Processing Coordinated Structures in PENG Light

Rolf Schwitter

Centre for Language Technology,
Macquarie University,
Sydney NSW 2109, Australia
Rolf.Schwitter@mq.edu.au

Abstract. PENG Light is a controlled natural language designed to write unambiguous specifications that can be translated automatically via discourse representation structures into a formal target language. Instead of writing axioms in a formal language, an author writes a specification and the associated background axioms directly in controlled natural language. In this paper, we first review the controlled natural language PENG Light and show how a discourse representation structure is generated for sentences written in PENG Light. We then discuss two different solutions of how discourse representation structures can be implemented for coordinated structures. Finally, we show how an efficient implementation of coordinated structures combined with a suitable parsing strategy affects the parsing performance of the controlled natural language.

Keywords: controlled natural language, parsing, coordination, discourse representation structures.

1 Introduction

PENG Light is a controlled natural language designed for representing knowledge in an unambiguous way [10]. Specifications written in PENG Light can be translated into a formal target language for automated reasoning. The language of PENG Light covers a strict subset of standard English and is defined by a controlled grammar and a controlled lexicon [12]. The language processor of PENG Light uses a unification-based phrase structure grammar that is based on a definite-clause grammar (DCG) notation [8]. In order to avoid redundant analysis and for practical reasons that we will discuss later, the DCG notation is automatically transformed into a format that can be processed easier by a chart (= tabular) parser [1].

In general, the DCG notation can be used to write context-free as well as context-sensitive grammars and these grammars can be executed directly using Prolog's top-down, depth-first, backtrack search. If no left recursion is present in context-free grammars, then the worst-case parsing performance is exponential in the size of the input, and if tabulation is used, then the performance is – in theory – cubic for context-free grammars (but tabulation is expensive in Prolog).

D. Wang and M. Reynolds (Eds.): AI 2011, LNAI 7106, pp. 658–667, 2011.

The situation, however, is more complex in our case, since the parsing performance is less clear for context-sensitive grammars because of the wide range of expressiveness that is allowed in these formats. The grammar of PENG Light uses, for example, feature structures in the arguments of the grammar rules and interleaves syntactic, semantic, and pragmatic information. Furthermore, the anaphora resolution algorithm of PENG Light is directly embedded into the grammar and investigates the discourse representation structure during parsing to find possible antecedents. Another thing that adds to the complexity is that the chart parser collects look-ahead information during parsing to support the writing process of the user. In this paper, we show that the way the grammar of PENG Light is written is critical for the parsing performance. We demonstrate this by example of coordinated structures in PENG Light and investigate how discourse representation structures can be constructed efficiently for coordinated structures. The parsing performance is then evaluated in an empirical way for an entire specification text.

The rest of this paper is structured as follows: In the next section we review the controlled natural language PENG Light and provide an example specification that we then use for our performance analysis. In Section 3, we look at the language processor of PENG Light and explain how chart parsing works with a special focus on chart parsing in Prolog. In Section 4, we first show how simple and complex discourse representation structures are generated in PENG Light, and then discuss different ways how discourse representation structures for coordinated structures can be implemented. In Section 5, we use the example specification introduced in Section 2 and evaluate the performance of the chart parser taking various settings for the parser as well as the proposed solutions for coordinated structures into consideration.

2 PENG Light

PENG Light is a controlled natural languages that can be used as a high-level knowledge representation language [12]. By design, PENG Light eliminates both ambiguity and complexity of full natural language but adheres to the same rigorous principles as formal languages do. At first glance, a specification written in PENG Light **looks seemingly informal** and is therefore easy to read and understand by a human; nevertheless, the specification can be translated unambiguously via discourse representation structures into a first-order logic theory. Below is an example of a specification text written in PENG Light that describes knowledge in a dynamic domain. This specification can be written as a coherent piece of text, but we can distinguish four different forms of knowledge (A-D) in this dynamic domain:

A. Knowledge about Events and their Effects

1. If a person arrives at a location then the person will be at that location.
2. If a person is at a location and a vehicle is waiting at that location and the person gets on that vehicle then the person will be in that vehicle.

3. If a person is in a vehicle and the vehicle leaves a location then the person will no longer be at that location and the vehicle will no longer be waiting at that location.

B. Terminological Knowledge

4. Every airport is a location.
5. Everybody who is John is a person.
6. Every bus is a vehicle.
7. Every Burwood bus is a bus.

C. Initial Domain State

8. The Burwood bus is waiting at the airport.

D. Sequence of Domain Events

9. John arrives at 10:10 with Qantas Flight QF2 at the airport of Sydney.
10. John gets on the Burwood bus.
11. The bus leaves the airport at 11:00.

The language processor of PENG Light translates this specification incrementally into the input language of the Simplified Event Calculus [5,6,11], and the author can query the resulting theory in controlled natural language as the events (9-11) unfold.

As the specification text illustrates, the syntactic structures of PENG Light sentences can be simple or complex. Simple sentences such as (8-11) have the following functional structure: subject + verb + [complements] + {adjuncts}. Complex sentences (1-7) are built from simpler sentences with the help of coordination, subordination, quantification and negation. Questions are derived from declarative sentences via movement of constituents, usually referred to as filler-gap dependencies [8]. Only restricted forms of anaphoric references are allowed in PENG Light: anaphoric references can be establish via definite noun phrases, proper nouns, and variables. For example, sentence (8) introduces a new object (*Burwood bus*) into the discourse and the corresponding noun phrases in (10) and (11) refer to this object. It is important to note that the author of a specification text does not need to remember the restrictions of the controlled natural language since these restrictions are enforced by a predictive authoring tool [9]. This authoring tool guides the writing process and informs the author with the help of look-ahead information about the words and phrases that can follow the current input.

3 The Language Processor

The language processor of PENG Light uses a chart parser and a unification-based phrase structure grammar to process the input text incrementally. The chart parser resolves anaphoric references, builds a syntax tree, a discourse representation structure, a paraphrase, and extracts look-ahead information from the chart. The grammar itself is written in DCG notation and consists of about

200 grammar rules; however, we do not use the DCG directly. Instead we use term expansion [13], a source-to-source transformation technique, to transform the DCG notation into a format that is easier to process with the help of a chart parser. We do this, because we want to avoid redundant analysis and because we need a way to process the input incrementally on a word-by-word basis so that we can extract look-ahead information from the chart to guide the writing process of the author.

The chart parser is an agenda-based active chart parser [1,3,4] that stores active edges (hypotheses about constituents) and passive edges (complete constituents) in the knowledge base. That means the chart parser uses the Prolog knowledge base directly as agenda. Every active edge represents a hypothesis that needs to be further explored. The fundamental rule [1] of chart parsing combines active edges with passive edges to generate new edges. These new edges can be either active or passive. We use a top-down rule invocation strategy and regard the agenda as a queue; new edges are added to the queue (by asserting them to the end of the knowledge base); this implements a breadth-first search strategy (we will discuss the impact of the search strategy on the parsing performance in Section 5). The edges of the chart are stored in the following form in the knowledge base:

12. `edge(SNum, Mode, SVNum, EVNum, LHS, Found, ToFind)`

Here, the first argument `SNum` stands for the sentence number, the second argument `Mode` for the mode of the sentence, the third argument `SVNum` for the number of the start vertex, the fourth argument `EVNum` for the number of the end vertex, the fifth argument `LHS` for the category on the left-hand side of a grammar rule, the sixth argument `Found` for the categories that have been found so far, and finally the seventh argument `ToFind` for the categories that still have to be found in order to complete an edge. For efficiency reasons, we index the sentence number (`SNum`), the number of the start vertex (`SVNum`), the number of the end vertex (`EVNum`), and the category on the left-hand side of a grammar rule (`LHS`). In SWI Prolog [13] up to four arguments of a predicate can be indexed, compound terms such as the left-hand side category of a grammar rule are indexed on the combination of their name and arity. We will discuss the impact of argument indexing on the parsing performance in Section 5.

4 Discourse Representation Structures (DRSs)

The grammar of PENG Light uses feature structures to generate a discourse representation structure (DRS) during parsing. A DRS consists of a set of discourse referents `U` and a set of conditions `Con`. Conditions can be either basic or complex: basic conditions store information about discourse referents or relations between discourse referents, and complex conditions contain embedded DRSs. DRSs are defined recursively and their nesting predicts which discourse referents are accessible via anaphoric expressions and which ones are not accessible. In contrast

to standard discourse representation theory [2], we use a neo-Davidsonian representation for events and thematic roles [7] to connect discourse referents that represent events with other discourse referents that are described in a sentence.

4.1 Building Basic and Complex DRSs

We represent a basic DRS in the grammar of PENG Light as a Prolog term of the form: `drs(U, Con)` where both `U` and `Con` are lists. This basic DRS is initially empty and processed with the help of a difference list that collects the discourse referents and relevant conditions during parsing. The following grammar rule `s0` takes the existing DRS `D1` as input and generates the DRS `D3` as output. The contribution of the verb phrase `v3` to the DRS `D3` is collected with the help of the difference list `D2-D3` and this information is combined with the contribution of the noun phrase `n3` (depending on the determiner of the noun phrase):

```
13. s0([ sem:[E], tree:[s0, T1, T2], drs:D1-D3, para:P1-P4,
         gap:G1-G3, snum:N ]) -->
      n3( [ syn:[agr:[Pers, Num, case:nom]], sem:[M], tree:T1,
            drs:D1-D3, sco:D2-D3, para:P1-P2, gap:G1-G2,
            snum:N ]),
      v3( [ crd:C, syn:[vform:fin, agr:[Pers, Num, case:nom]],
            sem:[E, M], tree:T2, drs:D2-D3, para:P3-P4,
            gap:G2-G3, snum:N ]).
```

Apart from the feature structures for the DRS (`drs, sco`), the grammar rule (13) contains feature structures for the processing of syntactic information (`crd, syn, tree, gap`), semantic information (`sem`), pragmatic information (`para`) and information for keeping track of the sentence number (`snum`). Let us illustrate how a simple DRS looks like. In PENG Light the processing of the sentence (14):

14. A bus leaves the airport at 11:00.

results in the following DRS:

```
15. drs([A, B, C, D],
        [object(A, bus), theta(B, agent, A),
         event(B, leaving),
         theta(B, theme, C), object(C, airport),
         theta(B, time, D), timex(D, '11:00')])
```

The single conditions that occur in this DRS are retrieved during parsing from the lexicon. For example, the lexical entry for the word form *leaves* contains among other information the three conditions: `theta(B, agent, A), event(B, leaving)`, and `theta(B, theme, C)`.

The contribution of the determiners to a DRS deserves closer investigation: determiners are the most important constituents to establish the DRS. Semantically, determiners have two arguments: a restrictor (`res`) and a scope (`sco`). The

restrictor consists of the information in the noun phrase minus the determiner, and the scope is the information outside of the noun phrase. In the case of an indefinite noun phrase (*a bus*), the lexical entry (16) for the indefinite determiner (*a*) specifies – among other things – that the output variable of the restrictor (`D2`) is same as the input variable for the scope (`D2`):

16. `drs:D1-D3, res:D1-`**`D2`**`, sco:`**`D2`**`-D3.`

This has the consequence that the conditions derived from the verb phrase are simply added to the current DRS (`D2`) that contains the previous contextual information, inclusive the information derived from the noun phrase. The situation gets a bit more complex if other determiners such as *every* or *no* occur in a sentence. The determiner *every* triggers a complex DRS (17) where the DRS for the restrictor (`drs(U2, C2)`) is combined with the DRS for the scope (`drs(U3, C3)`) and the result is embedded into the conditions of the superordinate DRS (`drs(U1, C1)`) using the operator `=>` to denote material implication. The lexical entry of this determiner contains the following information:

```
17. drs:D1-[drs(U1,[drs(U2, C2) => drs(U3, C3)|C1])|T],
    res:[drs([],[])|D1]-D2,
    sco:[drs([],[])|D2]-[drs(U3, C3), drs(U2, C2), drs(U1, C1)|T].
```

The determiner *no* triggers a similar structure (18) like *every*. The only difference is that the DRS built up in the scope is negated `~drs(U3, C3)` and embedded into a DRS in the consequent of the implication:

```
18. drs:D1-[drs(U1,[drs(U2, C2) => drs([],[~drs(U3, C3)])|C1])|T],
    res:[drs([],[])|D1]-D2,
    sco:[drs([],[])|D2]-[drs(U3, C3), drs(U2, C2), drs(U1, C1)|T].
```

4.2 Building DRSs for Coordinated Structures

In PENG Light, we can coordinate sentences, relative clauses, verb phrases, adjective phrases, and adverbial phrases by means of *and* and *or*. Coordination is only possible between complete constituents of the same syntactic type and the standard binding order of logic applies (that means the coordinator *and* binds stronger than the coordinator *or*). Our example specification contains two complex sentences (2 and 3) where coordination occurs between simple sentences. Apart from these syndetic cases of coordination where a coordinator marks the coordinated constituents, sentence (9) shows an asyndetic case of coordination where the coordinator is absent. In sentence (9), a number of prepositional phrases occur in adjunct position of the sentence without an explicit coordinator between these constituents, but the implementation of asyndetic coordination requires a very similar grammar rule as for syndetic coordination. In the following, we present two solutions of how coordinated structures can be implemented in PENG Light and focus on the coordination of simple sentences, coordination for the other constituents follows similar rules.

First Solution. The first solution relies on two grammar rules: one for conjunction and one for disjunction. Recall that a DRS has the form `drs(U, Con)` and that this structure is updated during parsing. For example, we can implement sentence coordination using the following two grammar rules (19 and 20):

```
19. s1([ crd:yes, tree:[s1, T1, T2, T3], drs:D1-D3, para:P1-P4,
         snum:N ]) -->
      s1( [ crd:no, tree:T1, drs:D1-D2, para:P1-P2, snum:N ]),
      crd([ cat:conj, tree:T2, para:P2-P3 ]),
      s1( [ crd:C, tree:T3, drs:D2-D3, para:P3-P4, snum:N ]).

20. s1([ crd:yes, tree:[s1, T1, T2, T3],
         drs:D1-[drs(U1, [D3 v D4|Con1])|Top],
         para:P1-P4, snum:N ]) -->
      s1( [ crd:no, tree:T1,
            drs:[drs([], [])|D1]-[D3|D2],
            para:P1-P2, snum:N ]),
      crd([ cat:disj, tree:T2, para:P2-P3 ]),
      s1( [ crd:C, tree:T3,
            drs:[drs([], [])|D2]-[D4, drs(U1, Con1)|Top],
            para:P3-P4, snum:N ]).
```

In the case of a conjunction (19), the DRS is passed through the grammar rule with the help of a difference list, first from `D1` to `D2` and then from `D2` to `D3`. In the case of a disjunction (20), each disjunct uses its own DRS, in our case `D3` and `D4`, and these DRSs are finally embedded into the existing DRS `drs(U1, Con1)` resulting in: `drs(U1, [D3 v D4|Con1])`. It is important to note that both disjuncts start with an empty DRS `drs([], [])` followed by the superordinate DRS `D1` and `D2`, respectively. At first glance, this solution looks intuitive, but it has the disadvantage that it duplicates the grammar rules for coordination on each level and this increases the processing time dramatically, as we will see in Section 5.

Second Solution. The second solution uses only one grammar rule for conjunction and disjunction and delegates the work for coordination to the category `crd` for coordination. This makes the grammar less speculative, since we provide the relevant structure for the DRS only when the coordinator is processed:

```
21. s1([ crd:yes, tree:[s1, T1, T2, T3], drs:D1-D3, para:P1-P4,
         snum:N ]) -->
      s1( [ crd:no, tree:T1, drs:D1-D2, para:P1-P2, snum:N ]),
      crd([ cat:coord, tree:T2, drs:D2-D3, hld:D1, sco:D4,
            para:P2-P3 ]),
      s1( [ crd:C, tree:T3, drs:D4, para:P3-P4, snum:N ]).
```

Here, the category `crd` takes the DRS `D1` as well as the DRS `D2` as input. The former one contains the information before the first conjunct `s1` is processed, and the latter one contains the information after the first conjunct has been processed. The individual grammar rules for the category `crd` then take care of the further processing as we will see now.

In the case of a conjunction, the grammar rule (22) makes sure that the DRS in `D2` can flow from the first conjunct to the second one and returns the result in `D3`:

```
22. crd([ cat:coord, tree:[conj, [and]], drs:D2-D3, hld:_,
          sco:D2-D3, para:P1-[[and]|P1] ]) -->
      [and],
      { lexicon([cat:conj, wform:[and]]) }.
```

In the case of a disjunction, the grammar rule (23) makes sure that the DRSs for the disjuncts are properly embedded into the existing DRS. Recall that the DRS `D1` contains the information before processing the first disjunct `s1` and the DRS `D2` contains the information after processing this disjunct. The difference between these two DRSs is the DRS `drs(U3, Con3)` that contains the relevant information for the first disjunct. The DRS for the second disjunct is then built up in `D4`:

```
23. crd([ cat:coord, tree:[disj, [or]],
          drs:D2-[drs(U5, [drs(U3, Con3) v D4|Con5])|Top],
          hld:D1,
          sco:[drs([],[])|D3]-[D4, drs(U5, Con5)|Top]
          para: P1-[[or]|P1] ]) -->
      [or],
      { lexicon([ cat:disj, wform:[or],
                  drs:D2-[drs(U3, Con3)|D3], hld:D1 ]) } .
```

Note that the actual difference between the DRS `D1` and `D2` is calculated with the help of the lexical rule for the coordinator `or` whenever this coordinator is processed.

5 Evaluation

We evaluated the performance of the chart parser for the two presented solutions using the example specification introduced in Section 2. We processed this specification on a 2.4 GHz Intel Core 2 Duo Windows machine with 2 GB RAM running SWI-Prolog (32 bits, Version 5.11.23). The lexicon that we used for this evaluation consists of 2,326 entries and is of a realistic size for a controlled language application, most of these entries represent content words.

In Figure 1, we compare the processing times for each sentence of the example specification and take the two solutions for processing coordinated structures into consideration. We used the chart parser with a top-down, breadth-first parsing

strategy and measured the processing times in milliseconds for the first solution with argument indexing (s1-a) and without argument indexing (s1-b), as well as for the second solution with argument indexing (s2-a) and without argument indexing (s2-b). Recall that the sentences are processed in context; that means the chart parser takes the DRS of the previous sentences into account while the current sentence is processed in order to resolve anaphoric references. The most efficient solution is the second solution with argument indexing (s2-a). This solution is on average 4.29 times faster than the first solution with argument indexing (s1-a). Argument indexing gives us a speed-up of about 15% for the second solution (s2-a) compared to the same solution without argument indexing (s2-b). In Figure 2, we compare the processing times for the best solution using a breadth-first parsing strategy (s2-a-bf) and a depth-first strategy (s2-a-df). The breadth-first strategy gives us an overall speed-up of about 12%, only sentence 4 was processed slightly faster using the depth-first strategy.

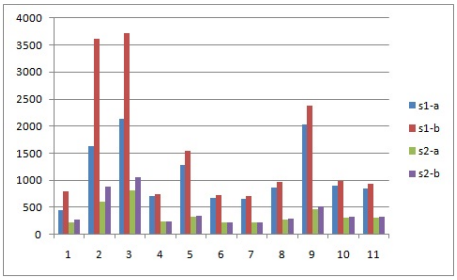

Fig. 1. Overall Comparison

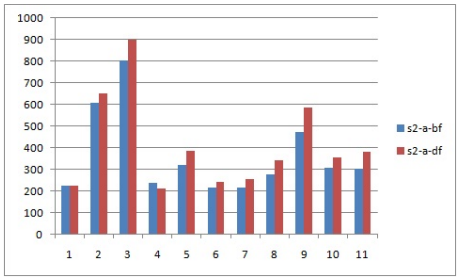

Fig. 2. Parsing Strategy

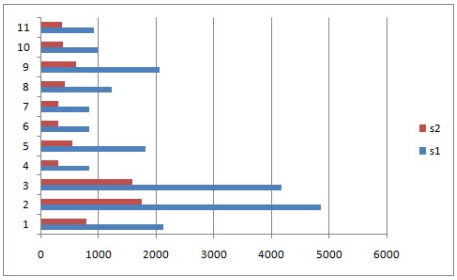

Fig. 3. Number of Edges

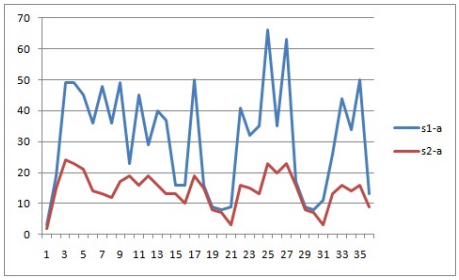

Fig. 4. Sentence 3

Figure 3 shows the number of edges that are generated for each sentence of the specification and compares the first solution for coordinated structures (s1) with the second solution (s2). The first solution creates 20.648 edges for the entire specification and the second solution 7.369 edges, that means the second solutions reduces the number of edges by a factor of 2.8. Since the specification text is usually written in an incremental fashion and look-ahead categories are generated for each word, it is interesting to compare the processing times for

each word of a sentence. In Figure 4, we compare the processing times for each word of sentence 3 taking the first solution (s1-a) and the second solution (s2-a) into consideration, using argument indexing and the top-down, breadth-first parsing strategy. The interesting result here is that the processing times for both solutions are in the millisecond range, in the case of the second solution (s2-a), the processing times are well below 30 milliseconds, that means the author will not experience any delay while typing a text, since a delay of up to 100 milliseconds is generally not perceivable by a human.

6 Conclusions

In this paper, we showed how discourse representation structures are implemented for coordinated structures in PENG Light. The choice of the implementation for coordinated structures has a dramatic effect on the overall parsing performance. Argument indexing and a suitable parsing strategy (in our case a top-down, breadth first search strategy) can further speed up the processing of a specification. Furthermore, we showed that our approach is fast enough to support the incremental processing of a specification text, since there will be no perceivable delay when an author writes a text in PENG Light.

References

1. Gazdar, G., Mellish, C.: Natural Language Processing in PROLOG. Addison Wesley (1989)
2. Kamp, H., Reyle, U.: From Discourse to Logic. Kluwer, Dordrecht (1993)
3. Kay, M.: The MIND system. In: Rustin, R. (ed.) Natural Language Processing, pp. 155–188. Algorithmics Press, New York (1973)
4. Kay, M.: Algorithm schemata and data structures in syntactic processing. Technical Report CSL-80-12, Xerox PARC, Palo Alto, CA (October 1980)
5. Kowalski, R., Sergot, M.: Logic-Based Calculus of Events. New Generation Computing 4, 67–95 (1986)
6. Mueller, E.T.: Automating Commonsense Reasoning Using the Event Calculus. Communications of the ACM 52(1), 113–117 (2009)
7. Parsons, T.: Events in the Semantics of English: A Study in Subatomic Semantics. Current Studies in Linguistics. MIT Press (1994)
8. Pereira, F.C.N., Shieber, S.M.: Prolog and Natural-Language Analysis. CSLI Publications (1987)
9. Schwitter, R., Ljungberg, A., Hood, D.: ECOLE – A Look-ahead Editor for a Controlled Language. In: Proceedings of EAMT-CLAW 2003, Controlled Translation, pp. 141–150. Dublin City University, Ireland (2003)
10. Schwitter, R.: Controlled Natural Language for Knowledge Representation. In: Proceedings of COLING 2010, pp. 1113–1121 (2010)
11. Schwitter, R.: Specifying Events and their Effects in Controlled Natural Language. In: Proceedings of PACLING 2011 (forthcoming, 2011)
12. White, C., Schwitter, R.: An Update on PENG Light. In: Pizzato, L., Schwitter, R. (eds.) Proceedings of ALTA 2009, Sydney, Australia, pp. 80–88 (2009)
13. Wielemaker, J.: SWI-Prolog 5.10.4 Reference Manual. Department of Computer Science VU University Amsterdam (2010)