. We introduce Gentzen-style rules to take care of two new operations, and an equivalent rewriting system. The extended pregroup calculus enjoys several properties shared with traditional dependency grammars, yet does not significantly expand the polynomial complexity of the syntactic analysis on the pregroup grammar [3].

6.7 A parameterized algorithm to explore formal contexts with a taxonomy

Participants: Peggy CellierSébastien FerréOlivier RidouxMireille 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 [5].

6.8 Formal concept analysis enhances fault localization in software

Participants: Peggy CellierMireille DucasséSébastien FerréOlivier Ridoux.

The current trend in debugging and testing is to cross-check information collected during several executions. Jones et al. [JHS02], 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

[JHS02] J. A. Jones, M. Harrold, J. T. Stasko, "Visualization of test information to assist fault localization", *in : Int. Conf. Software Engineering*, ACM, p. 467–477, 2002.

the rules. Casting the fault localization problem in the FCA framework helps analyze existing approaches, as well as alleviate some of their limitations [11, 12].

6.9 DeLLIS : debugging programs by localizing faults with a logical information system

Participants: Peggy Cellier.

The following is a summary of the PhD thesis of Peggy Cellier [2], supervised by Mireille Ducassé, Olivier Ridoux, and Sébastien Ferré.

When testing a program, some executions can fail. Fault localization gives clues to locate the faults that cause those failures. The first contribution of this thesis is a new data structure for fault localization: a lattice that contains information from execution traces. The lattice is computed thanks to the combination of association rules and formal concept analysis, two data mining techniques. The lattice computes all differences between execution traces and, at the same time, gives a partial ordering on those differences. Unlike existing work, the method takes into account the dependencies between elements of the traces thanks to the lattice.

The second contribution of this thesis is an algorithm that traverses the lattice in order to locate several faults in one pass of a test suite of the program. Experiments show that while the method takes into account multiple faults, it is not penalized, compared to existing work, when the program contains only one fault (in terms of number of lines to inspect). In addition, the study of the impact on the method of the dependences between faults shows that in three out of the four identified cases of dependency the faults can be located.

The third contribution is an algorithm to compute association rules. The particularity of that algorithm is that it can take into account taxonomies, such as the hierarchy of the abstract syntax tree, without redundancy. It is used to generate association rules to build the lattice for fault localization.

6.10 Tracer driver and dynamic analyses of CLP(FD)

Participants: Mireille Ducassé.

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

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[7].

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

6.11 Using polyhedral abstractions in constraint-based testing

Participants: Mireille Ducassé.

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

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. 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.

This activity comes with the PhD of Tristan Denmat [Den08], supervised by Arnaud Gotlieb from the IRISA Lande team and Mireille Ducassé.

6.12 Assessment of achievements

Participants: All participants.

The results achieved by the LIS team must be compared with the key issues presented in the objective part. Not all key issues have deserved attention yet. However, a few of them have been sufficiently well explored to start and draw conclusions.

[Den08] T. Denmat, *Contraintes et abstractions pour la génération automatique de données de test*, PdD Thesis, Insa de Rennes, Juin 2008.

We have now gained sufficient experience to claim that even if every application uses a specific logic, it is not necessary to design every specific logic from scratch. Not only do we have proposed a toolbox of logic components for building logic tools (parser, prover, printer, etc), but we also have designed a theory of logic composition that allows to prove meta-properties about the prover thus obtained. This methodology has been successfully applied to build Description Logic style provers, topological property provers, time comparators, text comparators, etc. Moreover, we have design a non-commutative logic comment that allows to build variants of Lambek logics.

We have also progressed on the metaphor issue. We have observed that all applications we have designed require a three-components interface. One component exposes a query, another one exposes a navigation tree, and the last one exposes a set of application oriented entities represented using an application oriented interface. The second component provides at the same time a summary of the answers to the query, and a set of navigation links that lead to related queries. The third component can be a map display for GIS applications, a thumbnail array for a picture application, or an agenda for a personal organizer application. In all applications the three interface components must be tightly connected so that acting on one component effects the two other components. The three components must always be kept coherent.

Finally, we have demonstrated that LIS principles can be successfully applied to the GIS domain, and to the software fault localization domain. In both cases, LIS principles have permitted new functionalities, and have opened new perspectives on possible developments. The most interesting result is that the same high-level LIS features give rise to different capabilities in different domains. This shows that more than a set of information management principles, LIS is also an application integration principle. This is due to the use of logic for describing data and to the lack of rigid data schematas.

7 Other Grants and Activities

7.1 International Collaborations

- Daniela Bargelli, from McGill Canada, has been re-invited in the LIS team. The collaboration involves the pregroup formalism and the development of actual pregroup grammars for some chosen natural languages, French in particular. D. Bargelli has organized an ACFAS colloquium on Pregroup and Typed Grammars, in which A. Foret gave a talk. She also gave a talk based on [3] at the meeting "50 years from the Syntactic Calculus", organized by Claudia Casadio in Chieti (Italy).

- The team LIS collaborates with the Belgian company *Mission Critical IT* on automated debugging [7], as well as on the use of Camelis for the management of a library of software components.

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 (until september

2008), and P. Allard's PhD (since october 2008).

- Annie Foret is external collaborator of LINA (research lab. Nantes), in TALN team (Natural Language Processing), and member of "Agence Universitaire de la Francophonie" (AUF) , LTT network on "Lexicologie, terminologie et traduction".

8 Dissemination

8.1 Involvement in the Scientific Community

- Olivier Ridoux has served in the program committee of ICCS'08. He is in the editorial board of Interstices. Olivier Ridoux has been a member in several doctoral committees: Alexandre Vautier "Mining Data without Information on Knowledge Structure – Application to Network Intrusion Detection ", Tristan Le Gall "Abstract Lattices for the Verification of Systems with Queues and Stacks", Mathieu Petit "Test statistique structurel par résolution de contraintes de choix", Ali Choumane "Traitement générique des références dans le cadre multimodal parole-image-tactile", Pascal Sotin "Quantitative Aspects of Program Analysis" et Jean-Marie Mottu "Oracles et qualification du test de transformation de modèle"; and one Habilitation committee: Jacques Nicolas "Syntaxe, raisonnement et génomes". He also serves in AERES expert committees.

- Mireille Ducassé has served in the program committee of PADL'08 (International Symposium on Practical Aspects of Declarative Languages), San Francisco, USA. She has been in five PhD committees : Nicolas Vanderavo et Xavier Martin, Université Catholique de Louvain, Belgique ; Yohann Thomas, Rennes ; Hugo Venturini, Unversité de Grenoble and Jacques Saraydaryan, Lyon.

  She is an elected member of the "Conseil National des Université (CNU) 27e section", a national assessment committee for the teaching and research staff. This amounts for more than 5 weeks of full time work per year. In February 2008, she has represented the CNU 27 at the AERES assessment visit of the LORIA and INRIA laboratories in Nancy.

- Sébastien Ferré has been a member of the program committees of ICFCA'08 (Int. Conf. Formal Concept Analysis), CLA'08 (Concept Lattices and their Applications), and FIND'08 (Faceted Search and Dynamic Taxonomies). He has also been a member of the program committees of special editions of journals: IJCIS (Int. J. Computing and Information Sciences), IJGS (Int. J. General Systems). He also served as an external reviewer of the conferences CARI (Colloque Africain sur la Recherche and Informatique), and RFIA (Reconnaissance des Formes en Intelligence Artificielle). He has been named as a Program Chair of ICFCA'09. He is also a member of the PhD committee of Nicolas Lebreton.

- Annie Foret has been a program committee member of the *Formal Grammar* 2008 International Conference.

8.2 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 teaches an introduction to computability and complexity at the Licence level, which led to the publication of a book [1]. He also teaches an introduction to the principles of IT systems at the Licence level.

- Mireille Ducassé is the head of the computer science department of the INSA of Rennes. She is also an elected member of the board of directors ("Conseil d'administration") of the Insa of Rennes. She has been a member of three recruitment committees ("commission de spécialistes") in computer science: at the Insa of Rennes, at the University of Rennes 1 and at the University of "La Réunion".

  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 has also been co-responsible of the 1st year of a master in bioinformatics until september 2008. Since september, he is in delegation at the CNRS.

- Annie Foret teaches university courses including formal logic, functional programming, and databases.

- Peggy Cellier teaches databases at Licence 2 level, C language at Licence 3 level and data mining at Master 1 level of Insa. Since october, she is ATER (Attachée Temporaire à l'Enseignement et la Recherche) at University of Rennes 1. She teaches algorithmics of graphs at Master 1 level of DIIC (Diplôme d'Ingénieur de l'IFISIC), object-oriented modelling (UML, JUnit) at Master 1 level of IFSIC, and an introduction to computability and complexity at the Licence level.

- Olivier Bedel teaches programming applied to Geographical Information Systems at Master 2 students (Geography and Urban planning) at University of Rennes 2.

- Pierre Allard teaches programming in C 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", *Journal Recherches linguistiques de Vincennes*, 2006.

[2] D. Bechet, A. Foret, "k-Valued Non-Associative Lambek Grammars are learnable from Generalized Functor-Argument Structures", *Journal of Theoretical Computer Science 355*, 2, 2006.

[3] S. Ferré, R. D. King, "A dichotomic search algorithm for mining and learning in domain-specific logics", *Fundamenta Informaticae – Special Issue on Advances in Mining Graphs, Trees and Sequences 66*, 1-2, 2005, p. 1–32.

[4] 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, p. 191–215, 2002.

[5] S. Ferré, O. Ridoux, "An Introduction to Logical Information Systems", *Information Processing & Management 40*, 3, 2004, p. 383–419.

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

[7] 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, and P. Szeredi (editors), Springer-Verlag, Lecture Notes in Artificial Intelligence 3010, 2004.

[8] Y. Padioleau, O. Ridoux, "A Logic File System", *in : Usenix Annual Technical Conference*, 2003.

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

[10] B. Sigonneau, O. Ridoux, "Indexation multiple et automatisée de composants logiciels", *Technique et Science Informatiques 25*, 1, 2006.

Books and Monographs

[1] O. Ridoux, G. Lesventes, *Calculateurs, calculs, calculabilité*, *Sciences Sup*, Dunod, 2008.

Doctoral dissertations and "Habilitation" theses

[2] P. Cellier, *DeLLIS : Débogage de programmes par Localisation de fautes avec un Système d'Information Logique*, PhD Thesis, Université de Rennes 1, 2008.

Articles in referred journals and book chapters

[3] D. Bechet, A. Dikovsky, A. Foret, E. Garel, "Introduction of option and iteration into pregroup grammars.", *in : Computational Algebraic Approaches to Natural Language. Polimetrica Publisher, Italy, pp. 85-108.*, 2008.

[4] O. Bedel, S. Ferré, O. Ridoux, E. Quesseveur, "GEOLIS: A Logical Information System for Geographical Data", *Revue Internationale de Géomatique 17*, 3-4, 2008, p. 371–390.

[5] P. Cellier, S. Ferré, O. Ridoux, M. Ducassé, "A Parameterized Algorithm to Explore Formal Contexts with a Taxonomy", *Int. J. Foundations of Computer Science (IJFCS) 19*, 2, 2008, p. 319–343.

[6] S. Ferré, "Camelis: a logical information system to organize and browse a collection of documents", *Int. J. General Systems*, Accepted for publication.

[7] L. Langevine, M. Ducassé, "Design and Implementation of a Tracer Driver: Easy and Efficient Dynamic Analyses of Constraint Logic Programs", *Theory and Practice of Logic Programming, Cambridge University Press 8*, 5-6, Sep-Nov 2008, `http://arxiv.org/abs/0804.4116`.

Publications in Conferences and Workshops

[8] P. Allard, S. Ferré, "Dynamic Taxonomies for the Semantic Web", *in : DEXA Int. Work. Dynamic Taxonomies and Faceted Search (FIND)*, A. M. Tjoa, R. R. Wagner (editors), IEEE Computer Society, p. 382–386, 2008.

[9] D. Bechet, A. Dikovsky, A. Foret, E. Garel, "Optional and Iterated Types for Pregroup Grammars", *in : Int. Conf. Language and Automata Theory and Applications (LATA)*, C. Martín-Vide, F. Otto, H. Fernau (editors), *LNCS 5196*, Springer, p. 88–100, 2008.

[10] 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), *LNAI 4933*, Springer, p. 241–257, 2008.

[11] 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), *LNAI 4933*, Springer, p. 273–288, 2008.

[12] P. Cellier, "Formal concept analysis applied to fault localization", *in : Doctorial Symposium of the Int. Conf. on Software Engineering (ICSE)*, Robby (editor), ACM, p. 991–994, 2008.

[13] M. Ducassé, S. Ferré, "Fair(er) and (almost) serene committee meetings with Logical and Formal Concept Analysis", *in : Proceedings of the International Conference on Conceptual Structures*, P. Eklund, O. Haemmerlé (editors), Springer-Verlag, July 2008. Lecture Notes in Artificial Intelligence 5113.

[14] S. Ferré, "Agile Browsing of a Document Collection with Dynamic Taxonomies", *in : DEXA Int. Work. Dynamic Taxonomies and Faceted Search (FIND)*, A. M. Tjoa, R. R. Wagner (editors), IEEE Computer Society, p. 377–381, 2008.