In Memoriam: Paris C. Kanellakis

Our colleague and dear friend, Paris C. Kanellakis, died unexpectedly and tragically on December 20, 1995, together with his wife, Maria-Teresa Otoya, and their beloved young children, Alexandra and Stephanos. En route to Cali, Colombia, for an annual holiday reunion with his wife's family, their airplane strayed off course without warning during the night, minutes before an expected landing, and crashed in the Andes.

As researchers, we mourn the passing of a creative and thoughtful colleague who was respected for his many contributions, both technical and professional, to the computer science research community. As individuals, we grieve over our tragic loss—of a friend who was regarded with great affection, and of a happy, thriving family whose warmth and hospitality were gifts appreciated by friends around the world. Their deaths create for us a void that cannot be filled.

Paris left unfinished several projects, including a paper on database theory intended for this special issue of *Computing Surveys*, to be written during his holiday visit to Colombia. Instead, we wish to offer here a brief biography of Paris, and a description of the research topics that interested Paris over the last few years, together with the contributions that he made to these areas. It is not our intention to outline definitive surveys or histories of these areas, but rather to honor the significant contemporary research of our friend.

Paris was born in Greece in 1953 to Eleftherios and Argyroula Kanellakis. In 1976, he received the Diploma in Electrical Engineering from the National Technical University of Athens; his undergraduate thesis was titled *Easy-to-test Criteria for Weak Stochastic Stability of Dynamical Systems*, advised by Prof. E. N. Protonotarios. Paris continued his studies at the graduate level in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology, where he received his M. Sc. in 1978, submitting the thesis *Algorithms for a Scheduling Application of the Asymmetric Traveling Salesman Problem*, supervised by Profs. R. Rivest and M. Athans, followed by his Ph. D. in 1982; his doctoral dissertation was *On the Complexity of Concurrency Control for Distributed Databases*, supervised by Prof. C. H. Papadimitriou.

In 1981, Paris joined the Computer Science Department of Brown University as assistant professor. He was promoted to associate professor with tenure in 1986, and to full professor in 1990. Intermittent with his appointment at Brown, Paris also held several temporary positions, including posts at the IBM Watson Research Center, the MIT Laboratory for Computer Science, GIP Altaïr, and INRIA Rocquencourt. He served as an associate editor of the new journal *Constraints*, as well as *Information and Computation*, *ACM Transactions on Database Systems*, *SIAM Journal of Computing*, *Theoretical Computer Science*, and *Journal of Logic Programming.* Paris served in addition as program committee member, program chair, and invited speaker at many of the prominent research conferences in computer science.

We take this opportunity to present some of Paris' contributions to database theory, including deductive, object-oriented, and constraint databases, as well as his work in fault-tolerant distributed computation and in type theory. In each of these areas, we recognize research contributions that were not merely examples of good problem *solving*, but also examples of insightful problem *formulation.*

In synchrony with his technical ability in solving problems, Paris added a mature editorial voice which, by proposing new kinds of research questions, and answering some of them in novel and sometimes surprising ways, helped to change our perceptions of what was technically significant. In several cases—for example, in deductive databases and type theory—Paris brought the tools and techniques of complexity theory and algorithmics to analyze the efficiency of constructs in programming language design. These themes were found again in the area of constraint databases, an area he played a major role in initiating, while guiding its development via sound and feasible algorithmic principles. In distributed computing, Paris advanced new computational frameworks intended to align algorithmic paradigms with salient aspects of realizable system architectures. In object-oriented databases, Paris worked to build a semantic foundation that provides an implementation-independent meaning for these systems, much in the same spirit that the relational model provides an implementation-independent meaning for relational databases. We recognize in all this work Paris' desire to understand better the theoretical foundations of practical systems, to study them with precise analytical tools, and to use the results to improve the functionality and performance of these systems.

The authors of this technical obituary feel honored by the privilege they had in collaborating with Paris on many of these projects. In mourning his tragic death, we miss his technical facility, his broad knowledge, his insight, his commitment, and his humor. To write a research paper with Paris was also an opportunity to observe his indefatigable attention to detail, and to engage in vigorous debate with his editorial voice. To write this obituary allowed us to to feel his voice once more, to understand better what a good scientist he was, and to appreciate his uncommon decency and kindness. It is our great loss that we will not hear his voice again.

1 Deductive Databases

Paris Kanellakis was a major contributor to the theoretical foundations of deductive databases. It has been recognized since the early 1980s that first-order database query languages such as SQL are lacking in expressive power. This insight lead to the investigation of many higher-order query languages, in particular *Datalog*, the language of logic programs without function symbols. A canonical use of Datalog is to compute transitive closure, where we think of the database as a directed graph:

$$\begin{array}{lcl} path(X,Y) & :- & edge(X,Y). \\ path(X,Y) & :- & path(X,Z), path(Z,Y). \end{array}$$

In this example, we take *edge* to be an *extensional database (EDB) predicate*, representing basic facts stored in the database. For example, $edge(1,5)$ is an EDB fact stating that there is an edge between vertices 1 and 5. The *intensional database (IDB) predicate path* represents facts deduced from the database via the logic program above: the first rule says every directed edge forms a path, and the second rule tells how paths can be joined together. We can now query, for instance, $path(1,7)$ or $path(2,V)$ to determine, respectively, whether there is a path from vertex 1 to vertex 7, or what vertices V are connected to vertex 2 by a path. The *path* facts are *deduced* from the *edge* facts, hence the name *deductive databases*.

Paris' work addressed the problem of finding efficient evaluation methods for Datalog queries. He viewed the challenge of deductive databases as the need to combine the technology of logic programming with the efficiency of database technology, providing a concrete step towards a new generation of computing. The focus of his research in this area was in identifying classes of Datalog queries that can be evaluated efficiently.

Datalog and Parallel Computation: Paris investigated what kind of Datalog queries can be sped-up by massive parallelism [CK86, Kan88]. He identified speed-up with the complexity class NC, which consists of the problems that can be computed in polylogarithmic time through the use of polynomially bounded hardware. Problems in NC are exactly those with a great deal of potential parallelism. In contrast, significant speed-ups cannot be achieved for PTIME-complete problems, unless NC=PTIME, which is widely believed not to be the case. Thus, PTIME-complete problems are often called inherently sequential.

Paris proposed to measure the computational complexity of Datalog programs both by their time complexity as well as by their *database complexity*, which measures the number of calls the Datalog query engine makes to the underlying relational database system. He proved that there are Datalog queries that are hard to evaluate in parallel, regardless of which complexity measure is being used. For example, Paris showed that the query

$$\mathit{reach}(X) \quad \text{:-} \quad \mathit{reach}(Y), \mathit{reach}(Z), \mathit{edge}(Y,X), \mathit{edge}(X,Z), \mathit{edge}(Z,Y)$$

is PTIME-complete and, furthermore, its database complexity is provably super-polylogarithmic. The latter bound is significant, since it does not depend on whether NC is a proper subclass of PTIME. Paris also proved a *gap theorem* for the database-complexity measure. He showed that the database complexity of every Datalog query is either $O(1)$ or $\Omega(\log n)$; surprisingly, there is nothing in between.

Bounded vs. Unbounded Queries: It is clear that recursive applications of Datalog rules make queries hard to evaluate. In particular, Datalog queries whose database complexity is in $O(1)$ can be evaluated in NC; such queries are called *bounded.* It is known that a Datalog query is equivalent to a first-order query iff it is bounded. This makes it highly desirable to be able to identify which Datalog queries are bounded. Unfortunately, the distinction between bounded and unbounded queries can be quite subtle. For example, the query

$$\begin{array}{lll} \mathit{buys}(X,Y) & \text{:-} & \mathit{likes}(X,Y). \\ \mathit{buys}(X,Y) & \text{:-} & \mathit{trendy}(X), \mathit{buys}(Z,Y). \end{array}$$

is bounded, while the query

$$\begin{array}{lll} \mathit{buys}(X,Y) & \text{:-} & \mathit{likes}(X,Y). \\ \mathit{buys}(X,Y) & \text{:-} & \mathit{knows}(X,Z), \mathit{buys}(Z,Y). \end{array}$$

is unbounded. This subtlety is not accidental; it is known that the problem of testing whether a given Datalog query is bounded or not is undecidable.

Paris was engaged in a long-term project whose goal was to delineate the boundary between the decidable and undecidable for classes of Datalog queries [CGKV88, AK89, HKMV95], that is, to identify maximal classes of Datalog queries whose boundedness problem is decidable and minimal classes of Datalog queries whose boundedness problem is undecidable.

One way of classifying Datalog programs is by the *arity* of their IDB predicates. For example, the *path* and *buys* queries in the examples above are binary, while the *reach* query is unary. Together with colleagues, Paris was able to show that the boundary between the decidable and the undecidable lies between the unary and the binary. More precisely, the boundedness problem for unary Datalog queries is decidable [CGKV88], while the boundedness problem for binary Datalog queries in undecidable [HKMV95].

2 Object-Oriented Databases

In 1988-89, Paris visited INRIA Rocquencourt, where he held a joint position in the Database Research Group and in the Altaïr R&D Group. At that time, object-oriented database systems were emerging from research venues and into product development with start-up companies such as Altaïr. The new technology, however, lacked the formal foundations of the relational model. Paris' goal, while participating in development efforts, was to bridge this gap.

While at Altaïr, Paris had a major influence on several aspects of the O_2 system, bringing his theoretical expertise to a practical project. His most visible impact was on the data model: he helped formulate the O_2 data model, and provided a final theoretical framework for it. He was also the lead editor of the definitive monograph [BDK92] documenting the relevant design and analysis of this multi-year research and development effort.

One of Paris' ambitions was to provide sound theoretical foundations to database systems. He strongly believed that understanding the functionalities of these systems, providing formal models for them, and finding connections with already mature theories were the keys to developing better systems. In this enterprise, Paris could rely on a wide culture in theoretical computer science and mathematics, as well as a strong intuition for practical issues.

Paris developed an object-based database model in a collaborative work [AK89, AK91]. The structural part generalized most of the known complex-object data models. The main contribution is the operational part of the data model, the query language IQL, which uses object identities (oid's) for three critical purposes: (1) to represent data-structures with sharing and cycles, (2) to manipulate sets, and (3) to express any computable database query. The language IQL can be statically type checked, can be evaluated bottom-up and naturally generalizes most popular rule-based database languages. The model was also extended to incorporate type inheritance, without changes to the language.

The main purpose of this work was to capture formally the essence of object database application programming, and to highlight the new dimension brought by object identity with regard to old questions such as duplicate elimination or query completeness. In particular, Paris observed that standard proofs of query completeness (e.g., [CH85]) did not work as usual in this setting because of the presence of oid's. He showed that an analogous value-based data model could be founded on regular infinite trees, thereby capturing fundamental aspects of object identity that had been overlooked by previous researchers.

Paris was also concerned with the essential novelty of programming applications in an object-oriented paradigm. In [AKRW94, HKR94] he investigated, with collaborators, a simple programming formalism for object-oriented databases, namely *method schemas*. The syntax is based on an application of program schemas, using a hierarchy of classes, method composition, recursion via function calls and name overloading. The semantics is based on term rewriting with late binding.

Paris and colleagues concentrated on understanding the problem of consistency checking, i.e., of testing at compile time whether in some interpretation, a method invocation would lead to an error. Not surprisingly, the problem is undecidable in general. Paris studied in depth decidable cases such as monadic schemas (methods having arity 1) or recursion-free schemas (absence of cycles in the method dependence graph). Some surprising consequences of *covariance*, a standard restriction imposed on method definitions, were demonstrated: for an important class of monadic, recursion-free schemas, consistency checking is complete for NLOGSPACE, whereas it is complete for DLOGSPACE if covariance is also imposed.

Paris was convinced that object-oriented databases were an important technological step, but that they needed to rely on previously developed theories and techniques. He viewed his more recent work on type theory (see Section 5) as part of a general program to provide formal foundations for database programming languages.

3 Constraint Databases

Paris was one of the founders of the area of Constraint Databases [KKR95]. While visiting the IBM T. J. Watson Research Center, he was shown a demonstration of the CLP(R) system. This system is an instance of the CLP(X) framework, an extension of logic programming in which rules contain, besides normal terms, constraints from a domain X. CLP(R), in which rules contain constraints from the domain of real numbers, uses a combination of lazy evaluation and an appropriate constraint solver, together with standard techniques from logic programming. Paris immediately wondered whether a database theory could be developed for such systems, by analogy with the way deductive databases were inspired by logic programming.

The direct consequence of this idea was his collaborative work on Constraint Query Languages [KKR95]. A class of database models and query languages was defined that could be instantiated by specific constraint domains, similar to the CLP(X) framework. The basic idea was to replace the notion of tuple in a relational database by that of a *generalized tuple*, i.e., a conjunction of constraints. A relational tuple $(a_1, \ldots, a_n)$, for example, can be regarded as a special case of a generalized tuple, i.e., $(x_1 = a_1) \wedge \cdots \wedge (x_n = a_n)$. By choosing more powerful classes of constraints, one can represent potentially infinite sets of points in a compact way. For example, a rectangle with corners at (a, b) and (c, d) can be represented by the generalized tuple $(a \leq x \leq b) \wedge (c \leq y \leq d)$. Other example are linear arithmetic constraints, which can be used to model many spatial applications (e.g., GIS), and polynomial constraints, which can be used to describe more complex spatial objects, such as those in CAD systems.

Of course, just extending the database model would be not be very interesting unless we were able to query the database. Paris realized that for the model to be of interest we would need query languages that were not just decidable, but of low complexity. This research goal was addressed in [KKR95] for dense-order constraints, i.e., inequalities $x < y$ where $<$ is a dense order (e.g., the rationals). Some results in this direction have also been obtained for more powerful query classes such as linear constraints and polynomial constraints.

Paris' original paper showed that the first-order query language with dense-order constraints could be evaluated in LOGSPACE. Subsequent joint work [KG94] improved these bounds to AC^0. One consequence of this result was to resolve, for dense-order constraints, a problem that Paris had proposed for first-order constraint query languages in general: whether the PARITY query—is the cardinality of a (finite) database even or odd?—is expressible in such languages. His conjecture was that the answer is negative, as is well-known to be the case for relational databases. This question turned out to be a very difficult: for polynomial constraints the negative result was only obtained recently [BDLW96] using sophisticated techniques from non-standard model theory.

The AC^0 result referred to above was obtained by defining an algebra for dense-order constraints and then analyzing its complexity. In one of his very last papers, [GK96], Paris extended this work and studied algebras for constraint databases in more detail. This research included further work on dense-order constraints (including update issues) and linear arithmetic constraints. In the latter case, he defined a particularly promising algebra for 2-variable constraints—for example, temporal constraints.

One of Paris' goals was to include recursion (e.g., transitive closure) in the model. Unfortunately, languages more powerful than dense-order constraints were not closed under recursion. For example, if $R(x, y)$ contained the generalized tuple $y = 2x$, the transitive closure of R would have to contain all tuples of the form $y = 2^n x$, for $n \geq 1$. On the other hand, it was possible to write recursive programs that did not have this problem, intuitively by ensuring that the rules one wrote did not create new objects. Defining such a notion in a precise way was an abiding interest of Paris'—unfortunately, it remained an uncompleted project.

Paris was always concerned about the practical side of the field, and was aware of the risk of it becoming a theoretical exercise with limited practical value. From the very beginning he was interested in the question of appropriate index structures for constraint databases, i.e., how to store constraints so that they could be accessed efficiently. He spent an extended visit to IBM studying the state of the art in index structures for spatial databases, and how applicable they were to constraints. He was immediately struck by the contrast between the elegant combination of theory and practice in the original B-tree paper, and the lack of such an analysis for spatial index structures. The result of this study was the paper [KRVV94], proposing a data structure for indexing constraints on one attribute. This index structure has optimal worst-case storage and query performance, and optimal amortized insert time (i.e., averaged over a sequence of inserts), with performance very close to that of the B-tree. By studying the techniques that he had used for indexing constraints [RK95], Paris was also able to develop index methods for object-oriented databases, and at the time of his death he was investigating their application to temporal databases.

4 Fault-Tolerant Parallel Computation

Paris Kanellakis, among other researchers, sought to bridge the gap between abstract models of parallel computation and realizable parallel architectures. The parallel random access machine (PRAM) model attracted significant attention, and numerous efficient parallel algorithms were developed for the model. The PRAM model elegantly combines the power of parallelism and the simplicity of the random access machine (RAM) model. Most parallel algorithms require a fault-free environment, where any unreliability of processors, interconnections, or synchronizations either eliminate efficiency, or result in incorrect computation. Paris proposed a formally defined notion of *robustness* that combines fault-tolerance and efficiency, and he led the development of deterministic parallel algorithms that remain efficient in the presence of arbitrary dynamic processor failure patterns. This work and relevant open problems were recently summarized in [KMS94, KS94].

Robust computation: The ultimate goal of this reasearch is the synergy between the *speed-up* potential of parallel computation and the *reliability* potential of distributed computation. Paris investigated fault models and parallel computation models where it is possible to achieve algorithmic efficiency (i.e., speed-ups close to linear in the number of processors) despite the presence of faults. Such combinations of fault and computation models illustrate constructive trade-offs between reliability and efficiency. This trade-off exists because reliability usually requires *introducing redundancy* in the computation in order to detect errors and reassign resources, whereas gaining efficiency by massively parallel computing requires *removing redundancy* from the computation to fully utilize each processor. Even allowing for some abstraction in the model of parallel computation, it is not obvious that there are any non-trivial fault models that allow near-linear speed-ups, especially considering the generally intimidating impossibility results for distributed and parallel computation. Thus it was rather surprising when Paris demonstrated in collaborative work [KS92] that it is possible to combine efficiency and fault-tolerance for many basic algorithms, expressed as concurrent-read concurrent-write (CRCW) PRAMs, in a fault model [KS92] allowing any pattern of dynamic fail-stop processor errors, as long as one processor remains alive. The approach and techniques pioneered by Paris in his work were shown to be readily extendible to all CRCW PRAMs. In effect, any PRAM algorithm can be made robust by efficiently simulating a fault-free machine on a fault-prone one. Papers by other researchers followed, pursuing similar problems in related shared-memory and message-passing models. It was also demonstrated by Paris, in collaborative work, that although optimal simulations are possible in some models, such oblivious simulations are not necessarily as efficient as "handcrafted" robust algorithms, e.g., for the all-important pointer doubling and parallel prefix algorithms [KS94].

Paris and colleagues extended the fault model to include *processor restarts* [KS91], *arbitrary initial memory initialization* [KS94], and *restricted memory-access* patterns through controlled memory access [KMS95].

Algorithms and lower bounds: A key primitive in the above work is the *Write-All* operation [KS92], defined as follows: *using P processors, write 1s into all locations of an array of size N, where $P \leq N$.* When $P = N$ this operation captures the computational progress that can be naturally accomplished in one time unit by a PRAM. Iterated *Write-All* forms the basis for the algorithm simulation techniques. Therefore, improved *Write-All* solutions lead to improved simulations of *all* parallel algorithms. Under dynamic failures, efficient deterministic solutions to *Write-All*, i.e., increasing the fault-free $O(N)$ work by small polylog(N) factors, are non-obvious. The first such solution, proposed by Paris in joint work [KS92], was an algorithm having worst-case work bound $O(N + P \log^2 N / \log \log N)$.

Memory-access concurrency is a major source of available redundancy in parallel algorithms, and deterministic robust (i.e., efficient and fault-tolerant) computation is impossible when concurrent writes are excluded [KMS94, KS92]. Paris believed that it should nevertheless be possible to limit the significant memory-access concurrency that was assumed by existing robust algorithms. Again, surprisingly, in [KMS95] he showed, with his colleagues, that concurrent writes are necessary only when faults actually occur and not in the anticipation of possible faults. *Write-All* algorithms can be constructed so that they can be executed on a fault-free exclusive-write machine, while on a fault-prone concurrent-write machine the number of concurrent writes is exactly the number of processor failures encountered during the execution.

The *Write-All* algorithms for the fail-stop model have optimal ranges of processors for which the work is $O(N)$. Optimality is achieved by taking advantage of parallel slackness. Are there optimal algorithms for the full range of processors $N = P$? Paris et al. showed that such algorithms do not exist [KS92]: even for the models with instant memory snapshot and arbitrarily powerful instruction set, an adversary can be constructed that will force any *Write-All* algorithm to do $\Omega(N \log N / \log \log N)$ work. For memory snaphosts this bound is tight—there is a simple algorithm with a matching upper bound [BKRS95].

While the current *Write-All* lower/upper bound gap is relatively small for the fail-stop model with dynamic failures, a much larger gap remains for the models with restarts and asynchrony. It was conjectured that for *Write-All* there is a quadratic lower bound in this case. Together with several colleagues [BKRS95], Paris constructed the first subquadratic deterministic algorithm with work $O(N \cdot P^{\log \frac{3}{2}})$. In the same paper a $\Omega(N \log N)$ lower bound was shown for the model. The bound stands even if memory snapshots are allowed, but in this case there is also a matching upper bound.

Paris considered the remaining gaps between lower and upper bounds to be interesting open problems, and intended to pursue their resolution. Left unfinished was a jointly-authored monograph bringing together the latest results in this area.

5 Type theory

Paris Kanellakis made fundamental contributions to the algorithmic analysis and complexity-theoretic understanding of several important topics in programming language design, including first- and higher-order unification, type inference for ML-like programming languages, and expressiveness in the typed lambda calculus. Paris' interests in these areas were natural extensions of his earlier work in logic programming and database theory. The work on these topics combined a technical expertise with a careful, understated iconoclasm: he wanted, and succeeded, in changing his colleagues' perceptions of the relevant states of the art.

First order unification: Unification is a ubiquitous building block in implementations of sophisticated programming languages. It is, for example, the workhorse of logic programming engines, and an essential component of compile-time type analysis. The dual emergence in the 1980s of logic programming and parallel computation, in both research and development, suggested to many computer scientists that parallel unification could greatly enhance the performance of logic programming systems. In this context, Paris' collaborative result [DKM84] that first-order unification is *complete for* PTIME—in other words, as hard as any polynomial-time problem—was an interesting technical result with an important editorial message.[1]

The theorem implied that unification was a formidable, if not essential, bottleneck in logic programming: any research program that succeeded in parallelizing unification—say, to run in sub-polynomial time—would have also succeeded in equivalently parallelizing *every* known polynomial time algorithm. Because of this seemingly insurmountable (or as most researchers believe that PTIME $\neq$ NC, impossible) hurdle, serious attempts to parallelize unification were largely put aside.

The key idea in this proof of the "linear nature" of unification, due to Paris, is that very simple unification problems can be constructed whose solution is isomorphic to the computation of Boolean logic gates: Boolean values are simulated by pairs of terms that are forced to unify iff the related value is "true." Since PTIME can be captured by certain easily-constructed polynomial-sized circuits, this technology permitted any decision problem solvable in polynomial time to be efficiently transformed into a unification problem. These ideas were effectively reused to analyze the complexity of type inference.

Complexity of type inference: Type checking is an important safety feature of programming languages ensuring that no run-time type errors cause programs to "go wrong" via type errors, e.g., adding pointers and strings. Type *inference* incorporates type checking with automatic *compile-time* mechanisms that deduce the types of all run-time data. Unification is essential to many type inference algorithms, since first-order terms can characterize data types; terms are built via functions like "pair," "list," or "function" over constants such as "integer," "boolean," or *type variables* having values constrained by term equations.

Many functional programming languages (e.g., ML, Haskell, Miranda) based on typed lambda calculus support type inference in the presence of so-called *parametric polymorphism.* In simpler terms, these combined features enable programmers to code implementations of abstract data

[1]The paper in which this result appeared was edited by the founding editor of the *Journal of Logic Programming*, J. Alan Robinson, and was published as the very first article in that journal.

types (trees, stacks, queues, etc.) once, and "reuse" the code on data of different types, without the obligation of adding type annotation at each use. By comparison, other languages with compile-time type checking—the canonical example being Pascal—require a procedure to be repeatedly coded for each data type at which it is used; the type checker requires redundant source code, each copy with different type annotation, even though identical target code is generated for each copy.

Although claims had been made in the research community that the ML type inference mechanism was efficient, Paris believed that such claims were unfounded. In a striking joint paper [KM89], the simpler problem of recognizing type-correct ML programs was shown to be PSPACE-hard (i.e., as hard as any problem that can be solved in polynomial space), and furthermore contained in EXPTIME. This result was both counterintuitive and surprising. The fundamental technology was extended via additional insights in another joint work [KMM91], showing the problem to be complete for EXPTIME.

The proof technology used in these theorems was a direct descendant of Paris' earlier work on unification. The additional type flexibility introduced by ML polymorphism allows the simulation, via unification, of more powerful complexity classes. In Paris' joint paper [KM89], ML polymorphism was used as an iterative mechanism to simulate quantifier elimination for Quantified Boolean Formulas,[2] resulting in the PSPACE-hardness bound, and in [KMM91] the same tools were enhanced to simulate arbitrary exponential-time computation. Later sophisticated extensions of his idea yielded significant lower bounds for other typed lambda calculi.

Expressibility: Paris was also interested in characterizing complexity classes (AC^0, PTIME, PSPACE, EXPTIME, EXPSPACE,...) by naturally defined classes of typed lambda terms, mirroring earlier work that had been done in the database community on logic and expressibility [Imm86, Var82]. Lambda calculus is the underlying theory of all functional programming languages, though its roots are found in the modern development of mathematical logic. Paris' work in this area was designed to provide foundations for functional database query languages, and also to rehabilitate the simply typed lambda calculus from a certain computational disrepute.

Interest in higher-order typed lambda calculi had been motivated in part by negative results that the only expressible functions on the canonical type for integers were the *extended polynomials,* made from addition, multiplication, and conditional equality with zero—noticeably absent is exponentiation, subtraction, and integer equality. These results suggested that the calculus was no good for useful computation.

Paris wanted to show that this was not the case: all feasible computation was comfortably found within the simply typed lambda calculus. The syntactic and (especially) semantic contortions of higher-order calculi, of certain mathematical interest, could then be shown to be computationally unnecessary. The means to this demonstration was a shift in the computation paradigm: instead of considering numerical computations, Paris suggested coding database queries. His suggestion was inspired by a lambda calculus encoding [Mai92] of the *decision problem for higher-order logic*, which Paris proposed to extend to an encoding of the *complex object algebra* [AB88], a powerful database query language. We briefly discuss some relevant details.

Quantified Boolean Formulas allows quantification only over Booleans, but can be generalized by allowing quantification over iterated powersets of Booleans, and replacing propositional variables by prime formulas $x \in y$, where x and y are typed to range over the appropriate powerset. This *decision problem for higher-order logic* has nonelementary complexity (not solvable in any

[2] Given a closed propositional formula F where each variable X is $\forall$- or $\exists$-bound, is F true under the naive interpretation?

fixed stack of exponentials), and can be used to show that deciding equivalence of two typed lambda terms is nonelementary [Mey74, Sta79]. In the related complex object algebra, quantification ranges over atomic constants, sets, and tuples, expressing exactly the generic elementary database queries that are computable in some fixed stack of exponentials. By implementing the complex objects algebra in the simply typed lambda calculus, the data complexity of logical queries, studied in [Imm86, Var82], could be replaced by a *procedural* variant. In this procedural implementation, the computational power of a query is measured by its integer *rank*, describing the degree of its higher-order functionality. In joint work, Paris began implementation of this research program [HKM93, HK94], where PTIME was given a precise characterization via query terms of fixed rank.

In further sophisticated extensions of this functional framework for descriptive computational complexity, Paris succeeded, again in collaborative work, in providing syntactic characterizations for many standard complexity classes [HK95]. These results rely on complex query evaluation strategies that use efficient data structures, and avoid the redundancies incurred by more usual reduction strategies in the lambda calculus. At the time of his death, Paris was considering the use of this technical machinery in providing an alternative, and perhaps more compelling, treatment of the issues addressed by the area of *optimal reduction* in the lambda calculus.

SERGE ABITEBOUL,
Stanford University and
Institut National de Recherche en Informatique et Automatique (INRIA)

GABRIEL M. KUPER,
European Computer-Industry Research Centre (ECRC)

HARRY G. MAIRSON,
Brandeis University

ALEXANDER A. SHVARTSMAN,
Massachusetts Institute of Technology

MOSHE Y. VARDI,
Rice University

References

Deductive Databases

[AK89] S. Abiteboul and P. C. Kanellakis. *Deciding Bounded Recursion in Database Logic Programs.* **SIGACT News** 20:4, Fall 1989.

[CGKV88] S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, and M. Y. Vardi. *Decidable Optimization Problems for Database Logic Programs.* **1988 ACM Symposium on Theory of Computing**, pp. 477–490.

[CK86] S. S. Cosmadakis and P. C. Kanellakis. *Parallel Evaluation of Recursive Rule Queries.* **1986 ACM Symposium on Principles of Database Systems**, pp. 280–293.

[HKMV95] G. G. Hillebrand, P. C. Kanellakis, H. G. Mairson, and M. Y. Vardi. *Undecidable Boundedness Problems for Datalog Programs.* **J. Logic Programming** 25 (1995), pp. 163–190. (A preliminary version of this paper, *Tools for Datalog Boundedness*, appeared in the **1991 ACM Symposium on the Principles of Database Systems**, pp. 1–13.)

[Kan88] P. C. Kanellakis. *Logic Programming and Parallel Complexity.* In **Foundations of Deductive Databases and Logic Programming**, (ed. J. Minker), Morgan Kaufmann, 1988, pp. 547–585.

[Kan90] P. C. Kanellakis. *Elements of Relational Database Theory.* In **Handbook of Theoretical Computer Science**, (ed. A.R. Meyer, M. Nivat, M.S. Paterson, D. Perrin and J. van Leeuwen), vol. B, Chapter 17. North Holland, 1990, pp. 1075–1144.

Object-Oriented Databases

[AK89] S. Abiteboul and P. C. Kanellakis. *Object Identity as a Query Language Primitive.* **J. of the ACM**, to appear. Full version available as Brown Univ. Tech. Rep. CS-89-26.

[AK91] S. Abiteboul and P. C. Kanellakis. *The Two Facets of Object-Oriented Data Models.* **IEEE Data Engineering Bulletin** 15:2 (1991), pp. 3–8.

[AKRW94] S. Abiteboul, P. C. Kanellakis, S. Ramaswamy, and E. Waller. *Method Schemas.* **J. of Computer and System Sciences**, to appear.

[BDK92] F. Bancilhon, C. Delobel, and P. C. Kanellakis. **Building an Object-Oriented Database System: The Story of O2.** Morgan Kaufmann, 1992.

[CH85] A. K. Chandra and D. Harel. *Horn Clause Queries and Generalizations.* **J. Logic Programming** 2 (1985), pp. 1–15.

[HKR94] G. G. Hillebrand, P. C. Kanellakis, and S. Ramaswamy. *Functional Programming Formalisms for OODB Methods.* In **Advances in Object-Oriented Databases**, (ed. A. Dogac, M. Tamer Özsu, A. Biliris, and T. Sellis). Springer-Verlag, Series F (Computer and Systems Sciences), vol. 130, Ch. 2.3, pp. 73–98, 1994.

Constraint Databases

[BDLW96] M. Benedikt, G. Dong, L. Libkin, and L. Wong. *Relational Expressive Power of Constraint Query Languages.* Manuscript, 1995.

[GK96] D. Q. Goldin and P. C. Kanellakis. *Constraint Query Algebras.* Manuscript, 1996.

[KG94] P. C. Kanellakis and D. Q. Goldin. *Constraint Programming and Database Query Languages.* Invited paper, **Symposium on Theoretical Aspects of Computer Software.** Springer LNCS 789, pp. 96–120, Sendai Japan, April 1994. (Note: Full version invited to the first issue of the new journal **Constraints**).

[KKR95] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. *Constraint Query Languages.* **J. Computer and System Sciences** (special issue, ed. Y. Sagiv) 51:1 (1995), pp. 26–52. (An extended abstract appeared in **1990 ACM Symposium on the Principles of Database Systems**, pp. 299–314.)

[KRVV94] P. C. Kanellakis, S. Ramaswamy, D. E. Vengroff, and J. S. Vitter. *Indexing for Data Models with Constraints and Classes.* J. Computer and System Sciences, to appear (special issue ed. C. Beeri). (An extended abstract appeared in **1993 ACM Symposium on the Principles of Database Systems**, pp. 233–243.)

[RK95] S. Ramaswamy and P. C. Kanellakis. *OODB Indexing by Class-Division.* in **1995 ACM Symposium on the Management of Data**, pp. 139–150.

Fault-Tolerant Parallel Computation

[BKRS95] J. Buss, P. C. Kanellakis, P. Ragde, and A. A. Shvartsman, *Parallel Algorithms with Processor Failures and Delays.* **J. Algorithms**, to appear.

[KMS94] P. C. Kanellakis, D. Michailidis, and A. A. Shvartsman. *Concurrency = Fault-Tolerance in Parallel Computation.* Invited paper, **1994 International Conference on Concurrency Theory**. Springer LNCS vol. 836, pp. 242-266.

[KMS95] P. C. Kanellakis, D. Michailidis, and A. A. Shvartsman. *Controlling Memory Access Concurrency in Efficient Fault-Tolerant Parallel Algorithms.* **Nordic Journal of Computing** 2 (1995), pp. 146-180. (A preliminary version appeared in **1993 International Workshop on Distributed Algorithms**, pp. 99–114.)

[KS92] P. C. Kanellakis and A. A. Shvartsman. *Efficient Parallel Algorithms Can Be Made Robust.* **Distributed Computing** 5:4 (1992), pp. 201–217. (A preliminary version appeared in **1989 ACM Symposium on Principles of Distributed Computing**, pp. 211–222.)

[KS91] P. C. Kanellakis and A. A. Shvartsman. *Efficient Parallel Algorithms On Restartable Fail-Stop Processors.* **1991 ACM Symposium on Principles of Distributed Computing**, pp. 23–37.

[KS94] P. C. Kanellakis and A. A. Shvartsman. *Fault-Tolerance and Efficiency in Massively Parallel Algorithms.* In **Foundations of Dependable Computing – Paradigms for Dependable Applications** (ed. G. M. Koob and C. G. Lau). Kluwer, 1994, pp. 125–154.

Type Theory

[AB88] S. Abiteboul and C. Beeri. *On the Power of Languages for the Manipulation of Complex Objects.* INRIA Research Report 846, 1988.

[DKM84] C. Dwork, P. Kanellakis, and J. C. Mitchell. *On the Sequential Nature of Unification.* **Journal of Logic Programming** 1:35–50, 1984.

[HK94] G. G. Hillebrand and P. C. Kanellakis. *Functional Database Query Languages as Typed Lambda Calculi of Fixed Order.* **1994 ACM Symposium on Principles of Database Systems**, pp. 222–231.

[HK95] G. G. Hillebrand and P. C. Kanellakis. *On the Expressive Power of Simply-Typed and Let-Polymorphic Lambda Calculi.* Manuscript.

[HKM93] G. G. Hillebrand, P. C. Kanellakis, and H. G. Mairson. *Database Query Languages Embedded in the Typed Lambda Calculus.* **Information and Computation**, to appear. (A preliminary version appeared in the **1993 IEEE Symposium on Logic in Computer Science**, pp. 332–343.)

[HKM94] G. G. Hillebrand, P. C. Kanellakis, and H. G. Mairson. *An Analysis of Core-ML: Expressive Power and Type Inference.* **1994 International Colloquium on Automata, Languages and Programming**. Springer LNCS 820, pp. 83–105.

[Imm86] N. Immerman. *Relational Queries Computable in Polynomial Time.* **Information and Computation** 68 (1986), pp. 86–104.

[KMM91] P. C. Kanellakis, H. G. Mairson, and J. C. Mitchell. *Unification and ML Type Reconstruction.* In **Computational Logic: Essays in Honor of Alan Robinson**, ed. J.-L. Lassez and G. D. Plotkin. MIT Press, 1991, pp. 444-478.

[KM89] P. C. Kanellakis and J. C. Mitchell. *Polymorphic Unification and ML Typing.* **1989 ACM Symposium on the Principles of Programming Languages**, pp. 105–115.

[Mai92] H. G. Mairson. *A Simple Proof of a Theorem of Statman.* **Theoretical Computer Science** 103:2 (September 1992), pp. 387–395.

[Mey74] A. R. Meyer. *The Inherent Computational Complexity of Theories of Ordered Sets.* **Proceedings of the International Congress of Mathematicians**, pp. 477–482, 1975.

[Sta79] R. Statman. *The Typed λ-Calculus is not Elementary Recursive.* **Theoretical Computer Science** 9 (1979), pp. 73–81.

[Var82] M. Y. Vardi. *The Complexity of Relational Query Languages.* **1982 ACM Symposium on Theory of Computing**, pp. 137–146.