Computer Processable English and McLogic

Jana Sukkarieh and Stephen Pulman*

University of Cambridge Computer Laboratory,
New Museums Site,
Cambridge CB2 3QG, UK.
England
{Jana.Sukkarieh, Stephen.Pulman}@cl.cam.ac.uk

September 15th, 1998

Abstract

This paper describes an experimental system which uses an artificially restricted subset of English (CPE) for knowledge representation and program specification, following a line of work described in Macias and Pulman 1995 [3], and Pulman 1996 [5]. The present work differs from its antecedents in several ways, the most obvious being that the target representation language is a natural-language oriented logic developed by McAllester [2], [4](which we call McLogic). We describe extensions to McLogic designed to allow for greater expressivity in the dialect of CPE used; an inference system for McLogic; and two experimental applications: the first is a well-known example from the Z programme specification literature ('Wing's library problem') and the second a well-known test case for theorem provers ('Schubert's Steamroller').

Keywords: controlled language, knowledge representation, theorem proving.

1 Introduction

Our eventual aim is to be able to use a simplified subset of English which is completely syntactically and semantically processable by a machine to carry out various knowledge representation and system specification tasks. We call this type of English 'Computer Processable English' to distinguish it from the real thing, which is much more ambiguous, elliptical, and difficult to process.

In the abstract, both of our intended application areas can be characterised as the performance of the following task:

1 *Given English utterances* $U_1, ..., U_n$ *and an English utterance C, a machine has to decide whether C follows from* $U_1 \wedge ... \wedge U_n$

*Also at SRI International, Cambridge

Obviously, undecidability and complexity issues mean that there is no completely general solution for this problem. Therefore, we consider a variant problem, as follows:

2 *Given a set of premises S=$\{U_1, ..., U_n\}$ ($n \geq 1$), where U_is belong to* ***a restricted dialect of English Language*** *U, and a queried conclusion $C \in U$ for which it is guaranteed that it is possible to know whether C follows from S or is negated by S, the goal is* ***to reach the right answer*** *and* ***to formulate transparent arguments*** *which prove that the answer is valid (not a guess nor a reasoning mistake)* ***in a reasonable time****, by using a representation language that is expressive and at the same time computationally efficient.*

The subset U is a dialect of Computer Processable English (CPE) [5]. The representation language, which we call **McLogic**, is an extension of the one defined in [2], [4]. This is a system for first-order logic (with one or two higher order extensions) which for our purposes has two possible advantages: firstly, the syntax is relatively English-like, making for transparency in the process of formulating and reasoning with specifications; and secondly, it has a polynomial-time inference procedure.

In this paper, we present the system we have developed for translation of CPE expressions to and from McLogic and the inference system associated with it. We briefly describe two experimental applications: the first is a well-known example from the Z programme specification literature ('Wing's library problem') and the second the well-known test case for theorem provers ('Schubert's Steamroller'). The motivation for choosing these was to stick to relatively well-behaved benchmarks while developing the system.

One problem not addressed in this paper is the issue of restricting users in the field to the appropriate dialect of CPE. While a significant issue in general, there is not space to address it here.

2 English to and from McLogic

To simplify the interpretation and disambiguation task we exclude from this dialect of CPE use of inter-sentence referring devices, and so texts are translated as if they are lists of isolated sentences. Hence, the problem of inter-sentential links was avoided. However, there are cases in both application areas that demand the use of intra-sentential references: e.g. "Every animal eats some animal that is smaller than itself". for which we adopted a solution presented in [3], [5]. By introducing as part of CPE a bit of 'logician's English' we translate sentences like this as: *Every animal X eats some animal that is smaller than X* . The translation from CPE $\longleftrightarrow$ McLogic has been implemented as an interactive program which receives as input Ss, that belong to CPE, which make use of purely linguistic processing by the application of lexical entries and syntactic and semantic grammar independently of the influence of context and which constructs expressions of McLogic. For some constructs a kind of quasi-logical form is constructed which is then translated into the appropriate McLogic expression: these include negation, conjunction, and temporal clauses.

A standard left to right, bottom up Prolog chart parser and a feature-based unification grammar, which were part of the Fracas Project[1], were used to achieve the translation in the McLogic direction. A head-driven generation algorithm based on the same grammar was used to translate in the opposite direction.

The basic notion in McLogic is a class expression, i.e. an expression that denotes a set. In general, any monadic predicate symbol of classical syntax can be used as a class expression, and for any class expression s and binary relation R one can construct the class expresssions *(R (some s))* and *(R (every s))*. For example, if *like* is a binary relation symbol and *woman* is a class symbol, then one can construct the class expressions *(like (some woman))* and *(like (every woman))*. These denote the set of all entities in the domain that like some woman and the set of all those who like all women respectively.

Some constructs in CPE and their corresponding representation in McLogic follow:

- common nouns, intransitive verbs, predicate nominals, and intersective adjectives are represented as monadic class expressions e.g. student, red, sleeps.
- NPs with determiners give rise to a formula like: (some X Y) or (every X Y) if in subject position, but to a formula like (some X) if in a non-subject position. This is a non-compositional feature of McLogic motivated by linguistic transparency, but we shall see below that this prevents a one-step translation for conjunctions of noun phrases to McLogic.
- Names are represented as a quantified expression e.g. (R (some/every john)) or (some/every john X). Since we assume the 'unique name' requirement, then the class denoted by 'john', 'some john' (or 'every john') will be identical, and formulating inference rules is facilitated by taking the quantified form as standard.
- transitive verb phrases translate to a class expression of the form (R (some s))/ (R (every t)) e.g. (own (some computer)) or (like (every mary))

For example, *some student borrows every book* is represented as (some student (borrow (every book)) where (borrow (every book))= $\{y/\forall x.x \in book \longrightarrow y\ borrow\ x\}$. The formula is true iff $student \cap (borrow\ (every\ book)) \neq \emptyset$. Formulas in McLogic then are of the form (every s t), (some s t), (some s exists), (at_most_one s) where s, t are class expressions. In addition, it is worth mentioning that a lambda class expression in McLogic is of the form $\lambda x.\phi(x)$ where $\phi(x)$ is a formula in McLogic.

$$\lambda x.\phi(x) = \{x/\phi(x) is\ true\}.$$

We see an example of its use in section 5.2. A formula not containing a lambda expression is called quantifier-free formula.

[1] `http://www.cogsci.ed.ac.uk/ fracas/`

3 Two applications

'Schubert's Steamroller' was presented in 1978 by Lenhart Schubert as a challenge to automated-deduction systems [7]. Its statement in CPE is as follows:

> Every wolf is an animal. Every fox is an animal. Every bird is an animal.
> Every caterpillar is an animal.Every snail is an animal. Some wolf exists.
> Some fox exists. Some bird exists. Some caterpillar exists.
> Some snail exists. Every grain is a plant. Some grain exists.
> Every caterpillar is smaller than every bird.
> Every snail is smaller than every bird.
> Every bird is smaller than every fox.
> Every fox is smaller than every wolf.
> It is not true that some wolf eats some fox.
> It is not true that some wolf eats some grain.
> Every bird eats every caterpillar.
> It is not true that some bird eats some snail.
> Every caterpillar eats some plant.
> Every snail eats some plant.
> Every animal X eats every plant or every animal that is smaller than X and eats some plant.

The conlusion to be proved is:

> Some animal eats an animal that eats some grain.

'Wing's Library Problem' (ref to Z textbook) concerns the specification of allowable library operations originally informally described by Wing (1988). We will present here only one transaction (as illustration) instead of the five transactions allowed by the system: checking out and returning books:

3 *Given a user U, a copy of a book C and a member of the staff_member M, the problem is to find out whether U can borrow or return C and whether the transaction can be done by M.*

The following CPE sentences (partially) define the specification of the library operations:

> Every copy is available or checked_out.
> No copy is available and checked_out.
> Every borrower is a registered user. Every registered user is a borrower.
> Every staff_member is an administrator.
> Every administrator is a staff_member.
> No staff_member is a registered user.
> Every staff_member is an authorised requestor.
> Every authorised requestor is a staff_member.
> Every borrowed copy is checked_out.
> Every checked_out copy is borrowed.
> The_University_Library owns every available copy.

Every registered user who has borrowed less than ten copies can borrow everu available copy.
Every user who can borrow an available copy is a registered user and has borrowed less than ten copies.
one copy is less than ten copies. two/.../nine copies are less than ten copies.
every authorised requestor can give every registered user some copy.

As will be seen from these examples, even this restricted fragment of English contains several constructs which have no obvious translation into McLogic. Our first task therefore was to extend the logic in order that the expressive power needed could be obtained.

4 Extensions to McLogic

In this section we will introduce a number of extensions to McLogic and provide English examples that motivate them.

Adjectives

We simplify the interpretation of adjectives by treating them as class expressions. Nominal modification, like "happy man" is thus treated intersectively. We define an operator + between class expressions where s+t denotes s $\cap$ t, the intersection of two sets.

VP modification

An Advp or a PP modifying a verb are not represented independently as sets. A construct *drives fast* refers to a subset of entities who drive and a construct *sees with some telescope* refers to a subset of entities who see. Hence, the introduction of a new operator, #, for which (A # B) $\cap$ A = A # B. *drives fast, drives slowly, drives carefully, drives with a, etc.*, all refer to subsets of the class expression *drive*. As for a complex adverbs as in *drives slowly and carefully*, it would be translated into drive # slowly + drive # carefully[2].

Relative clauses

A (subject) relative clause "who/which/that vp" is represented as (the representation of) vp. *every student who reads every article succeeds* is represented as (every (student+(read(every article))) succeed). Non-subject relatives are not treated yet.

Conjunction and Negation

We assume the usual boolean connectives to describe sentence level conjunction, disjunction, and negation. Other types of conjunction are translated as follows:

Adjectives:

[2]adverbs and adjectives with a degree like *very fast or very handsome* can be considered by adding a new operator. However, we did not deal with them yet.

Using the operator, +, defined earlier the intersection between class expressions is represented. For example, *some book is new and shiny* $\Rightarrow$ (some book **(new + shiny)**).
For 'or', we define an operator, $, between class expressions where s $ t denotes s $\cup$ t, the usual union between sets. $ represents an inclusive 'or'. To represent *some book is new or old (but not both)*, we need the disjoint union of two class expressions. Hence, the introduction of the operator (-) such that s-t = { x/ x $\in$ (s $\cup$ t), x $\notin$ (s $\cap$ t) }.

Verb phrase and nominal conjunction can now be handled:
John laughs and loves a funny girl $\Rightarrow$ (some john **(laugh + (love (some funny + girl))))**.
every teacher and administrator snored $\Rightarrow$ (every **(teacher + administrator)** snore).

NP conjunction:
To treat NP conjunction we need a meaning for NP independent of VP. McAllester and Givan didn't define such a meaning to simplify the exposition of their semantics. If NP is in a subject position then the translation is done in one step. However, when NP is not in a subject position such construct is mapped to a quasi-logic form first then mapped (using simple equivalence relations) to a sentence in McLogic. For example,

(1) a. 'some teacher and every student snores' is mapped into

b. ((some teacher snore) and (every student snore))

While

(2) a. 'some fox eats some grain or some plant ' is mapped into

b. (some fox (eat or ((some grain) (some plant)))) then

c. (some fox (eat (some grain)) $ (eat (some plant)))

Note that when np is a predicative as in *some man is a client or a manager* then in one step that sentence is represented as (some man client $ manager). Note that collective predicates would not be correctly interpreted under such a treatment.

We also extend the negation operator to apply to class expressions:
Intuitively, in case of *some teacher is not a borrower* if x is not a borrower then we can conclude that x could be anything in the domain, D, except a borrower. Define (not s) with the meaning (not s) $=\{x/x \in D \text{ and } x \notin s\}$. Hence, the sentence is represented as (some teacher (not borrower)).

Ditransitive Verbs

In McAllester's original language, a binary relation R can be transformed into a function R' from elements to sets such that y is an element of $R'(x) = \{y/\langle x, y\rangle \in R\}$. When we consider sentences such as *John gave Mary a book* or *the postman handed me a parcel*, etc, then we are no longer restricting the work to binary relations. If we extend the language

to account for ditransitive verbs and hence to 3-ary relations R then such an R can be transformed into a function R' from elements to functions from elements in the domain to sets. Hence, given $x_i \in$ Domain, $R'(x_i) = R'_{x_i} : Domain \longrightarrow P(Domain)$ such that $R'_{x_i}(y) = \{z/\langle x, z, y\rangle \in R\}$ where x denotes the subject, z the direct Object(dO) and y the indirect object(iO). In particular, the following hold:

(R(some s)(every t))= $\{\ y/\exists x \in s.\forall z.z \in t \longrightarrow \langle y, z, x\rangle \in R\ \}$
(R(some s)(some t))= $\{\ y/\exists x \in s \wedge \exists z \in t \wedge \langle y, z, x\rangle \in R\ \}$
(R(every s)(every t)) = $\{\ y/\forall x.x \in s \longrightarrow \forall z.z \in t \longrightarrow \langle y, z, x\rangle \in R\ \}$
(R(every s)(some t)) =$\{\ y/\forall x.x \in s \longrightarrow \exists z.z \in t \wedge \langle y, z, x\rangle \in R\ \}$.

In all the above cases the representation is of the form (V' (iO') (dO')).
John gave Mary a book will be translated to (some john (give (some mary) (some book))).
If the sentence *John gave a book to Mary* was considered then its representation should be equivalent to that of *John gave Mary a book*. The representation will take the form (V (dO) (to,iO)) where (R (some s) (to,every t))= $\{\ y/\exists x \in s.\forall z.z \in t \longrightarrow \langle y, x, z\rangle \in R\ \}$. In the above case, the relation R is transformed into a function R' such that $R'(x_i)$, where x_i is an element in the domain, is a function that takes an element in the domain y and returns $R'_{x_i}(y) = \{z/\langle x, y, z\rangle \in R\}$. The particle "to" doesn't have a semantics on its own. Instead, to reduce the need for more inference rules we can consider the form (V(dO (to,iO)) as an intermediate form to be translated into (V (iO) (dO)).

Passives

We only considered the monotransitive and ditransitive verbs.
We define $(R^{-1}(some\ s)) = \{x/\exists y.y \in s\ \wedge\ yRx\}$ and *some book was borrowed by John* will be represented as $(some\ book(\ borrow^{-1}(some\ john)))$.
Can a similar concept hold for 3-ary relation?
For a ditransitive verb, it is always the first object that becomes subject in the passive.
We define $R^{-1}(x, y, z)$ to be R(y,z,x). In that case,
$(R^{-1}(some\ s)(every\ t)) = \{y/\exists x \in s.\forall z.z \in t \longrightarrow \langle x, z, y\rangle \in R^{-1}$
$(if\ dO\ is\ first\ in\ the\ sentence)\ or\ \langle x, y, z\rangle \in R^{-1}\ (if\ iO\ is)\}$.

(3) a. John gave a book to some student

b. a book was given to some student by John

c. (some book ($give^{-1}$ ((some student) (some john))))

(4) a. John gave some student a book

b. Some student was given a book by John

c. (some student ($give^{-1}$ ((some book) (some john))))

Sentences like *a book was borrowed, a book was given, a book was given to some client, a client was given a book* are yet to be considered.

Comparatives and Numerals

Comparatives are not treated compositionally. We assume a comparative version of every adjective, thus: 'every bird is smaller than every horse' is translated as (every bird (smaller_than (every horse))). To deal with comparatives and numerals as in sentences like *Some man borrowed less than ten books*, we need to define the following:

1. we first extend the language to represent sentences like *Ten books are old.* We introduce:

   (a) the class expresssion s^p = P(s)= Power set of the class expression s

   (b) A particular class expression, N, representing a positive integer N and denoting the set { X/ X is a set of N objects in the domain }[3].

   (c) A special operator $*$ between such two class expressions. $*$ would have the same meaning as '+' i.e. intersection between sets. However, the introduction of a new operator is to emphasize the fact that $*$ is an intersection between sets of sets and not sets of entities.

   Hence, expressions like N$*$t={ X/X is a set of N t's }. The latter is a set of sets. The formula (N$*$s t) in this case is true iff some element of $N * s$, k, is a subset of t i.e. when the formulas (every k t) and (some/every k N$*$s) are true. Note that in the first formula k is considered a set of elements while in the second it is considered a singleton set consisting of one element which is itself a set.

   ten books are old $\equiv$ (ten$*$book old)[4].

2. Then we define the following:

   (a) (R(less_than $(N * s^p)$)) =
   $\{x/\exists y \in (N \; and \; s^p) \wedge \exists z.\phi \subset z \subset y \wedge \forall i.(i \in z \longleftrightarrow xRi)\} \cup \{x/\forall y.y \in s \longrightarrow < x, y > \notin R\}$.

   (b) (R (greater_than $N * s^p$)) =
   $\{x/\exists y \in (N and s^p) \wedge \exists z.y \subset z \wedge \forall i.(i \in z \longleftrightarrow xRi)\} \cup \{x/\forall y.y \in s \longrightarrow < x, y > \notin R\}$

   (c) (less_than $N * s^p$ t) true iff $s \cap t \subset x \; for \; some \; x \in N \; and \; s^p$
   $i.e. \; 0 < card(s \cap t) < N$

   (d) (greater_than $N * s^p$ t) true iff $x \subset t \cap s \; for \; some \; x \in N \; and \; s^p$

Hence, the representation (some man (borrow (less_than (ten$*$book)))).

Time Reference

To represent sentences such as "Some student sleeps before the instructor begins the lecture", first we consider a particular kind of class expressions which denote elements in

[3] N does not have a semantics independently of entities in the domain.

[4] Note the p can be dropped from the representation of s^p. It is understood from the existence of $N*$.

the domain defining an interval of time which we call **time class expressions**. Let t_1, t_2 be two time class expressions then

$$t_1 = [a_1, b_1] = \{x/x \leq b_1 \; and \; x \geq a_1\}$$

and

$$t_2 = [a_2, b_2]$$

where a_i, b_i are positive integers denoting a_i *and* b_i units of time respectively. We have defined earlier a positive integer N as { X/ X is a set of N objects in the domain }. In this particular case, the objects are units of time. If, for example, $t_1 = [4, 5]$ then 4= { X/X is a set of 4 units of time (sec, mn, hours, etc) }. Second, we augment the arguments of any relation, R, considered till now, by one argument, a time class expression.

$$read\ an\ article \longrightarrow (read\ (some\ t)(some\ article))$$

$$snore \longrightarrow (snore\ (some\ t))$$

It is like introducing an event variable with each action verb for example see [1].

(5) a. Some student sleeps before the instructor begins the lecture

b. ((some student (sleep (some t1))) and (some instructor (begin (some t2)(some lecture))) and (every t1 (before (every t2))))

(6) a. Some instructor examines Mary after the lecture is given by John

b. ((some instructor (examine (some t1) (some mary))) and (some lecture ($give^{-1}$ (some t2) (some john))) and (every t1 (after (every t2))))

Similarly, states or facts like, some man is happy or the sun is round, are true at a particular point of time. The same question may be addressed to nominals, a student is so at a particular point of time. We will consider these issues later.

5 Current status of the implementation

Precis 1 *Given rules of the form* ***rule(Head, Body, Number)*** *- which represent the 32 inference rules in [4] (+ their contrapositives) and additional rules that account for some of the extensions - where Head and Body are expressions, written in McLogic, defining respectively the conclusion and antecedants of each rule, and given the premises of some problem (e.g. Schubert's Steamroller) already translated to McLogic using the translation process described above, a Prolog meta-interpreter shows that the conclusion of the considered problem is true and gives an explanation on how the conclusion was reached. One can ask for the explanation if one wants to.*

5.1 The inference mechanism

In [2], it was shown that for the quantifier-free fragment of the language, satisfiability is polynomial time decidable. The inference procedure is characterised by the inference rules in table 1. It was proved that the literal satisfiability problem[5] for this fragment of the language is NP-complete and this arises from the fact that for any given class expression appearing in the input, we may not know whether or not that expression denotes the empty set. Hence, the claim:

Theorem 1 *If for each class expression C in the input, either [some,C,exists] or [not,[some,C,exists]] is known then the satisfiability of a set of quantifier-free literals is polynomial time decidable.*

This theorem is proved by showing:

1. if σ is a set of quantifier-free literals where for each class expression C in σ it is known whether C is empty or not then σ is satisfiable iff $\sigma \longrightarrow F$ [6]

2. $\sigma \longrightarrow \phi$ is determined in polynomial time.

where $\sigma \longrightarrow \phi$ if ϕ can be proven using the rules in table 1 such that every class expression appearing in the proof appears in σ.
The above definition ensures that to determine whether $\sigma \longrightarrow \phi$ we need only consider formulas all of whose class expressions appear in σ. The number of such formulas is always less or equal than $K\ card(\sigma)^2$ where K is a constant and $card(\sigma)$ is the number of class expressions in σ since the inference rules allow only formulas of the form (every s t), (some s t) and (at_most_one s) . Hence, one can determine in polynomial time in the size of σ whether or not $\sigma \longrightarrow \phi$ by enumerating all derivable formulas.
The rule set in [4] include the set in table 1 except for 5 rules. It is only conjectured but not proved in [4] that though the syntax presented is more elaborate, the time complexity of the proof process is still polynomial in the number of class expressions occuring in the quantifier-free premises of the problem at hand.

The proof procedure is implemented as a backward (goal-directed) search procedure using a simple Prolog meta-interpreter. Instead of generating all permissible class expressions for a certain problem and only allow variables to be instantiated to these class expressions, we made sure that such variables are not instantiated to formulas. We are in the process of improving the efficiency[7] of the proof system.
The extension of the language required the addition of more inference rules. However, we have not proved that with these additions the proof procedure is still complete. Below, we present a running example.

[5] A literal is an atomic formula i.e. of the form (every s t) or (some s t) or a negation of an atomic formula.

[6] For a proof see Appendix B in [2].

[7] Solving Schubert's steamroller is slow relative to other implementations in the literature e.g. [6].

Table 1: Inference rules for quantifier-free literals. r,s,t range over class expressions, c ranges over constant symbols, and R ranges over relation symbols.

$$\frac{(every\ s\ t)}{(every\ (R\ (some\ s))(R\ (some\ t)))}$$

$$\frac{(every\ s\ t)}{(every\ (R\ (every\ t))\ (R\ (every\ s)))}$$

$$\frac{(every\ r\ s)\wedge(every\ s\ t)}{(every\ r\ t)}$$

$$(every\ t\ t)$$

$$\exists\ c$$

$$(at-most-one\ c)$$

$$\frac{\exists(R\ (some\ s))}{\exists\ s}$$

$$\frac{\exists\ r\ \wedge(every\ r\ t)}{\exists\ t}$$

$$\frac{at-most-one\ t\wedge(every\ r\ t)}{at-most-one\ r}$$

$$\frac{\neg(every\ r\ t)}{\exists\ r}$$

$$\frac{\exists\ s\ \wedge at-most-one\ t\ \wedge(every\ s\ t)}{every\ t\ s}$$

$$\frac{\exists\ r\ \wedge(every\ r\ s)\wedge(every\ r\ t)}{(every\ (R\ (every\ s))\ (R\ (some\ t)))}$$

$$\frac{\neg\exists\ s}{(every\ t\ (R\ (every\ s)))}$$

$$\frac{(at-most-one\ t)\wedge(every\ s\ t)}{(every(R\ (some\ s))(R\ (every\ t)))}$$

$$\frac{(every\ r\ s)\wedge(every\ r\ t)\wedge r}{(some\ s\ t)}$$

$$\frac{\psi\wedge\neg\psi}{False}$$

5.2 A running example

Given the NL specification in section 3, the program outputs the McLogic translation for both Schubert's Steamroller and the library transaction problems. Since McLogic is so similar to Natural Language the proofs below should be transparent even with no prior knowledge of the inference rules in [4] or the additional rules.

☞ Schubert's Steamroller

(every, wolf, animal). (every, fox, animal). (every, bird, animal]. (every, caterpillar, animal). (every, snail, animal). (some, wolf, exists). (some, fox, exists). (some, bird, exists). (some, caterpillar, exists). (some, snail, exists). (every, grain, plant). (some, grain, exists). (every, caterpillar, (smaller_than, (every, bird))). (every, snail, (smaller_than, (every, bird))). (every, bird, (smaller_than, (every, fox))). (every, fox, (smaller_than, (every, wolf))). (not, (some, wolf, (eat, (some, fox)))). (not, (some, wolf, (eat, (some, grain)))). (every, bird, (eat, (every, caterpillar))). (not, (some, (bird, (eat, (some, snail))))). (every, caterpillar, (eat, (some, plant))). (every, snail, (eat, (some, plant))).
(every,animal,(eat,(every,plant)) $ lambda(X,(every,X,(eat,(every,animal+((smaller_than,(some,X))+(eat,(some,plant)))))))).
Conclude: (some,animal,(eat,(some, animal+(eat,(some,grain))))).

Note that only the last premise is not quantifier-free (or lambda-free).
The full proof is automatically done but its trace is long and at the moment not very illuminating. We plan to provide a compact version but here is an illustration, reconstructed by hand, of a part of the proof.

Problem 1 *Given: the above premises and (every x_w wolf) and (focus_on x_w)*[8]*, conclude that It_is_not_the_case_that some wolf eats every plant and hence, every wolf X eats every animal that is smaller than X and eats some plant.*

Proof

$$\frac{\frac{some\ grain\ exists}{some\ grain\ grain}}{(every\ (eat\ (every\ grain))(eat\ (some\ grain)))\ \alpha_1}$$

In addition,

$$\frac{\neg(some\ wolf\ (eat\ (some\ grain)))}{\neg(some\ (eat\ (some\ grain))\ wolf)\ \alpha_2}$$

$$\frac{\alpha_1, \alpha_2}{\neg(some\ (eat\ (every\ grain))\ wolf)\ \alpha_3}$$

Moreover,

$$\frac{(every\ grain\ plant)}{(every\ (eat\ (every\ plant))(eat\ (every\ grain)))\ \alpha_4}$$

[8] This formula restricts the instantiation of the λ-quantifier to x_w and has no semantic content [4].

$$\frac{\dfrac{\alpha_3,\ \alpha_4}{\neg(some\ (eat\ (every\ plant))wolf)}}{\neg(some\ wolf\ (eat\ (every\ plant)))\alpha_5}$$

$$\frac{(every\ wolf\ animal), (every\ animal\ ((eat\ (every\ plant)) \cup \lambda x.\phi(x)))}{(every\ wolf\ ((eat\ (every\ plant)) \cup \lambda x.\phi(x)))\ \alpha_6}$$

where,
$\phi(x) = (every, x, (eat, (every, animal + ((smaller\_than, (some, x)) + (eat, (some, plant))))))$.

$$\frac{\alpha_5, \alpha_6}{(every\ wolf\ \lambda x.\phi(x))\ \alpha_7}$$

$$\frac{(every\ x_w\ wolf),\ \alpha_7}{(every\ x_w\ \lambda x.\phi(x))\ \alpha_8}$$

$$\frac{\alpha_8,\ focus\_on\ x_w}{\phi(x_w) = x_w\ (eat\ (every\ animal \cap (smaller\_than\ x_w) \cap (eat\ (some\ plant)))\ \alpha_9}$$

☞ Wing's library rules:

(every,copy,(available,'or',checked_out)).
(not,(some,copy,(available,'and',checked_out))).
(every,registered+user,borrower). (every,borrower,registered+user).
(every, staff_member,administrator). (every,administrator,staff_member).
(not,(some,staff_member,registered+user)).
(every staff_member authorised+requestor).
(every authorised+requestor staff_member).
((every,borrowed+copy,checked_out+copy).
(every,checked_out+copy, borrowed+copy)).
(some,UL,(own,(every,available+copy))).
(every,registered+user+(has_borrow (less_than (ten∗copy))),(can_borrow,(every,available+copy))).
(every user (can_borrow (some available+copy)) registered+user+(has_borrow
(less_than (ten∗copy)))).
(some staff_member authorised+requestor).
(one∗copy (less_than (ten∗copy))). (nine∗copy (less_than (ten∗copy))). (every
authorised+requestor (can_give ((every registered+user) (some+copy)))).

Problem 2 *Given the set of premises:*

{ *John is a registered user. John has borrowed 9 books.*
'Computational Linguistics: an intro' is an available copy.
Anna is an authorised requestor }

translated into the set

{ *(every/some john registered+user). (every/some john (has_borrow (nine∗copy))).*
(every/some 'computational linguistics: an intro' available+copy).
(every/some anna authorised+requestor) }

conclude that: John can borrow 'Computational Linguistics: an Intro' or (some john (can_borrow (some 'computational linguistics: an intro'))).

Proof

$$\frac{(some\ john\ (has\_borrow\ (nine * copy))), (nine * copy\ (less\_than\ (ten * copy)))}{(every\ john\ (has\_borrow\ (less\_than\ (ten * copy))))\delta_1}$$

$$\frac{\delta_1, (every\ john\ registered + user)}{(every\ john\ registered + user + (has\_borrow\ (less\_than\ (ten * copy))))\delta_2}$$

$$\frac{\delta_2, (every\ registered + user + (has\_borrow\ (less\_than\ (ten * copy)))\ (can\_borrow\ (every\ available + copy)))}{(every\ john\ (can\_borrow\ (every\ available + copy)))\ \delta_3}$$

$$\frac{(every\ 'computational\ linguistics : an\ intro'\ available + copy), \delta_3}{(every\ john\ (can\_borrow\ (every\ 'computational\ linguistics : an\ intro')))}$$

The full paper will include more examples and output.

6 Future work

Our future aim is to be able to formulate knowledge in a form of English and to be able to draw conclusions from that knowledge, also in English, in such a way that the reasoning involved is completely transparent to the user. If people cannot see how some conclusion does or does not follow from the knowledge base they will be unwilling to trust advice or information provided by such a system, quite rightly. In order to avoid the problems posed by the degree of ambiguity and vagueness found in real English, we use an artificially restricted subset that can be fully processed. The trick here is to try to define a dialect of Computer Processable English that is sufficiently restrictive to be able to process accurately, while still being expressive enough to state what is required clearly and naturally.
The work reported here has tried to develop some of the components of such a system. We translate between a restricted subset of English and a logic capable of supporting as a knowledge representation and reasoning system. The translation is transparent: as we hope will be apparent from some of the examples above, a McLogic formulation of a problem is quite succinct and English-like, and certainly a lot easier to read and follow than, say, a first order predicate calculus formulation of the same problems. This is an important property for developers, but also perhaps for some of the eventual users of such a system. Many people would like to be able to translate between natural languages and formal languages (like Z), but for most of these formal languages the expertise required in

order to judge the translation accurate is sufficient to make the need for translation less than compelling.
We have tested our implementation on two standard benchmarks: one to test the ability of Computer Processable English to formulate a standard problem of the type that formal specification languages are designed to deal with; and one to test the ability of the inference mechanism to deal with natural but complex chains of inference.
Two components need further work: the first is the user interface: keeping the user to the requisite dialect of English is not easy unless they are prepared to undergo a certain amount of training. The second is the output of the inference system. Translating inference steps one at a time back into English does not usually make for an informative description of what is going in it. We plan to experiment with proof manipulation techniques to try to turn long tedious proofs into compact informative ones before generating the natural language justification of the inference.

References

[1] Davidson D. *The Logic of Decision and Action (ed. N. Rescher)*, chapter The Logical Form of Action Sentences, pages 81–95. University of Pittsburgh Press, 1967.

[2] McAllester D. and Givan R. Natural language syntax and first-order inference. *Artificial Intelligence*, 56:1–20, 1992.

[3] B. Macias and S.G. Pulman. A method for controlling the production of specifications in natural language. *The Computer Journal*, 38(4):310–318, 1995.

[4] Givan R. McAllester D. and Sameer Shalaby. Natural language based inference procedures applied to schubert's steamroller. *AAAI*, 1991.

[5] S.G. Pulman. Controlled language for knowledge representation. In *First International Workshop on Controlled Language Applications, Katholieke Universiteit Leuven, Belgium*, pages 233–242, 1996.

[6] Manthey R. and Bry F. Satchmo: A theorem prover implemented in prolog. In *CADE 88 (9th Conference on Automated Deduction)*, pages 1–17, May 1988.

[7] Mark E. Stickel. Schubert's steamroller problem: Formulations and solutions. *Journal of Automated Reasoning*, 2:89–101, 1986.