Many-sorted First-Order Logic Database Language

J. S. H. YANG*, Y. H. CHIN† AND C. G. CHUNG*

* *Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, 30050, R.O.C.*

† *Institute of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, 30043, R.O.C.*

A database languages based on Many-Sorted First-Order Logic (MSFOL) have many advantages over one based on One-Sorted First-Order Logic (OSFOL). The advantages includes ease-of-expressiveness, efficiency, and the abstraction mechanism. Many database researchers have used OSFOL to view the Relational Data Model (RDM); however, no RDM has been modelled by MSFOL. This paper first gives a formal definition for MSFOL and then its advantages of expressiveness and of abstraction are illustrated. Two reduction algorithms which can transform an MSFOL-based language into/from an OSFOL-based language are given. The semantic equivalence between languages based on MSFOL and Typed OSFOL is also proved. Recent extensions of RDMs require aggregation, classification, and generalisation/specialisation mechanisms which MSFOL-based languages can provide, but OSFOL-based languages cannot.

Received January 1989, revised August 1989

1. INTRODUCTION

Many researchers[1,2] use OSFOL to interpret the RDM[3] from the model-theoretic viewpoint. In particular, Reiter[4,5] added some enhancements by using *type predicates* with OSFOL (hereafter, called Typed OSFOL) to view the RDM. Pirotte[6] mention that MSFOL can be used to view the RDM, but they did not describe how it could be used, none did, they describe its advantages in relation to OSFOL.

A many-sorted logic-based language is easy to use because of its expressiveness and more efficient than a One-sorted logic-based language. The *sort structuring* of MSFOL can provide abstraction mechanisms, such as *classification* and *generalisation*.[7] The concept of such mechanisms, adopted from Artificial Intelligence, enhances the semantic capabilities of the RDM. The major contributions of this paper are (1) to illustrate the differences between these two logics; (2) to give two syntactic reduction algorithms between the wffs of an MSFOL-based language and a Typed OSFOL-based language; (3) to prove that these two types of languages are semantically equivalent, and (4) to describe the advantages of using an MSFOL to view the RDM and extended RDM.

In Section 2, an example is presented for the purpose of illustration. Section 3 reviews the model-theoretic view of the RDM by using OSFOL and Typed OSFOL. Section 4 gives a formal definition of MSFOL and compares the definition with that of OSFOL. Section 5 gives the two reduction algorithms and proves their correctness. Section 6 explains the advantages of an MSFOL-based language. Finally Section 7 summarises the results of the paper and discusses the possible extensions of the logic.

2. RELATIONAL DATABASE EXAMPLES

A library application in Table 1 is used as an illustration. Domain names such as NAME, BOOK_NAME, ... divide the library application universe into different domains. Relations are defined as the aggregation[7] of the column names of LIBRARY_NAME, LOCATION, ... One or more column names can be defined over the same domain. They are aliases of the defined domains. The range of those column names is the same. This two-level abstraction, domain and relation, provides a logical representation distinct from the physical implementation. However, the modelling capabilities of the RDM are still limited by the tuple-oriented structure.

3. RELATIONAL DATABASE VIEWED THROUGH ONE-SORTED FIRST-ORDER LOGIC

In this section, previous approaches of using OSFOL and Typed OSFOL to view the RDM are illustrated. The abstraction capabilities are examined by using the example given in Section 2.

3.1 Model-theoretic view of using OSFOL

From the viewpoint of OSFOL (without function symbols), the RDM[3] is a first-order formal system, and the relational calculus is based on the first-order predicate calculus.[6] A relational database (with the schema and instances) is viewed as a first-order interpretation with respect to model-theory.[1,5] The union of all domains is the universe D. Column names are variables. Relation names are predicates. Tuples (instances) of relations are the ground wffs, and relational queries correspond to the open wffs of OSFOL. Query evaluation is the process of truth functional evaluation of the first-order open wffs with respect to an interpretation. With the accommodation of integrity constraints (which are closed wffs), an interpretation should be a *model* of the integrity constraints. Some assumptions (implicit rules) of RDMs such as the Closed World Assumption, Unique Name Assumption, ...etc.,[5] can also be represented by first-order closed wffs.

The following examples are the wffs corresponding to the queries and constraints in Table 1.

In these examples, all the variables/constants are defined in one universe. Consequently, we only use the variable/constant and the predicate to represent the meaning of the query. for instance, the second query of Table 2, 'Get the BOOK_TITLE of BOOKs BORROWed by John', is represented by the wff (BOOK(x,_,_,_,_,_)|BORROW(John,x,_)), in which

Table 1. A library application relational data model

(1) Relational Database Scheme

```
DOMAINS:
  NAME CHAR(20), PLACE_NAME CHAR(20), BOOK_NAME CHAR(60),
  PERSON_NAME CHAR(20), YEAR NUMBER(4), AMOUNT INTEGER(2),
  ID NUMBER(6), DEPART_CODE CHAR(6), LEVEL_CHAR(2),
  PLACE_CODE CHAR(8), DATE NUMBER(2)"/"NUMBER(2)"/"NUMBER(2).
RELATIONS:
  LIBRARY(LIBRARY_NAME: DOMAIN NAME, LOCATION: DOMAIN PLACE_NAME)
  BOOK(BOOK_TITLE: DOMAIN BOOK_NAME, AUTHOR: DOMAIN PERSON_NAME,
      PUBLISHER: DOMAIN NAME, YEAR: DOMAIN YEAR,
      LIBRARY_NAME: DOMAIN NAME, PLACE: DOMAIN PLACE_CODE,
      #_OF_COPIES: DOMAIN AMOUNT)
  BORROWER(BORROWER_NAME: DOMAIN PERSON_NAME,
      #_OF_BOOKS: DOMAIN AMOUNT)
  STUDENT(ID: DOMAIN ID, STUDENT_NAME: DOMAIN PERSON_NAME,
      DEPART: DOMAIN DEPART_CODE, LEVEL: DOMAIN LEVEL_CODE)
  FACULTY(ID: DOMAIN ID, FACULTY_NAME: DOMAIN PERSON_NAME,
      DEPART: DOMAIN DEPART_CODE)
  BORROW(BOOK_TITLE: DOMAIN BOOK_NAME,
      BORROWER_NAME: DOMAIN PERSON_NAME, DATE: DOMAIN DATE).
```

(2) Queries

1. Find BORROWER_NAME of BORROWERs who BORROWed more than 3 BOOKs.
2. Get the BOOK_TITLE of BOOKs BORROWed by John.
3. Find the BOOK_TITLE and BORROWER_NAME of BOOKs that are BORROWed by some AUTHOR.
4. Find the BOOKs whose AUTHOR is also the PUBLISHER.

(3) Constraints

1. A BORROWER cannot BORROW more than 5 BOOKs.
2. A BORROWER must be either a STUDENT or a FACULTY.

John is a constant of the column name BORROWER_NAME, x is a variable for BOOK_TITLE ('_' means that we do not care about the column value.). The meaning of variables/constants and their corresponding column names cannot be expressed in one-sorted logic. For example, if we present as BORROW(BORROWERNAME(John),BOOKTITLE(x)), then this is a second-order formula. Also, those variables/constants, in the view of OSFOL, are ranged in the only sort, the universe; hence the search space of each variable/constant is bigger.

3.2. Typed One-Sorted First-Order Logic

In Ref. 4, a Typed One-sorted First-order Logic (typed OSFOL) is defined. Except for a class of unary predicate symbols, typed OSFOL is the same as OSFOL. These unary predicates are called simple types. Other types can be created from simple types by using the operators $\vee$, $\wedge$ and $\neg$. Type predicates are used to restrict the range of variables/constants. Typed wffs are also defined as the wffs of OSFOL, except that all the variables are typed.

The following example represents the queries and constraints of Table 1 in a Typed OSFOL-based language.

Type predicates can restrict the range of variables/constants in the formula. For instance, in the first query, the unary predicate BORROWER_NAME(x) restricts the usage of variable x in the domain of BORROWER_NAME. Such a restriction not only improves the efficiency of a language by reducing the search space of a variable, but also provides a mechanism for expressing the *classification* (Ref. 7) of a variable. According to Ref. 4, complex type predicates can be

Table 2. Queries and constraints of Table 1 represented in an OSFOL-based language

(1) Queries

1. Find BORROWER_NAME of BORROWERs who BORROWed more than 3 BOOKs.
   $(\mathrm{BORROWER}(x, \_) \mid (\exists y)(\mathrm{BORROWER}(x, y) \wedge y > 3))$
2. Get the BOOK_TITLE of BOOKs BORROWed by John.
   $(\mathrm{BOOK}(x, \_, \_, \_, \_, \_, \_) \mid \mathrm{BORROW}(\mathrm{John}, x, \_))$
3. Find the BOOK_TITLE and BORROWER_NAME of BOOKs that are BORROWed by some author.
   $((\mathrm{BOOK}(x, \_, \_, \_, \_, \_, \_), \mathrm{BORROWER}(y, \_)) \mid \mathrm{BORROW}(y, x, \_))$
4. Find the BOOKs whose AUTHOR is also the PUBLISHER.
   $(\mathrm{BOOK}(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \mid x_2 = x_3)$

(2) Constraints

1. A BORROWER cannot BORROW more than 5 BOOKs.
   $(\forall x)\ (\forall y)\ (\mathrm{BORROWER}(x, y) \rightarrow y \leqslant 5)$
2. A BORROWER must be either a STUDENT or a FACULTY.
   $(\forall x)\ (\exists y)\ (\exists z)\ (\mathrm{BORROWER}(x, \_) \rightarrow (\mathrm{STUDENT}(\_, y, \_, \_) \wedge x = y) \vee (\mathrm{FACULTY}(\_, z, \_) \wedge x = z))$

Table 3. Queries and Constraints of Table 1 represented in a Typed OSFOL-based language

(1) Queries

1. Find BORROWER_NAME of BORROWERs who BORROWed more than 3 BOOKs.
   $(\mathrm{BORROWER\_NAME}(x) \mid (\exists y)\ (\#\_\mathrm{OF\_BOOKS}(y) \wedge \mathrm{BORROWER}(x, y) \wedge y > 3))$
2. Get the BOOK_TITLE of BOOKs BORROWed by John.
   $(\mathrm{BOOK\_TITLE}(x) \mid \mathrm{BORROWER\_NAME}(\mathrm{John}) \wedge \mathrm{BORROW}(\mathrm{John}, x, \_))$
3. Find the BOOK_TITLE and BORROWER_NAME of BOOKs that are BORROWed by some AUTHOR.
   $((\mathrm{BOOK\_TITLE}(x), \mathrm{BORROWER\_NAME}(y)) \mid \mathrm{BORROW}(y, x, \_))$
4. Find the BOOKs whose AUTHOR is also the PUBLISHER.
   $(\mathrm{BOOK}(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \mid$
   $\mathrm{BOOK\_TITLE}(x_1) \wedge \mathrm{AUTHOR}(x_2) \wedge \mathrm{PUBLISHER}(x_3) \wedge \mathrm{YEAR}(x_4) \wedge \mathrm{LIBRARY\_NAME}(x_5) \wedge \mathrm{PLACE}(x_6)$
   $\wedge\ \#\_\mathrm{OF\_COPIES}(x_7) \wedge x_2 = x_3)$

(2) Constraints

1. A BORROWER cannot BORROW more than 5 BOOKs.
   $(\forall x)\ (\forall y)\ (\mathrm{BORROWER\_NAME}(x) \wedge \#\_\mathrm{OF\_BOOKS}(y) \wedge \mathrm{BORROWER}(x, y) \rightarrow y \leqslant 5)$
2. A BORROWER must be either a STUDENT or a FACULTY.
   $(\forall x)\ (\exists y)\ (\exists z)\ (\mathrm{BORROWER\_NAME}(x) \wedge \mathrm{STUDENT\_NAME}(y) \wedge \mathrm{FACULTY\_NAME}(z) \rightarrow (x = y) \vee (x = z))$

defined from simple type predicates; hence the *generalisation* (Ref. 7) concept in the variable level (not in the predicate level) can be supported. For example, the following definition

$$\mathrm{BORROWER\_NAME} = \mathrm{STUDENT\_NAME} \vee \mathrm{FACULTY\_NAME}$$

means that 'A BORROWER_NAME is either a STUDENT_NAME or a FACULTY_NAME' and has the same meaning as the second constraint of Table 3. However, an MSFOL-based language (to be explained in section 4) is proposed to enhance the expressive power and to support the explicit representation capabilities of those abstraction mechanisms.

4. MANY-SORTED FIRST-ORDER LOGIC (MSFOL) BASED LANGUAGE

The main difference between OSFOL and MSFOL is that in MSFOL, the universe is divided into sorts; consequently, variables and constants are defined in sorts. Predicates belong to the product of sorts. The motivation for defining sorts in MSFOL is the same as defining type predicates in Typed OSFOL (Ref. 4), and a sort is used to restrict the usage of symbols. For example, $(\forall x)\,(\forall y)\,(\exists z)$ $\mathrm{BORROW}(x, y, z)$ is an OSFOL formula, where variables x, y, and z are ranged over the whole universe. To make the sentence more meaningful, we can insert some type predicates to restrict the range of x, y and z, and make it

$$(\forall x)\ (\forall y)\ (\exists z)\ \mathrm{BORROWER\_NAME}(x) \wedge \mathrm{BOOK\_TITLE}(y) \wedge \mathrm{DATE}(z) \rightarrow \mathrm{BORROW}(x, y, z).$$

This is a Typed OSFOL formula, which is obviously more meaningful than an OSFOL formula. In MSFOL, the corresponding sentence is simply written as

$$(\forall x^{\mathrm{BORROWER\_NAME}})\ (\forall y^{\mathrm{BOOK\_TITLE}})\ (\exists z^{\mathrm{DATE}})\ (\mathrm{BORROW}(x, y, z)),$$

where x, y, and z are defined in the sorts BORROWER_NAME, BOOK_TITLE, and DATE respectively. Intuitively, this formula is easier to express than that of a Typed OSMOL formula by simply counting the number of predicates in the formula. The advantages will be further illustrated in Section 6.

MSFOL was mentioned in Refs. 6, 8; however, they described neither the way of using MSFOL nor the differences between OSFOL/Typed OSFOL and MSFOL. MSFOL has a powerful mechanism, called *sort structuring*. Sort structuring provides the same effect as the *classification* and *generalisation/specialisation* abstraction mechanism (Ref. 7) which is commonly used in most of current extensions of the RDM, and in Semantic Data Models.[9]

Sort Structure

Three kinds of sort structures can be defined in MSFOL, namely: disjoint, hierarchical, and lattice. They are used to divide the universe of discourse. The disjoint sort structure is in which all sorts are disjoint. The hierarchy sort structure is in which a sort can be defined as a *subsort/supersort* of another sort. For example, STUDENT_NAME and FACULTY_NAME can be defined as two *subsorts* of BORROWER_NAME, and consequently BORROWER_NAME is the *supersort* of STUDENT_NAME and FACULTY_NAME. The last one, the lattice sort structure is in which sorts can be defined as the union, intersection, and difference on the *subsort/supersort* of other sorts. Figure 1 shows an example:

```
                          BORROWER_NAME
                                /    \
                STUDENT_NAME        FACULTY_NAME
                      /    \        /     \
UNDER_STUDENT_NAME GRA_STUDENT_NAME LECTURER_NAME
```

Figure 1.

The meaning is that 'GRA_STUDENT_NAME is a *subsort* of FACULTY_NAME and STUDENT_NAME'.

As shown in above, the *supersort* corresponds to the concept of generalisation, and *subsort* corresponds with the concept of specialisation. These two abstraction mechanisms can be used to classify objects and to represent a hierarchical/network relationship among objects in designing a database.

The following section defines MSFOL, and illustrates the features of using MSFOL.

Formal definition of MSFOL

The formal definition of MSFOL is in four parts; the language's definition is given in Definition 1, the semantics of a relational interpretation are given in Definition 2.0, the truth assignment is given in Definition 2.1, and the model definition is given in Definition 2.2. A brief discussion precedes each formal definition.

The MSFOL based language

There are some variations in the definitions of the MSFOL-based language which is derived from an OSFOL. A new symbol, called the sort symbol, is used to restrict the range of variables/constants and a relation that of predicates. Complex sorts can be defined from simple sorts. A sort can be either a *subsort* or a *supersort* of another sort. A hierarchical or a lattice sort structure can match the structure of real world applications, and the sort's subsort/supersort relationships can be explicitly represented. The added structure of MSFOL-Based languages make it possible to more efficiently perform more kinds of operations than with OSFOL-based languages.

The definition of equality predicates puts some restrictions on the variables and constants which can be compared. If two symbols are defined from the same sort or in the same path of a sort hierarchy, they are comparable; otherwise they cannot be compared as the formula is not well-formed. If BOOK_TITLE is a sort and JSH Yang is a constant of the sort STUDENT_NAME, then the predicate $=_{\mathrm{NAME}}(x^{\mathrm{BOOK\_TITLE}}$, 'JSH Yang') is illegal, because the constant 'JSH Yang' and the variable x are in two different sorts, notwithstanding the two sorts, STUDENT_NAME and BOOK_TITLE, are both subsorts of the sort NAME.

Each sort has distinct universal and existential quantifiers which are used with the variables of that sort. Sorted variables/constants and sorted quantifiers have an effect similar to the *data type* used in a programming languages. Furthermore, the two levels of abstraction in MSFOL, the sorts/variables and predicates, is more flexible to use than OSFOL with the *type predicate* approach (Ref. 5) (to be explained in the Definition 2).

Definition 1: MSFOL based language

An MSFOL-based language is a pair (A, W), where A is an alphabet of symbols and W is a set of syntactically well-formed formulae the symbols of A which comprises:

1. Logical symbols: punctuation symbols such as parentheses: (,), ⟨,⟩ and sentential connective symbols: $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$.

2. Sorts: a non-empty finite set of symbols S, each of which represents a simple sort and to which the following rules are applied when the sorts are defined: (*a*) a simple sort is a sort. (*b*) if s_1 and s_2 are sorts, then $s_1 \wedge s_2$, $s_1 \vee s_2$, and $\neg s_1$ are also sorts.

Example 1

If STUDENT_NAME and FACULTY_NAME are simple sorts, then we can define BORROWER_NAME as the supersort of STUDENT_NAME and FACULTY_NAME. That is, BORROWER_NAME = STUDENT_NAME ∨ FACULTY_NAME. This is the concept of generalisation[7] and STUDENT_NAME/FACULTY_NAME is a *specialisation*[7] of BORROWER_NAME.

3. Variables: for each sort s, there are finite number of variables $v_1^s, v_2^s, \ldots$.

Example 2

If x is a variable defined over sort NAME, then it is denoted as x^{NAME}.

4. Quantifier symbols: for each sort s, there is a universal quantifier symbol $\forall_s$ and an existential quantifier symbol $\exists_s$.

When a variable is quantified, it is denoted as $\forall_s x^s$ or $\exists_s x^s$. Without ambiguity, we may write $\forall x^s$, $\exists x^s$ or $\forall_s x, y, \exists_s x, y$.

Assume s is a subsort of s', then $\forall_s x^s$, $\forall_s x^{s'}$, or $\exists_s x^s$, $\exists_s x^{s'}$ are legal and x should range in sort s only.

5. Constant symbols: for each sort s, there is a finite set of constant symbols, each of which is said to be of sort s. Constants can be superscribed as 'JSH Yang $^{\mathrm{STUDENT\_NAME}}$'. In most cases, the superscripts can be omitted when the meaning is clear.

6. Predicate symbols: for each n-tuple $\langle s_1, \ldots, s_n \rangle$ of sorts for $n > 0$, there is a finite set of n-place predicate symbols, each of which is said to be of sort $\langle s_1, \ldots, s_n \rangle$.

Example 3

BORROW(BOOK_TITLE, BORROWER_NAME, DATE) is the predicate of sort ⟨BOOK_NAME, PERSON_NAME, DATE⟩ (the concept of *aggregation*[7]).

7. Equality Symbols: for each $s \in S$, there is an equality symbol $=_s$. It is a predicate symbol of sort $\langle s, s \rangle$. That means, the two terms of this predicate are in sort s. For subsort s_1 of s_2, the symbol $=_{s_1}$ could be used as the predicate symbol of sort $\langle s_1, s_2 \rangle$.

Example 4

If STUDENT_NAME is defined as a subsort of BORROWER_NAME, then $=_{\mathrm{STUDENT\_NAME}}(x^{\mathrm{STUDENT\_NAME}}, y^{\mathrm{BORROWER\_NAME}})$ is a legal predicate. The infix form of the above predicate is $x^{\mathrm{STUDENT\_NAME}} =_{\mathrm{STUDENT\_NAME}} y^{\mathrm{BORROWER\_NAME}}$.

8. Terms: any variable or constant symbol of sort s is a term of sort s.

9. Atomic Formulae: an atomic formula $P(r_1, \ldots, r_n)$ consists of a predicate symbol P of sort $\langle s_1, \ldots, s_n \rangle$ and terms $r_1, \ldots, r_n$ of sort $s_1, \ldots, s_n$, respectively. W is defined as:

10. Well-formed formulae (wffs): Wffs are defined recursively by using the connectives $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ and the quantifiers $\forall_s$ and $\exists_s$ as follows: (*a*) An atomic formula is a wff. (*b*) If w_1 and w_2 are wffs, then $\neg(w_1)$, $(w_1) \vee (w_2)$,

$(w_1) \wedge (w_2)$, $(w_1) \rightarrow (w_2)$, and $(w_1) \leftrightarrow (w_2)$ are wffs. (*c*) If w is a wff and $x \in s$, then $\forall_s x^s w(x^s)$ and $\exists_s x^s w(x^s)$ are wffs.

The meaning of $\forall_s x^s w(x^s)$ is 'For all x which belong to sort s, w is the case' and $\exists_s x^s w(x^s)$ is 'There is an x which belongs to the sort s, such that w is the case.'

11. Open wff: a wff which contains one or more free variables. A free variable is a variable which is not quantified.

Example 5

Assume STUDENT_NAME is defined as the subsort of sort BORROWER_NAME. Then $(\forall x^{\text{STUDENT\_NAME}})$ $((z^{\text{DEPARTMENT}} \quad y^{\text{BORROWER\_NAME}} | \text{STUDENT}(\_, x, z, \_) \wedge x =_{\text{NAME}} y))$ and $\text{STUDENT}(x^{\text{ID}}, y^{\text{STUDENT\_NAME}}$, 'Computer Engineering', 5) are both open wffs.

The meaning of the first open wff is 'List the departments and names of borrowers who are students'. the meaning of the second one is 'List the id. and names of the students who are in the Computer Engineering department'.

12. Closed wffs: a wff which contains no free variable.

Example 6

$(\forall_{\text{STUDENT\_NAME}} x)$ $(\forall_{\text{BOOK\_TITLE}} y)$ $(\exists_{\text{DATE}} z)$ (BORROW(x, y, z)) which means 'For all x and y which are STUDENT_NAMES, and BOOK_TITLES, respectively, there exists a DATE in the predicate relation BORROW.'

13. Ground wffs: a wff which contains no variables (only constants).

Example 7

STUDENT(173215, 'JSH Yang', 'computer engineering').

Relational interpretation

The interpretation (semantics) of an MSFOL-based language is described as follows. For each sort, there associates a domain D_s. Quantifiers are defined over sorts and restrict the quantified variables to be in the domain of the sort. For example, if $c_1, c_2, \ldots, c_n$ are the finite number of constants of sort s, then $\forall_s x^s w(x^s)$ has the same meaning as $w(c_1) \wedge w(c_2) \wedge \ldots \wedge w(c_n)$, and $\exists_s x^s w(x^s)$ means $w(c_1) \vee w(c_2) \vee \ldots \vee w(c_n)$. Predicate is defined and ranged over the product of sorts of its terms. Equality or other comparison/arithmetic operators are defined as binary predicates of sort $\langle s, s' \rangle$ (where s and s' are *compatible* sorts). The result of equality comparison is to assign a truth value.

Definition 2.0: Relational interpretation of an MSFOL based language

The interpretation I of an MSFOL-based language is defined as follows:

1. To the quantifier symbol $\forall_s$ and $\exists_s$, I assigns a non-empty set D_s. D_s is called the universe of I of sort s.

The universe of discourse is the union of sorts' universe, denoted as $D = \bigcup_s D_s$.

2. To each constant symbol c of sort s, I assigns a point c in D_s. Without loss of generality, hereafter, all constant symbols are treated the same as the point in D and are denoted as c^s or just c).

3. To each predicate symbol P of sort $\langle s_1, \ldots, s_n \rangle$, I assigns a relation $P \subseteq D_{s_1} x \ldots x D_{s_n}$.

4. To each variable v^s in V^s of sort s, I assigns a function $f \colon V^s \rightarrow D_s$ which maps each variable into the corresponding domain.

5. For each equality predicate the following general properties apply:

- Reflexivity $\forall_s x(x =_s x)$,
- Symmetry $\forall_s x, y((x =_s y) \rightarrow (y =_s x))$,
- Transitivity $\forall_s x, y, z((x =_s y) \wedge (y =_s z) \rightarrow (x =_s z))$,
- Substitution $\forall_{s_1} x_1, y_1, \ldots, \forall_{s_n} x_n, y_n$ $(P(x_1, \ldots, x_n) \wedge (x_1 =_{s_1} y_1) \wedge \ldots \wedge (x_n =_{s_n} y_n) \rightarrow P(y_1, \ldots, y_n))$.

Remarks

Other arithmetic/comparison operators such as $+$, $-*$, ..., $>$, $<$, ... etc. can also be defined as predicates in the same way as equality, and have their own usual properties.

Truth Assignment and Satisfactions of MSFOL Wffs

Truth assignment and satisfaction are similar to OSFOL. The true value of a ground atomic formula is evaluated by finding out whether the formula is in the definition of the predicate. For example, to evaluate the true value of STUDENT(173215, 'JSH Yang', 'Computer Engineering', 5) we find whether the tuple ⟨173215,'JSH Yang','Computer Engineering', 5⟩ is in the relation of the predicate STUDENT(ID,STUDENT_NAME,DEPART,LEVEL). Wffs constructed by using logical connectives (i.e. $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$) are evaluated with their usual meaning using the truth tables as defined below. The evaluation of the truth value of a closed wff is to find out whether it is in, or is implied by, the axioms. The evaluation of the truth value of an open wff is to find a relation, and a set of n-tuples, which satisfy the wff. From the above two definitions of Definition 2.0, we can easily find out that the yes/no query evaluation in an RDM corresponds to the evaluation of a ground wff, and the evaluation of common queries corresponds to the evaluation of open wffs. The deductive capabilities or integrity constraints maintenance of a RDM corresponds to the mechanism of specification, derivation, and evaluation of closed wffs.

Definition 2.1: Truth assignment and satisfactions

1. Truth assignment

The truth assignment for a closed wff (atomic formula/ground wff), with respect to an interpretation I, is the truth functional result of the terms of the wff. The following truth table show the truth assignment for the closed wffs with logical connectors: (w_1 and w_2 are assumed to be closed wffs).

Truth Table

w_1	not w_1	w_2	$w_1 \vee w_2$	$w_1 \wedge w_2$	$w_1 \rightarrow w_2$	$w_1 \leftrightarrow w_2$
true	false	true	true	true	true	true
true	—	false	true	false	false	false
false	true	true	true	false	true	false
false	—	false	false	false	true	true

2. Satisfaction

The satisfaction of a closed wff, with respect to the interpretation I, means that the wff with the truth assignment to the terms of the wff is *true* in the interpretation. It is denoted by $\models_I w$. Assume w_1, w_2 and w are closed wffs in set W. P is a predicate, and $t_1, t_2, \ldots, t_n$ are terms, the following are the definitions:

(*a*) $\models_I P(t_1, \ldots, t_n)$ iff $P(t_1, \ldots, t_n) \in W$.
(*b*) $\models_I w_1 \wedge w_2$ iff $\models_I w_1$ and $\models_I w_2$.
(*c*) $\models_I w_1 \vee w_2$ iff $\models_I w_1$ or $\models_I w_2$.
(*d*) $\models_I \neg w$ iff not $\models_I w$.
(*e*) $\models_I w_1 \rightarrow w_2$ iff $\models_I \neg w_1 \vee w_2$.
(*f*) $\models_I w_1 \leftrightarrow w_2$ iff $\models_I (w_1 \rightarrow w_2) \wedge (w_2 \rightarrow w_1)$.
(*g*) $\models_I (\forall_s x^s) w$ iff for all constants c of D_s, which are assigned to variable and $\models_{I[x \rightarrow c]} w$.
(*h*) $\models_I (\exists_s x^s) w$ iff $\models_{I[x \rightarrow c]} \neg(\forall_s x^s) \neg w$.

(i.e. for some constant c of D_s which is assigned to variable x of same type and $\models_I w$.)

Valid wffs and model

Some assumptions used by the RDM, such as the Closed World Assumption, Unique Name Assumption, ...etc. are the common implicit rules of RDM (Ref. 5). These rules can also be represented by first-order closed wffs and they are valid wffs. Similarly, the integrity constraints can be represented as the closed wffs in MSFOL. The database (interpretation) should be a *model* of these integrity constraints.

Definition 2.2: Valid wffs and model

1. Valid Wffs:

A wff w is valid iff $\models_I w$, that is, w is true for every interpretation.

2. Model:

An interpretation is called a model for some set of wffs, if it is an interpretation in which all wffs in the set are true.

Notice, MSFOL is a first order variation; hence the completeness and compactness of first-order logic are reserved; however, we shall contrast the expressiveness, efficiency, ease-of-use of an MSFOL-based language as opposed to an OSFOL-based language.

The following examples represent the queries and constraints of Table 1 in an MSFOL-based language.

5. REDUCTION ALGORITHMS BETWEEN A TYPED FIRST-ORDER LANGUAGE AND A MANY-SORTED FIRST-ORDER LANGUAGE

From the above definitions of MSFOL, the major difference between an MSFOL-based language and an OSMOL-based language are the syntactic variations of sorts and sorted symbols. The interpretation of an MSFOL is similar to OSFOL's, except the restriction of using the language symbols and the sort structuring.

In this section, two algorithms and examples are given to make a syntactic reduction between the wffs of an MSFOL-based language and a Typed OSFOL-based language in order to prove that these two types of wffs are syntactically equivalent. Since a first-order language is composed of the wffs of the underlying logic (please refer to 'Definition 1: MSFOL based language'), the equivalence of these two kinds of wffs also implies the equivalence of these two languages. The correctness of the two algorithms is proved from the point of view of interpretation (semantics) of the given language; therefore these two languages are semantically equivalent in that sense.

Syntactic reduction from MSFOL to typed OSFOL

Reduction algorithm 1: (To reduce an MSFOL wff to a Typed OSFOL wff)
Input: an MSFOL wff
Output: the corresponding typed OSFOL wff
Procedure:

(1) Replace each sorted universal/existential quantifier by a universal/existential quantifier (i.e. Replace $\forall_s$ and $\exists_s$ with $\forall$ and $\exists$).

(2) For each sorted variable/constant, insert the corresponding unary predicate in the left-hand side of the formula.

(3) Connect the unary predicates inserted by the previous step with the $\wedge$ sign and use the $\rightarrow$ sign to connect with the formula.

(4) Replace each sorted variable/constant by an unsorted variable/constant, such as x for $x^s, \ldots$, and etc. (e.g. If the MSFOL wff is $(\forall_s x^s)$ $(\forall_{s'} y^{s'})(w(w^s, y^{s'})$, the result of the above three steps is $(\forall x)(\forall y)$ $(s(x) \wedge s'(y) \rightarrow w(x, y))$.)

(5) Repeat the above steps until there are no sorted variables/constants and sorted universal/existential quantifiers left.

Table 4. Queries and constraints of Table 1 represented in an MSFOL-based language

1. Find BORROWER_NAME of BORROWERs who BORROWed more thant 3 BOOKs.
   $(x^{\text{BORROWER\_NAME}} \mid (\exists y^{\text{\#\_OF\_BOOKS}})\ (\text{BORROWER}(x, y) \wedge y > 3))$
2. Get the BOOK_TITLE of BOOKs BORROWed by John.
   $(x^{\text{BOOK\_TITLE}} \mid \text{BORROW}(\text{John}, x, \_))$
3. Find the BOOK_TITLE and BORROWER_NAME of BOOKs that are BORROWed by some AUTHOR.
   $((x^{\text{BOOK\_TITLE}}, y^{\text{BORROWER\_NAME}}) \mid \text{BORROW}(y, x, \_))$
4. Find the BOOKs whose AUTHOR is also the PUBLISHER.
   $((x_1^{\text{BOOK\_TITLE}}, x_2^{\text{AUTHOR}}, x_3^{\text{PUBLISHER}}, \ldots, x_7^{\text{\#\_OF\_COPIES}}) \mid x_2 = x_3)$

(2) Constraints

1. A BORROWER cannot BORROW more than 5 BOOKs.
   $(\forall x^{\text{BORROWER\_NAME}})\ (\forall y^{\text{\#\_OF\_BOOKS}})\ (\text{BORROWER}(x, y) \rightarrow y \leqslant 5)$
2. A BORROWER must be either a STUDENT or a FACULTY.
   $(\forall x^{\text{BORROWER\_NAME}})\ (\exists y^{\text{STUDENT\_NAME}})\ (\exists z^{\text{FACULTY\_NAME}})\ (x = y) \vee (x = z))$

Example 8

Let $(\forall y^{\text{BORROWER\_NAME}})\ (\exists y^{\text{STUDENT\_NAME}})\ (\exists z^{\text{FACULTY\_NAME}})$ $((x = y) \vee (x = z))$ be a closed wff of MSFOL (with abbreviation for $\forall$ and $\exists$). following the above algorithm, the corresponding type closed wff of OSFOL after the reduction is

$$(\forall x)\ (\exists y)\ (\exists z)\ ((\text{BORROWER\_NAME}(x) \\ \wedge \text{STUDENT\_NAME}(y) \\ \wedge \text{FACULTY\_NAME}(z)) \rightarrow (x = y) \vee (x = z))$$

Syntactic Reduction from Typed OSFOL to MSFOL

Reduction algorithm 2: (To reduce a Typed OSFOL wff to an MSFOL wff)
Input: a Typed OSFOL wff
Output: the corresponding MSFOL wff
Procedure:

(1) Find each variable/constant which is in a unary predicate on the left-hand side of the formula, and the unary predicate is one of the special *type* predicate (e.g. x of $s(x)$, c of $s'(c)$, ...).

(2) Replace each variable/constant with the sorted variable/constant by using the predicate symbol as its sort symbol, and eliminate the predicate (e.g. replace x with x^s and eliminate $s(x)$, ...).

(3) Replace each universal/existential quantifier with the sorted universal/existential quantifier by using the variable's sort of quantified as its sort.

(4) Repeat the above steps until all the variables/constants and universal/existential quantifiers are sorted, then eliminate the $\rightarrow$ sign if there is no predicate remaining on its left-hand side.

Example 9

Let

$$(\forall x)\ (\exists y)\ (\exists z)\ ((\text{BORROWER\_NAME}(x) \\ \wedge \text{STUDENT\_NAME}(y) \\ \wedge \text{FACULTY\_NAME}(z)) \rightarrow (x = y) \vee (x = z))$$

be a typed closed wff of OSFOL. Following the above algorithm,

$$(\forall x^{\text{BORROWER\_NAME}})\ (\exists y^{\text{STUDENT\_NAME}}) \\ (\exists z^{\text{FACULTY\_NAME}})\ ((x = y) \vee (x = z))$$

is the corresponding closed wff of MSFOL (with abbreviation for $\forall$ and $\exists$).

Correctness of the reduction algorithms

Using induction and two connective symbols $\neg$ and $\vee$, the correctness of the Reduction algorithm can be proven. Other connective symbols can be derived from $\neg$ and $\vee$; therefore they are omitted.

Theorem 1

The two reduction algorithms are correct.

Proof

Assume in s and s', there are $c_1^s, c_2^s, \ldots, c_n^s$ and $c_1^{s'}, c_2^{s'}, \ldots, c_m^{s'}$ finite number of constants respectively, and the correctness of the algorithm can be proven by the following induction:

(1) $\forall_s x^s(\neg w(x^s))$ is equivalent to $\forall x(s(x) \rightarrow \neg w(x))$.

Proof

By the definitions of MSFOL, the meaning of $\forall_s x^s(\neg w(x^s))$ is that $\neg w(c_1) \vee \neg w(c_2) \vee \ldots \vee \neg w(c_n)$ is true. We can say that $\forall x(s(x) \rightarrow \neg w(x))$ means 'If x is restricted to the range of s then $\neg w(x)$ is true.' By the OSFOL definition, it has the same meaning.

(2) $\exists_s x^s(\neg w(x^s))$ is equivalent to $\exists x(s(x) \rightarrow \neg w(x))$.

Proof

The proof is similar to (1) above. The meaning of $\exists_s x^s(\neg w(x^s))$ is the same as $\exists x(s(x) \rightarrow \neg w(x))$ both equivalent to $\neg w(c_1) \wedge \neg w(c_2) \wedge \ldots \wedge \neg w(c_n)$.

(3) $\forall_s x^s \forall_{s'} y^{s'}(w(x^s) \vee w'(y^{s'}))$ is equivalent to $\forall x\, \forall y(s(x) \vee s'(y) \rightarrow w(x) \vee w'(y))$

Proof

By the definition of MSFOL, the meaning of $\forall_s x^s \forall_{s'} y^{s'}(w/x^s) \vee w'(y^{s'}))$ is $(w(c_1^s) \vee w'(c_1^{s'})) \wedge (w(c_2^s) \vee w'(c_1^{s'})) \wedge \ldots \wedge (w(c_n^s) \vee w'(c_1^{s'})) \wedge (w(c_1^s) \vee w'(c_2^{s'})) \wedge \ldots \wedge (w(c_n^2) \vee w'(c_m^{s'}))$ is true, and similarly the meaning of $\forall x\, \forall y(s(x) \wedge s'(y) \rightarrow w(x) \vee w'(y))$ is the same as above.

(4) $\exists_s x \exists_{s'} y(w(x) \vee w'(y))$ is equivalent to $\exists x\, \exists y(s(x) \wedge s'(y) \rightarrow w(x) \vee w'(y))$

Proof

Similar as to the above, the meaning is $(w(c_1^s) \vee w'(c_1^{s'})) \vee (w(c_2^s) \vee w'(c_1^{s'})) \vee \ldots \vee (w(c_n^s) \vee w'(c_1^{s'})) \vee (w(c_1^s) \vee w'(c_{\frac{s'}{2}})) \vee \ldots \vee (w(c_n^s) \vee w'(c_m^{s'}))$ is true.

6. ADVANTAGES OF MANY-SORTED FIRST-ORDER LANGUAGE

It has been shown that an MSFOL-based language has the same expressive power as a Typed OSFOL-based language; however, the restricted usage of symbols and the sort structuring in MSFOL can provide more advantages than either an OSFOL-based language or a Typed OSFOL-based language. The followings are the illustrations.

More efficient implementation

Observation 1

The implementation of an MSFOL-based language can be improved by reducing the search space of variables.

In an MSFOL-based language, all variables are defined in sorts, there is no need to use a unary type predicate to restrict the usage of variables; hence the number of predicates in the MSFOL wff is less than the corresponding wff in Typed OSFOL; therefore, implementation of an MSFOL-based language is more efficient than that of a typed OSFOL-based (or OSFOL-based) language, if the criteria for efficiency is the number of predicates to be evaluated.

Early detection of syntax errors

Observation 2

In an MSFOL-based language, a sort syntax error in a logical (or comparison) predicate such as equality, can be detected earlier than in a Typed OSFOL-based language.

Since variables are defined in sorts and each sort has its own quantifiers and logical/comparison predicates; syntax errors can be detected by syntactic analysis rather than at the interpretation (execution) level as with a Typed OSFOL-based language does.

Operation Prevention

Observation 3

Some semantically meaningless operations can be detected when parsing MSFOL-based language.

For reasons similar to those above, if the terms in a formula of a sorted logic have no sorts in common, they cannot intersect with each other. In one-sorted logic, there is no such restriction. Therefore the problem of the *natural join ambiguity* which could occur with an OSFOL-based language would be presented by an MSFOL-based language.

Example 10

Compare the following two operations:

(1) The natural join of the relations BOOK and BORROW on the domain BOOK_TITLE,

(2) The natural join of the relations LIBRARY and BOOK on the domains LIBRARY_NAME and BORROWER_NAME.

The first operation is allowed by an OSFOL, a Typed OSFOL, or an MSFOL-based language since the join operation is based on the same domain. The second operation is intuitively meaningless, since LIBRARY_NAME and BORROWER_NAME are in two different domains. Such a situation can be detected by an MSFOL-based language as an syntax error while an OSFOL-based language would permit it. In a Typed OSFOL-based language, the variables for LIBRARY_NAME and BORROWER_NAME can be restricted by using different types predicates, such meaningless operations can also be prohibited during the evaluation time. However, for an MSFOL-based language, the syntax error can be detected at compile/interpret time rather than at evaluation time as OBSERVATION 2 describes.

Abstraction mechanism

The sort structuring of an MSFOL-based language provides the generalisation/specialisation abstraction mechanisms to divide the universe into a hierarchy or a lattice structure. This is neither supported by an OSFOL-based language nor by a Typed OSFOL-based language.

7. CONCLUSIONS AND FURTHER RESEARCHES

The formal definition of an MSFOL-based language is given and the semantic equivalence between a Typed OSFOL-based language and an MSFOL-based language is proven. The efficiency and abstraction mechanism in an MSFOL-based language provides an appropriate interpretation to the recent extensions of the RDM (Ref. 10). The complexity of both Reduction algorithms is $O(mn)$; where m is the number of quantifiers and n is the number of variable/constant symbols.

With other abstraction mechanisms such as attribute/role, classification/instantiation, functional relationship, and object oriented features provided in recent semantic data models (Ref. 9), it would seem that an MSFOL-based language is more promising than an OSFOL-based language for interpretations of these models. The comparison of abstraction mechanisms of an OSFOL-based language, a Typed OSFOL-based language, and an MSFOL-based language is summarised in the following table.

Table 5. Comparison of abstraction mechanisms

RDM	OSFOL	Typed OSFOL	MSFOL
aggregation	yes	yes	yes
classification	no	implicitly	explicitly
generalisation	no	no	yes

From the taxonomy of data models (Ref. 10), the current trend in data modelling is to provide more structural and dynamic mechanisms to reflect the real world application in an object-based manner. Examples are the semantic data models such as TAXIS,[11] SDM,[12] IFO[13] or SHM+.[14] These structural and dynamic mechanisms also provide abstraction capabilities for designing a database. The results provide a formal basis to interpret database models through the extensions of classical logic. MSFOL gives structural mechanisms and provides a better interpretation than that of OSFOL or Typed OSFOL. The logic interpretation of behaviour modelling and constraints handling should be studied next.

A proper and sound logic interpretation for a database model could, by using MSFOL, give a formal basis for evaluating the power of various data models, and can provide a deep understanding of the capability of a data model. The symbolic deduction capabilities of logic might also provide the features for knowledge reasoning and deduction required in the field of Artificial Intelligence.

Acknowledgement

We are grateful to professor P. J. H. King of Birkbeck College, University of London, and professor Daniel J. Buehrer for their dedication to revise the paper.

REFERENCES

1. J. M. Nicolas and H. Gallaire, Data Base: Theory *vs.* Interpretation, In *Logic and Databases* (H. Gallaire and J. Minker eds.), pp. 33–54. Plenum Press (1978).
2. R. Reiter, Data Bases: A Logical Perspective, In *Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modeling*, pp. 174–176. Pingree Park, Colorado (1980).
3. E. F. Codd, A Relational Model of Data for Large Shared Data Banks, *Communication of ACM* **13** (6) 377–387 (1970).

4. R. Reiter, On the Integrity of Typed First Order Data Bases. In *Advances in Data Base Theory*, edited H. Gallaire, J. Minker and J. M. Nicolas, pp. 137–157. Plenum Press (1981).
5. R. Reiter, Toward a Logical Reconstruction of Relational Database Theory. In *On conceptual Modelling*, edited M. L. Brodie, J. Mylopoulos and J. W. Schmidt, pp. 191–223. Springer-Verlag, N.Y., (1984).
6. A. Pirotte, High Level Data Base Query Languages. In *Logic and Databases*, edited H. Gallaire, and J. Minker, pp. 409–436. Plenum Press (1978).
7. J. M. Smith and D. C. P. Smith, Database Abstractions: Aggregation and Generalization, *ACM Transaction On Database Systems* **2** (2) 105–133 (1977).
8. H. Gallaire, J. Minker and J. M. Nicolas, Logic and Database: A Deductive Approach, *ACM Computing surveys*, **16** (2) 153–185 (1984).
9. R. Hull and R. King, Semantic Database Modeling: survey, Application, and Research Issues, *ACM Computing Surveys*, **19** (3) 201–260 (1987).
10. M. L. Brodie, On the Development of Data Models. In *On Conceptual Modelling*, edited M. L. Brodie, J. Mylopoulos and J. W. Schmidt, pp. 19–47. Springer Verlag (1984).
11. A. Borgida, J. Mylopoulos and H. K. T. Wong, Generalization/Specialization as a basis for Software Specification. In *On conceptual Modelling* edited M. L. Mylopoulos and J. W. Schmidt, pp. 87–114. Springer-Verlag (1984).
12. M. Hammer and D. McLeod, Database description with SDM: A Semantic Database Model, *ACM Transaction on Database Systems* **6** (3) 351–386 (1982).
13. S. Abiteboul and R. Hull, IFO: a Formal Semantic Database Model (Preliminary Report). In *Proceedings of ACM SIGART-SIGMOD Symposium on Principles of Database Systems*, pp. 119–132 (1984).
14. M. L. Brodie and D. Ridjanovic, On the Design and Specification of Database Transactions. In *On Conceptual Modelling*, edited M. L. Brodie, J. Mylopoulos and J. W. Schmidt, pp. 277–306. Springer-Verlag (1984).

Announcements

17–20 June 1992

ICCAL' 92 Fourth International Conference on Computers and Learning, Acadia University, Nova Scotia, Canada. 50 papers, 6 invited speakers, workshops, tutorials, panels, exhibitors.

Contact: Dr Ivan Tomek, Jodrey School of Computer Science, Acadia University, Wolfville, Nova Scotia, Canada, B0P 1X0. Tel: (902) 542-2201. Fax: (902) 542-7224. E-mail: internet:iccal@AcadiaU.ca.

24–26 June 1992

Thirteenth International Conference on Application and Theory of Petri Nets, Sheffield. Petri Nets Tutorial, Sheffield, 22–23 June 1992

The Thirteenth Annual International Petri Net Conference will be organised by The School of Computing and Management Sciences, Sheffield City Polytechnic, England. Papers will be presented in areas of application and theory of Petri nets. The language of the conference is English.

Topics

System design and verification using nets
Causality/partial order theory of concurrency
Analysis and synthesis, structure and behaviour of nets
Net-based semantical, logical and algebraic calculi
Higher-level net models
Timed and stochastic nets
Relationships between net theory and other approaches
Symbolics net representation (graphical, textual, ...)
Computer tools for nets
Experience with using nets, case studies
Educational issues related to nets
Applications of nets to:

- office automation
- flexible manufacturing
- programming languages
- protocols and interfaces
- hardware structures
- real-time systems
- performance evaluation
- operations research
- embedded systems

The conference takes place under the auspices of: AFCET SIG 'Systèmes Parallèles et Distribués' and CNRS-C^3, AICA, BCS SIG 'Formal Aspects of Computing Science', EATCS and GI SIG 'Petri Nets and Related System Models'.

Tools, Posters, Projects, Meetings and Courses

The conference will also comprise:

- An *exhibition of posters* describing theoretical and practical results. Posters are displayed throughout the conference.
- An *exhibition of computer tools* for Petri nets. Tuesday will be the main day of the tool exhibition and each tool will have its own scheduled time for a coherent presentation for a large audience. The length of each presentation will be approx. 60 min. Moreover, periods are set aside during the conference in which tools can be demonstrated for small groups. The organizers will provide Macintosh, IBM PC and SUN equipment. There will be an overhead projection system for Macintosh and IBM PC. It may, upon request, be possible to supply other kinds of computer equipment.
- Short *presentations of projects* where nets are put into practice. This section of the conference allows the presentation of experiences of using nets in on-going or completed projects. The presentations will take place in a special session during the conference and each of them may be supplemented by a brief report in the proceedings.
- *Meetings and courses* with relation to Petri Nets. Monday and Tuesday will be the days for this activity. It is possible to arrange meetings for different groups, e.g. participants in international Petri Net projects. It is also possible to arrange small educational courses, e.g. with respect to some of the demonstrated Petri Net tools. All activities are free of charge for the participants.

Tutorial

The tutorial will concentrate on the basic notions and fundamental concepts of Petri Nets. There will be talks on Elementary Net Systems, Place/Transition Systems, High Level Nets, Coloured Petri Nets, Timed and Stochastic Nets and Performance Evaluation.

History of the Conference

The conference was formerly called the European Workshop on Applications and Theory of Petri Nets and the meetings have taken place annually since 1980. The aim of the conferences is to create a forum for discussing progress in the application and theory of Petri Nets. Typically, the conferences have 100–150 participants – one-third of these coming from industry while the rest are from universities and research institutions. The conference takes place in the last week of June.

Organising Committee Chairman

Geoff Cutts, School of Computing and Management Sciences, Sheffield City Polytechnic, 100 Napier Street, Sheffield S11 8HD, England. Tel: +44 742 533 117. Fax: +44 742 533 161. Telex: 54680 SHPOLY G.

Details of the conference may be obtained from:

Conference Services, Sheffield City Polytechnic, 36 Collegiate Crescent, Sheffield S10 2BP, England. Tel: +44 742 532 576. Fax: +44 742 532 579.