Open and Closed World Types in NLP Systems

Dale Gerdemann
Seminar für Sprachwissenschaft, Universität Tübingen
Kl. Wilhelmstr. 113
72074 Tübingen, Germany
Email: dg@sfs.nphil.uni-tuebingen.de

1 Introduction

The use of an open-world versus a closed-world type system has often been considered as a criterion upon which to evaluate NLP systems. I will argue in this paper that this is a false criterion of evaluation.

The essential difference between the two kinds of systems is that in a closed-world approach all types are declared, whereas in an open-world approach some types may be left implicit. The open-world approach may be simulated in a closed-world system such as TROLL [6] simply by filling in all of these implicit types. Open-World systems cannot, however, always be used to simulate a closed-world system. In some open-world systems (ALE [2] and $\mathcal{TDL}$ [10]), it is simply impossible to rule out all of the unwanted implicit types. Thus a closed-world system such as TROLL is strictly more expressive than these systems. But this does not mean that closed-world systems in general are better. It simply means that open-world systems need to designed so that implicit types can be ruled out when the grammar writer so desires. The CUF system, for example, is open world. But it uses type axioms which are quite sufficient to rule out any implicit types. Thus the open-world type system of CUF and the closed-world types of TROLL are equally expressive.

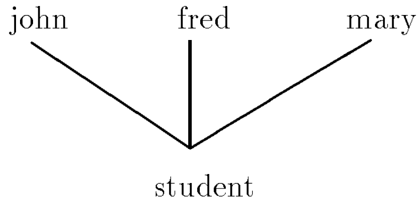
The organization is as follows. In §2, I identify two kinds of closed-world assumptions, the *partition condition* and the *disjoint species condition*. These are discussed with respect to the systems ALE and $\mathcal{TDL}$ in §3 and §4 respectively. In §5, I show that the type axioms of CUF can be translated into a closed-world type hierarchy and vice versa. Conclusions are in §6.

2 Closed vs. Open World

The terms *closed world* and *open world* are most familiar from Prolog and logic programming but as applied to type hierarchies they have a somewhat extended meaning. In Prolog, the closed-world hypothesis means essentially that the Prolog data base is complete. Given the following data base we can conclude that John, Fred and Mary are the only students in our domain:

```
student(john).
student(mary).
student(fred).
```

Similarly, this data base could be expressed as a type hierarchy:[1]



For the type hierarchy representation, however, we may identify two different kinds of closed-world assumptions [7] [8]. The *partition condition* asserts that if an object is of type t then the object is of at least one species[2] subsumed by t and the *disjoint species condition* asserts that the object is of no more than one species subsumed by t. So the partition condition rules out the possibility of some person other than John, Fred or Mary being a student. And the disjoint species condition rules out the possibility of John and Fred being the same person. The term "open-world assumption" is loosely applied to a system that violates either one of these assumptions.

3 The Exhaustive Species Condition

In ALE, an appropriateness partial function $\mathsf{Approp}\colon \mathsf{Type} \times \mathsf{Feat} \rightharpoonup \mathsf{Type}$ must be specified and a feature structure[3] is deemed to be *well-typed* iff for all nodes q in the feature structure, if $\lambda(q) = t$ and $\delta(q, f)$ is defined, then $approp(t, f)$ is also defined and $approp(t, f) \sqsubseteq \lambda(\delta(q, f))$.

The problem with the well-typedness test is that it allows feature structures that are (in Carpenter's terms) "ungroundable".[4] Suppose, for example, there is a type $a[f : bool, g : bool]$, where the appropriateness specification is stated in the square brackets. And suppose further that a has subtypes $a'[f : +, g : -]$ and $a''[f : -, g : +]$. With these types, the feature structure $a[f : \boxed{1}, g : \boxed{1}]$ is well typed. But this is a peculiar result. What kind of object would this feature structure describe? Clearly, it describes an object of type a which is neither an a' nor an a''. So there must be a third species subsumed by a, which was not included in the hierarchy.

It should be noted that maintaining well-typedness represents an attempt at maintaining groundedness. Why would any system perform well-typing otherwise? The well-typing test works, however, only as a sufficient test for excluding ungroundable feature structures, i.e., if a feature structure is not well-typable, then it is ungroundable. Thus it is possible to find cases where ALE would exclude an ungroundable feature structure where a system like $\mathcal{TDL}$, which does

[1] The hierarchy here follows the "information ordering" as in [1] and [12]. In terms of a "types-as-sets" denotational semantics, increasing information content decreases the denotation, i.e., if $t \sqsubseteq t'$ then $[\![t]\!] \supseteq [\![t']\!]$. Not all systems explicitly employ this "types-as-sets" semantics. However, such a semantics can provide a useful common ground for comparison.

[2] A *species* is a maximally specific type.

[3] A feature structure is a Moore machine, i.e., a finite state automata with a type labelling each node. I will use δ as the transition function and λ as the labelling function.

[4] In fact [8] introduced the closed-world assumptions into the TROLL system precisely in order to ensure that TROLL feature structures would be groundable. For a proof that TROLL feature structures are groundable, see [9].

not enforce run-time well-typedness would not. Suppose, for example, that the type $t[f : bool, g : bool]$ has the subtypes $t'[f : +, g : +]$ and $t''[f : +, g : -]$. Furthermore, assume that there is a type s that subsumes t'' and u (with no features appropriate for s and u). In ALE, it is not possible for $t[f : \boxed{1} bool, g : \boxed{1}]$ to unify with s. In $\mathcal{TDL}$, on the other hand, this unification is allowed. Thus the $\mathcal{TDL}$ system assumes that there is another implicit type subsumed by both t and s, which is impossible to exclude (Krieger, p.c.).

It is, in fact, possible to translate an ALE hierarchy to a closed-world hierarchy by filling in all implicit types. First define a *depth-one feature structure of type* t as a well-typed feature structure for which $\delta(q, f)$ is defined just in case:

- $q = \overline{q}$ (the root),
- $\lambda(q) = t$, and
- $approp(t, f)$ is defined.

A depth-one feature structure F of type t is minimal if there exists no distinct F', also a depth-one feature structure of type t, such that $F' \sqsubseteq F$.[5] Similarly, a depth-one feature structure F of type t is maximal if there exists no distinct F', also a depth-one feature structure of type t, such that $F \sqsubseteq F'$.[6] So, for an example based on the above hierarchy and appropriateness specification, $a[f : bool, g : bool]$ is a minimal depth-one feature structure of type a and $a[f : +, g : -]$ is a maximal depth-one feature structure of type a. The feature structure $a[f : bool]$, on the other hand, fails to be depth-one since the feature g is missing.

Now, if there exists a maximal depth-one feature structure of type t which is not unifiable with any minimal depth-one feature structure of type t' such that $t \sqsubseteq t'$, then add a new species s (a previously unused type name), such that t (and only t) directly subsumes s. Define the appropriateness specification of s identically to that of t, i.e., for each f appropriate on t, let $approp(s, f) = approp(t, f)$ and let all other features be inappropriate for s.[7]

It is clear that all implicit types can be filled in by this method. So there is a translation from an ALE hierarchy to the TROLL closed-world hierarchy. However, it is also clear that there is no general transformation in the reverse direction. If we wanted the type a to have just a' and a'' as subtypes as in the above hierarchy, then there is no way to express this in ALE.[8]

4 The Disjoint Species Condition

In addition to the exhaustive species condition, some systems, such as $\mathcal{TDL}$ [10], violate the disjoint species condition as well. The sort of open-worldness that

[5] I assume that feature-structure subsumption is defined in terms of a homomorphism between typed feature structures as in [1] and [11].

[6] Note that a maximal depth-one feature structure may contain reentrancies.

[7] This is somewhat complicated since the ALE hierarchy is only open world when the appropriateness specification forces it to be. As a simpler example, suppose we give the *student* hierarchy above a non-exhaustive interpretation. Then to make a closed-world hierarchy, we would simply make one more species with a name like *other-students* and we allow this new species to be directly subsumed only by the type *student*.

[8] See [7] [8] for discussion of how this limitation of ALE affects the ability to express feature cooccurrence restrictions. Copestake et al [3] also encountered this limitation in a large scale linguistic application.

this violation produces does not, however, affect the expressivity of the type hierarchy. Such a type hierarchy can be translated into a closed-world hierarchy simply by filling in implicit types.

First, let us call the maximal types in a such a hierarchy pseudo-species. The species in the closed-world equivalent of such a hierarchy will be the consistent sets of pseudo-species, where "consistent" means that there are no disjointness statements between any two pseudo-species.[9] A pseudo-species will subsume one of these newly constructed species just in case the pseudo-species is a member of the species.

Then we need to ensure that the closed-world type hierarchy is bounded complete. For any two non-maximal types t and t', let $\mathrm{I} = \{s \mid t \sqsubseteq s\} \cap \{s \mid t' \sqsubseteq s\}$. If $I = \emptyset$, then t and t' are inconsistent. If $I = \{s\}$, a singleton set, then the least upper bound of t and t' already exists. And if $I = S$, a non-singleton, non-empty set, and if there is no non-maximal type t''' which subsumes just the species in S, then introduce t''', a previously unused type name, and let t''' subsumes just the species in S. This provides a translation into a closed-world hierarchy. Translation in the reverse direction is even easier: simply provide a disjointness statement between each pair of species.

So the question of whether or not a system should maintain the disjoint species condition is really a question of whether consistency of types should be stated positively or negatively. The way in which user specifies the type hierarchy, however, in no way affects the underlying logic of the system. It is simply a user interface issue.

5 Boolean Type Axioms and Type Hierarchies

Finally, we come to the method of using boolean expression as type axioms. This approach was used by [4] and incorporated into the CUF system [5]. The idea is that the interpretation of a type is determined by the denotation of the type axioms, which is given as follows:

$$
\begin{array}{lll}
[\![T]\!]^I & := & T^I \qquad (T \text{ is a primitive type term}) \\
[\![t \wedge s]\!]^I & := & [\![t]\!]^I \cap [\![s]\!]^I \\
[\![t \vee s]\!]^I & := & [\![t]\!]^I \cup [\![s]\!]^I \\
[\![\neg s]\!]^I & := & D^I - [\![s]\!]^I \ (D \text{ is the domain})
\end{array}
$$

The use of such type axioms might well be considered open-world. Dörre and Eisele [5], themselves emphasize this point:

> In CUF any interpretation of the types which is not invalidated by the axioms is possible. Hence, in the absence of other information, intersections of types are note considered to be empty.

The Boolean axioms of CUF are, however, quite expressive. So they can be used to rule out unwanted unifications or implicit types that may be impossible to rule out in other open-world systems. Thus, it will be possible to translate in both directions between type axioms and closed-world type hierarchies.

[9] Typically, systems that violate the disjoint species condition allow the user to make such disjointness statements.

Translation from a closed-world type hierarchy into a conjunctive set of boolean axioms is trivial. First translate each direct subsumption relation $t \sqsubseteq t'$ into the formula $\neg t' \vee t$. If the set of species is $\{s_1, \ldots, s_n\}$ then the exhaustive species condition can be translated as $\bot \doteq s_1 \vee \ldots \vee s_n$, where $s \doteq t$ is an abbreviation for $(\neg s \vee t) \wedge (s \vee \neg t)$, and the disjoint species condition can be translated as $\{\neg(s \wedge s') \mid s, s' \in \{s_1, \ldots, s_n\}\}$.

To translate in the reverse direction, first convert the type axioms into disjunctive normal form. Then ensure that this DNF is maximal, i.e., if for some type term s, a disjunct $d = (t_1 \wedge \ldots \wedge t_n)$ contains neither the conjunct s nor $\neg s$, then replace d by the two disjuncts $(t_1 \wedge \ldots \wedge t_n \wedge s)$ and $(t_1 \wedge \ldots \wedge t_n \wedge \neg s)$. Each of these disjuncts is now as fully specified as possible given the type terms used in the type axioms. The set of such disjuncts represents the set of species allowed by the type axioms. The actually allowable species are, however, specified by interaction between the type axioms and the feature declarations, so some of these potential species must be ruled out because of inconsistent feature declarations (see [5]). The set of species so constructed, however, does not constitute a full set of species since CUF also allows undeclared *afs* (atomic feature structure) types, which should also be considered species. A list of these undeclared types could be obtained, for example, by searching through the grammar to see what undeclared types are actually being used.[10]

We could then form a power-set lattice of these species as the full type hierarchy. However, this would produce more non-maximal types than necessary given the type terms used in the type axioms. It would be more desirable to be able to reinterpret the input type terms as types in the type hierarchy.[11] Let us say that if a type term occurs as a positive literal in a species, then that species *contains* that type term. Let *spec* be the function from type terms to the set of species that contain that type term. If for some type terms t, t':

- $spec(t) = \emptyset$, then exclude t from the hierarchy.
- $spec(t) \subset spec(t')$, then let t' subsume t in the type hierarchy.
- $spec(t) = spec(t')$, then arbitrarily let t subsume t' or t' subsume t.[12]

Finally, the type hierarchy needs to be closed under least upper bounds. If, for any two types t, t' in the type hierarchy, it is the case that there is no type t'' such that $spec(t'') = spec(t) \cap spec(t')$, then let t''' be a new type name, let both t and t' subsume t''' in the type hierarchy and let $spec(t''') = spec(t) \cap spec(t')$.

6 Conclusion

I have shown in this paper that open-world and closed-world type hierarchies are potentially interchangeable. It is possible to translate in both directions between open-world CUF-style type axioms and a TROLL-style closed-world type hierarchy. So the choice between these two approaches is arbitrary. One could even imagine a system that allowed both forms of grammar specification.

[10] The few built-in types of CUF can also be translated on an ad hoc basis.

[11] This correspondence will be necessary if we want to be able to translate the feature terms used in a CUF grammar into another formalism using a type hierarchy.

[12] This is an exclusive *or*. As long as t and t' are on a unary branch, they will have the same denotation, regardless of which one subsumes the other [1].

On the other hand, some open-world systems (ALE, $\mathcal{TDL}$) have been shown to be seriously lacking in expressiveness. But the problem with these systems is not the open-world assumptions per se. It is the fact that typing system for these programs is not strong enough to exclude an open-world interpretation in those cases where this interpretation is not desirable. A system using closed-world types, on the other hand, can always simulate an open-world interpretation simply by filling in all implicit types.

References

[1] Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science 32. Cambridge University Press, 1992.

[2] Bob Carpenter and Gerald Penn. ALE *The Attribute Logic Engine, User's Guide*, 1993.

[3] Ann Copestake, Antonio Sanfilippo, Ted Briscoe, and Valeria de Paiva. The ACQUILEX LKB: An introduction. In *Inheritance, Defaults, and the Lexicon*, pages 148–163. Cambridge University Press, 1993.

[4] Michael Dorna. Erweiterung der Constraint-Logiksprache CUF um ein Typsystem. Master's thesis, Institut für Informatik, Universität Stuttgart, 1992. Diplomarbeit Nr. 896.

[5] Jochen Dörre and Michael Dorna. CUF — a formalism for linguistic knowledge representation. In Jochen Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description, Dyana Deliverable R1.2.A, DYANA-2 — ESPRIT Basic Research Project 6852*, pages 8–22, 1991.

[6] Dale Gerdemann and Thilo Götz. Troll: Type resolution system, user's guide. Technical report, University of Tübingen, 1994.

[7] Dale Gerdemann and Paul John King. Typed feature structures for expressing and computationally implementing feature cooccurrence restrictions. In *Proceedings of 4. Fachtagung der Sektion Computerlinguistik der Deutschen Gesellschaft für Sprachwissenschaft*, pages 33–39, 1993.

[8] Dale Gerdemann and Paul John King. The correct and efficient implementation of appropriateness specifications for typed feature structures. In *COLING 94, Proceedings*, pages 956–960, 1994.

[9] Paul John King. Typed feature structures as descriptions. In *COLING 94, Proceedings*, pages 1250–1254, 1994.

[10] Hans-Ulrich Krieger and Ulrich Schäfer. $\mathcal{TDL}$—a type description language for constraint-based grammars. In *COLING-94*, pages 893–899, 1994.

[11] Michael Moshier. *Extensions to Unification Grammar for the Description of Programming Languages*. PhD thesis, University of Michigan, 1988.

[12] Carl Pollard and Ivan Sag. *Head-Driven Phrase Structure Grammar*. CSLI Lecture Notes. Chicago University Press, Chicago, 1994.