CL-CONLLU
Dependências Universais em Common Lisp

Henrique Muniz
Fundação Getulio Vargas
Escola de Matemática Aplicada (EMAp)
Brazil
hnmuniza@gmail.com

Fabricio Chalub
IBM Research
Brazil
fchalub@br.ibm.com

Alexandre Rademaker
IBM Research,
Fundação Getulio Vargas
Escola de Matemática Aplicada (EMAp)
Brazil
alexrad@br.ibm.com

I. Introdução

O uso de diferentes etiquetas para anotações morfológicas e sintáticas, bem como convenções de anotações distintas, dificulta o desenvolvimento de ferramentas de análise sintática multi-idioma e o estudo de fenômenos linguísticos comuns entre diferentes idiomas [1, p. 1659]. Para resolver esse problema, o projeto *Universal Dependencies* (UD) tem como objetivo a criação de anotações linguísticas consistentes entre diferentes idiomas, utilizando uma análise lexicalista e de dependências. Seu esquema de anotação é baseado no Interset, uma interlíngua para sistemas de rotulação morfossintática [2], nas classes gramaticais (POS) Google Universal Tagset [3], bem como nas Stanford Dependencies [4].

Recentemente, o projeto UD lançou a versão 2.0 dos seus treebanks [5], já utilizados na shared task da 'Conference on Computational Natural Language Learning' (CoNLL 2017).

O avanço do projeto UD demanda a criação de diferentes ferramentas para produção e revisão de corpora com anotações linguísticas. Em particular, como parte do projeto de revisão e expansão do corpus UD_Portuguese,[1] tivemos a necessidade de uma biblioteca para manipulação de arquivos no formato CoNLL-U. Esta biblioteca deveria prover funcionalidades como: ler e gravar arquivos CoNNL-U, permitir transformações em lote ("*batch*") de anotações, diferentes visualizações de árvores sintáticas e validação das anotações. Estas foram as motivações que nos levaram a criar a biblioteca `cl-conllu`, em Common Lisp, descrita neste artigo.

II. O Formato CoNLL-U

Por sua adoção do lexicalismo, em UD as unidades básicas de anotação são *palavras* sintáticas (e não palavras ortográficas ou fonológicas) [1, p. 1660]. Por isso, contrações e clíticos são divididos. Por exemplo, *do* é quebrado em *de* e *o*. Já seguindo um modelo sintático de dependências, é considerado que cada palavra é dependente de alguma outra (à exceção da raiz da frase), dita seu *head* (cabeça), através de uma relação de dependência específica (isto é, um rótulo na dependência).

Para representação das anotações seguindo esses princípios, é usado o formato CoNLL-U. Este formato é uma evolução do formato CoNLL-X, que por sua vez foi uma extensão do formato Malt-TAB de Joakim Nivre [6, pp. 150-151]. Cada arquivo pode conter múltiplas frases, separadas por uma linha em branco. Cada frase é iniciada por uma ou mais linhas contendo metadados, como identificador da frase e texto original. Em seguida, as palavras (tokens), são descritos um em cada linha. Também tokens multi-palavra (tokens ortográficos que foram quebrados em mais de uma palavra) recebem uma linha própria. Cada palavra ou tokens contém 10 campos separados por um único caractere de tabulação. São eles: numeração sequencial (ID); forma original da palavra no texto (FORM); lema, a forma canônica de uma palavra (LEMMA), classe gramatical universal (UPOSTAG), classe gramatical específica da linguagem (XPOSTAG), atributos morfológicos (FEATS), índice da palavra da qual depende (HEAD), a relação universal de dependência (DEPREL), relação de dependências *enhanced* (DEPS) e informações diversas (MISC). Para o token raiz da frase, a relação de dependência é `root` e seu head é zero. Alguns campos, como o campo `upostag`, admitem uma lista de valores, separados por '|'. Na figura 1 é apresentado um exemplo de uma sentença no formato CoNLL-U.

III. A biblioteca cl-conllu

O pacote cl-conllu que está sendo desenvolvido em Common Lisp tem como objetivo facilitar o trabalho com arquivos no formato CoNLL-U. A biblioteca já possui importantes funcionalidades como: leitura e escrita de arquivos no formato em questão, funções para conversões de formatos, funções para avaliação de anotações e comparação de diferentes anotações de uma mesma sentença.

Dentre as conversões de formato, atualmente a biblioteca suporta a conversão de arquivos CoNLL-U para Prolog. A conversão de anotações sintáticas para cláusulas Prolog permite que a biblioteca seja usada em etapas de processamento de linguagem posteriores à anotação sintática nos moldes de [7].

Duas importantes recentes adições à biblioteca são: (1) uma linguagem de regras para facilitar correções em lote; (2) a geração de uma visualização das anotações em árvores sintáticas deitadas. Estas recentes adições serão foco deste artigo.

[1] http://github.com/universaldependencies/UD_Portuguese/.

```
1-2 Pela _ _ _ _ _ _ _ _
1 Por por ADP _ _ 3 case _ _
2 a o DET _ Definite=Def|Gender=Fem|Number=Sing|PronType=Art 3 det _ _
3 lei lei NOUN _ Gender=Fem|Number=Sing 7 obl _ SpaceAfter=No
4 ,,PUNCT _ _ 3 punct _ _
5 os o DET _ Definite=Def|Gender=Masc|Number=Plur|PronType=Art 6 det _ _
6 partidos partido NOUN _ Gender=Masc|Number=Plur 7 nsubj _ _
7 recebem receber VERB _ Mood=Ind|Number=Plur|Person=3|Tense=Pres|VerbForm=Fin 0 root _ _
8 recursos recurso NOUN _ Gender=Masc|Number=Plur 7 obj _ _
9-10 do _ _ _ _ _ _ _ _
9 de de ADP _ _ 11 case _ _
10 o o DET _ Definite=Def|Gender=Masc|Number=Sing|PronType=Art 11 det _ _
11 governo governo NOUN _ Gender=Masc|Number=Sing 8 nmod _ _
12 de de ADP _ _ 13 case _ MWE=de_acordo
13 acordo acordo NOUN _ Gender=Masc|Number=Sing 7 obl _ _
14 com com ADP _ _ 16 case _ _
15 os o DET _ Definite=Def|Gender=Masc|Number=Plur|PronType=Art 16 det _ _
16 resultados resultado NOUN _ Gender=Masc|Number=Plur 13 nmod _ _
17-18 da _ _ _ _ _ _ _ _
17 de de ADP _ _ 20 case _ _
18 a o DET _ Definite=Def|Gender=Fem|Number=Sing|PronType=Art 20 det _ _
19 última último ADJ _ Gender=Fem|Number=Sing|NumType=Ord 20 amod _ _
20 eleição eleição NOUN _ Gender=Fem|Number=Sing 16 nmod _ SpaceAfter=No
21 . . PUNCT _ _ 7 punct _ _
```

Figura 1. A sentença 'Pela lei, os partidos recebem recursos do governo de acordo com os resultados da última eleição.' no formato CoNNL-U.

A. Estruturas de Dados

As estruturas de dados principais da biblioteca são as classes `sentence`, `token` e `mtoken` (multiword token). Uma sentença tem como atributo principal a lista dos seus tokens e multiword tokens. Optamos por manter os tokens e multi-word tokens em listas separadas para facilitar o uso destas estruturas por várias outras funções da biblioteca. No fragmento de código abaixo, os atributos das classes `token` foram omitidos para poupar espaço.

```
(defclass token ()
  (...))

(defclass mtoken ()
  ((start    :initarg :start
             :accessor mtoken-start)
   (end      :initarg :end
             :accessor mtoken-end)
   (form     :initarg :form
             :accessor mtoken-form)
   (misc     :initarg :misc
             :initform "_"
             :accessor mtoken-misc)))

(defclass sentence ()
  ((start    :initarg :start
             :initform 0
             :accessor sentence-start)
   (meta     :initarg :meta
             :initform nil
             :accessor sentence-meta)
   (tokens   :initarg :tokens
             :initform nil
             :accessor sentence-tokens)
   (mtokens  :initarg :mtokens
             :initform nil
             :accessor sentence-mtokens)))
```

As funções `read-conllu` e `write-conllu` são as funções para leitura e escrita de arquivos CoNNL-U, respectivamente. A primeira recebe um caminho completo para um arquivo no disco e retorna uma lista encadeada de objetos da classe `sentence` e a segunda recebe uma lista encadeada de objetos da classe `sentence` e um nome de arquivo e escreve as sentenças no arquivo.

B. Edição de arquivos CoNLL-U

Durante a revisão e melhoria de um corpus, normalmente encontramos casos que requerem edição manual de árvores sintáticas e casos em que várias sentenças podem ser corrigidas em lote, quando um erro sistemático é identificado. Durante o projeto de criação do corpus `UD_Portuguese`, ambos os tipos de edição de árvores ocorreram.

O processador de regras que será descrito na seção III-C foi criado para correção em lote de anotações (sintáticas e morfológicas) no corpus. Em particular, com a nova versão 2.0 do *Universal Dependencies*, foram necessárias mudanças sistemáticas de anotações.

Por outro lado, não apenas alterações em lote são necessárias durante a revisão do corpus. Alterações que envolvem a criação ou remoção de tokens ou mudanças nas ligações de mais de um token da sentença também podem ocorrer e podem afetar a numeração sequencial dos tokens. Para estes casos, implementamos na biblioteca a função `adjust-sentence` que permite a rápida reordenação dos tokens da sentenças e ajustes nas dependências de acordo.

C. Regras

Para permitir transformações em lote em arquivos CoNLL-U, foi implementada a função `apply-rules-from-file`, para aplicar regras de transformação em sentenças. Tal funcionalidade foi inspirada no programa "Corte e Costura" [8]. A função recebe uma lista de regras e as aplica em uma lista de frases gerando uma nova lista de frases modificadas.

A função em questão recebe um nome de arquivo (o corpus, no formato CoNLL-U), um arquivo contendo as regras, o nome do arquivo que será produzido e o nome do arquivo de 'log' para registro das operações efetuadas. As regras possuem um lado esquerdo e um lado direito. O esquerdo contém um padrão

de características que são procuradas em uma sequência de tokens. Já no lado direito se encontram as informações que devem ser modificadas ou adicionadas. A linguagem de regras é descrita na listagem 1.

Listing 1. Linguagem de regras

```
rule        ::= (=> rls rhs)
rls, rhs    ::= pattern+
pattern     ::= (var condition ...) | * | ?
var         ::= ?identifier
condition   ::= (op field expression)
op-lhs      ::= (= field string)
              | (member field string+)
              | (match field regex)
op-rhs      ::= (set field string)
              | (add field string)
expression  ::= string | regex
```

Observando o exemplo 3 podemos notar que cada lado de uma regra é formada por um ou mais padrões (‘patterns’). Tais padrões seguem a seguinte estrutura, começam com uma variável que é seguida por uma lista de condições. As variáveis são identificadores CL começando com o carácter "?"e contendo pelo menos um outro carácter. As condições são formadas por um operador, um campo do token que estamos interessados em testar e uma string que pode ser uma expressão regular, dependendo do operador.

Listing 2. Exemplo de regra

```
(=> ((?a (match lemma "[aA]te"))
     (?b (= lemma "entao")))
    ((?a (set upostag "ADV"))
     (?b (set upostag "ADV"))))
```

Os operadores do lado esquerdo da regras são: ‘=’ busca uma simples igualdade; ‘match’ aplicação de expressões regulares; ‘member’ como alguns campos admitem mais de um valor separados por ‘|’, esse operador busca se a sequência de strings passadas esta presente na lista de valores do campo. Os operadores do lado direito da regras são: ‘set’ que altera o campo para a string passada; e ‘add’ que adiciona ao campo a string passada.

O operador $*$ indica uma quantidade indeterminada de tokens entre dois padrões. O operador ? indica opcionalidade do padrão seguinte à ele.

A função `apply-rules-from-files` recebe um arquivo CoNLL-U, este arquivo é lido e a partir dele é criado uma lista contendo as instâncias da classe `sentence`. Para cada objeto `sentence` são testados todas as regras. Cada regra é testada avançando token à token da sentença, tentando assim o casamento do padrão descrito no lado esquerdo da regra com qualquer posição na lista de tokens da sentença. Apenas depois que uma regra foi exaustivamente tentada na sentença, a função avança para a próxima regra. Se for encontrada uma sequência de tokens que case com os padrões do lado esquerdo de uma regra, o lado direito da regra é aplicado na sentença.

Como já exposto a principal inspiração para esta funcionalidade de aplicação de regras foi o programa corte e costura. As duas ferramentas possuem uma ideia inicial muito similar, a aplicação de regras para permitir transformações em lote. Apesar desta grande intercessão os dois programas são bem distintos tanto no sentido de manipularem corpus em formatos diferentes tanto em suas implementações. O desenvolvimento desta biblioteca em Common Lisp proporcionou um código bem mais conciso e claro. Enquanto em Perl o programa possui cerca de 500 linhas, em nossa implementação foi necessário apenas a metade. Também ganhou-se em modularidade, em fácil manutenção e entendimento.

D. Visualização das árvores sintáticas

Outra função recentemente adiciona na biblioteca é a `tree-sentences`. Esta função produz a visualização das árvores sintáticas no formato apresentado na figura 2. Esta função foi implementada a partir de função similar na biblioteca UDAPI. [2].

A função espera receber uma lista encadeada de objetos da classe `sentence` e gera a visualização das árvores de cada sentença. Na figura 2 está a árvore gerado pelo programa correspondente ao exemplo da sentença contida na figura 1.

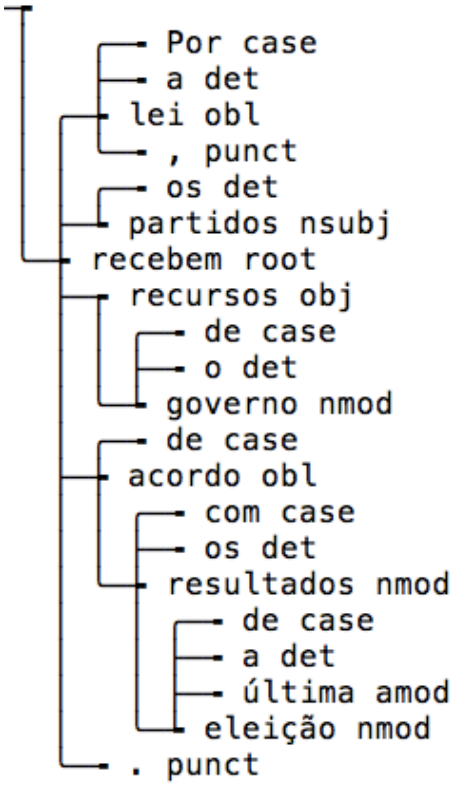

Figura 2. Árvore correspondente a sentença ‘Pela lei, os partidos recebem recursos do governo de acordo com os resultados da última eleição.‘ no formato CoNNL-U.

A visualização dos dados é essencial para verificar as dependências em uma sentença. Muitas ferramentas de visualização de dependências apresentam suas árvores em uma estrutura horizontal, esta função produz a visualização vertical da sentença. Optamos por esta forma de estruturação pois

[2]https://github.com/udapi/udapi-python

linguistas de nosso time a consideram especialmente útil para sentenças longas.

A partir de cada objeto da classe `sentence` o programa cria duas listas, ambas com o mesmo número de elementos, a quantidade de tokens da sentença. Em uma lista são armazenadas as strings de caracteres de cada linha da árvore. Em outra lista, o comprimento de cada linha, em caracteres. A árvore é produzida de forma interativa, coluna à coluna, em várias passadas pelas listas de tokens. Ao final das interações, a lista de strings que contém cada linha da árvore é simplesmente enviada para o STDOUT.

E. Filtros

Outra funcionalidade comumente associada a treebanks é a de filtragem de padrões (veja por exemplo [9]), como por exemplo, "todos os tokens que são verbos", ou "todos os tokens relacionados pela dependência `advcl`". Para tanto foi criada uma simples linguagem de filtro que funciona como uma abstração em cima operações sobre conjuntos cujos elementos são tokens. Temos os seguintes operadores: (a) filtragem por predicado (e.g., a propriedade `upostag` é igual a `VERB`); (b) filtragem por conexão de dependência entre dois ou mais tokens (e.g., tokens que estejam relacionados via dependência `nsubj`); (c) operações sobre estes conjuntos como `and` (intersecção) e `or` (união).

O fragmento abaixo exemplifica um tipo de filtro que combina filtros por predicados, dependências, e operações sobre conjuntos.

Listing 3. Exemplo de filtro

```
(execute '(nsubj (advcl (and (upostag ~ "VERB")
                             (lemma ~ "correr"))
                         (upostag ~ "VERB"))
                 (upostag ~ "PROP"))
         tokens)
```

É importante ressaltar que a implementação desta funcionalidade ficou muito facilitada pelo uso de Common Lisp, dado que a linguagem de filtragem em si é baseada em *s-expressions* e pela sua composicionalidade (todas as operações são essencialmente operações sobre conjuntos), o que permite uma implementação trivial de seu parsing e consequente expansão e execução do código derivado.

F. Testes

Para garantir o bom funcionamento das funcionalidades de uma biblioteca é muito importante a presença de testes. Por esse motivo foram adicionadas dois novos pacotes a biblioteca `cl-conllu`, `conllu-visualize-test` e `conllu-rule-test`, com a principal função cada pacote sendo `conllu-visualize-test:test-all` e `conllu-rulr-test:test-all`. Os pacotes ainda estão em uma fase inicial então é claro que ainda há muito para ser desenvolvido e agregadado a elas.

No escopo da `conllu-rule-test` a função `test-all` verifica principalmente se as alterações, de um conjunto de regras e um arquivo predefinidos, estão ocorrendo da forma correta e quando esperado. Já no campo do `conllu-visualize-test` a função `test-all` é responsável por atestar a criação de árvores de dependência bem estruturadas e corretas de sentenças predeterminadas.

IV. Trabalhos Relacionados

No âmbito das regras um trabalho que se relaciona fortemente com a biblioteca é o Corte e Costura [8]. Ele é um recurso criado pela Linguateca[3] para modificar, eliminar e acrescentar novos atributos a corpos "no formato AC/DC", previamente anotado pelo PALAVRAS.

Especificamente em relação ao formato CoNLL-U, vários grupos trabalhando no projeto UD desenvolveram ferramentas com diferentes propósitos como: visualização, transformação, validação etc. Várias destas ferramentas estão listadas na página http://universaldependencies.org/tools.html. Mais diretamente relacionada com os objetivos da nossa biblioteca estão: 'DKPro Core CoNLL-U reader/writer' e o udapy. O artifo Udapi: Universal API for Universal Dependencies[10] explica melhor os objetivos e aplicações dessa última biblioteca. E em uma rápida pesquisa é possível encontrar algumas outras ferramentas, por exemplo, Conllu-clj [4] e Conllu [5], mas é fácil também notar a simplicidade delas.

V. Conclusão e trabalhos futuros

Pretendemos continuar acrescentando funcionalidades na biblioteca cl-conllu como, por exemplo: (1) melhor suporte para validação de sentenças; (2) expansão da linguagem de regras com suporte a variáveis sobre expressões e não apenas variáveis para referência a tokens; e (3) suporte para edição de sentenças de forma interativa e outras formas de visualização das árvores sintáticas. Finalmente, pretendemos adicionar ainda mais casos de testes para aumentar a robustez da biblitoeca.

Referências

[1] J. Nivre, M. de Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. T. McDonald, S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, and D. Zeman, "Universal dependencies v1: A multilingual treebank collection," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, May 23-28, 2016.*, N. Calzolari, K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, and S. Piperidis, Eds. European Language Resources Association (ELRA), 2016. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2016/summaries/348.html

[2] D. Zeman, "Reusable tagset conversion using tagset drivers," in *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2008, 26 May - 1 June 2008, Marrakech, Morocco*. European Language Resources Association, 2008. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2008/summaries/66.html

[3] S. Petrov, D. Das, and R. T. McDonald, "A universal part-of-speech tagset," in *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*, N. Calzolari, K. Choukri, T. Declerck, M. U. Dogan, B. Maegaard, J. Mariani, J. Odijk, and S. Piperidis, Eds. European Language Resources Association (ELRA), 2012, pp. 2089–2096. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2012/summaries/274.html

[3] http://www.linguateca.pt.

[4] https://github.com/ysmiraak/conllu-clj

[5] https://github.com/EmilStenstrom/conllu

[4] M. de Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning, "Universal stanford dependencies: A cross-linguistic typology," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014, Reykjavik, Iceland, May 26-31, 2014.*, N. Calzolari, K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, and S. Piperidis, Eds. European Language Resources Association (ELRA), 2014, pp. 4585–4592. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2014/summaries/1062.html

[5] J. Nivre, Ž. Agić, L. Ahrenberg, M. J. Aranzabe, M. Asahara, A. Atutxa, M. Ballesteros, J. Bauer, K. Bengoetxea, R. A. Bhat, E. Bick, C. Bosco, G. Bouma, S. Bowman, M. Candito, G. Cebiroğlu Eryiğit, G. G. A. Celano, F. Chalub, J. Choi, Ç. Çöltekin, M. Connor, E. Davidson, M.-C. de Marneffe, V. de Paiva, A. Diaz de Ilarraza, K. Dobrovoljc, T. Dozat, K. Droganova, P. Dwivedi, M. Eli, T. Erjavec, R. Farkas, J. Foster, C. Freitas, K. Gajdošová, D. Galbraith, M. Garcia, F. Ginter, I. Goenaga, K. Gojenola, M. Gökırmak, Y. Goldberg, X. Gómez Guinovart, B. Gonzáles Saavedra, M. Grioni, N. Grūzītis, B. Guillaume, N. Habash, J. Hajič, L. Hà Mỹ, D. Haug, B. Hladká, P. Hohle, R. Ion, E. Irimia, A. Johannsen, F. Jørgensen, H. Kaşıkara, H. Kanayama, J. Kanerva, N. Kotsyba, S. Krek, V. Laippala, P. Lê Hông, A. Lenci, N. Ljubešić, O. Lyashevskaya, T. Lynn, A. Makazhanov, C. Manning, C. Mărănduc, D. Mareček, H. Martínez Alonso, A. Martins, J. Mašek, Y. Matsumoto, R. McDonald, A. Missilä, V. Mititelu, Y. Miyao, S. Montemagni, A. More, S. Mori, B. Moskalevskyi, K. Muischnek, N. Mustafina, K. Müürisep, L. Nguyễn Thị, H. Nguyễn Thị Minh, V. Nikolaev, H. Nurmi, S. Ojala, P. Osenova, L. Øvrelid, E. Pascual, M. Passarotti, C.-A. Perez, G. Perrier, S. Petrov, J. Piitulainen, B. Plank, M. Popel, L. Pretkalniņa, P. Prokopidis, T. Puolakainen, S. Pyysalo, A. Rademaker, L. Ramasamy, L. Real, L. Rituma, R. Rosa, S. Saleh, M. Sanguinetti, B. Saulīte, S. Schuster, D. Seddah, W. Seeker, M. Seraji, L. Shakurova, M. Shen, D. Sichinava, N. Silveira, M. Simi, R. Simionescu, K. Simkó, M. Šimková, K. Simov, A. Smith, A. Suhr, U. Sulubacak, Z. Szántó, D. Taji, T. Tanaka, R. Tsarfaty, F. Tyers, S. Uematsu, L. Uria, G. van Noord, V. Varga, V. Vincze, J. N. Washington, Z. Žabokrtský, A. Zeldes, D. Zeman, and H. Zhu, "Universal dependencies 2.0," 2017, LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University. [Online]. Available: http://hdl.handle.net/11234/1-1983

[6] S. Buchholz and E. Marsi, "Conll-x shared task on multilingual dependency parsing," in *Proceedings of the Tenth Conference on Computational Natural Language Learning*, ser. CoNLL-X '06. Stroudsburg, PA, USA: Association for Computational Linguistics, 2006, pp. 149–164. [Online]. Available: http://dl.acm.org/citation.cfm?id=1596276.1596305

[7] A. Lally and P. Fodor, "Natural language processing with prolog in the IBM Watson System," *The Association for Logic Programming (ALP) Newsletter*.

[8] C. Mota and D. Santos, "Corte e costura no ac/dc: auxiliando a melhoria da anotação nos corpos," *Setembro de*, 2009.

[9] J. Luotolahti, J. Kanerva, S. Pyysalo, and F. Ginter, "Sets: Scalable and efficient tree search in dependency graphs," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. Association for Computational Linguistics, 2015, pp. 51–55. [Online]. Available: https://aclweb.org/anthology/N/N15/N15-3011.pdf

[10] M. Popel, Z. Žabokrtskỳ, and M. Vojtek, "Udapi: Universal api for universal dependencies," in *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies, 22 May, Gothenburg Sweden*, no. 135. Linköping University Electronic Press, 2017, pp. 96–101.