Tregex and Tsurgeon: tools for querying and manipulating tree data structures

Roger Levy* **and Galen Andrew†**

*School of Informatics
University of Edinburgh
rlevy@inf.ed.ac.uk
†Microsoft Research
galena@microsoft.com

Abstract

With syntactically annotated corpora becoming increasingly available for a variety of languages and grammatical frameworks, tree query tools have proven invaluable to linguists and computer scientists for both data exploration and corpus-based research. We provide a combined engine for tree query (Tregex) and manipulation (Tsurgeon) that can operate on arbitrary tree data structures with no need for preprocessing. Tregex remedies several expressive and implementational limitations of existing query tools, while Tsurgeon is to our knowledge the most expressive tree manipulation utility available.

1. Introduction

Syntactically annotated corpora have become available for a wide variety of languages and grammatical frameworks, and currently play a major role in much syntactic research in theoretical linguistics, computational linguistics, and psycholinguistics. Researchers in all of these disciplines are often interested in examples or statistics involving detailed co-occurrence patterns, which has created a need for expressive *tree query* tools. A number of such tools are already available (König et al. 2003; Cassidy and Harrington 2001; McKelvie et al. 2001; Bird et al. 2005; see Lai and Bird 2004 for a critical review) and in varying degrees of use among the research community. In addition to tree query, the need for *tree manipulation* frequently arises, in systematic correction of annotation errors, conversion between annotation conventions, and adaptation of existing syntactic resources for new purposes. In this paper, we show how a highly expressive tree manipulation language can be built on top of a tree query language, and present an open-source implementation allowing combined tree query and manipulation.

2. Tree query with Tregex

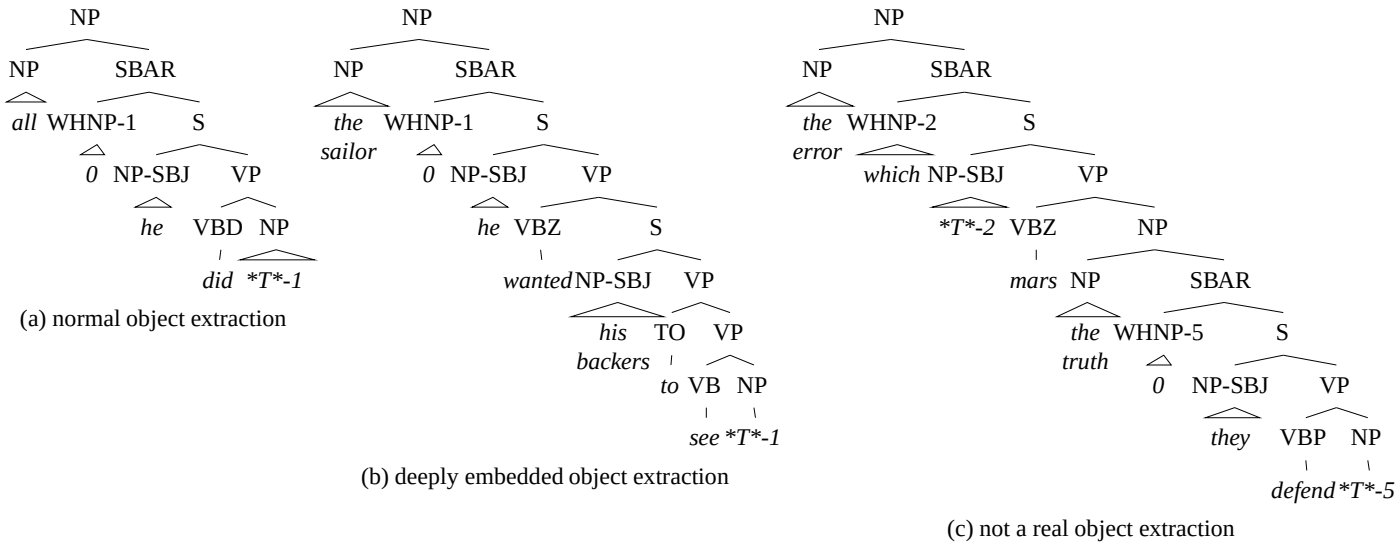
We model the syntax, semantics, and relational inventory of our tree query language after the `tgrep2` query language (Rohde, 2005), which is already in widespread use among the linguistics community. `tgrep2` makes available a wide range of relational operators derived from the primitive relations of *immediate dominance* and *precedence*. `tgrep2` also allows boolean conjunction, disjunction (expressed with `|`), and negation (expressed with `!`) over relational statements; regular-expression matching of node labels; and tree node identity constraints enforced by names, or handles, assigned to nodes in a pattern. Table 1 lists a sample of the Tregex node relations. Tregex replicates the functionality of `tgrep2`, and extends its expressivity in three key respects:

1. *Constrained dominance and precedence*: two nodes can be required to be in a relation of dominance or precedence through an unbroken change of nodes all of whose labels match some regular expression. For example, whereas the query `S < (VP < NP)` matches any top-to-bottom tree node path of S-VP-NP, the query `S <+(VP) NP` matches any top-to-bottom path of S-VP*-NP, where VP* denotes any number of VPs. Constrained dominance is particularly useful in querying syntactic trees with nested adjunction or coordinated structures; together with constrained precedence, it fills much of the need for closures articulated by Lai and Bird (2004).

2. *Headship* is added as a primitive relation. The criteria determining the head of a given tree node can be specified by the user; headship modules are included for the Penn Treebanks of English, Chinese, and Arabic, and for the NEGRA and TIGER treebanks of German. Combined with other relations and boolean operators, headship as a primitive enables a variety of linguistically interesting queries such as: "maximal projection of node X", "head terminal of node Y", and "governor of word Z". For example, the pattern

   ```
   (__ !< __) >># (VP < (PP <<# on))
   ```

   matches at leaf nodes (i.e., words) projecting to a VP that governs a PP headed by the word "on"—



(a) normal object extraction

(b) deeply embedded object extraction

(c) not a real object extraction

```
NP < (SBAR < /^WH.*-([0-9])+/#1%j << (VP < (NP <: /^\*T\*-([0-9])+/#1%j)))
```

Figure 1: Variable groups in Tregex.[1]The pattern does not match the root node of 1c, because the indices of the top SBAR (4) and the bottom trace (5) cannot both be identified with the same variable `%j`.

more informally, verbs taking an *on*-PP as an adjunct or complement.

3. *Variable groups*: when a regular expression R is used to match a node label, any group in R can be assigned a variable name. If a variable name appears more than once in a given match pattern, the pattern will only match if all the groups assigned that name capture the same substring. This facility is particularly useful for imposing nonlocal coindexation relationships among nodes in a pattern, such as for extraction or pronominal anaphora. For example, the match pattern in Figure 1 matches only relative clauses where the relative pronoun is extracted from the object of some embedded verb, because the index of each is assigned to the same variable `%j`. To our knowledge, no other tree query engine provides a comparable facility.

`A << B`	A dominates B
`A < B`	A immediately dominates B
`A <<: B`	B is a unary descendent of A
`A $++ B`	A is a left sister of B
`A $+ B`	A is the immediate left sister of B
`A <+(C) B`	A dominates B through a chain of nodes each of which matches description C
`A . B`	A immediately precedes B
`A .+(C) B`	A precedes B through a chain of nodes each of which matches description C
`A <# B`	B is the head daughter of A
`A <<# B`	B heads A (through transitive closure of `<#`)

Table 1: Some of the node-node relations in Tregex[2]

3. Tree manipulation with Tsurgeon

Despite the applicability both for machine-learning approaches to NLP and for data management, tools for tree-manipulation operations have not been widely developed.[3] Because Tregex pattern nodes can be as-

[1]The `/⟨expr⟩/` syntax for a node label means that ⟨`expr`⟩ is to be interpreted as a regular expression. The node label suffix #n%⟨`string`⟩ means to assign group n of the regular expression to the variable %⟨`string`⟩. Note also that boolean relational operators are left-associative, so the pattern asserts that both the WH-phrase and the VP are in a domination relationship with the SBAR.

[2]Following *tgrep2* convention, every relation R containing $<$ has a corresponding "passivized" relation R' where every $<$ is replaced with $>$. For example, `A > B` means that A is immediately dominated by B.

[3]To our knowledge, the most expressive previously existing tree-manipulation tools are the *TrEd* tree viewer/editor (Hajic et al., 2001) and *tsed* (Blaheta, 2003). TrEd was designed primarily for interactive tree manipulation through a graphical interface, however, and the inventory of tree-based pattern matching relations for use in batch tree-processing is not as rich as those in Tregex. (We are grateful to an anonymous reviewer to bringing TrEd to our attention.) The *tsed* tool is unfortunately is no longer available. The tree-manipulation facilities of the *Treep* tool

delete ⟨name$_1$⟩ ... ⟨name$_n$⟩	prune ⟨name$_1$⟩ ... ⟨name$_n$⟩
relabel ⟨name⟩ ⟨new_label⟩	insert ⟨name⟩ ⟨position⟩
coindex ⟨name$_1$⟩... ⟨name$_n$⟩	insert ⟨tree⟩ ⟨position⟩
move ⟨name⟩ ⟨position⟩	replace ⟨name$_1$⟩ ⟨name$_2$⟩
excise ⟨name$_{top}$⟩ ⟨name$_{bottom}$⟩	adjoin ⟨tree⟩ ⟨target_name⟩

Table 2: Tsurgeon operations and their syntax

signed handles, however, implementing a tree manipulation engine turned out to be a relatively easy task.

A Tsurgeon pattern is defined to consist of a single Tregex match pattern P, combined with any number of Tsurgeon operations that are to be executed when when P matches. Nodes in a Tregex match pattern can be *names*, which can then be referred to as manipulation targets in Tsurgeon operations.[4] Table 2 lists the manipulation operations available. The **delete**, **relabel**, **insert**, **move**, and **replace** operations are intuitive; the **prune** operation differs from **delete** in that the former recursively deletes any nonterminal nodes that are left with no children—preventing nonterminal nodes from becoming terminals—whereas the latter does not. The **excise** and **adjoin** operations are closely interrelated. **Excise** "flattens" a tree fragment by splicing out a vertical chain of nodes, and re-attaching all children of spliced-out nodes into the parent of the highest removed node. **Adjoin** reverses this process via the formal Tree-Adjoining Grammar operation of adjunction (Joshi, 1985), splicing a tree fragment in at a target site. Figure 2 illustrates one of the uses of **excise** and **adjoin**: converting between nested and flat adjunction structures. Finally, **coindex** allows multiple nodes captured in a single match pattern to be assigned a common index.

One difficulty in designing semantics for a tree manipulation formalism is encountered when a single query pattern matches a tree in more than one way. The simplest alternative, to allow an operation to apply only once per tree, is unsatisfactory: if a single tree includes two coordinated clauses, for example, the user will almost always want all transformation operations to apply in each clause. It would seem ideal to identify in parallel all possible matches in the tree, and then to apply the corresponding manipulation at each match. Determinism cannot be maintained under this approach, however, without severely restricting the scope of both query and manipulation operations. For

(Chiang and Bikel, 2002) are limited to node relabeling.

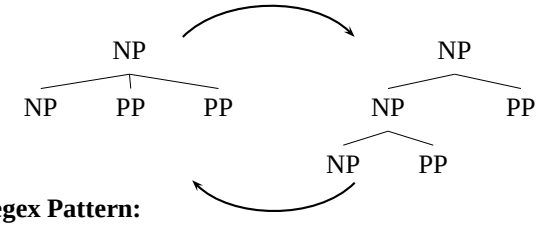
[4]The Tregex syntax for naming a node is to append =⟨**name**⟩ to the description of the node label, as seen for the names `np` and `pp` in Figure 2. The string ⟨**name**⟩ can then be referred to later in the pattern, or in asociated Tsurgeon operations.

Tregex Pattern:

```
NP=np < (NP $+ (PP $+ PP=pp2))
```

Tsurgeon Operations:

```
adjoin (NP=new_np NP@) np
move pp2 >- new_np
```



Tregex Pattern:

```
NP < (NP=np < (NP $+ PP) $+ PP)
```

Tsurgeon operation:

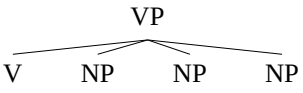
```
excise np np
```

Figure 2: Tsurgeon patterns for converting between flat and nested adjunction structures using **excise** and **adjoin**[5]

example, if a Tsurgeon pattern of the following form

```
V $++ (NP=left $+ NP=right)
relabel  left   NP-LEFT
relabel  right  NP-RIGHT
```

were applied to a tree of the form



then the final label for the middle NP would be indeterminate. Instead, we have chosen *cyclical* application for Tsurgeon rules: when an initial match is found, all manipulation operations apply immediately, and the resulting tree is then rescanned from the beginning to see whether the Tsurgeon operation can be applied again. When a single Tsurgeon pattern applies more than once to a given tree, the order of application is therefore determined by the order of search specified by Tregex. It is incumbent upon the user to write Tsurgeon patterns in a manner that prevents unintended interactions; we have found this to be the most flexible approach. As a simple example, suppose we wanted to add an explicit zero copula node as immediate left sister of non-verbal predicates in clauses with no overt copula. One pattern for this might be:

```
S < /-PRD/=prd !< /^V/ !< COPULA
  insert (COPULA 0) $+ prd
```

Without the `!< COPULA` portion of the Tregex pattern, execution of the pattern would never terminate after an initial successful match.

[5]In Figure 2, the symbol `@` in the `adjoin` operation marks the *foot* node of the auxiliary tree (the node to which the daughters of the adjunction target are attached). Note

4. Implementation and Applications

Tregex and Tsurgeon are implemented in Java and therefore enjoy advantages of platform independence and internal Unicode character representation, both desirable in language technology. Unlike other tree query tools including `tgrep` and `tgrep2`, they require no preprocessing of the input prior to search. They can be invoked from the command line or incorporated into Java programs through a concise API. The Tregex API is modeled after the `java.util.regex` library, providing a high degree of control to the user: multiple tree matcher objects can be spawned from a single compiled match-pattern object, and iteration over successful matches can be controlled using ordinary Java loop constructs. Both Tregex and Tsurgeon are already in use in a number of research projects. These include:

- Use of the API to construct feature templates for semantic role identification (Toutanova et al., 2005) and to transform questions into statements for textual inference (Raina et al., 2005);
- Extraction of detailed information from relative clauses for a psycholinguistic study of syntactic production (Jaeger et al., 2005);
- Conversion of treebanks and Tree-Adjoining Grammars for use in the parsing of Arabic dialects, and standardization and transformation of tree annotation conventions, at the 2005 Johns Hopkins Center for Language and Speech Processing Summer Workshop.

Tregex and Tsurgeon are available, under the GPL, at *http://nlp.stanford.edu/software/tregex.shtml*.

5. Conclusion

In Tregex and Tsurgeon we have presented an expressive and flexible system for tree query and manipulation. This system fills several expressive gaps in existing tree query languages, and provides a specialized high-level interface for specifying and carrying out arbitrary tree manipulations. The system is already in use as a component in several research projects. Future work will involve further refinement of both the query and manipulation languages, and potentially extensions to *multiple-tree* data structures, such as parse trees over parallel text, or overlapping phonological/syntactic parses of a single linguistic string.

6. Acknowledgements

Development of Tregex and Tsurgeon was supported by a grant from ARDA's Advanced Question Answering for Intelligence Program, and by the JHU Center for Language and Speech Processing Summer Workshop. Tregex was developed at Stanford University, and we are grateful to members of the Stanford NLP group for feedback during development.

References

Bird, S., Chen, Y., Lee, S. D. H., and Zheng, Y. (2005). Extending XPath to support linguistic queries. In *Proceedings of Programming Language Technologies for XML*, pages 35–46.

Blaheta, D. (2003). *Function Tagging*. PhD thesis, Brown University.

Cassidy, S. and Harrington, J. (2001). Multi-level annotation in the Emu speech database management system. *Speech Communication*, 33(1–2):61–77.

Chiang, D. and Bikel, D. (2002). Recovering latent information in treebanks. In *Proceedings of COLING*.

Hajic, J., Vidová-Hladká, B., and Pajas, P. (2001). The Prague Dependency Treebank: Annotation structure and support. In *Proceedings of the IRCS Workshop on Linguistic Databases*, pages 105–114. University of Pennsylvania.

Jaeger, F. T., Levy, R., Wasow, T., and Orr, D. M. (2005). The absence of "that" is predictable if a relative clause is predictable. Presented at AMLaP 2005.

Joshi, A. K. (1985). How much context-sensitivity is necessary for characterizing structural descriptions – Tree Adjoining Grammars. In Dowty, D., Karttunen, L., and Zwicky, A., editors, *Natural Language Processing – Theoretical, Computational, and Psychological Perspectives*. Cambridge.

König, E., Lezius, W., and Voormann, H. (2003). *TIGERSearch 2.1 User's Manual*. IMS, University of Stuttgart.

Lai, C. and Bird, S. (2004). Querying and updating treebanks: A critical survey and requirements analysis. In *Proceedings of the Australasian Language Technology Workshop*, pages 139–146. Macquarie University, Sydney.

McKelvie, D., Isard, A., Moller, M. B., Gross, M., and Klein, M. (2001). The MATE workbench—an annotation tool for XML coded speech corpora. *Speech Communication*, 33(1–2):97–112.

Raina, R., Haghighi, A., Cox, C., Finkel, J., Michels, J., Toutanova, K., MacCartney, B., de Marneffe, M.-C., Manning, C. D., and Ng, A. Y. (2005). Robust textual inference using diverse knowledge sources. In *First PASCAL Challenges Workshop*.

Rohde, D. L. T. (2005). *TGrep2 User Manual*, version 1.15 edition.

Toutanova, K., Haghighi, A., and Manning, C. D. (2005). Joint learning improves semantic role labeling. In *Proceedings of ACL*.

also that nodes of an auxiliary tree can be named (as in `=new_np`) and referred to in later operations. The `>-new_np` argument of the `move` operation specifies that the `pp2` node should move to the last daughter position of `new_np`. Finally, in the `excise` operation, the target node chain is of length one, so the top and the bottom node of the chain are both the `np` node in the Tregex pattern.