The Quick Check Pre-unification Filter for Typed Grammars: Extensions

Liviu Ciortuz
CS Department, University of Iaşi, Romania
ciortuz@infoiasi.ro

The so called quick check (henceforth QC) pre-unification filter for feature structure (FS) unification was introduced by (Kiefer et al., 1999). QC is considered the most important speed-up technique in the framework of non-compiled FS unification. We present two potential ways in which the design of the quick check can be further extended: consistency sort check on coreferenced paths, and pre-unification type-checking. We analyse the effect of these extensions on LinGO, the large-scale HPSG grammar for English (Flickinger et al., 2000) using the compiler system LIGHT (Ciortuz, 2002).

1 Coreferenced Based Quick Check

Suppose that the FS φ is going to be unified with ψ, and that φ contains the coreference $\varphi.\pi \doteq \varphi.\pi'$. In this setup, if for a certain path η it happens that *sort*$(\varphi.(\pi\eta)) \wedge$ *sort*$(\psi.(\pi\eta)) \wedge$ *sort*$(\psi.(\pi'\eta)) = \bot$, then certainly φ and ψ are not unifiable. There is no a priori reason why, on certain typed grammars, coreference-based sort inconsistency would not be more effective in ruling out FS unification than sort inconsistency on mutual paths. Moreover, the integration of the two forms of QC is not complicated. However, up to our knowledge no system parsing LinGO-like grammars included the above newly presented form of (coreference-based) pre-unification QC test.

On the GR-reduced form LinGO (Ciortuz, 2004) we identified 12 pairs of non-cross argument coreferences inside rule arguments (at LinGO's source level). Interestingly enough, all these coreferences occur inside key arguments, belonging to only 8 (out of the total of 61) rules in LinGO.

To perform coreference-based QC, we computed the closure of this set Λ of coreference paths. The closure of Λ will be denoted $\bar{\Lambda}$. If the pair π_1 and π_2 is in Λ, then together with it will be included in $\bar{\Lambda}$ all pairs of QC-paths such that $\pi_1\eta$ and $\pi_2\eta$, where η is a feature path (a common suffix to the two newly selected paths). For the GR-reduced form of LinGO, the closure of Λ defined as above amounted to 38 pairs. It is on these pairs of paths that we performed the coreference-based QC test.

Using all these coreference paths pairs, 70,581 unification failures (out of a total of 2,912,623 attempted unifications) were detected on the CSLI test suite. Only 364 of these failures were not detectable through classical QC. When measuring the "sensitivity" of coreferenced-based QC to individual rule arguments, we found that out of a total of 91 rule arguments in LinGO only for 4 rule arguments the coreference-based QC detects inconsistencies, and the number of these inconsistencies is far lower than those detected by the classical QC on the same arguments. None of the pairs of coreference paths exhibited a higher failure detection rate than the first ranked 32 QC-paths. If one would work with 42 QC-paths, then only 4 of the pairs of coreference paths would score failure detection frequencies that would qualify them to be taken into consideration for the (extended form of) QC-test.

As a conclusion, it is clear that for LinGO, running the coreference-based QC test is virtually of no use. For other grammars (or other applications involving FS unification), one may come to a different conclusion, if the use of non-cross argument coreferences balances (or outnumbers) that of cross-argument coreferences.

2 Type Checking Based Quick Check

Failure of run-time type checking — the third potential source of inconsistency when unifying two typed FSs — is in general not so easily/efficiently detectable at pre-unification time, because this check requires calling a type consistency check routine which is much more expensive than the simple sort consistency operation.

While exploring the possibility to filter unification failures due to type-checking, the measurements we did using LinGO (the GR-reduced form) on the CSLI test suite resulted in the following facts:

1. Only 137 types out of all 5235 (non-rule and non-lexical) types in LinGO were involved in (either successful or failed) type-checks.[1] Of these types, only 29 types were leading to type checking failure.[2]

2. Without using QC, 449,779 unification failures were due to type-checking on abstract instructions, namely on intersects_sort; type-checking on test_feature acts in fact as type unfolding. When the first 32 QC-paths (from the GR-set of paths) were used (as standard), that number of failures went down to 92,447. And when using all 132 QC-paths (which have been detected on the non GR-reduced form of LinGO), it remained close to the preceding figure: 86,841.

3. For QC on 32 paths, we counted that failed type-checking at intersect_sort occurs only on 14 GR-paths. Of these paths, only 9 produced more than 1000 failures, only 4 produced more than 10,000 failures and finally, for only one GR-path the number of failed type-checks exceeded 20,000.

The numbers given at the above second point suggest that when trying to extend the 'classical' form of QC towards finding all/most of failures, a considerably high number of type inconsistencies will remain in the FSs produced during parsing, even when we use all (GR-paths as) QC-paths. Most of these inconsistencies are due to failed type-checking. And as shown, neither the classical QC nor its extension to (non-cross argument) coreference-based QC is able to detect these inconsistencies.

[1]For 32 QC-paths: 122 types, for 132 QC-paths: also 122.

[2]For 32 QC-paths and 132 QC-paths: 24 and 22 respectively.

```
$s$ = GR$\varphi$[ i ] $\wedge$ GR$\psi$[ i ];
if   $s \neq$ GR$\varphi$[ i ] and $s \neq$ GR$\psi$[ i ] and
     $\varphi.\pi_j$ or $\psi.\pi_j$ is defined for
             a certain non-empty path $\pi_j = \pi_i \pi$,
             an extension of $\pi_i$,
         such that $\pi_j$ is a QC-path,
then
     if   $\Psi(s).\pi \wedge$ GR$_\psi$[i] = $\perp$, where
             a $\Psi(s)$ is the type corresponding to $s$,
         or type-checking $\psi.\pi_i$ with $\Psi(s)$ fails
     then $\varphi$ and $\psi$ do not unify.
```

Figure 1: The core of a type-checking specialised compiled QC sub-procedure.

The first and third points from above say that in parsing the CSLI test suite with LinGO, the failures due to type checking tend to agglomerate on certain paths. But due to the fact that type-checking is usually time-costly, our *conclusion*, like in the case of non-cross argument coreference-based QC, is that extending the classical QC by doing a certain amount of type-checking at pre-unification time is not likely to improve significantly the unification (and parsing) performances on LinGO.

For another type-unification grammar one can extend (or replace) the classical QC test with a *type-check QC filter procedure*. Basically, after identifying the set of paths (and types) which most probably cause failure during type-checking, that procedure works as shown in Figure 1.

References

L. Ciortuz. 2002. LIGHT —a constraint language and compiler system for typed-unifi cation grammars. *KI-2002: Advances in Artificial Intelligence*. M. Jarke, J. Koehler and G. Lakemeyer (eds.), pp. 3–17. Springer-Verlag, vol. 2479.

L. Ciortuz. 2004. On two classes of feature paths in large-scale unifi cation grammars. *Recent Advances in Parsing Technologies*. H. Bunt, J. carroll and G. Satta (eds.). Kluwer Academic Publishers.

D. Flickinger, A. Copestake and I. Sag. 2000. HPSG analysis of English. *Verbmobil: Foundations of speech-to-speech translation*. Wolfgang Wahlster (ed.), pp. 254–263. Springer-Verlag.

B. Kiefer, H-U. Krieger, J. Carroll and R. Malouf. 1999. A bag of useful techniques for effi cient and robust parsing. *Proceedings of the 37th annual meeting of the Association for Computational Linguistics*, pp. 473–480.