Models and Equality for Logical Programming[1]

Joseph A. Goguen and José Meseguer
SRI International, Menlo Park CA 94025
Center for the Study of Language and Information, Stanford University 94305

Abstract: We argue that some standard tools from model theory provide a better semantic foundation than the more syntactic and operational approaches usually used in logic programming. In particular, we show how initial models capture the intended semantics of both functional and logic programming, as well as their combination, with existential queries having logical variables (for both functions and relations) in the presence of arbitrary user-defined abstract data types, and with the full power of constraint languages, having any desired built-in (computable) relations and functions, including disequality (the negation of the equality relation) as well as the usual ordering relations on the usual built-in types, such as numbers and strings. These results are based on a new completeness theorem for order-sorted Horn clause logic with equality, plus the use of standard interpretations for fixed sorts, functions and relations. Finally, we define "logical programming," based on the concept of institution, and show how it yields a general framework for discussions of this kind. For example, this viewpoint suggests that the natural way to combine functional and logic programming is simply to combine their logics, getting Horn clause logic with equality.

1 Introduction

This paper argues that some very significant benefits are available to logic programming from using certain concepts from first order model theory, namely:

- order-sorted logic and models;
- initial models;
- interpretation into fixed models for certain fixed sorts, functions and relations; and
- true semantic equality.

These techniques, which are all standard in the theory of abstract data types [17, 22, 14], provide an attractive alternative to the more syntactical and operational approaches generally favored in logic programming. Moreover, they provide a powerful approach that supports:

- user-defined abstract data types;
- built-in data types;
- combined logic and functional programming; and
- constraint-based programming, in a way that can utilize standard algorithms for standard problems, such as linear programming.

In addition, we suggest that the more recent theory of institutions [10] may provide conceptual insight and clarification, as well as a broadening of the general scope of logic programming, so as to encompass any logical system satisfying certain simple restrictions.

In a sense, this paper is an attempt to explicate our previous paper on Eqlog [11], by giving a fuller account of its mathematical semantics, as well as further details, polemics, and comparisons with the

[1]Supported in part by Office of Naval Research Contracts N00014-85-C-0417 and N00014-86-C-0450, and a gift from the System Development Foundation.

existing literature. One reason that [11] may have been obscure to many readers, is the large number of new ideas that it tried to introduce all at once; here, we attempt to highlight certain ideas by ignoring others. Among the features of Eqlog deliberately downplayed here are: modules, both hierarchical and generic; theories and views; and "attributes" of operators (e.g., associativity and commutativity). Although these features greatly increase the expressive power of Eqlog, they would also distract from the basic foundational and semantic issues that we wish to emphasize here. For similar reasons, this paper does not develop most issues concerning the operational semantics of the various systems that are discussed. Thus, unification, term rewriting, narrowing and resolution are only touched upon. They are discussed in somewhat more detail in [11], and will receive full treatment in [23] and [26].

1.1 Order-Sorted Logic

Ordinary unsorted logic offers the dubious advantage that anything can be applied to anything; for example,

$$3 * \texttt{first-name(age(false))} < 2^{\texttt{birth-place(temperature(329))}}$$

is a well-formed expression. Although beloved by hackers of Lisp and Prolog, unsorted logic is too permissive. The trouble is that the usual alternative, many-sorted logic, is too restrictive, since it does not support overloading of function symbols such as _+_ for integer, rational, and complex numbers. In addition, an expression like

```
(-4 / -2)!
```

does not, strictly speaking, parse (assuming that factorial only applies to natural numbers). Here, we suggest that **order-sorted logic**, with subsorts and operator loading, plus the additional twist of **retracts** (although they are not discussed here; see [14]), really does provide sufficient expressiveness, while still banishing the truly meaningless.

Although the specialization of many-sorted logic to many-sorted algebra has been very successfully applied to the theory of abstract data types, many-sorted algebra can produce some very awkward specifications in practice, primarily due to difficulties in handling erroneous expressions, such as dividing by zero in the rationals, or taking the top of an empty stack. In fact there *is no* really satisfactory way to define either the rationals or stacks with MSA. However, order-sorted algebra overcomes these obstacles through its richer type system, which supports subsorts, overloaded operators, and total functions that would otherwise have to be partial. Moreover, order-sorted algebra is the basis of both OBJ [9] and Eqlog [11]. Finally, order-sorted algebra solves the **constructor-selector problem**, which, roughly speaking, is to define inverses, called selectors, for constructors; the solution is to restrict selectors to the largest subsorts where they make sense. For example, **pop** and **top** are only defined for non-empty stacks. [15] shows not only that order-sorted algebra solves this problem, but also that many-sorted algebra *cannot* solve it.

The essence of order-sorted logic is to provide a *subsort* partial ordering among the sorts, and to interpret it semantically as subset inclusion, among the carriers of a model, and to support operator overloading that is interpreted as restricting functions to subsorts. Two happy facts are that order-sorted logic is only slightly more difficult than many-sorted logic, and that essentially all results generalize from the many-sorted to the order-sorted case without complication. See [14] for a comprehensive treatment of order-sorted algebra. This paper broadens the logical framework to allow

not only algebras, but also models of arbitrary first-order signatures, with both function and predicate symbols, including equality, and gives rules of deduction for Horn clauses in such a logic, proving their completeness and several other basic results that are directly relevant to our model-theoretic account of logic and functional programming, including initiality and Herbrand theorems.

1.2 Models

Perhaps the origins in proof theory explain the obsession of logic programming theorists with syntactic and proof theoretic constructions. In any case, we believe that more semantic and more abstract tools provide a basis that is both broader and more powerful. In particular, we feel that the usual Herbrand Universe construction is too syntactic and is also unnecessarily restrictive, because:

1. it does not provide for built-in types, such as numbers and infinite trees;
2. it does not provide for user-defined abstract data types;
3. it does not (directly) address the phenomenon of representation independence for terms and for data types, whether built-in or user-defined; and
4. the proofs are more concrete and computational than necessary[2].

Of course, these deficiencies can all be patched without great difficulty -- for example, [19] shows how to include built-in numbers -- but after a few such patches, you have something enough like the initial model approach that you might as well, or better, take advantage of the powerful machinery associated with that tradition.

The reason for being interested in models is just that a standard model can provide the implementer with a clear standard for correctness, and can also provide the programmer and user with a clear model for what to expect when programs are actually run.

The reason for being interested in standard interpretations into particular semantic domains on some sorts, functions and relations (while leaving others free) is that then one can use standard algorithms to solve particular problems over such domains, for example, linear programming algorithms over the real numbers. This gives a great deal of flexibility, since one can still use initiality (i.e., abstraction) over other sorts. We argue below that this provides an elegant foundation for constraint-based programming.

1.3 Equality

Equational logic, which is essentially the logic of substitution of equals for equals, provides a foundation for functional programming languages. For example: [18] gives (what can be seen as) an equational description of Backus' FP [2]; [24] describes an "equational programming" language[3]; and [9] describes OBJ2, a language that combines initial algebra semantics for executable "objects" (defined by very general sets of user-supplied conditional order-sorted equations), with "loose" algebra semantics for non-executable "theories" (defined by arbitrary sets of equations).

---

[2]Not everyone will regard this as a deficiency!

[3]This language has some very strong restrictions, including: no repeated variables on lefthand sides, no overlap among equations, only one sort of data, no conditional equations, and a strong sequentiality condition; on the other hand, it is much easier to compile efficient code from sets of equations that satisfy such restrictions.

In the context of first order logic, equality is generally treated as a special relation, interpreted as real semantic equality in models, rather than merely axiomatized. This is the sense in which one speaks of "first order logic with equality" and of "Horn clause logic with equality." Complete sets of rules of deduction are well-known for these logical systems, and the latter has been used to combine logic and functional programming [11]. This paper later gives corresponding rules for order-sorted Horn clause logic with equality.

Equality is also useful in understanding constraint-based programming, because equations can be used to define the basic data structures, and then various relations of special interest can be defined recursively over these data structures, and/or provided as built-ins.

1.4 Initiality

Initial models free one from commitment to any particular representation; that is, they support *abstraction*. In particular, initiality handles abstract data types for logical programming languages with great fluency and convenience, and similarly it can be used to define functions and relations over built-in types [11]. Initial models also provide an account of the conceptual world of a program, in the sense of being "closed worlds" or "standard models." In particular, they provide a standard of correctness for the implementer, as well as a model for what results to expect for the programmer. Finally, initiality is a so-called "universal property," that there exists a unique mapping satisfying certain conditions, and it is well-known that, in many cases, one gets a much cleaner mathematical theory, with simpler and more conceptual proofs, from using universal characterizations of objects of interest, as compared to using concrete constructions for them [21]. In fact, the familiar characterization of "free" by the existence of a unique mapping with certain properties that extends another, is a special case of initiality.

One can better understand initiality through the so-called "no junk" and "no confusion" conditions (originally from [7]); these can also be seen as "completeness" and "soundness" conditions, respectively. Assume that signatures provide symbols for construcing sentences, including functions and constants (in Σ) and relations (in Π), and that models contain "data elements." Given a signature Σ,Π and a set C of Σ,Π-sentences, call a Σ,Π-model **standard** if and only if:

1. **No junk**: Every data item is denoted by a term using the function (and constant) symbols in Σ. (A data item that cannot be so constructed is "junk.")
2. **No confusion**: a predicate holds of some data elements if and only if it can be proved from the given sentences; in particular, two elements are identified if and only if they can be proved equal from the given sentences. (Two data items that are equal but cannot be proved so are "confused.")

For Horn clause logic, either with or without equality, either order-sorted, many-sorted, or unsorted, these two conditions define the data items ***uniquely*** up to renaming, i.e., they define a model up to isomorphism. Moreover, "no junk" is equivalent to structural induction over the signature, and the two conditions together are equivalent to the "unique homomorphism" condition called **initiality** (see [22] for details).

1.5 Constraints

In its general sense, a **constraint** is a logical relation that one wishes to impose on a set of potential solutions. In principle, such constraints could be arbitrary first order sentences involving arbitrary (interpreted and uninterpreted) relations; but in practice, constraints are limited to sets of atomic sentences, such as

$a * X + b * Y < c * Z + d$,

$a * X * X + b * X + c = 0$,

$a * X * Y \cup b * X + c \subseteq Z$,

where the variables in the first two constraints range over some kind of number (e.g., integers, or rationals, or complexes), and in the second range over sets of strings from some fixed alphabet ($*$ is multiplication in the first two, and is concatenation, extended to sets, in the third). Although Prolog would, in principle, be ideal for *constraint-based programming*, it does not suffice in practice, because of the limited capabilities of the built-in relations. Moreover, the usual semantic basis of Prolog does not extend to built-ins without some extra fuss and awkwardness (e.g., as in [19]).

We refer to sorts, functions, and relations upon which interpretation into a fixed (standard) model are imposed as **built-ins**. Two obvious examples of such models are numbers and infinite trees, with their usual functions and relations. The pioneering work of Jaffar and Lassez [19] and of Jaffar and Michaylov [20] treat these and a number of other examples, in the context of a constraint logic programming language called CLP. These authors also treat a number of other topics, some of which are not considered here, including negation as failure and compactness conditions [19].

1.6 Logical Programming

Various aspects of programming languages are captured by various aspects of logic. The functional aspect of programming is captured by equational logic [9]. Strong typing is captured by many-sorted logic. Logic programming (which might be less misleadingly called relational or Horn clause programming) is captured by Horn clause logic. Object-oriented programming is captured by reflective logic, in which there is an abstract data type of program texts built into the language [12]. The perspective of logical programming suggests that the right way to combine various programming paradigms is to discover their underlying logics, combine them, and then base a language upon the combined logic. This permits one to mix and match various programming language features. For example, combined functional and logic programming is captured by Horn clause logic with equality [11]. Combined functional and object-oriented programming is captured by reflective equational logic (we call this language FOOPS, see [12]). We currently feel that reflective order-sorted Horn clause logic with equality is a good candidate for unifying the functional, relational and object-oriented paradigms into a single simple programming language which also has powerful database capabilities.

The theory of institutions [10] can provide a formal basis for the notion of logical programming. Informally, an institution is a logical system, with formal notions of sentence, model, and satisfaction. Then, a **logical programming language** $\mathcal{L}$ has an associated logical system (i.e., institution) I such that:

- the statements of $\mathcal{L}$ are sentences from I;
- the operational semantics of $\mathcal{L}$ is (a reasonably efficient) deduction in I; and
- the denotational semantics of $\mathcal{L}$ is given by a class of models in I (preferably initial models).

2 Order-Sorted Algebra

The following assumes familiarity with S-sorted sets and functions (for S a family of sorts) and with many-sorted algebra signatures (S,Σ), Σ-algebras and Σ-homomorphisms [22], and generalizes these concepts to order-sorted algebra.

Definition 1: An **order-sorted signature** is a triple $(S,\leq,\Sigma)$ such that (S,Σ) is a many-sorted signature, $(S,\leq)$ is a partially ordered set[4], and the operators satisfy the following **monotonicity condition**[5]:

if $\sigma\in\Sigma_{w1,s1}\cap\Sigma_{w2,s2}$ and if $w1\leq w2$, then $s1\leq s2$.

When the sort set S is clear, we write Σ for (S,Σ). Similarly, when the partial order $(S,\leq)$ is clear, we write Σ for $(S,\leq,\Sigma)$. Also, we may write σ: $w\rightarrow s$ for $\sigma\in\Sigma_{w,s}$ to emphasize that σ denotes a function with **arity** w and co-arity (or **value sort**) s. An important special case is $w=\lambda$, the empty string; then $\sigma\in\Sigma_{\lambda,s}$ denotes a **constant** of sort s. □

Regular signatures allow one to define the least sort of a term, and to give an order-sorted generalization of the usual term algebra construction. Intuitively, regularity says that overloaded operators with argument sorts greater than a given sort string are consistent under the restriction of arguments to subsorts.

Definition 2: An order-sorted signature Σ is **regular** iff given $w0\leq w1$ in S^* and σ in $\Sigma_{w1,s1}$ there is a least $(w,s)\in S^*\times S$ such that $\sigma\in\Sigma_{w,s}$ and $w0\leq w$. If, in addition, each connected component[6] of the sort poset has a top element, then the regular signature is called **coherent**. □

Definition 3: Let $(S,\leq,\Sigma)$ be an order-sorted signature. Then an $(S,\leq,\Sigma)$-**algebra** is an (S,Σ)-algebra A such that

(1) $s\leq s'$ in S implies $A_s\subseteq A_{s'}$ and

(2) $\sigma\in\Sigma_{w1,s1}\cap\Sigma_{w2,s2}$ with $s1\leq s2$ and $w1\leq w2$ implies A_σ: $A_{w1}\rightarrow A_{s1}$ equals A_σ: $A_{w2}\rightarrow A_{s2}$ on A_{w1} .

When $(S,\leq)$ is clear, $(S,\leq,\Sigma)$-algebras may be called **order-sorted** Σ-**algebras**. We may write $A_\sigma^{w,s}$ instead of A_σ: $A_w\rightarrow A_s$. □

Definition 4: Let $(S,\leq,\Sigma)$ be an order-sorted signature, and let A and B be order-sorted $(S,\leq,\Sigma)$-algebras. Then a $(S,\leq,\Sigma)$-**homomorphism** h: $A\rightarrow B$ is a (S,Σ)-homomorphism satisfying the following **restriction** condition

$s\leq s'$ and $a\in A_s$ imply $h_s(a)=h_{s'}(a)$.

When the partially ordered set $(S,\leq)$ is clear, $(S,\leq,\Sigma)$-homomorphisms are also called **order-sorted** Σ-**homomorphisms**. The $(S,\leq,\Sigma)$-algebras and $(S,\leq,\Sigma)$-homomorphisms form a category that we denote $\mathbf{OSAlg}_\Sigma$. □

Order-sorted algebra *strictly generalizes* many-sorted algebra, in the sense that any many-sorted (S,Σ)-algebra is an order-sorted $(S,\leq,\Sigma)$-algebra for $\leq$ the trivial ordering on S, with $s\leq s'$ iff $s=s'$;

---

[4] We extend the ordering $\leq$ on S to strings of equal length in S^* by $s_1\ldots s_n\leq s'_1\ldots s'_n$ iff $s_i\leq s'_i$ for $1\leq i\leq n$. Similarly, $\leq$ extends to pairs (w,s) in $S^*\times S$ by $(w,s)\leq(w',s')$ iff $w\leq w'$ and $s\leq s'$.

[5] Although not needed for validity of our results, this very natural condition rules out some bizarre models.

[6] Given a poset $(S,\leq)$, let $\equiv$ denote the transitive and symmetric closure of $\leq$. Then $\equiv$ is an equivalence relation whose equivalence classes are the **connected componets** of $(S,\leq)$.

then $\mathbf{OSAlg}_\Sigma = \mathbf{Alg}_\Sigma$ and the forgetful functor $\mathbf{OSAlg}_\Sigma \to \mathbf{Alg}_\Sigma$ is the identity. Similarly, the rules of order-sorted deduction given later specialize to many-sorted rules for this ordering.

Definition 5: For $(S,\leq,\Sigma)$ an order-sorted signature and A an order-sorted Σ-algebra, an **order-sorted Σ-congruence** $\equiv$ on A is a many-sorted Σ-congruence $\equiv=\{\equiv_s \mid s\in S\}$ such that

for each $s,s'\in S$ and $a,a'\in A_s$, whenever $s\leq s'$ then $a\equiv_s a'$ iff $a\equiv_{s'} a'$.

□

Let f: A→B be an order-sorted Σ-homomorphism. Then the **kernel** of f is an S-sorted family of equivalence relations $\equiv_f$ defined by $a\equiv_{f,s} a'$ iff $f_s(a)=f_s(a')$ and denoted ker(f). It is easy to check that ker(f) is an order-sorted Σ-congruence. For $(S,\leq,\Sigma)$ a coherent order-sorted signature, A an order-sorted Σ-algebra, and $\equiv$ an order-sorted Σ-congruence on A, the **quotient** of A by $\equiv$ is the order-sorted Σ-algebra $A/\equiv$ defined as follows: for each sort s, the carrier of maximal sort in its connected component is $(A/\equiv)_{max(s)} = A_{max(s)}/\equiv_{max(s)}$ and the carrier of sort s is $(A/\equiv)_s = q_{max(s)}(A_s)$, for $q_{max(s)}: A_{max(s)} \to (A/\equiv)_{max(s)}$ the natural projection of each element a into its $\equiv_{max(s)}$-equivalence class. The operations are defined by: $(A/\equiv)_\sigma([a1],...,[an]) = [A_\sigma(a1,...,an)]$, which is well-defined since $\equiv$ is an order-sorted Σ-congruence. The order-sorted algebra $A/\equiv$ has a natural surjective order-sorted Σ-homomorphism q: A→A/$\equiv$ defined by restricting the $q_{max(s)}$ to the remaining sorts and such that ker(q) = $\equiv$. This homomorphism is called the **quotient map** associated to the congruence $\equiv$.

Lemma 6: Universal Property of Quotients: Let Σ be a coherent order-sorted signature, A an order-sorted Σ-algebra and $\equiv$ an order-sorted congruence on A. Then the quotient map q: A→A/$\equiv$ satisfies the following universal property: If f: A→B is any order-sorted Σ-homomorphism such that $\equiv\subseteq$ker(f), then there is a unique Σ-homomorphism u: A/$\equiv$→B such that uoq=f. □

Given an order-sorted signature Σ, we construct the **order-sorted Σ-term algebra** T_Σ as the least family $\{T_{\Sigma,s} \mid s\in S\}$ of sets satisfying the following conditions (note our somewhat pedantic, but temporary, use of ⌊ and ⌋ for parentheses as formal syntactic symbols):

- $\Sigma_{\lambda,s}\subseteq T_{\Sigma,s}$ for $s\in S$;
- $T_{\Sigma,s'}\subseteq T_{\Sigma,s}$ if $s'\leq s$;
- if $\sigma\in\Sigma_{w,s}$ and if $ti\in T_{\Sigma,si}$ where $w=s1...sn\neq\lambda$, then (the string) $\sigma⌊t1...tn⌋\in T_{\Sigma,s}$.
- Also, for $\sigma\in\Sigma_{w,s}$ let $T_\sigma: T_w\to T_s$ send t1,...,tn to (the string) $\sigma⌊t1...tn⌋$.

These terms are ground terms, i.e., they involve no variables. Terms with variables arise from ground terms by enlarging the signature Σ with additional constants for the variables. We assume that each variable comes with a unique associated sort, so that variables form an S-sorted set X with $X_s\cap X_{s'}=\emptyset$ if $s\neq s'$. This gives an extended signature $\Sigma(X)$ (that is also regular) and an algebra $T_{\Sigma(X)}$ also denoted $T_\Sigma(X)$. Although a term in an order-sorted term algebra may have many different sorts, we still have

Fact 7: For Σ a regular signature, each term t in T_Σ has a least sort, denoted LS(t). □

Definition 8: Let Σ be an order-sorted signature. Then an order-sorted Σ-algebra I is **initial** in the class of all order-sorted Σ-algebras iff there is a unique order-sorted Σ-homomorphism from I to any other order-sorted Σ-algebra. □

Theorem 9: Let Σ be a regular order-sorted signature. Then T_Σ is an initial order-sorted Σ-algebra. □

Fact 10: Let A be an order-sorted Σ-algebra, X an S-sorted set of variables, and let f: X→A be an S-sorted function (with X disjoint from Σ); such a function is called an **assignment** from X to A. Then (by initiality) there is a unique order-sorted Σ-homomorphism $f^*: T_\Sigma(X)\to A$ that extends f. □

This fact is usually expressed by saying that $T_\Sigma(X)$ is the **free** order-sorted Σ-algebra on X.

For $(S,\leq,\Sigma)$ a coherent order-sorted signature and X,Y two S-sorted variable sets, a **substitution** is an S-sorted map θ: $X \to T_\Sigma(Y)$; this is a special assignment, where the values of variables lie in a term algebra. The unique order-sorted Σ-homomorphism θ^*: $T_\Sigma(X) \to T_\Sigma(Y)$ that extends θ is also be denoted θ.

3 Order-Sorted Model Theory

Order-sorted models generalize order-sorted algebras by allowing predicates instead of just function symbols, and order-sorted Horn clause logic allows arbitrary Horn clauses instead of just conditional equations. Order-sorted Horn clause logic has a number of advantages over unsorted logic; in general, these advantages are extensions of the advantages of order-sorted algebra over unsorted algebra, such as the great expressive power given by overloading and strong but still flexible typing, and the capacity to handle errors. An additional advantage of order-sorted Horn clause logic that has great practical importance is that sometimes the search space and required time for theorem proving can by drastically cut by making use of subsorts, and shown, for example, by work of Walther [27] on the steamroller example.

This section first extends the basic notions from algebras to models, and then gives rules of deduction for order-sorted Horn clause logic with equality, with a proof of their completeness. We then construct the initial model associated to a set of Horn clauses, and prove Herbrand's theorem, both in general, and for the special case where all sorts are non-empty. Finally, we prove the existence of free extensions, and apply this to built-ins.

3.1 Basic Definitions

Definition 11: An **order-sorted signature with predicates** is a quadruple $(S,\leq,\Sigma,\Pi)$ such that $(S,\leq,\Sigma)$ is a coherent order-sorted signature, and $\Pi=\{\Pi_w \mid w\in S^+\}$ is a family of **predicate symbols** satisfying the conditions below. We use capital letters $P,Q,P_1,P_2,\ldots$ to denote predicate symbols. We may write "P: w" for $P\in\Pi_w$ and then call w an **arity** of P. As with function symbols, we allow overloading of predicate symbols, i.e., P can have several arities. We assume the following two conditions:

1. There is an equality predicate symbol $=$ of arity ss in Π iff s is the top of a connected component of the sort poset S.
2. Regularity: For each w0 such that there is a P: w1 with $w0\leq w1$, there is a least w such that P: w and $w0\leq w$.

When the sort set S and partial order $\leq$ are clear, we write Σ,Π for $(S,\leq,\Sigma,\Pi)$. □

Definition 12: Let $(S,\leq,\Sigma,\Pi)$ be an order-sorted signature with predicates. Then an $(S,\leq,\Sigma,\Pi)$-**model** is an order-sorted $(S,\leq,\Sigma)$-algebra M together with an assignment to each P: w in Π of a subset ${M^w}_P \subseteq M_w$ such that:

(i) for P the identity predicate $=$: ss the assignment is the identity relation, i.e., $(M^{ss}{}_{=})=\{(a,a) \mid a\in M_s\}$; and

(ii) whenever P: w1 and P: w2 and $w1\leq w2$ then ${M^{w1}}_P = M_{w1} \cap {M^{w2}}_P$.

□

Definition 13: Let Σ,Π be an order-sorted signature with predicates and let M and M′ be order-sorted Σ,Π-models. Then a Σ,Π-**homomorphism** h: $M \to M'$ is an order-sorted Σ,Π-homomorphism such that for each P: w in Π,

$f_w(a_1,...,a_n) \in M'^w_P$ whenever $(a_1,...,a_n) \in M^w_P$.
The Σ,Π-models and Σ,Π-homomorphisms form a category that we denote $\mathbf{Mod}_{\Sigma,\Pi}$. □
Order-sorted model theory *strictly generalizes* many-sorted model theory, in the sense that any many-sorted (S,Σ,Π)-model is an order-sorted $(S,\leq,\Sigma,\Pi)$-model for $\leq$ the trivial ordering on S, with $s \leq s'$ iff $s=s'$. Similarly, the rules of order-sorted deduction given later specialize to rules for many-sorted deduction for the trivial ordering.

Given an order-sorted signature Σ,Π with predicates, we can define the Σ,Π-model $\mathcal{T}_{\Sigma,\Pi}$ consisting of the Σ-term algebra $\mathcal{T}_\Sigma$ and having $(\mathcal{T}_{\Sigma,\Pi}{}^w)_P = \emptyset$ for each P: w in Π different from the identity predicate (= is of course interpreted as actual identity). For X an S-sorted set of variables disjoint form Σ, the model $\mathcal{T}_{\Sigma,\Pi}(X)$ is defined in a similar way by enlarging Σ with the constants in X. It is then immediate to generalize to models the notion of an initial algebra and to check the following:
Theorem 14: Let Σ,Π be an order-sorted signature with predicates. Then $\mathcal{T}_{\Sigma,\Pi}$ is an initial order-sorted Σ,Π-model. □
Fact 15: Let M be an order-sorted Σ,Π-model, X an S-sorted set of variables disjoint from Σ, and let f: X→M be an S-sorted assignment. Then, by initiality, there is a unique order-sorted Σ,Π-homomorphism f*: $\mathcal{T}_{\Sigma,\Pi}(X) \to M$ that extends f. This fact is usually expressed by saying that $\mathcal{T}_{\Sigma,\Pi}(X)$ is the **free** Σ,Π-model on X. □

We now introduce additional notation for atoms and Horn clauses. For Σ,Π an order-sorted signature with predicates we define Σ,Π-**atoms** as expressions $P(t_1,...,t_n)$ such that the $t_1,...,t_n$ are Σ-terms (possibly with variables) and there is a $w=s_1...s_n$ with P: w in Π such that t_i has sort s_i for i=1,...,n. Thus, an equation $t=t'$ is an atom where P is the identity relation. We will use symbols $A,B,A_1,A_2,...$ for atoms. For θ a substitution and $A=P(t_1,...,t_n)$ an atom, θA denotes the atom $A=P(\theta[t_1],...,\theta[t_n])$. We interpret a *set* of atoms $\{A_1,...,A_n\}$ as a conjunction. Although we may drop the curly brackets and write $A_1,...,A_n$ we still assume that this denotes a set, so that the order does not matter and there are no repeated atoms. The empty set of atoms has the empty notation, i.e., it is denoted as a blank. By a Σ,Π-**Horn clause** we mean an expression

$$(\forall X)\ A \Leftarrow B_1,...,B_m$$

where X contains all the variables occurring in all the terms in the atoms $A,B_1,...,B_m$. We call A the **head** and the set $B_1,...,B_m$ the **body**. When the body is empty, the Horn clause is called an **atomic** (universal) **sentence** and the notation "$(\forall X)\ A \Leftarrow$" is abbreviated to "$(\forall X)\ A$." When all atoms have the equality predicate as their predicate symbol, the Horn clause is called a **conditional equation**; if, in addition, the body is empty it is called an **equation**.
Definition 16: For Σ,Π an order-sorted signature with predicates, M a Σ,Π-model and $(\forall X)\ A \Leftarrow B_1,...,B_m$ a Σ,Π-Horn clause, we say that M **satisfies** $(\forall X)\ A \Leftarrow B_1,...,B_m$ if (assuming $A=Q(t_1,...,t_n)$, $B_i=P_i(t_{i1},...,t_{in_i})$) for any assignment f: X→M such that[7]

$$(f^*(t_{i1}),...,f^*(t_{in_i})) \in M^{wi}_{P_i} \text{ for } i=1,...,m,$$

then also

$$(f^*(t_1),...,f^*(t_n)) \in M^w_Q.$$

Similarly, for $\mathcal{C}$ a set of horn clauses we say that M **satisfies** $\mathcal{C}$ iff it satisfies each clause in $\mathcal{C}$; such a model is then called a $\Sigma,\Pi,\mathcal{C}$-**model**, and the category of all such models is denoted $\mathbf{Mod}_{\Sigma,\Pi,\mathcal{C}}$. □

---

[7] Note that, by regularity, the least arity wi of P_i is uniquely defined.

3.2 Order-Sorted Deduction

This section presents a new completeness theorem for order-sorted Horn clause logic with equality. This theorem provides a basis for the correctness of implementations for languages like Eqlog [11], as well as the basis for our development of a semantic theory for constraint languages.

Given an order-sorted signature Σ,Π with predicates and a set C of Σ,Π-Horn clauses, the following are the rules for deriving atomic sentences:

(1) Reflexivity: Each equation $(\forall X)\ t=t$ is derivable.

(2) Symmetry: If $(\forall X)\ t=t'$ is derivable, then so is $(\forall X)\ t'=t$.

(3) Transitivity: If the equations $(\forall X)\ t=t'$, $(\forall X)\ t'=t''$ are derivable, then so is $(\forall X)\ t=t''$.

(4) Congruence: If $t_i,t'_i \in T_\Sigma(X)_{s_i}$ for i=1,...,n, and if $(\forall X)\ t_i=t'_i$ is derivable for i=1,...,n, then:

- for any σ: $s_1...s_n \to s$ in Σ the equation $(\forall X)\ \sigma(t_1,...,t_n)=\sigma(t'_1,...,t'_n)$ is derivable;
- for any P: $s_1...s_n$ in Π (other than the identity predicate) if the sentence $(\forall X)\ P(t_1,...,t_n)$ is derivable, then so is $(\forall X)\ P(t'_1,...,t'_n)$.

(5) Modus Ponens: If $(\forall X)\ A \Leftarrow B_1,...,B_m$ is in C and if θ: $X \to T_\Sigma(Y)$ is a substitution such that for each B_i in the body of the clause the atomic sentence $(\forall Y)\ \theta B_i$ is derivable, then so is $(\forall Y)\ \theta A$.

For X an S-sorted set of variables, we define a Σ,Π-model $T_{\Sigma,\Pi,C}(X)$ as follows: First, notice that for t and t' terms of the same sort, the property

$(\forall X)\ t=t'$ is derivable from C using rules (1)-(5)

defines an order-sorted Σ-congruence $\sim^C_{(X)}$ on $T_\Sigma(X)$ since, by rules (1)-(4), $\sim^C_{(X)}$ is a many-sorted Σ-congruence relation and, in addition, $\sim^C_{(X)}$ is an order-sorted Σ-congruence relation, since for any sort s such that $t, t' \in T_\Sigma(X)_s$ we have $t \sim^C_{(X),s} t'$ iff $(\forall X)\ t=t'$ is derivable from C using rules (1)-(5), a property that does not depend on s. Thus, we can define an order-sorted Σ-algebra $T_{\Sigma,C}(X)$ as the quotient of $T_\Sigma(X)$ by the order-sorted congruence $\sim^C_{(X)}$. We can then give a Σ,Π-model structure $T_{\Sigma,\Pi,C}(X)$ to $T_{\Sigma,C}(X)$ by defining $([t_1],...,[t_n]) \in T_{\Sigma,\Pi,C}(X)^w{}_P$ iff $(\forall X)\ P(t_1,...,t_n)$ is provable from C by (1)-(5) (where $w=s_1,...,s_n$ and t_i has sort s_i). This definition is independent of the representatives t_i by (4). We can now prove

Lemma 17: $T_{\Sigma,\Pi,C}(X)$ satisfies C.

Proof: Let $(\forall Y)\ A \Leftarrow B_1,...,B_m$ be a Horn clause in C (with, say, $A=Q(t_1,...,t_n)$, $B_i=P_i(t_{i1},...,t_{in_i})$) and let θ_0: $Y \to T_{\Sigma,\Pi,C}(X)$ be an S-sorted assignment such that $(\theta_0{}^*(t_{i1}),...,\theta_0{}^*(t_{in_i})) \in T_{\Sigma,\Pi,C}(X)^{wi}{}_{P_i}$ for i=1,...,m. By choosing a representative $t \in T_\Sigma(X)_s$ for each $[t]=\theta_0(y)$, $y \in Y_s$, we then obtain a substitution θ: $Y \to T_\Sigma(X)$ such that, by hypothesis (and by initiality of $T_{\Sigma,\Pi}(Y)$ making $\theta_0{}^*=q\circ\theta$ for q the quotient homomorphism q: $T_{\Sigma,\Pi}(X) \to T_{\Sigma,\Pi,C}(X)$) we have $(\forall X)\ \theta B_i$ derivable from C using rules (1)-(5) for i=1,...,m. Thus, by (5) we then have $(\forall X)\ \theta A$ derivable from C using rules (1)-(5), i.e., we have $(\theta_0{}^*(t_1),...,\theta_0{}^*(t_n)) \in T_{\Sigma,\Pi,C}(X)^{wi}{}_Q$ as desired. □

Theorem 18: Completeness Theorem: For Σ,Π an order-sorted signature, C a set of Σ,Π-Horn clauses and $(\forall X)\ A$ a Σ,Π-atomic sentence, the following are equivalent:

- $(\forall X)\ A$ is derivable from C using rules (1)-(5).
- $(\forall X)\ A$ is satisfied by all order-sorted Σ-algebras that satisfy C .

Proof: We leave the reader to check soundness, i.e., that the first assertion implies the second, and concentrate here on proving completeness, i.e., that any sentence that is satisfied is provable. Let $(\forall X)\ P(t_1,...,t_n)$ be a sentence satisfied by all Σ,Π,C-models. Then, it is satisfied by $T_{\Sigma,\Pi,C}(X)$. In particular, for the assignment q: $X \to T_{\Sigma,\Pi,C}(X)$ associated to the quotient homomorphism q: $T_{\Sigma,\Pi}(X) \to T_{\Sigma,\Pi,C}(X)$) we have

$([t_1],...,[t_n])\in T_{\Sigma,\Pi,C}(X)^w{}_P$
which by definition just means that the sentence $(\forall X)\ P(t_1,...,t_n)$ is derivable from C with rules (1)-(5).
□

Corollary 19: Initiality Theorem: For Σ,Π an order-sorted signature with predicates and C a set of Σ,Π-Horn clauses, $T_{\Sigma,\Pi,C}(\emptyset)$ (henceforth denoted $T_{\Sigma,\Pi,C}$) is an initial model in the class of all Σ,Π,C-models, and $T_{\Sigma,\Pi,C}(X)$ is a free model on X in that same class.

Proof: Let M be an order-sorted model satisfying C, and let $\underline{a}$: X→M be an assignment. We have to show that there is a unique order-sorted Σ,Π-homomorphism $\underline{a}^{\&}$: $T_{\Sigma,\Pi,C}(X)\to M$ extending $\underline{a}$, i.e., such that $\underline{a}^{\&}(q(x))=\underline{a}(x)$ for each $x\in X$, where q denotes the quotient homomorphism q: $T_{\Sigma,\Pi}(X)\to T_{\Sigma,\Pi,C}(X)$. Existence of $\underline{a}^{\&}$ follows from the Completeness Theorem, by (i) and (ii) below:

(i) $\underline{a}^*(t)=\underline{a}(t')$ for each equation $(\forall X)\ t=t'$ provable from C by the rules (1)-(5).

This means that $\sim^{C}_{(X)}\subseteq\ker(\underline{a}^*)$ and by the universal property of quotients (Lemma 6) there is a unique order-sorted Σ-homomorphism $\underline{a}^{\&}$: $T_{\Sigma,C}(X)\to M$ with $\underline{a}^*=\underline{a}^{\&}\circ q$.

(ii) $([t_1],...,[t_n])\in T_{\Sigma,\Pi,C}(X)^w{}_P$ iff (for representatives t_i of sort s_i, with $w=s_1,...,s_n$) $(\forall X)\ P(t_1,...,t_n)$ is provable from C by the rules (1)-(5) iff (by the Completeness Theorem) $(\forall X)\ P(t_1,...,t_n)$ holds in all models that satisfy C.

Thus, $(\underline{a}^{\&}[t_1],...,\underline{a}^{\&}[t_n])=(\underline{a}^*(t_1),...,\underline{a}^*(t_n))\in M^w{}_P$ which shows that $\underline{a}^{\&}$ is a Σ,Π-homomorphism. Uniquenes of $\underline{a}^{\&}$ then follows by combining the universal property of $T_{\Sigma,\Pi}(X)$ as a free order-sorted model on X with the universal property of q as a quotient. Indeed, let $\underline{a}^{\$}$: $T_{\Sigma,\Pi,C}(X)\to M$ be another order-sorted homomorphism such that $\underline{a}^{\$}(q(x))=\underline{a}(x)$ for each $x\in X$. Then, since $T_{\Sigma,\Pi}(X)$ is a free order-sorted model on X, we have $\underline{a}^*=\underline{a}^{\$}\circ q$ and by the universal property of q as a quotient we have $\underline{a}^{\$}=\underline{a}^{\&}$ as desired. For $X=\emptyset$ the empty S-sorted family, this proves that $T_{\Sigma,\Pi,C}(\emptyset)=T_{\Sigma,\Pi,C}$ is an initial model in the class of models that satisfy C. □

3.3 Herbrand's Theorem

Usual logic programming practice considers queries of the form $B_1,...,B_n$. Such queries are answered positively if a substitution θ is found such that each θB_i is provable from the clauses C that constitute the program. Such a query is simply an existential formula of the form $(\exists X)\ B_1,...,B_n$ and what is being established is that the formula holds for all models that satisfy C. Herbrand's theorem reduces provability of such a formula to satisfaction in the initial model. This subsection extends the rules of order-sorted deduction to handle existential quantification of conjunctions of atoms by giving a very simple proof of a general Herbrand's theorem that hold for any order-sorted Horn clause logic with equality specification (Σ,Π,C). Before the advent of resolution in the theorem-proving literature [25], the methods used to prove that an existential formula was satisfied were brute-force, "saturation" methods that enumerated all possible elements of the initial model until a successful instance was found. Assuming that all the sorts of the initial model are nonempty, we can give simpler rules of deduction that dispense with explicit quantification, and we can prove a second version of Herbrand's Theorem. This second version provides a foundation for proofs by resolution in the extended context of order-sorted logic with equality.

Definition 20: For Σ,Π an order-sorted signature with predicates, we call an existential formula of the form $(\exists X)\ B_1,...,B_n$ where X is an S-sorted variable set that contains all variables that appear in terms of the Σ,Π-atoms $B_1,...,B_n$ a Σ,Π-**existential conjunction (of atoms)**. Given a Σ,Π-model M, we say that M **satisfies** the above existential conjunction iff there is an assignment $\underline{a}$: X→M, called a **witness** such that (assuming, say, $B_i=P_i(t_{i1},...,t_{in_i})$ with P: $s_1...s_n$ and t_i of sort s_i) we have
$(\underline{a}^*(t_1),...,\underline{a}^*(t_n))\in M^w{}_P$. □

Theorem 21: Herbrand's Theorem: For Σ,Π an order-sorted signature with predicates, C a set of Σ,Π-Horn clauses, and $(\exists X)\ B_1,...,B_n$ a Σ,Π-existential conjunction, the following are equivalent:

- $(\exists X)\ B_1,...,B_n$ is satisfied by all Σ,Π,C-models.
- $(\exists X)\ B_1,...,B_n$ is satisfied by the initial Σ,Π,C-model $T_{\Sigma,\Pi,C}$.

Proof: Of course, if $(\exists X)\ B_1,...,B_n$ is satisfied by all Σ,Π,C-models, it is satisfied by the initial one. Conversely, if θ: $X \to T_{\Sigma,\Pi,C}$ is a witness for $(\exists X)\ B_1,...,B_n$, in $T_{\Sigma,\Pi,C}$ and if M is a Σ,Π,C-model with h: $T_{\Sigma,\Pi,C} \to M$ the unique Σ,Π,C-homomorphism, then it follows from the homomorphic property of h that $h\circ\theta$ is a witness for $(\exists X)\ B_1,...,B_n$ in M. □

Corollary 22: Adding the following rule to the rules (1)-(5) of order-sorted deduction gives a sound and complete set of rules to derive, for C a set of Σ,Π-Horn clauses, all valid universal atomic formulas and all valid existential conjunctions of atoms.

(6) Existential Introduction: If, for $1=1,..,n$, $(\forall \emptyset)\theta B_i$ is derivable, with θ: $X \to T_\Sigma$, then $(\exists X)\ B_1,...,B_n$ is derivable.

□

So far nothing in this paper has required that all sorts of a model are nonempty. There are good reasons for not imposing such a restriction on the models. Indeed, excluding empty sorts one would in general fail to have initial models, and parameterization would be awkward [13]. Nevertheless, in the context of existential quantification there are definite advantages in the case where models do not have empty sorts. This happens exactly when all the sorts of the initial algebra T_Σ are nonempty, for Σ the order-sorted signature of function symbols in question; we will then say that the order-sorted signature with predicates Σ,Π has **nonempty sorts**. We will show that the rules of deduction become simpler in this case. We first prove a lemma that holds for any signature at all. The simplest proof of this lemma would be a model-theoretic soundness proof. However, we prefer to give a proof-theoretic proof that shows explicitly why the substitution rule is not needed.

Lemma 23: Substitution Lemma: For Σ,Π an order-sorted signature with predicates and C a set of Σ,Π-Horn clauses,

Substitution: If $(\forall X)$ A is derivable from C by (1)-(5) and if θ: $X \to T_\Sigma(Y)$ is a substitution, then $(\forall Y)\ \theta A$ is also derivable.

Proof: The derivable formulas can be obtained in a countable number of stages, with stage n+1 adding the new formulas that can be obtained from the those in stage n by (1)-(5) using C. In the first stage, we just have all equations $(\forall X)$ t=t (by reflexivity) and all atomic sentences $(\forall Y)\ \theta A$ for each $(\forall X)$ A in C and arbitrary $\theta X \to T_\Sigma(Y)$ (by Modus Ponens). Consequently, the first stage is trivially closed under substitution. Let us assume as our induction hypothesis that stage n is closed under substitution; we only need to show that stage n+1 also is:

- If $(\forall X)$ $t=t'$ is in stage n, then, by symmetry, $(\forall X)$ $t'=t$ is in stage n+1 and so is $(\forall Y)$ $\theta(t')=\theta(t)$ since $(\forall Y)$ $\theta(t)=\theta(t')$ is in stage n by hypothesis.
- If the equations $(\forall X)$ $t=t'$, $(\forall X)$ $t'=t''$ belong to stage n, then by transitivity, $(\forall X)$ $t=t''$ belongs to stage n+1 and so is $(\forall Y)$ $\theta(t)=\theta(t'')$ since $(\forall Y)$ $\theta(t)=\theta(t')$, $(\forall Y)$ $\theta(t')=\theta(t'')$ are in stage n by hypothesis.
- If $t_i,t'_i \in T_\Sigma(X)_{s_i}$, $i=1,...,k$, and if the equations $(\forall X)$ $t_i=t'_i$ belong to stage n for $i=1,...,k$, then by congruence:
  - for any σ: $s_1...s_k \to s$ in Σ the equation $(\forall X)$ $\sigma(t_1,...,t_k)=\sigma(t'_1,...,t'_k)$ belongs to stage n+1 and so does the equation $(\forall Y)$ $\sigma(\theta(t_1),...,\theta(t_k))=\sigma(\theta(t'_1),...,\theta(t'_k))$, by applying the induction hypothesis to the original equations;

- for any P: $s_1...s_k$ in Π (other than the identity predicate) if the sentence $(\forall X)\ P(t_1,...,t_k)$ belongs to stage n, then $(\forall X)\ P(t'_1,...,t'_k)$ belongs to stage n+1 and so does $(\forall Y)\ P(t'_1,...,t'_k)$ by applying the induction hypothesis to the original equations and to $(\forall X)\ P(t_1,...,t_k)$.

- If $(\forall X)\ A \Leftarrow B_1,...,B_m$ is in C, and if θ: $X \to T_\Sigma(Y)$ is a substitution such that for each B_i in the body of the clause the atomic sentence $(\forall Y)\ \theta B_i$ belongs to stage n, then, by Modus Ponens, $(\forall Y)\ \theta A$ belongs to stage n+1 and so does $(\forall Z)\ \rho\theta A$ by applying the induction hypothesis to the sentences $(\forall Y)\ \theta B_i$ i=1,...,m.

□

Using the Substitution Lemma we can now establish the following version of Herbrand's Theorem for signatures with nonempty sorts:

Corollary 24: Herbrand's Theorem for Nonempty Sorts: For Σ,Π an order-sorted signature with predicates and nonempty sorts, C a set of Σ,Π-Horn clauses, and $(\exists X)\ B_1,...,B_n$ a Σ,Π-existential conjunction the following are equivalent:

- $(\exists X)\ B_1,...,B_n$ is satisfied by all Σ,Π,C-models.
- There is a substitution θ: $X \to T_\Sigma(Y)$ such that $(\forall Y)\ \theta B_i$ is provable from C using (1)-(5) for i=1,...,n.

Proof: By Herbrand's Theorem we only have to prove that the second condition is satisfied by some substitution θ: $X \to T_\Sigma(Y)$ iff it is satisfied by some ground term substitution θ': $X \to T_\Sigma$. This is so since, by the nonempty sort assumption, we can always define a substitution ρ: $Y \to T_\Sigma$ and, by the Substitutivity Lemma, we can satisfy the condition with $\theta' = \rho\theta$. □

Nonempty sorts allow dispensing with quantification over sets of variables. This also follows easily from the Substitutivity Lemma, by the following corollary, which says that the choice of a superset X of the set of variables occurring in an atom A is immaterial for the validity of the universally quantified sentence:

Corollary 25: For Σ,Π an order-sorted signature with predicates and nonempty sorts, for A a Σ,Π-atom, and C a set of Σ,Π-Horn clauses, then

$(\forall X)$ A is provable from C using (1)-(5) iff $(\forall vars(A))$ A is also,

where vars(A) is the S-sorted set of variables occurring in A (and, of course, $vars(A) \subseteq X$).

Proof: The "if" part is clear, using rule of substitution applied to the inclusion $vars(A) \subseteq X \subseteq T_\Sigma(X)$. For the "only if" part, by the nonempty sort assumption one can always define a substitution $X \to T_\Sigma(vars(A))$ that is the identity on vars(A) by sending the variables in X-vars(A) to ground terms. □

We finish this subsection with a simplified and generalized set of sound and complete proof rules for the case of nonempty sorts. We leave the universal quantification implicit for atoms and Horn clauses, and treat not only atomic sentences, but also conjunctions $B_1,...,B_n$. Similarly, we simplify existential quantification by omitting the set of variables, since our last corollary, combined with Herbrand's Theorem for nonempty sorts, shows that choice of a superset of variables is also immaterial for existential quantification; we just write $\exists\ B_1,...,B_n$. Here are the rules of deduction for Σ,Π an order-sorted signature with predicates and nonempty sorts and C a set of Σ,Π-Horn clauses to derive (implicitly universal) conjunctions of atoms and existential conjunctions of atoms:

(1) Reflexivity: Each equation t=t is derivable.

(2) Symmetry: If t=t' is derivable, then so is t'=t.

(3) Transitivity: If $t=t', t'=t''$ is derivable, then so is $t=t''$.

(4) Congruence: If $t_i, t'_i \in T_\Sigma(X)_{s_i}$ for i=1,...,n, and if $t_1=t'_1,...,t_n=t'_n$ are derivable then:

- for any σ: $s_1...s_n \rightarrow s$ in Σ the equation $\sigma(t_1,...,t_n)=\sigma(t'_1,...,t'_n)$ is derivable;
- for any P: $s_1...s_n$ in Π (other than the identity predicate) if $P(t_1,...,t_n)$ is derivable, then so is $P(t'_1,...,t'_n)$.

(5) Modus Ponens: If $A \Leftarrow B_1,...,B_m$ is in C, and if θ: $X \rightarrow T_\Sigma(Y)$ is a substitution such that $\theta B_1,...,\theta B_n$ is derivable, then so is θA.

(6) Conjunction Introduction: If B_1 and ... and B_m are derivable, then $B_1,...,B_m$ is derivable.

(7) Existential Introduction: If $\theta B_1,...,\theta B_n$ is derivable for some substitution θ, then $\exists\ B_1,...,B_m$ is derivable.

By assigning extra variables from X to elements of a model M denoted by ground terms it is easy to check that, for signatures with nonempty sorts, satisfaction of a Horn clause or an existential conjunction of atoms does not depend on the superset of variables being quantified, and can thus be defined for implicitly quantified sentences. Satisfaction of a conjunction $B_1,...,B_m$ is defined as simultaneous satisfaction of each one of its conjuncts. We then obtain,

Theorem 26: Completeness Theorem for nonempty sorts: For Σ,Π an order-sorted with predicates and nonempty sorts, for C a set of Σ,Π-Horn clauses and S a (impicitly universal) conjunction $B_1,...,B_m$ of Σ,Π-atoms, or an existential conjunction of Σ,Π-atoms $\exists\ B_1,...,B_m$ the following are equivalent:

- $(\forall X)$ S is derivable from C using rules (1)-(7).
- $(\forall X)$ S is satisfied by all order-sorted Σ-algebras that satisfy C .

□

The rules (1)-(7) are the foundation for the much more efficient operational semantics rules for Eqlog, which is based on term rewriting, narrowing, order-sorted unification, and resolution. Since Eqlog computation is restricted to objects, and not allowed for theories, we can allow nonempty sorts in theories for convenience with parameterization, while still requiring nonempty sorts for the executable parts. For example, the theory **POSET** of partially ordered sets can have an empty **Element** sort, but any instantiation along a view from **POSET** to an object should map **Element** to a nonempty sort.

3.4 Operational semantics

The two sets of rules of deduction that we have given, one fully general, and the other for the case where all sorts are non-empty, are not in themsleves efficient enough to implement a langauge that would really be used for programming, although the second is an improvement on the first. Rather, the purpose of these rule sets is to support theoretical developments, such as completeness and Herbrand theorems, and to serve as a bridge to a really efficient operational semantics. Thus, our view is that an operational semantics for a logical programming language is just an efficient proof theory for its logic. For example, order-sorted rewriting [16] provides an efficient proof theory for order-sorted equational logic, and thus for OBJ [9], just as ordinary term rewriting provides an efficient proof theory for ordinary equational logic. Similarly, Horn clause resolution (with some tricks for the incremental accumulation of substitutions, etc.) provides an efficient operational semantics for ordinary Horn clause logic, and thus for Prolog. Finally, the efficient operational semantics for Eqlog [11] is a combination of order-sorted resolution with order-sorted narrowing (which itself is a combination of rewriting and unification); our joint work with Gert Smolka on the order-sorted

operational semantics of Eqlog [26], and more specifically on order-sorted unification [23], spells all this out in detail.

3.5 Free Extensions

So far we have considered initial and free models for order-sorted Horn clause logic with equality. However, initiality and freeness can be considered in a relative way: given a model of a subspecification, can we extend it to a model of a superspecification so that the extension is initial or free? This is a question naturally associated with the semantics of parameterized modules, where a model of a (parameter) requirement theory is freely extended to a model of of the theory of the "body." Free extensions are also associated with models that are partly built-in and partly interpreted as initial models of a set of axioms (i.e., a program) in that these models are just free extensions of their built-in parts. This subsection proves that such free extensions exist and can be obtained by a proof-theoretic construction. With slight variations to accomodate changes in syntax and possibly noninjective maps between sorts, our proof generalizes straightforwardly to give a construction of free extensions along a specification morphism (called a view in Eqlog and OBJ) V: $(\Sigma,\Pi,C)\rightarrow(\Sigma',\Pi',C')$, i.e., to a proof that order-sorted Horn clause logic with equality is a *liberal* institution (in the sense of [10]).

Consider a containment of order-sorted Horn clause logic with equality specifications J: $(\Sigma,\Pi,C)\subseteq(\Sigma',\Pi',C')$, i.e., $\Sigma\subseteq\Sigma'$, $\Pi\subseteq\Pi'$, and $C\subseteq C'$. Then any Σ',Π',C'-model M' can be regarded as a Σ,Π,C-model just by forgetting the operations in Σ-Σ' and the predicates in Π-Π'; of course, this process of "forgetting" is a functor $\mathbf{Mod}_{\Sigma',\Pi',C'}\rightarrow\mathbf{Mod}_{\Sigma,\Pi,C}$. Intuitively, by an "extension" of a Σ,Π,C-model M we probably understand a Σ',Π',C'-model M' such that M is a Σ,Π,C-submodel of M'. However, if we insist on the submodel relation, such an M' in general may not exist, due to the fact that C' (having more axioms than C) may force identifying different elements of M. Thus, it is better to generalize our intuitive notion and just require that M and M' are linked by a Σ,Π-homomorphism f: M$\rightarrow$M', i.e., that M' contain an *image* of M. In this way we obtain a category $\mathbf{Ext}_J(M)$ with objects all such extensions f: M$\rightarrow$M' and with morphisms h: (f: M$\rightarrow$M')$\rightarrow$(f': M$\rightarrow$M'') those Σ',Π'-homomorphisms h that preserve the image of M, i.e., such that f'=hof. The existence problem for free extensions can then be made precise by asking the following question:

Does $\mathbf{Ext}_J(M)$ *have an initial object?*

This is exactly the same question as whether the forgetful functor $\mathbf{Mod}_{\Sigma',\Pi',C'}\rightarrow\mathbf{Mod}_{\Sigma,\Pi,C}$ has a left adjoint [21]. The answer is *Yes*, and we denote such an initial object by $F_J(M)$. We will actually show that $\mathbf{Ext}_J(M)$ can be axiomatized as a class of models definable by Horn clauses, and since we have already proved that those classes have initial models the result then follows directly. We need only construct the signature $\Sigma'(M)$ that adds the elements of M disjointly as new constants, and introduce the notion of the **positive diagram** of the model M, denoted $\mathrm{Diag}^+(M)$, which is just the set of $\Sigma(M),\Pi$-sentences of the form $(\forall\emptyset)\ \sigma(a1,...,an)=M^w{}_\sigma(a1,...,an)$ for $(a1,...,an)\in M_w$ and σ: w$\rightarrow$s in Σ, or of the form $(\forall\emptyset)\ P(a1,...,an)$ for $(a1,...,an)\in M^w{}_P$ and P:w in Π.

Lemma 27: For J: $(\Sigma,\Pi,C)\subseteq(\Sigma',\Pi',C')$ an inclusion of order-sorted Horn clause logic with equality specifications and M a Σ,Π,C-model, the categories $\mathbf{Ext}_J(M)$ and $\mathbf{Mod}_{\Sigma'(M),\Pi',C'\cup\mathrm{Diag}^+(M)}$ are isomorphic.

Proof: Giving a $\Sigma'(M),\Pi',C'$-model is the same as giving a $\Sigma',\Pi'C'$-model M' together with an S-sorted function f: M$\rightarrow$M' that gives an interpretation in M' for each constant in M (S is the set of sorts for Σ). Similarly, a $\Sigma'(M),\Pi'$-homomorphism is just a Σ',Π'-homomorphism h: M'$\rightarrow$M'' that "preserves the

constants" in M, i.e., such that $f'=h \circ f$ for f and f' the functions interpreting the M-constants in M' and M''. So, we just need to check that a $\Sigma'(M),\Pi',C'$-model f: $M \to M'$ satisfies $Diag^+(M)$ iff f is a Σ,Π-homomorphism, and this is trivial by construction. □

Corollary 28: For Σ,Π an order-sorted signature with predicates and M a Σ,Π-model, we have $M \simeq \mathcal{T}_{\Sigma(M),\Pi,Diag^+(M)}$.

Proof: Pick J to be the identity inclusion $(\Sigma,\Pi,\emptyset) \subseteq (\Sigma,\Pi,\emptyset)$ and notice that then the identity homomorphism 1_M: $M \to M$ is obviously the initial model of $\mathbf{Ext}_J(M)$. □

We can now discuss a particular kind of free extension where the Σ,Π-model M is considered to be "built-in" and no new sorts or function symbols are added, but only new predicate symbols, yielding a new predicate signature Π' with a new "program" C' axiomatizing the new predicate symbols, which is a set of Horn clauses with only predicates in Π'-Π in their heads. This is also the approach taken by [19], but our free extension construction obtains by general principles what they describe as the "least fixpoint" of a somewhat complicated continuous operator.

Corollary 29: For J: $(\Sigma,\Pi,\emptyset) \subseteq (\Sigma,\Pi',C')$ an inclusion of order-sorted Horn clause logic with equality specifications, with C' a set of Σ,Π'-Horn clauses such that all the heads are in Π'-Π, and M a Σ,Π-model, the extension map $M \to F_J(M)$ is a Σ,Π-isomorphism.

Proof: $F_J(M) = \mathcal{T}_{\Sigma(M),\Pi,C' \cup Diag^+(M)}$ but, considering the rules of order-sorted deduction, it is clear that the clauses in C' do not contribute any new Σ,Π-sentences to those obtainable from $Diag^+(M)$. Since the function symbols are the same, this just means that, after forgetting about the predicate symbols in Π'-Π, $\mathcal{T}_{\Sigma(M),\Pi,C' \cup Diag^+(M)}$ becomes $\mathcal{T}_{\Sigma(M),\Pi,Diag^+(M)}$ and we have already seen that this last model is isomorphic to M. □

3.6 Built-Ins and Constraint Languages

A Σ,Π-model M can alway be axiomatized by its diagram. However, such an axiomatization is not in general finite. Even if there is a finite axiomatization satisfying the requirements for correctness of the operational semantics (e.g., that the equations are confluent and terminating rewrite rules, etc.) and one could rely on it to solve queries using the standard operational semantics for order-sorted algebra with equality, one would still prefer to have more efficient "built-in" special-purpose algorithms to solve such queries. Such algorithms are available for many widely used models, such as numbers, infinite trees, etc. Simplifying the picture a bit, we are assuming an algorithm Sol_M that when given a query (i.e., an existential formula $\exists\, B_1,...,B_n$) provides a solution iff the formula is satisfied by M. Such an algorithm is *complete* iff it enumerates a **complete set of solutions**, i.e., a set $Sol_M(B_1,...,B_n)$ of substitutions θ such that every witness f: $X \to M$ solving $\exists\, B_1,...,B_n$ factors as $f = f' \circ \theta$ for some θ in the set, and conversely, every composition of one such θ with *any* assignment to M is a witness. In other words, every "concrete" solution f is described by some member of a family of "generic" solutions.

Suppose now that we want to build in the Σ,Π-model M using an algorithm Sol_M and that we also want to freely extend M by some additional predicates and clauses[8] in such a way that the map

---

[8] This is the case treated by [19] using least fixpoint techniques, and is actually rather restrictive. In general, one might want to extend M by additional sorts and functions, as well as predicates and clauses, in such a way that the free extension $F_J(M)$ **protects** the built-in model M, i.e., such that when $F_J(M)$ is resticted to the signature of the built-in, one gets a model isomorphic to M. If the additional function symbols involved are all *constructors*, what we say here generalizes easily. However, the most general case needs further study.

$M \to F_J(M)$ is an isomorphism. The last subsection already proved that it is an isomorphism when the heads of the new clauses only involve new predicate symbols, and this subsection will assume that condition throughout.

How can we combine a solution algorithm Sol_M with the standard order-sorted operational semantics for the non built-in part of the extension in order to get a complete algorithm to solve queries? An answer can be gathered from the description of the free extension $F_J(M)$ as the initial model $\mathcal{T}_{\Sigma(M),\Pi',C'\cup Diag^+(M)}$ where (Σ,Π',C') is the specification (with nonempty sorts) containing the old function symbols and both the old and new predicates, and the new Horn clauses. Now, any query to be solved is necessarily of the form $\exists\ C_1,...,C_j,B_1,...,B_k$, where the $C_1,...,C_j$ are Σ,Π-atoms (i.e., only use the syntax of the built-in) and the B_i are all atoms whose predicate symbol is in Π'-Π. By Herbrand's Theorem, the query has a solution iff there is a witness $f: X \to F_J(M)$. Such a witness is *a fortiori* a witness for the query $\exists\ C_1,...,C_j$ so that there is a substitution $\theta \in Sol_M(C_1,...,C_j)$ such that f factors through θ. This means that the query $\exists\ C_1,...,C_j,B_1,...,B_k$ will have a solution iff for some $\theta \in Sol_M(C_1,...,C_j)$ the query $\exists\ \theta B_1,...,\theta B_k$ has a solution. Now, remember that $F_J(M) \simeq \mathcal{T}_{\Sigma(M),\Pi',C'\cup Diag^+(M)}$. Thus, after choosing to explore a particular θ in $Sol_M(C_1,...,C_j)$, an additional step in the search for a possible solution is, of course, resolution. This can only take place with the clauses in C', since the positive diagram only involves the old predicate symbols. Specifically, if $\theta B_i = P(t_1,...,t_n)$ and C' contains a clause of the form $P(t_1',...,t_n') \Leftarrow C'_1,...,C'_p,B'_1,...,B'_q$ then we can create a new subgoal by replacing θB_1 by $t_1=t'_1,...,t_n=t'_n,C'_1,...,C'_p,B'_1,...,B'_q$. An additional next step can then be to choose a substitution θ' in $Sol_M(t_1=t'_1,...,t_n=t'_n,C'_1,...,C'_p)$, thus yielding the overall subgoal $\theta'B'_1,...,\theta'B'_q,\theta'\circ\theta B_2,...,\theta'\circ\theta B_k$. Proceeding in this way, that is, by alternating calls to Sol_M with resolution steps using the clauses in C', gives a complete strategy for solving our original query.

However, this strategy may be too costly if calls to Sol_M are expensive. And it may be impracticable if our simplifying assumption that Sol_M applies to any conjunction of the Σ,Π-sentences does not hold, in a case where the acceptable sentences for some algorithm have a more restricted form that can ony be reached after being sufficient instantiated by substitutions. Thus, an alternative, and equaly complete strategy, is to *delay* calls to Sol_M until after all the atoms involving new predicates have been removed from the goal by resolution steps. This means "giving the benefit of the doubt" to atoms involving the built-in predicates, so that the search space, although it can be explored faster as far as the non-built-in predicates are concerned, is nevertheless broadened. It is also possible to have a "mixed" strategy, in which Sol_M is called immediately for certain predicates for which it happens to be efficient (such as perhaps equality, which might reduce to unification when only constructors are involved, or to unification without the occur check when M consists of infinite trees), whereas calls to Sol_M could be delayed for predicates where it is inefficient or where certain instantiations of the arguments are required by the algorithm. All these alternatives remain complete, since all are equivalent to satisfaction in the model $\mathcal{T}_{\Sigma(M),\Pi',C'\cup Diag^+(M)}$ which is our standard for correctness.

The above sketches an approach to constraint langauges based upon order-sorted Horn clause logic with equality. Denotational semantics is given by any model which is initial among all those having some fixed sorts, functions, and relations interpretated into a fixed order-sorted model (a "built-in"); and operational semantics is given by a combination of resolution and built-in algorithms. The main reason for taking this approach is to implement such a language efficienly, and we have discussed some of the more delicate issues that arise in this regard.

4 Logical Programming

This section discusses our broad perspective on logic programming, based on the notion of "institution" [10], which combines formal syntax with model theory and deduction at a very high level of generality. At the present stage of development, we can only claim that institutions provide a useful general framework and viewpoint, rather than a formal foundation for all aspects of our research on programming paradigms and their combinations. However, what we have so far actually supports most of our research, and also provides much broader perspective toward what might be the proper concern of "logic programming" than any other available foundation. Moreover, the areas that are not yet fully formalized represent some interesting opportunities for further research.

The abstractness and generality of the material treated here makes it very appropriate to use category theory as a language for expressing the concepts and developing their properties. Since a paper of this length cannot develop from scratch the category theory that will be needed, we assume that the reader is already familiar with the basics; for some introduction to these concepts, see [21, 1].

4.1 Institutions and Logical Programming Languages

Today's computer science is undergoing enormous growth, in both its artifacts and its theories. This creates an equally enormous need for conceptual unification, since otherwise we will drown in the flood of undigested information. In particular, there is a population explosion among the logical systems that are being used by computer scientists, including first order logic, equational logic, modal logic, higher order logic, temporal logic, intuitionistic logic, dynamic logic, order-sorted Horn clause logic, and the many specialized logics used in various verification systems and theorem provers.
Institutions [10, 6] formalize the notion of a logical system, and seem useful in capturing programming and specification language paradigms and features independently of any specific language or logic, as well as in unifying programming paradigms. Institutions avoid commitment to any particular logical system by doing constructions once and for all at a higher level of generality. In addition to the sentences, models and satisfaction already mentioned, institutions also encompass homomorphisms of models and proofs between sentences. One view is that institutions generalize classical model theory by *relativizing* it over signatures. This intuition is stated in the following slogan:

Truth is invariant under change of notation.

This subject is closely related to "abstract model theory" as studied by logicians, e.g., [3]. The formal definition will use the following notation: categories are underlined, and $|\underline{C}|$ denotes the class of objects of $\underline{C}$.

Definition 30: An **institution** I consists of:

- a category $\underline{Sign}$ of **signatures**
- a functor $\underline{Mod}: \underline{Sign} \to \underline{Cat}^{op}$ giving Σ-**models** and Σ-**morphisms**
- a functor $\underline{Sen}: \underline{Sign} \to \underline{Cat}$ giving Σ-**sentences** and Σ-**proofs**
- a **satisfaction** relation $\models_\Sigma \subseteq |\underline{Mod}(\Sigma)| \times |\underline{Sen}(\Sigma)|$ for each $\Sigma \in |\underline{Sign}|$

such that

- **satisfaction**: $m' \models_{\Sigma'} Sen(\phi)s$ iff $Mod(\phi)m' \models_\Sigma s$ for each $m' \in |\underline{Mod}(\Sigma')|$, $s \in |\underline{Sen}(\Sigma)|$, $\phi: \Sigma \to \Sigma'$ in $\underline{Sign}$, and
- **soundness**: $m \models_\Sigma s$ and $s \to s' \in Sen(\Sigma)$ imply $m \models_\Sigma s'$ for $m \in |\underline{Mod}(\Sigma)|$.

□

We now give a somewhat informal explication of the notion of logical programming.

Definition 31: A **logical programming language** $\mathcal{L}$ has an associated institution I such that:

- a **program** P of $\mathcal{L}$ consists of a signature Σ from I and a finite set of Σ-sentences[9];
- the **operational semantics** of an $\mathcal{L}$-program P is given by proofs in I, from some given query to an answer in some normal form[10]; and
- the **denotational semantics** of an $\mathcal{L}$-program P with signature Σ is given by a class of Σ-models in I[11].

□

Functional programming takes some institution of equational logic as its basis, perhaps unsorted, many-sorted, or order-sorted, perhaps first order or higher order, and then computation reduces a given sentence to a normal form. FP [2] uses a fixed higher order program P, and computation reduces user-supplied expressions to their normal form. OBJ [9] is based on first order order-sorted equational logic; the user first supplies the program P, perhaps only implicitly as a combination of smaller programs (i.e., "modules"); computation is again reduction of a well-formed expression to a normal form. Pure logic programming takes some institution of first order Horn clause logic as its basis, perhaps unsorted, many-sorted, or even order-sorted. The user supplies a program P and a "query" Q containing some variables; the operational semantics then tries to prove some substitution instance of Q from P, and returns the substitution if it succeeds. Eqlog [11] does much the same, but is based on the order-sorted Horn clause logic with equality institution, so that its sentences are much more general than in ordinary logic programming. Eqlog returns both the substitution and the substitution instance of the query, so that functional programming in the general style of OBJ is also a special case. FOOPS [12] is a combination functional and object-oriented programming language, based upon a "reflective" first order, order-sorted equational logic, and not only the user-supplied expression, but also the program, are modified by computation. Although we know the sentences and rules of deduction for this logic, we do not yet know what models are appropriate, so it is not yet an institution.

Actually, Definition 31 is even more general than we have led you to believe, since it also includes specification languages. The difference between a logical programming language and a specification language is the *efficiency* of deduction. Since this is partly subjective, and partly a matter of technology, we have not put it in a formal definition. Rather, we can just say that a **specification language** is an *inefficient* logical programming language. Some specification languages are "loose", in the sense that they take the class of *all* models that satisfy the given sentences as their denotation; for example, Clear [5]. In this case, the language needs some rather fancy sentences, such as the "data constraints" [10], to define abstract data types. Other specification languages take just the initial models for the denotation for a program, such as ACT ONE (see [8]), and some are even restricted to a fixed set of data structures, such S-expressions or numbers; for example, the original Boyer-Moore

---

[9] In practice, some sort of modularization mechanism, perhaps based on colimits as in [6], may be provided to facilitate constructing a new program by combining old programs.

[10] In general, the answer will consist not only of some I-sentence which has been proven, but also some information obtained during the proof, such as a substitution.

[11] In many cases, this class will be the class of initial models satisfying P.

theorem prover [4]. Instead of giving an expression to be reduced or a query to be answered, the user of a specification language can usually pose quite general hypotheses about the properties of a program; therefore, computation is rather general theorem proving. Also, the mode of user interaction is generally much freer than for a programming language, with the user directly participating in some aspects of the proof process. In this context, OBJ and Eqlog are "wide spectrum" languages, in that they have subsets which are quite efficient, other subsets that are tollerably efficient and would be useful for rapid prototyping (e.g., associative commutative matching), and still other subsets (e.g., the verification of theories) that are really only suitable for specification and design.

5 Conclusions

This paper has suggested that techniques somewhat more model-theoretic than those usually used in the logic programming literature may have some advantages, for example, in giving the semantics of constraint languages, as well as in combining logic and functional programming, and in reaping the various benefits of order-sorted logic that we have tried to make clear. We have also argued that order-sorted Horn clause logic with equality, especially in connection with built-ins, provides about as expressive and general a logical programming language as one might want, provided one is willing to do without states and objects. Moreover, we have given further details of the semantics of Eqlog that complete the picture given in [11]; in particular, we have given two sets of rules of deduction for order-sorted Horn clause logic with equality, one fully general, and the other for the case where all sorts are non-empty, and we have also proven completeness, initiality, Herbrand and free extension theorems. It is the latter which serves as a semantic foundation for constraint languages, and we argue that this model-theoretic approach is an attractive alternative to more syntactic approaches. One nice point is that abstraction, i.e., representation independence, is an explicit part of the formalism. We have also developed in some detail a very general framework for "logical programming," based upon the institution notion, and we have shown how the various cases discussed in the paper fit into that framework. Finally, a number of new questions have been raised, including various more general free extension theorems, and finding an appropriate model theory for object-oriented programming.

References

1. Michael A. Arbib and Ernest Manes. *Arrows, Structures and Functors.* Academic Press, 1975.

2. John Backus. "Can Programming be Liberated from the von Neumann Style?". *Communications of the Association for Computing Machinery 21*, 8 (1978), 613-641.

3. Jon Barwise. "Axioms for Abstract Model Theory". *Annals of Mathematical Logic 7* (1974), 221-265.

4. Robert Boyer and Moore, J. *A Computational Logic.* Academic Press, 1980.

5. Rod Burstall and Joseph Goguen. "Putting Theories together to Make Specifications". *Proceedings, Fifth International Joint Conference on Artificial Intelligence 5* (1977), 1045-1058.

6. Rod Burstall and Joseph Goguen. *Lecture Notes in Computer Science.* Volume 86: The Semantics of Clear, a Specification Language. In *Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification*,
Springer-Verlag, 1980, pp. 292-332.

7. Rod Burstall and Joseph Goguen. Algebras, Theories and Freeness: An Introduction for Computer Scientists. In *Proceedings, 1981 Marktoberdorf NATO Summer School*,
Reidel, 1982.

8. Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer-Verlag, 1985.

9. Kokichi Futatsugi, Joseph Goguen, Jean-Pierre Jouannaud and José Meseguer. Principles of OBJ2. In *Proceedings, Symposium on Principles of Programming Languages*,
Association for Computing Machinery, 1985, pp. 52-66.

10. Joseph Goguen and Rod Burstall. Institutions: Abstract Model Theory for Computer Science. CSLI-85-30, Center for the Study of Language and Information, Stanford University, 1985. Also submitted for publication; a preliminary version appears in *Proceedings, Logics of Programming Workshop*, edited by Edward Clarke and Dexter Kozen, Volume 164, Springer-Verlag Lecture Notes in Computer Science, pages 221-256, 1984.

11. Joseph Goguen and José Meseguer. Eqlog: Equality, Types, and Generic Modules for Logic Programming. In *Functional and Logic Programming*, Douglas DeGroot and Gary Lindstrom, Eds., Prentice-Hall, 1986, pp. 295-363. An earlier version appears in *Journal of Logic Programming*, Volume 1, Number 2, pages 179-210, September 1984.

12. Joseph Goguen and José Meseguer. Extensions and Foundations for Object-Oriented Programming. In preparation for *Research Directions in Object-Oriented Programming*, edited by Bruce Shriver and Peter Wegner. Preliminary version in *SIGPLAN Notices*, Volume 21, Number 10, pages 153-162, October 1986.

13. Joseph Goguen and José Meseguer. "Remarks on Remarks on Many-Sorted Equational Logic". *Bulletin of the European Association for Theoretical Computer Science 30* (October 1986), 66-73. Also to appear in *SIGPLAN Notices*..

14. Joseph Goguen and José Meseguer. Order-Sorted Algebra I: Partial and Overloaded Operations, Errors and Inheritance. To appear, SRI International, Computer Science Lab, 1987. Given as lecture at Seminar on Types, Carnegie-Mellon University, June 1983.

15. Joseph Goguen and José Meseguer. Order-Sorted Algebra Solves the Constructor-Selector Problem. In preparation.

16. Joseph Goguen, Jean-Pierre Jouannaud and José Meseguer. Operational Semantics of Order-Sorted Algebra. In *Proceedings, 1985 International Conference on Automata, Languages and Programming*,
Springer-Verlag, 1985. Summary presented at IFIP WG2.2 (Boston MA) June 1984.

17. Joseph Goguen, James Thatcher and Eric Wagner. An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types. RC 6487, IBM T. J. Watson Research Center, October, 1976. Appears in *Current Trends in Programming Methodology, IV*, edited by Raymond Yeh, Prentice-Hall, 1978, pages 80-149.

18. Joseph Y. Halpern, John H. Williams, Edward L. Wimmers and Timothy Winkler. Denotational Semantics and Rewrite Rules for FP. In *Proceedings, Symposium on Principles of Programming Languages*,
Association for Computing Machinery, 1985, pp. 108-120.

19. Joxan Jaffar and Jean-Louis Lassez. Constraint Logic Programming. Monash University, Australia, 1986. Draft.

20. Joxan Jaffar and Spiro Michaylov. Methodology and Implementation of a Constraint Logic Programming System. Monash University, Australia, 1986. Draft.

21. Saunders Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.

22. José Meseguer and Joseph Goguen. Initiality, Induction and Computability. In *Algebraic Methods in Semantics*, Maurice Nivat and John C. Reynolds, Eds., Cambridge University Press, 1985, pp. 459-541. Also SRI CSL Technical Report 140, December 1983.

23. José Meseguer, Joseph Goguen and Gert Smolka. Order-Sorted Unification. SRI International, 1987. In preparation.

24. Michael O'Donnell. *Equational Logic as a Programming Language*. MIT Press, 1985.

25. J. Alan Robinson. "A Machine-oriented Logic Based on the Resolution Principle". *Journal of the Association for Computing Machinery 12* (1965).

26. Gert Smolka, Joseph Goguen and José Meseguer. Order-Sorted Equational Computation. SRI International, 1987. In preparation.

27. C. Walther. Unification in Many-sorted Theories. In *Advances in Artificial Intelligence - Proceedings, Sixth European Conference on Artificial Intelligence*,
North-Holland, 1984, pp. .