実用的 HPSG パーザ

鳥澤健太郎　牧野貴樹　吉田稔　二宮崇　今井久夫　光石豊　金山博
建石由佳　宮尾祐介　辻井潤一
東京大学

Practical HPSG Parsers

TORISAWA Kentaro, MAKINO Takaki, YOSHIDA Minoru, NINOMIYA Takashi, NISHIDA Kenji, IMAI Hisao, MITSUISHI Yutaka, KANAYAMA Hiroshi, TATEISI Yuka, MIYAO Yusuke, TSUJII Jun'ichi
University of Tokyo

1 Introduction

This note is a progress report of the project aiming at developing HPSG-based NLP systems for practical purposes such as information extraction. A major obstacle for implementing a practical HPSG-based NLP systems is inefficiency of current HPSG parsers. We have developed efficient low-level substrates for efficient HPSG-based NLP systems, including a programming language LiLFeS [1, 2], a parallel substrate for typed feature structures (PSTFS) [3], and sequential/parallel parsers on LiLFeS and PSTFS [4, 5]. Currently, we aim at developing wide-coverage grammars on the substrate and integrating HPSG framework with accomplishments of other research paradigms such as statistical methods. In this report, we describe our accomplishment on LiLFeS, sequential HPSG parser, and wide-coverage grammars.

2 LiLFeS

LiLFeS is a programming language for processing typed feature structures[6]. A LiLFeS program consists of definite clauses including a specification of typed feature structures. Its control structure is similar to that of Prolog. Source code in LiLFeS is compiled into abstract machine code. Descriptions of typed feature structures in the source code are also compiled to sequences of abstract machine code that performs unification. Currently, we have two versions of LiLFeS compilers, namely WAM-based LiLFeS compiler [1] and BAM-based LiLFeS compiler [2].

The abstract machine that we have designed for the WAM-based LiLFeS compiler is based on WAM [7] for Prolog and Carpenter's abstract machine for attribute value logic (AMAVL) [8]. The generated code is executed by an emulator of the

表 1: Performance Comparison

	System	Simple	HPSG
(1)	LiLFeS Native-Code Compiler	2.54	1.30
	OLD-LiLFeS	3.89	1.61
	ProFIT on SICStus Native	1.00	5.17
	ProFIT on SICStus Emulator	2.12	5.87
	ALE 3.1 on SICStus Native	24.85	3.86
	ALE 3.1 on SICStus Emulator	70.65	9.77
(2)	LiLFeS Native-Code Compiler	0.64	0.45
	LiLFeS Byte-Code Emulator	2.23	0.97
	ProFIT on SICStus Emulator	3.28	5.07
	ALE 3.1 on SICStus Emulator	55.70	7.72

(Environment: (1) Sun UltraSparc 1/250MHz (2) Intel Pentium II/400MHz)
(Simple: toy parser that parses English sentence)
(HPSG: HPSG parser and grammar distributed with ALE)

abstract machine. On the other hand, the BAM-based LiLFeS compiler generates the code for a different abstract machine that is designed by introducing features of AMAVL to Berkeley Abstract Machine for Prolog[9]. The BAM-based compiler is more efficient than the WAM-based one. This is due to the facts that 1) execution of programs was done by native code converted from the abstract machine code and 2) the code is optimized by using static analysis, including abstract interpretation of LiLFeS code. The optimization becomes possible because BAM's instructions are finer-grained than WAM instructions and one can eliminate redundant instructions without changing behaviors of programs.

In addition to the abilities to compile a source code into efficient abstract machine code and to execute it effectively, we observed that 1) efficient copying and equivalence checking of typed feature structures, and 2) efficient normal/association array access support are crucial for building practical HPSG-based NLP systems.

LiLFeS provides an efficient built-in copy routine for typed feature structures and built-in equivalence checking. Furthermore, the memory organization is designed so that large tables of feature structures can be supported. The copy routine of LiLFeS performs *normalization* of feature structures as well as copying. Copied feature structures occupies a continuous region of memory space according to a certain order. Then, further copying and equivalence checking do not require recursive traverse of feature structures that accompanies expensive stack operations. We have observed that these functionalities lead to significant speed-up in parsing with HPSG.

In addition, LiLFeS language and its compiler provides mechanisms that enable efficient access of large arrays of typed feature structures for the efficient access to large data required by most recent accomplishments in NLP, such as corpus or arrays of statistical parameters. For instance, LiLFeS provides association array that can store hundreds Megabytes of typed feature structures.

Table 1 shows the performance comparisons of WAM/BAM-based LiLFeS compilers with other typed feature structure processing engines. As one can see, the BAM-based LiLFeS compiler outperforms other engines in the tasks including parsing or other applications using typed feature structures.

3 HPSG Parsers

Currently, we have implemented two different HPSG parsers, namely, a CKY-style parser and a 2-phased parser. This section focuses on the most efficient parser, the 2-phased parser. The key idea for the 2-phased parser is in compilation of HPSG formalism to simpler devices, CFG and simple procedures augmenting CFG[4].

The compilation is done by *partial evaluation* of grammar descriptions prior to parsing. Roughly, at compile time, rules in HPSG are applied to each lexical entry in a recursive way. Though in a normal bottom-up parsing, all the feature structures representing daughters must be given to the rules, only one feature structure is given to principles or ID-schemata as a head-daughter. The feature structures generated by this process can be seen as a *prediction* of feature structures to be generated at parsing time, and it is possible to obtain CFG and procedures that restricts possible parsing paths from the *predicted* feature structures.

The parsing process using the compiled grammar is divided into two phases. At Phase 1, an input sentence is parsed with the augmented CFG compiled from HPSG. Thus, Phase 1 generates skeletal parse trees from the input sentence. At Phase 2, unification is applied to the parse trees completed at Phase 1. Since the augmented CFG does not keep all the constraints described in the original HPSG, all the resulting parse tress of the Phase 1 parsing are not valid with respect to the original HPSG. Then, the validity of the parse

表 2: Speed-up by the 2-Phased Parsing Algorithm

Algorithm Algorithm	Average parsing time per sentence
CKY-style Parser	1,110 msec
2-Phased Parser	552.7 msec

Input : randomly selected 1000 sentences in the EDR corpus (Average Length 21.4 words), Settings : DEC AlphaServer 4100 (400MHz, 4GByte Memory)

trees generated at Phase 1 must be verified by unification at Phase 2. However, parsing with the augmented CFG is far less expensive than naive parsing with unification. By restricting the possible parse trees with the less expensive operation, amount of unification is significantly reduced, and parsing time by a whole algorithm is shortened. The performance of current 2-phased parser is shown in Table 2.

Since the performance is measured using WAM-based LiLFeS compiler, further speed-up by the BAM-based LiLFeS compiler is expected. In addition, we used a practical grammar for Japanese, which will be described in the next section. This means that future application systems will parse real-world sentences in this speed.

We are now developing two more optimization techniques. The first is a packing of feature structures[10], and the second is unification of normalized feature structures[11]. The packing method for feature structures are developed for handling many lexical entries for one word. We are expecting that our grammar is enriched by more detailed information, including semantic information, in future. Then the number of lexical entries given to each word will be larger. We have to keep the current efficiency even in that case. Our packing method creates a lexical entry from several lexical entries by introducing disjunctive feature structures. Since the common part of those lexical entries are processed just one time in the packed lexical entry, the total parsing time can be reduced. We observed 3-4 times speed up in parsing with our English grammar. The second method, unification of normalized feature structures, reduces a frequency of stack operation during unification. Usually, feature structures are represented as a directed graph with pointers, and their unification requires recursive traversal of the feature structures with pushing/popping pointers to a stack. The stack operations consume a large portion of parsing time. Our method eliminate the stack operations from unification process by giving a fixed size of block to a feature structure and fixed indexing for nodes in the feature structure. More precisely, there are one-to-one correspondence between a node in a feature structure and an address off-set in which the node is stored, in memory images of normalized feature structures. The unification of the normalized feature structures can be done by a simple loop with fewer stack operations because of the correspondence. We observed 1.6 to 2.5 times speedup in preliminary experiments.

4 Wide Coverage HPSG-based Grammars

As noted in the Introduction, we have started to implement wide coverage grammars on our system. The target languages are Japanese and English. We took different approaches to realize a grammar for each language. For Japanese, we relied on yet another desirable characteristics of HPSG, underspecification. Underspecification makes it possible to write a concise, but wide-coverage, grammar. In addition, we observed that the HPSG framework allows us to accomplish rather high-precision in parsing with a concise grammar description.

For implementing English grammar, we translated an existing wide-coverage grammar to our formalism[12]. We chose Tree Adjoining Grammar (TAG) known as the XTAG[13] as the orig-

Grammar	Coverage	Precision
(a)	98.3%	75.1%
(b)	87.5%	78.8%

(a) Grammar without heuristics
(b) Grammar with heuristics
Input: randomly selected 20295 sentences in the EDR-corpus (agerage length 21.3 words, or 8.9 bunsetsu's).
Coverage means the ratio of sentences that the grammar can generate parse trees successfully. Precision is a precision of dependencies between bun-setsu, which is a basic unit of Japanese consisting of a few words.

表 3: Parsing results with Japanese HPSG-based grammars

inal grammar. The purpose of this conversion is not only to obtain a wide-coverage grammar, but also to examine theoretical relationships between HPSG and TAG.

The next section describes our Japanese grammar SLUNG (Small Lexicon Underspecified Nipponese Grammar) in more detail.

4.1 SLUNG

SLUNG consists of only 8 rule schemata, 286 lexical entries, and 110 lexical entry templates. Lexical entry templates are underspecified lexical entries given to each part of speech, not to each individual word. In other words, lexical entry templates describes general behavior of words belonging to each part of speech and they contain less information than a specific lexical entry to be given to an individual word. Unlike other grammar formalism, a lack of specific description does not mean a parsing failure in HPSG, and one can parse complex sentences in real-world text successfully using those templates.

Another characteristic of our Japanese grammar is in a treatment of heuristics. For analyzing Japanese syntactic structures, commercial MT systems and practical parsers such as KNP [14] use several heuristics, whose origins are found in traditional descriptive linguistics. Unlike the analysis of linguistic phenomena found in theoretical linguistic literatures, those heuristics are not always valid and their introduction to HPSG has not been discussed. But we observed that some of the heuristics can be treated in HPSG without changing the overall architecture of HPSG. For instance, a postpositional phrase marked by postposition "wa" in Japanese tends to be an argument of the verb in the end of the sentence as far as another noun phrases in the sentence is not marked by another "wa." This kind of heuristics can be described in typed feature structures easily and it can be applied by simply unifying the feature structure with other principles. Table 3 shows the coverage and the precision of the current grammar. In the experiments, a unique parse tree was picked up from possible trees by the simple rule that any constituent is attached to the closest constituent as far as the grammar allows. The performance of the grammar without heuristics and the grammar with heuristics is compared. Note that basic grammar description showed rather high precision in spite of the simple tree-choosing rule. This figure is even higher than reported results of some statistical methods. When other heuristics, including one for "wa," are unified to principles, the precision was improved around 5%.

We think that the simple method for treating heuristics in a grammar description may suggest a solution to the problem that current commercial MT system or practical parsing systems suffer, unmanageable complexity and size of their grammar descriptions. The problem becomes serious in existing systems because of their ad-hoc grammar formalisms that do not allow independent implementation of core part of grammar and heuristics. In other words, linguistically valid grammar and heuristics cannot be distinguished easily in grammar descriptions in such current practical NLP systems. Our partition of purely linguistic grammar description and heuristics in the HPSG framework, and their merging by simple unification may provide alternative view for modularizing grammar descriptions properly. We are going to make the treatment more sophisticated by introducing default reasoning and

statistics to the HPSG framework.

5 Concluding Remarks and Future Work

We have reported the current status of our HPSG-based parsers and grammars. Our HPSG-based parsers are efficient and practical enough to apply them to real-world text. We are going to use the parsers and the grammars for knowledge acquisition, including more sophisticated statistical methods, and information extraction from unlimited corpora. We will also explore the possibility of using our HPSG parsers in real-time applications such as natural language interfaces. We are expecting that our programming tools provide a uniform representation for various types of applications, and contribute to rapid improvement of those systems' quality.

参考文献

[1] T. Makino, K. Torisawa and J. Tsujii: "LiLFeS — practical unification-based programming system for typed feature structures", Proc. NLPRS'97, pp. 239–244 (1997).

[2] T. Makino, M. Yoshida, K. Torisawa and J. Tsujii: "LiLFeS – towards a practical HPSG parser", Proc. COLING-ACL '98, pp. 807–811 (1998).

[3] T. Ninomiya, K. Torisawa and J. Tsujii: "An efficient parallel substrate for typed feature structures on shared memory parallel machines", Proc. COLING-ACL '98, pp. 968–974 (1998).

[4] K. Torisawa and J. Tsujii: "Computing phrasal-signs in HPSG prior to parsing", Proc. 16th COLING, pp. 949–955 (1996).

[5] T. Ninomiya, K. Torisawa and J. Tsujii: "Parallel HPSG parser", The 4th Annual Meeting of Japanese Association for Natural Language Processing (1998).

[6] B. Carpenter: "The Logic of Typed Feature Structures", Cambridge University Press (1992).

[7] H. Aït-Kaci: "Warren's Abstract Machine, A Tutorial Reconstruction", The MIT Press (1991).

[8] B. Carpenter and Y. Qu: "An abstract machine for attribute-value logics", Proc. of the 4th International Workshop on Parsing Technologies (1995).

[9] P. L. Van Roy: "Can logic programming execute as fast as imperative programming?", Technical Report CSD-90-600, University of California, Berkeley (1990).

[10] Y. Miyao, K. Torisawa, Y. Tateisi and J. Tsujii: "Packing of feature structures for optimizing the HPSG-style grammar translated from TAG", Proc. TAG+4, pp. 104–107 (1998).

[11] K. Nishida, K. Torisawa and J. Tsujii: "Path-address correspondence method for unification of typed feature structures", Submitted to 37th ACL.

[12] Y. Tateisi, K. Torisawa, Y. Miyao and J. Tsujii: "Translating the XTAG English grammar to HPSG", Proc. TAG+4, pp. 172–175 (1998).

[13] The XTAG Research Group: "A lexicalized tree adjoining grammar for english", IRCS Research Report 95-03, University of Pennsylvania (1995). available in `http://www.cis.upenn.edu/~xtag/tech-report/tech-report.html`.

[14] S. Kurohashi and M. Nagao: "A syntactic analysis method of long japanese sentences based on the detection of conjunctive structures", Computational Linguistics, **20**, 4, pp. 507–534 (1994).