In Proceedings of the 7th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA96), Sozopol, Bulgaria, 1996, pp. 86-95.

XI: A Simple Prolog-based Language for Cross-Classification and Inheritance

R. Gaizauskas and K. Humphreys
Department of Computer Science, University of Sheffield
{robertg,kwh}@dcs.shef.ac.uk

Abstract

This paper describes a simple Prolog-based knowledge representation language called XI – *X* for cross-classification and *I* for inheritance – which is designed to represent knowledge about individuals, about classes of individuals, and about inclusion relations between classes of individuals. XI allows for straightforward definitions of cross-classification hierarchies and for the association of arbitrary attribute-value information with classes or with individuals. XI also provides a simple inheritance mechanism which allows attribute values to be inherited by classes or individuals lower in the hierarchy. XI is simple, flexible, theoretically well-founded, and fully implemented. The utility of the language is demonstrated by discussing how it has been motivated by, and used in, a natural language processing system to represent the conceptual and world knowledge needed to perform co-reference resolution.

Keywords: knowledge representation, natural language, coreference resolution

1 Introduction

Knowledge representation (KR) has long been a central area of research in the 'symbolic' AI tradition and it remains so, justifiably, since so many aspects of intelligence appear to require the application of large amounts of general conceptual and world knowledge. A prime example of this is the co-reference resolution task in natural language understanding, that is, the task of determining which expressions in a text refer to the same entity [2]. This task is difficult because co-referring expressions are generally not textually identical (pronouns and their antecedents are one obvious instance of this phenomenon) and because the information needed to perform the task is very often not present in the text, but rather is background conceptual or world knowledge which the reader brings to the text. For example, in the sentences *Bill trod on the dog's paw. He yelped and ran away.* we have no difficulty resolving *He* with *the dog*, rather than with *Bill*, because we know that treading on paws causes pain and that experience of pain leads to responses such as yelping. To computationally model co-reference resolution, therefore, requires representing large amounts of this general knowledge in a computer and then applying it appropriately during the text interpretation process.

In the following we describe a simple language called XI – *X* for cross-classification and *I* for inheritance – that allows one to represent, access, and manipulate this sort of knowledge. In section 2, after a brief characterisation of the main features of the language, we present an example, then turn to presenting XI's syntax and its inference rules in more detail. The semantics of the language is given by showing how a XI theory can be mapped onto a definite clause theory in such a fashion as to guarantee the soundness of XI inferences. We briefly highlight some of the non-logical extensions to XI that make it more useful in practice. In section 3 we describe how XI has been used to do co-reference resolution as part of the discourse interpretation stage of a large applied NLP system. This system took part in the recent ARPA-sponsored Sixth Message Understanding Conference (MUC-6) [1] in which there was an evaluated co-reference task. The XI-based component of the system performed very well at this task.

XI is not particularly unique in conception. It may be viewed as yet another logic-based or frame-based inheritance system in the general tradition of KL-ONE [3], [10] or more generally as a form of semantic network [12], though the original motivation for its design came from Dahlgren's work on naive semantics [4]. It strengths are that is simple to use, simple to implement (the core of the language is less than 100 lines of Prolog), flexible (arbitrarily complex computations may be carried out if necessary), and theoretically well-founded. A full, efficient and freely available implementation exists, which may easily be extended or incorporated into existing Prolog programs[1].

2 XI

XI is a language for representing knowledge about individuals, about classes of individuals, and about inclusion relations between classes of individuals. It allows for straightforward definition of cross-classification (multiple inheritance) hierarchies and for the association of attribute-value structures with classes or with individuals. The attribute-value structures are sequences of attribute-value associations where the values may either be atomic or may be expressed via definite clauses [9] which specify how the value of an attribute is to be derived from other information held in the hierarchy (the 'impure' version of XI allows negated literals to appear in bodies of the attribute clauses, where negation is interpreted as 'negation as failure' as in Prolog). XI provides a simple inheritance mechanism which allows attribute values to be inherited by classes or individuals lower in the hierarchy.

At one level XI may be viewed simply as a declarative formalism with its own syntax and a semantics based on FOPC. But an interpreter and compiler for XI exists, together with ancillary commands that allow XI models to be incorporated into Prolog programs. In this way XI may be viewed as something akin to Prolog – a language which runs but which has some claim to a declarative semantics.

Throughout, the design philosophy has been to 'keep it simple'. Classes are represented as unary predicates and individuals are atoms. Attributes are binary predicates, the first argument identifying the class or individual of which the attribute holds, and the second being the value (which may be a variable to be instantiated by the execution of a rule associated with the attribute). The language has two components, a *definitional* component and a *derivational* component. The definitional component allows a hierarchy to be defined and attributes to be associated with nodes in the hierarchy. The definition of a cross-classification hierarchy we refer to as an *ontology*; the definition of a mapping between nodes in an ontology and attributes we refer to as an *attribute knowledge base (AKB)*. Together, an ontology and its associated AKB form what we term a *world model*.

The derivational component allows one to determine just two sorts of things: whether one node dominates another in the hierarchy and what value an attribute has at a given node. Attribute values are determined first at the given node and then are inherited by working depth-first up the hierarchy from left to right. Multiple values may be obtained by backtracking and nothing is done to prohibit these values from being contradictory – this is left to the application. The application programmer is free to implement a default inheritance scheme on top of XI.

2.1 An Example

Here is an example of the XI expressions that define a fragment of a crude world model to do with vehicles. First, the ontology (Figure 1 illustrates these hierarchical relationships graphically):

[1] XI may be obtained from `ftp://ftp.dcs.shef.ac.uk/home/kwh/xi.tar.gz`

```
top(X) ==> object(X) v event(X) v property(X).
object(X) ==> vehicle(X) v country(X) v person(X).
    vehicle(X) ==> (car(X) v lorry(X) v motorcycle(X)) &
                   (commercial(X) v private(X)).
    car(X) ==> (rover(X) v toyota(X) v renault(X)) &
                   (twodoor(X) v fourdoor(X)).
    e1 <-- private(X) & toyota(X) & fourdoor(X).
    e2 <-- country(X).
    e3 <-- country(X).
event(X) ==> drive(X) v own(X).
    e4 <-- drive(X).
property(X) ==> single_valued_prop(X) v multi_valued_prop(X).
    made_in <-- single_valued_prop(X).
    colour_of <-- single_valued_prop(X).
```

Terms on the left of the `==>` arrow are superordinate classes and terms on the right are subclasses. Each conjunct on the right is a dimension of classification and the disjuncts within each conjunct represent mutually exclusive alternative classifications within the given dimension. So, for example, a `vehicle` will be at most one of a `car`, a `lorry`, or a `motorcycle` and for whichever of these is chosen it may also be classified as either `commercial` or `private`. Terms on the left of the `<--` arrow are instances, denoted with constant terms of the form `e1`, `e2`, ..., and terms on the right of this arrow are the classes of which the terms on the left are instances. So, `e1` is an instance of something which is `private`, a `toyota`, and `fourdoor`.

The XI expressions should not be read as implicit universal quantifications, but more accurately as expressions relating classes denoted by unary predicate terms which are implicit lambda abstractions (e.g. `object(X)` denotes the class of objects and might more accurately, but more awkwardly, be written λ`X.object(X)`).

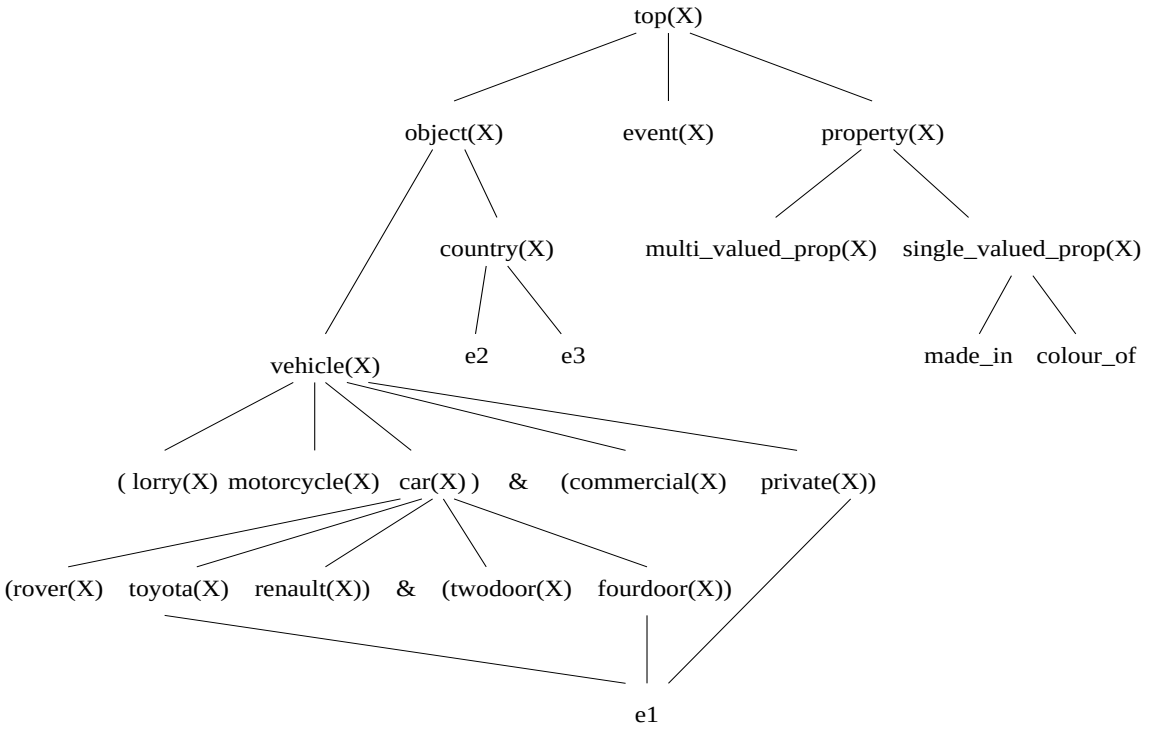

Figure 1: A Sample XI Ontology

Here is the associated XI attribute knowledge base:

```
props(vehicle(X), [function(X, transportation]).
props(rover(X), [made_in(X,e2)]).
props(toyota(X), [(made_in(X,e2) :- X <- twodoor(_)),
                  (made_in(X,e3) :- X <- fourdoor(_))]).
props(drive(X),[lsubj_type(X,person(_)), lobj_type(X,vehicle(_))].
props(e1,[colour_of(e1,blue)]).
props(e2,[name(e2,uk)]).
props(e3,[name(e3,japan)]).
props(made_in,[instance_type(object(_)),value_type(country(_))]).
props(colour_of,[instance_type(object(_)),value]).
```

Note that an attribute value may either be directly associated with a class or an instance (as in `props(rover(X),[madein(X,uk)])` or `props(e1,[colour(e1,blue)])` which tell us, respectively, that `rovers` are made in the UK and that `e1` is `blue`) or indirectly via a rule which must be evaluated for the value to be determined (as in the case of the `toyotas` which if `twodoor` are made in the UK and if `fourdoor` are made in `japan`). Note also that by treating attributes themselves as entities we can store information about attributes, such as the types of their arguments, within a XI model.

Given this world model we may now ask questions such as:

```
?- e1 <- vehicle(X).                          /* Is e1 a vehicle ? */
yes
?- hasprop(e1,colour_of(e1,X)).             /* What colour is e1 ? */
X = blue
?- hasprop(e1,made_in(e1,C)),               /* Where was e1 made ? */
   hasprop(C,name(C,N)).
C = e3, N = japan
?- hasprop(e1,P).                  /* What attributes hold of e1 ? */
P = colour_of(e1,blue) ;
P = made_in(e1,e3) ;
P = function(e1,transportation) ;
no
```

2.2 Ontologies, Attributes, World Models

The definitional component of XI allows for the specification of ontologies, attribute knowledge bases, and world models. We define these more formally here.

Since XI is an extension to Prolog, much of its syntax bears a close resemblance to that of Prolog. The *alphabet of XI*, $\mathcal{A}_{XI}$, is just like Prolog with the addition of two reserved functor symbols `props` and `hasprop` and several new connectives. $\Longrightarrow$ and $\longleftarrow$ are used in defining hierarchies to assert immediate class inclusion and immediate instance inclusion respectively. $\Rightarrow$ and $\leftarrow$ are used to indicate class inclusion (not necessarily immediate) and instance inclusion (not necessarily immediate). $\vee$ and $\wedge$ are used to define cross-classification hierarchies.

A *term* is one of a variable, a 0-ary functor symbol (also called an *instance symbol*), or an expression of the form $f(t_1, \ldots, t_n)$ where f is a functor symbol and $t_1, \ldots, t_n$ are terms. Terms of this last sort are called *complex*. In XI, classes are denoted by complex terms with a class functor and exactly one variable – we call these *class terms*. Attributes are asserted with *attribute terms* – complex terms with attribute functors and exactly two arguments. If t is a term we let $\mathrm{var}(t)$ denote the set of variables occurring in t.

We now define three further classes of expressions: O-clauses (for defining an ontology), G-clauses (for defining goals or queries), and P-clauses (for associating attributes with classes, i.e. for defining an AKB).

O-clauses have one of two forms:

1. $c \Longrightarrow D_1 \& \cdots \& D_n$, where c is a class term and each D_j is a disjunction of class terms of the form $d_{j,1} \vee \cdots \vee d_{j,p_j}$ and each class term $d_{j,k}$ on the right of the O-clause contains the variable occurring in the class term on the left of the O-clause – i.e. $\mathrm{var}(d_{j,k}) = \mathrm{var}(c)$;
2. $e \longleftarrow c_1 \& \cdots \& c_n$ where each c_i is a class term containing the same variable, i.e. $\mathrm{var}(c_1) = \cdots = \mathrm{var}(c_m)$ and e is an instance symbol.

$x \Longrightarrow y$ may be read 'y is an immediate subclass of x' and $x \longleftarrow y$ may be read 'x is an immediate instance of y'. Each conjunct (D_i) on the right hand side of an O-clause of type 1 may be thought of as a *classificatory dimension* and the disjoined terms within in it are intended to be mutually exclusive classes (this exclusiveness is enforced below through constraints on the sets of O-clauses that may form an ontology).

If $\mathbf{O}$ is a set of O-clauses we define the transitive, reflexive relation $\leq_{\mathbf{O}}$ to hold between two terms t_1 and t_2 which occur in clauses in $\mathbf{O}$ as follows. $t_1 \leq_{\mathbf{O}} t_2$ if:

1. $t_1 = t_2$; or
2. t_2 occurs on the left hand side of an O-clause of type-1 in $\mathbf{O}$ and t_1 occurs on the right hand side of the same O-clause; or
3. t_2 occurs on the right hand side of an O-clause of type-2 in $\mathbf{O}$ and t_1 occurs on the left hand side of the same O-clause; or
4. there exists a term t_3 occurring on the right hand side of an O-clause of type-1 in $\mathbf{O}$ in which t_2 occurs on the left and $t_1 \leq_{\mathbf{O}} t_3$.

G-clauses have one of the forms:

1. $c_1 \Rightarrow c_2$ where $c1$ and c_2 are class terms or variables;
2. $e \leftarrow c$ where e is an instance symbol or a variable and c is a class term or a variable;
3. $\mathtt{hasprop}(e, p)$ where e is an instance symbol or a variable and p is an attribute term or a variable;
4. G_1, G_2 where G_1 and G_2 are G-clauses.
5. $G_1; G_2$ where G_1 and G_2 are G-clauses.

$x \Rightarrow y$ may be read 'y is a subclass of x' and $x \leftarrow y$ may be read 'x is an instance of y'.

P-clauses have the form $\mathtt{props}(c, \mathbf{V})$ where c is either an instance symbol or a class term and $\mathbf{V}$ is a set whose members are of the form

1. $p(t_1, t_2)$ where p is a functor and t_1 is c if c is an instance term and $var(c)$ otherwise; or
2. $p(t_1, t_2) :- G$ where p is attribute term as in 1) and G is a G-clause.

A *world model* $\mathcal{W}$ is a pair $(\mathbf{O}, \mathbf{P})$ where

1. $\mathbf{O}$ is a set of O-clauses such that:
   (a) there exists a unique term r such that for all terms t occurring in $\mathbf{O}$ $t \leq_{\mathbf{O}} r$; and
   (b) for any O-clause $c \Longrightarrow D_1 \& \cdots \& D_n$ if t_1 and t_2 both occur in some D_i, i.e. if D_i has the form $t_1 \vee t_2 \vee \cdots \vee tn$ then there is no term t in any O-clause in $\mathbf{O}$ such that $t \leq_{\mathbf{O}} t_1$ and $t \leq_{\mathbf{O}} t_2$.
2. $\mathbf{P}$ is a set of P-clauses such that for each P-clause $\mathtt{props}(c, \{V_1, \ldots, V_n\})$ in $\mathbf{P}$, c occurs in some O-clause in $\mathbf{O}$.

Some remarks about this definition of world model are in order. The conditions on the O-clauses in **O** ensure that

1. the ontology has a greatest element (the universal class);
2. no class can inherit from two classes within the same classificatory dimension; i.e. since the classes within a classificatory dimension (one of the conjuncts D_i on the right of an O-clause of the form $c \Longrightarrow D_1 \& \cdots \& D_n$) are meant to be mutually exclusive then clearly no subclass can be a subclass of two or more of them.

The O-clauses in a world model may be seen as an extensional definition of a partial ordering relation on a subset of the class terms of $\mathcal{L}_{XI}$. The conditions on $\mathbf{P}$ ensure that all the classes which have properties associated with them in P-clauses are defined somewhere in the ontology.

The *language of XI*, $\mathcal{L}_{XI}$, is the set of strings in $\mathcal{A}^*_{XI}$ which are O-, G-, or P-clauses.

2.3 Derivation in XI

Only G-clauses may be derived in XI. There are five types of G-clauses: those asserting dominance relations between classes, those asserting dominance relations between classes and instances, those asserting that an attribute holds of an instance (possibly by inheritance), and conjunctions and disjunctions of these three basic types.

To establish that one class is a subclass of another or that an instance is an instance of some class requires recursively exploring the set of O-clauses in the world model to see if the terms in query are appropriately related; i.e. it involves seeing whether for a given set of O-clauses **O** and terms t_1 and t_2 the ordering relation $\leq_{\mathbf{O}}$ holds between them. To establish that an attribute holds of a given instance requires checking to see if the attribute is recorded in the set of attributes associated with the instance and this may in turn require determining that other G-clauses hold, if the attribute is conditional. If the attribute does not hold 'locally' then the ontology is recursively explored upwards to see if the attribute is associated with any dominating node. Note that the attribute we are seeking to establish may be a partially instantiated term. Thus, seeing that the attribute is recorded at a class or instance node requires seeing if the goal term may be unified with (the head of) any term stored in the set of attributes at the class or instance node. Conjoined goals are established by establishing each conjunct. Disjoined goals are established by establishing either disjunct.

As with Prolog, XI may be extended to permit the 'pure' underlying logic (definite clause logic in the case of Prolog) to include negated goals. As in Prolog, these are established by failing to establish the corresponding unnegated goal ('negation as failure'). Thus there are no inference rules explicitly for negation and we first define a notion of derivation for pure, negation-free XI, and later extend this notion to include negated goals.

The inference rules in XI are the following:

1. $$\frac{\begin{array}{ll} c \Longrightarrow D_{1,1} \& \cdots \& D_{1,n_1} & \text{(a type-1 O-clause)} \\ c_1 \Longrightarrow D_{2,1} \& \cdots \& D_{2,n_2} & (c_1 \text{ any class term in some } D_{1,j}) \\ \vdots & \\ c_{k-1} \Longrightarrow D_{k,1} \& \cdots \& D_{k,n_k} & (c_{k-1} \text{ any class term in some } D_{k-1,j}) \end{array}}{c \Rightarrow d} \quad (d \text{ any class term in some } D_{k,j})$$

2. $$\frac{\begin{array}{ll} c \Rightarrow d & \\ e \longleftarrow d_1 \& \cdots \& d_m & \text{(a type-2 O-clause with } d = d_i \text{ for some } 1 \leq i \leq m) \end{array}}{e \leftarrow c}$$

3. (a) $\dfrac{\texttt{props}(e, [\ldots, p(e,t)\ldots])}{\texttt{hasprop}(e, p(e,t))}$ (e an instance, p an attribute term, t any term)

   (b) $\dfrac{\texttt{props}(c(X), [\ldots, p(X,t), \ldots]) \quad e \leftarrow c(X)}{\texttt{hasprop}(e, p(e,t))}$ ($c(X)$ a class term, p an attribute term, t any term; e an instance term)

4. (a) $\dfrac{\texttt{props}(e, [\ldots, p(e,t) :\!- G, \ldots]) \quad G\theta}{\texttt{hasprop}(e, p(e,t)\theta)}$ (e an instance term, p an attribute term t any term, G a G-clause; θ a substitution for variables in G)

   (b) $\dfrac{\texttt{props}(c(X), [\ldots, p(X,t) :\!- G, \ldots]) \quad G\theta \quad e \leftarrow c}{\texttt{hasprop}(e, p(e,t)\theta)}$ ($c(X)$ a class term, p an attribute term, t any term, G a G-clause; θ a substitution for variables in G; e an instance term)

5. $\dfrac{G_1 \quad G_2}{G_1, G_2}$ (G_1 and G_2 G-clauses)

6. $\dfrac{G_1}{G_1; G_2}$ 7. $\dfrac{G_2}{G_1; G_2}$ (G_1 and G_2 G-clauses)

A *XI-derivation* of a G-clause G from a world model $\mathcal{W}$ is a finite sequence of O-, P-, and G-clauses such that G is the last clause in the sequence and every other clause is either an O-clause or a P-clause in $\mathcal{W}$ or is a G-clause that follows from one or more clauses preceding it in the sequence according to one of the XI inference rules. If such a derivation exists we say G is *XI-derivable from* $\mathcal{W}$, written $\mathcal{W} \vdash_{XI} G$.

2.4 Semantics

Rather than specify a semantics for XI 'from the ground up', a semantics may be given to XI indirectly by showing how a XI world model can be translated via a translation function $\mathcal{T}$ into a first order theory which may then be given the standard first order semantics. It is then possible to show that for a given world model $\mathcal{W}$ and goal G, if $\mathcal{W} \vdash_{XI} G$ then $\mathcal{T}(\mathcal{W}) \vdash_{FOPC} \mathcal{T}(G)$. This establishes the soundness of derivations in XI with respect to first order logic. We do not appeal to any particular first order proof theory, as the first order derivations we suppose are trivial; any natural deduction system such as [11] suffices. Of course directly supplying models for world models is another, arguably preferable, method for providing a semantics for XI; doing this should be relatively straightforward, but has not yet been done.

Details of the translation function $\mathcal{T}$ may be found in [5]. In brief:

1. each type 1 O-clause and type 1 G-clause is translated into one or more material implications in which some right-hand side class term from the O- or G-clause is the antecedent of the implication and the left-hand side class term is the consequent;
2. each type 2 O-clause and type 2 G-clause is translated into a ground fact in which the instance term is substituted for the variable in the class term;

3. `hasprop`(e, p) G-clauses are translated simply as goals p;
4. conjoined and disjoined G-clauses are translated as conjunctions and disjunctions of the translations of their subformulas;
5. P-clauses of the form `props`$(c, \{V_1, \ldots, V_n\})$ are translated as material implications $c \supset p$ if V_i is p, or $\mathcal{T}(G), c \supset p$ if V_i is p:– G.

Given this translation it is then possible to prove

Proposition *Let $\mathcal{W} = (\mathbf{O}, \mathbf{P})$ be a world model and G be a G-clause. If $\mathcal{W} \vdash_{XI} G$ then $\mathcal{T}(\mathcal{W}) \vdash_{FOPC} \mathcal{T}(G)$.*

Again, details of the proof may be found in [5].

2.5 Impure XI

There are four sorts of extension to XI, as it is implemented, which make it much more powerful but which affect the simple semantics given above. These extensions parallel similar features in Prolog. The first sort includes just the metalogical `not` operator. This operator can be applied to G-clauses only, but hence may occur in the G-clause components of attribute clauses within P-clauses. The semantics of `not` are as in Prolog: `not` G follows from a world model $\mathcal{W}$ iff there is no derivation of G from $\mathcal{W}$. Introduction of this operator forces us to extend the notion of XI-derivation to that of XINF-derivation (XI Negation as Failure).

The second extension includes mechanisms for limiting search – the `xi_cut` operator which is analogous to the cut operator in Prolog but in this case forces a halt to searching upwards in an ontology for attribute inheritance, and the `nodeprop` operator which limits property inheritance to the current node.

The third includes mechanisms for dynamically altering the hierarchy by adding or removing instances, classes or properties (analogous to `assert` and `retract` in Prolog).

The fourth includes mechanisms for input and output of world models.

3 The Application of XI to Coreference Resolution

We can identify several sorts of knowledge required to perform coreference resolution:

1. taxonomic or classification information (frequently anaphors are super- or subordinate class terms of the antecedent, for example *the vehicle* may be used to refer to a previously mentioned *car*);
2. attribute information about an individual or a class, together with an inheritance mechanism to associate such information with all individuals in a class (for example, that a typical precondition of a crashing event is the involvement of the agent in a driving event). Since the extensions of attributes may not be known, we will also need to express conditions on attribute values (i.e. the fact that an attribute has value V_1 if condition C_1 holds and value V_2 if C_2 holds);
3. attribute information about other attributes (for example, to preclude co-resolving references to a *blue car* and a *red car*, we must know that the colour attribute is single-valued for a given object at a given time – unlike, say, the *beside* attribute which can have several values for a given object at a given time).

Note that what might be called *property-based* logic programming languages [9] such as Prolog are inadequate for this task, at least if used naively. In such a language it is relatively easy to discover what things have a given property (e.g. to find all green things issue the query `?- green(X).`) but very difficult, without changing representations, to discover all the properties which hold of an individual. To do this an *object-based* logic is

required in which attributes are associated with individuals or classes. Of course another option is to adopt a second, or higher order, logic, but computational problems with these languages (e.g. second-order unification is undecidable [8]) together with their excessive formal power suggest, at least initially, the exploration of simpler approaches. XI is capable of representing the knowledge types identified above, is therefore useful for coreference resolution, and is a simple approach.

3.1 A Coreference Algorithm

A set of basic coreference heuristics can be easily represented in a XI attribute knowledge base through the use of a `distinct` attribute, specifying restrictions on particular nodes in an ontology. For example, instances of type `object` which have the properties of being animate and named by pronouns can be restricted to coreferences with instances of type `person` only. Text position information of referring expressions can also be represented via properties, allowing the use of 'distance measures' to restrict, say, pronouns to corefer only within the current paragraph.

A coreference mechanism to make use of an ontology and knowledge base which includes such heuristics can then adopt a simple overall strategy of attempting to resolve each new instance derived from a text with all existing instances, subject to the restrictions that the instances inherit from the knowledge base. The mechanism, as currently implemented, proceeds as follows.

Each pair of new and existing instances under consideration are initially compared for 'kind consistency', to ensure that the instances' semantic types are ordered in the ontology. The potential resolution is immediately rejected if the semantic types have no dominance relation between them. For example, a new `vehicle(e3)` instance will be kind-consistent with an existing `car(e2)` instance but not `person(e1)`.

If the instances have compatible types then any inherited values of the `distinct` attribute are established, to determine whether any of the coreference heuristics rule out the potential resolution. For example, a heuristic prohibiting any new indefinite object to corefer with anything before it in the text would apply here if, say, the new `vehicle(e3)` instance was derived from a referring expression with an indefinite determiner.

If no general heuristics apply, then the values of all other properties of each instance are established and checked for inconsistencies. Attributes are classified in the ontology as either single- or multi-valued, and two instances are 'attribute-consistent' only if the values of single-valued attributes common to both are equivalent. If the attribute `colour` is defined in the ontology as single-valued, then the potential resolution of a *blue vehicle* and a *red car* will be rejected at this point. Equivalence tests specific to semantic type are used for particular attributes such as `name`, for example to compare abbreviated forms of person names with the full form (e.g. “Smith” with “Mr. John Smith”).

Each kind-consistent, non-distinct and attribute-consistent pair of instances is then assigned a similarity score based on the distance (in nodes) between the two instances in the ontology, together with a count of any multi-valued attributes with the same value for both instances. After all pairs have been considered the highest scoring pair (if any score at all) is selected as the best candidate for coreference. If several pairs have equally high similarity scores then the pair with the closest referring expressions in the text is preferred. The selection of a best candidate causes the complete removal of the least specific instance of the pair from the world model, and the addition of all its attributes to the remaining instance. Any other references to the removed instance are then updated, effectively merging two instances into a single referent and therefore representing a coreference in the original input.

3.2 The Coreference Task in MUC-6

The algorithm outlined above was implemented in the LaSIE system, Sheffield University's entry to MUC-6 [7]. The XI language was used to construct a small domain-specific ontology based on the MUC task specifications. For example, the specifications explicitly distinguished between company and government organisations, and between people and company posts. These distinctions could be directly represented in the XI formalism, allowing an initial task-specific ontology to be rapidly constructed. Further sub-classifications were then made as needed, based on performance evaluations of training data. The small set of coreference heuristics were also refined in this way. The system achieved a high level of precision in the coreference task evaluation (approximately 70%, against a top score of 72%), and a reasonable level of recall (54% against a top 63%). Further details of the coreference algorithm and its performance can be found in [6].

4 Concluding Remarks

XI has proved to be a powerful and easy-to-use language for representing the general conceptual and world knowledge needed to perform coreference resolution. Coreference resolution is, however, just one example of a problem area in AI where general world knowledge needs to be brought to bear in order to solve a more immediate problem. Others include object matching in vision, reasoning in expert systems, planning and qualitative reasoning. XI's usefulness in the coreference task leads us to believe it could well find more general application in AI.

References

[1] ADVANCED RESEARCH PROJECTS AGENCY. *Proceedings of the Sixth Message Understanding Conference (MUC-6)* (1995), Morgan Kaufmann.

[2] ALLEN, J. *Natural Language Understanding*, 2nd ed. Benjamin/Cummings, 1995.

[3] BRACHMAN, R., AND SCHMOLZE, J. An overview of the KL-ONE knowledge representation system. *Cognitive Science 9* (1985), 171–216.

[4] DAHLGREN, K. *Naive Semantics for Natural Language Understanding*. Kluwer, 1988.

[5] GAIZAUSKAS, R. XI: A knowledge representation language based on cross-classification and inheritance. Tech. Rep. CS-95-24, Department of Computer Science, University of Sheffield, 1995.

[6] GAIZAUSKAS, R., AND HUMPHREYS, K. Quantitative evaluation of coreference algorithms in an information extraction system. In *Proceedings of the Discourse and Anaphora Resolution Colloquium* (1996), University of Lancaster. in press.

[7] GAIZAUSKAS, R., WAKAO, T., HUMPHREYS, K., CUNNINGHAM, H., AND WILKS, Y. University of Sheffield: Description of the LaSIE system as used for MUC-6. In [1].

[8] GOLDFARB, W. The undecidability of the second-order unification problem. *Theoretical Computer Science 13* (1981), 225–230.

[9] LLOYD, J. *Foundations of Logic Programming*, 2nd ed. Springer-Verlag, 1987.

[10] MACGREGOR, R. The evolving technology of classification-based knowledge representation systems. In *Principles of Knowledge Representation*, J. Sowa, Ed. Morgan Kaufmann, 1991.

[11] SMULLYAN, R. *First-Order Logic*. Springer-Verlag, 1968.

[12] SOWA, J., Ed. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, 1991.