Automated Generation of Test Suites for Error Analysis of Concept Recognition Systems

Tudor Groza[1,2]
[1]School of ITEE
The University of Queensland
[2]Garvan Institute of Medical Research
Australia
`tudor.groza@uq.edu.au`

Karin Verspoor
Dept of Computing and Information Systems
The University of Melbourne
Australia
`karin.verspoor@unimelb.edu.au`

Abstract

We present an architecture and implementation of a system that builds structured test suites for concept recognition systems. The system applies provided test case definitions to a target concept vocabulary, to generate test cases organised according to those definitions. Test case definitions capture particular characteristics, or produce regular transformations, of concept terms. The test suites produced by the system enable detailed, systematic, error analysis of the performance of concept recognition systems.

1 Introduction

In this paper, we introduce a framework to automate development of test suites for ontology concept recognition systems. The objective of the work is to enable the assessment of system competence and performance, by organising test cases into groups based on carefully defined characteristics. While failure analysis is often done in terms of such characteristics, it is generally done in an unsystematic manner. By providing a framework for automatically building test suites, we aim to enable more systematic investigation of errors.

We focus in this initial work on ontology concept recognition systems, that is, systems that aim to detect concepts defined in an ontology in natural language text. Prior work has demonstrated substantial differences in the performance of such systems, due to linguistic variability in the expression of ontology concepts (Funk et al., 2014). The use of *structured test suites* has been shown to enable identification of performance errors of such systems (Cohen et al., 2010), as well as being useful for finding bugs (Cohen et al., 2008). Structured test suites enable systematic evaluation, exhaustivity, inclusion of negative data, and control over data (Oepen et al., 1998). They can focus on specific linguistic phenomena, that can be presented in isolation and in controlled combinations.

Evaluation of the performance of NLP methods is typically done with respect to annotated training data. Methods are assessed based on their ability to reproduce human performance on a task, as measured in terms of the standard metrics of *precision*, *recall*, and *F-score*. Such metrics provide a quantitative basis for comparing performance of different methods. However, they are by their nature aggregative, considering all annotations in the corpus as equal for evaluation purposes. Furthermore, such metrics do not provide insight into the nature of errors made by the methods. As stated by (Cohen et al., 2004), testing a system on an annotated corpus “tells you how often the system failed, but not what it failed at; it tells you how often the system succeeds, but not where its strengths are.” Yet investigation of the strengths and failures of a system can reveal information meaningful for improving system performance, and is a critical component of error analysis. This approach is commonly applied in software testing. The methodology of *equivalence partitioning* (Myers, 1979) explicitly involves partitioning the input into equivalence classes that are representative of a range of possible test cases.

This paper introduces a framework for supporting automatic generation of test suites, with the goal of supporting more rigorous testing and evaluation of ontology concept recognition system. Concept recognition (CR) aims to link ontological concepts, defined in a specified ontology, to free text spans denoting entities of interest. CR is a natural evolution of the more traditional Named Entity Recognition (NER) task, which focuses only on detecting the mentions of entities of interest within unstructured textual sources, without aligning them to ontological terms. Well-studies CR tasks include, in particular, gene and protein normalisation (Lu et al., 2011), which involves entity linking of gene/protein mentions to biological data bases. In the context of large structured vocabularies, the CR task involves mapping of terms to specific vocabulary identifiers. The set of NER categories in CR is therefore effectively as large as the number of primary terms in the vocabulary.

Our test suite generation framework consists of 3 main components:

1. **An Input Wrapper** that loads terminology from an ontology, controlled vocabulary, or other target resource.
2. **Test Case Definitions** that specify characteristics of target terms to be incorporated as cases into the test suite.
3. **A Test Suite Factory** that produces a structured test suite for the input ontology, from the test case definitions.

Together, these components support automatic creation of a structured test suite, that can be used to systematically assess the performance of a concept recognition system. Each test case defines an equivalence class of terms, along a defined dimension. The framework is available at `https://github.com/tudorgroza/cr_testsuites`. We welcome contributions of new test case definitions and input wrappers.

2 Background

2.1 Concept Recognition Systems

The class of NLP system that we are primarily concerned with testing is the *concept recognition system*. These are systems that aim to detect mentions of terms corresponding to concepts from an ontology or controlled vocabulary in natural language text. These could be named entity recognition systems, where the set of named entities is defined by a target resource (e.g., the set of all registered US corporations, or the set of all genes in GenBank[1]).

In the biomedical domain, ontology concept recognition systems have been a recent focus of development, due to a proliferation of biomedical ontologies[2]. A number of systems have recently been developed, or deployed, to address this domain, including the US National Library of Medicine's MetaMap tool (Aronson and Lang, 2010), the NCBO Annotator (Jonquet et al., 2009), ConceptMapper (Tanenblatt et al., 2010; Funk et al., 2014), WhatIzIt (Rebholz-Schuhmann and others, 2008) and Neji (Campos et al., 2013).

These systems could equally make use of machine learning, or rule-based methods. Rule-based systems have the advantage of being flexibly redeployable to new ontologies or vocabularies that might be defined, as they do not require training data. Furthermore, in a normalisation context in which specific vocabulary items must be detected and normalised to an identifier (e.g., not just recognising a US corporation mention, but mapping that mention to a specific register ID), the number of target classes is effectively the number of concept classes. This can be prohibitive for an effective machine learning technique.

2.2 Use of Test Suites in NLP

A structured test suite consists of a set of carefully selected test cases that are designed to test specific functionality or the performance of an algorithm on a controlled input. In the development of software systems, test suites are used for acceptance and regression testing, to ensure that the software satisfies a given set of requirements and that a change to the code does not inadvertently break a given required functionality. In NLP, a test suite can be used to automatically verify the performance of an algorithm on specific linguistic phenomena. Test suites rely on *controlled variation* of the linguistic inputs, and allow analysis to be performed along particular dimensions of variation. This is in stark contrast to standard annotated corpora that reflect natural linguistic variation and natural distribution of entities, which is dependent on the collection strategy for the corpus. In error analysis of a task using an annotated corpus, the categorisation of annotations and errors into coherent groups is typically done in *post-hoc* analysis. It has been demonstrated that this can be both challenging to implement and insightful with regards to the generalisability of algorithms (Stoyanov et al., 2009; Kummerfeld et al., 2012; Kummerfeld and Klein, 2013). Using a test suite, it is done *a priori* through the test suite construction.

The use of test suites has long benefited development of NLP systems for syntactic analysis

[1]`http://www.ncbi.nlm.nih.gov/genbank`

[2]There are 384 ontologies, containing close to 6 million concept classes in total, listed in the US National Center for Biomedical Ontology's BioPortal, `http://bioportal.bioontology.org` (Whetzel et al., 2011).

(Oepen et al., 1998; Oepen, 1999), as well as from systematic organisation of grammatical phenomena along typological dimensions. The LinGO Grammar Matrix (Bender et al., 2010; Bender et al., 2002) captures linguistic variation along a number of defined dimensions, and enables creation of an initial grammar based on a library of syntactic structures. One of the key elements of the Matrix is support for regression testing via automated tests, such that any change to a grammar or the linguistic phenomena captured in the system can be automatically assessed for impact to the performance on previously existing phenomena. Such test suites are used for validation and exploration of changes to a grammar, during grammar engineering (Bender et al., 2008).

However, the approach has had limited adoption beyond analysis of deep parsing systems. A methodology and data resources was introduced to support feature-based evaluation of molecular biology entity recognition systems (Cohen et al., 2004). The data resources included examples of entity names across four categories of variation, orthographic (length, token "shape", presence of Greek letters, etc.), morphosyntactic (prefixes, suffixes, presence function words, etc.), source (e.g., a dictionary or a database), and lexical (e.g., status with respect to a language model or vocabulary). That work demonstrated that a test suite can be a good predictor of performance on named entities with particular typographic characteristics.

The approach was later applied to ontology concept recognition systems (Cohen et al., 2010). That work identifies a core set of terminological features that was common to the ontology concept recognition context and the named entity recognition context: *a*) Length *b*) Numerals *c*) Punctuation *d*) Function/stopwords *e*) Source or authority *f*) Canonical form in source (e.g., ontology or database); and *g*) Syntactic context.

In each case of this prior work, the test suites have been generated manually and contain a limited number of examples.

Other frameworks supporting evaluation of NLP systems, including ontology concept recognition systems, have been developed. U-compare (Kano et al., 2009) provides a sophisticated evaluation environment, specifically targeting evaluation and comparison of workflows for document annotation, including syntactic annotation, NER, and information extraction of events. The framework allows multiple systems to be compared over the same data, producing quantitative results in terms of precision, recall, and F-score, as well as supports visual inspection of annotations and annotation differences (Kano et al., 2011). However, there is no direct support for quantitative error analysis.

3 A Framework for Ontology Test Suite generation

We propose a framework to automate development of test suites for ontology concept recognition systems. Given an ontology definition file, and a set of specifications of the typological dimensions of interest, the framework generates a test suite. This test suite organises the ontology terms and any synonyms defined in the ontology according to the typological dimensions of interest.

Figure 1 depicts the high-level architecture of the framework. This comprises three major components: (i) the Input Wrapper – handling the processing of a given ontology or term resource, according to a specification file; (ii) the Test Case – defining specific characteristics along a dimension of interest; and (iii) the Test Case Factory – generating a test suite from a given input according to a set of defined test cases. In the following sections we describe each of these components.

3.1 Input Wrapper

The Input Wrapper processes a given terminological input resource, according to a provided *specification*, and provides an iterator over the *entity profiles* defined by the dataset. Generically, the InputWrapper does not rely on any assumptions about the input resource, but rather delegates these assumptions to the underlying implementation. This means that the input resource could be an explicitly structured ontology or dictionary, as well as an annotated (gold standard) corpus, for which the target vocabulary for a particular set of entities or concepts can be inferred from the annotations.

An Entity Profile captures the terminological representation of an individual concept or named entity, and is expected to include the following properties: (i) a unique identifier – i.e., the URI of a concept in the case of an ontology, or a plain identifier in the case of an annotated corpus; (ii) the list of labels – i.e., preferred and/or alternative labels for ontological concepts, or a canonical

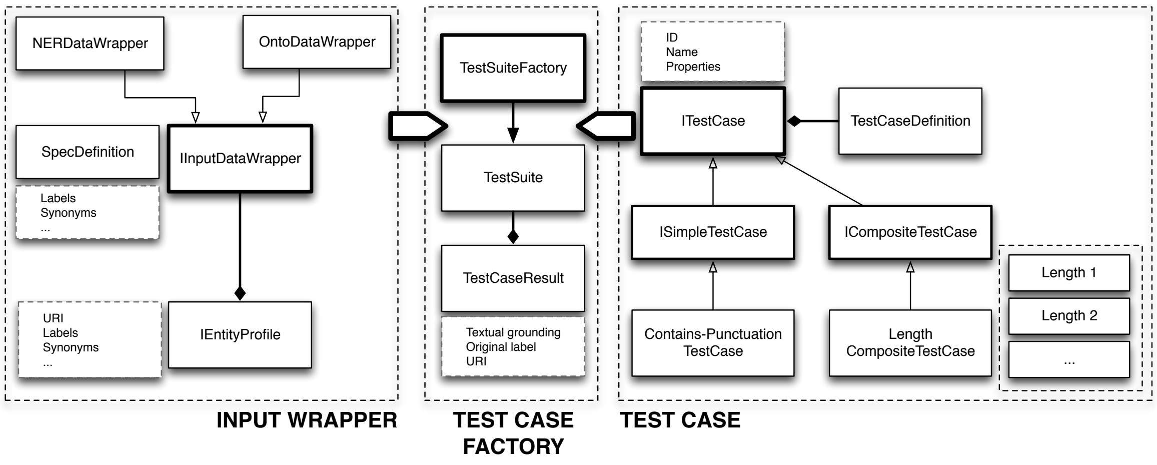

Figure 1: High-level overview of the test suite framework and its three major components: Input Wrapper – handling the input and producing Entity Profiles; Test Case – defining specific test case scenarios; Test Case Factory – bridging the provided input and a set of defined test cases.

textual representation for a concept or entity derived from a corpus annotation; and (iii) the list of synonyms – i.e., exact, related, broader or narrower synonyms for ontological concepts, or alternative textual representations for a concept or entity, as inferred from an annotated corpus.

3.1.1 Ontology term resources

The underlying Input Wrapper implementation is also responsible for defining the structure of the specification, according to which the processing is done. The current implementation of the framework provides an Ontology Data Wrapper that is able to perform the above-listed steps for a given ontology. The format of the ontology should be one of the formats supported by the OWL Api (Horridge and Bechhofer, 2011) – e.g., OWL, OBO, or RDF/XML. The resulting entity profiles will correspond to ontological concepts described via their URIs and the labels or synonyms defined in the specification. The actual specification is independent of the ontology, the ontology format or the implementation of the ontology processing mechanism within the OntoDataWrapper and it is defined using a simple JSON configuration file. This enables one to create and process the same ontology using different configurations.

The structure of the configuration file specifies:

- `conceptTypes`, the types of concepts to be processed
- `labelProperties`, the label properties to be considered
- `synonymProperties`, the synonyms properties to be considered, including a possible filtering based on the synonym type
- `uriPatterns`, URI patterns that should be included or excluded from the processing

Below we provide an example of an ontology specification (for an `OntoDataWrapper`) that will process only classes and will use the standard `dfs:label` and `skos:prefLabel` properties, in addition to any exact synonyms, defined by the pair `ono:synonym` – `ono:synonym type`. Furthermore, the specification excludes from processing three particular URIs (HP:0000001, HP:0000004 and HP:0000005). Please note that for brevity purposes, we do not list the complete URI of the properties.

```
{
  "conceptTypes": ["CLASS"],

  "labelProperties":{
    "http://.../rdf-schema#label":{},
    "http://.../skos#prefLabel":{},
  },

  "synonymProperties":{
    "http://.../obo/synonym":{
      "http://.../obo/synonymtype":["EXACT"]}
  },

  "uriPatterns":{
    "http://.../obo":[
      "*",
      "~HP_0000001",
      "~HP_0000005",
      "~HP_0000004"]
  }
}
```

3.1.2 Annotated corpus resources

We can straightforwardly extend the basic framework developed for ontology concepts to standard text corpora with annotations of ontology concepts or named entities over naturalistic text data. The framework enables organisation of annotated examples according to typological characteristics.

At a minimum, all that is required to achieve this at the basic technical level is to define an appropriate `InputDataWrapper`, e.g. `NERDataWrapper` in Figure 1. This Input Wrapper must know how to parse the relevant corpus representation. It would iterate through each annotation in the corpus, and either generate a new Entity Profile, or augment an existing Entity Profile when a new synonym or alternate form of an existing Entity is encountered.

3.2 Test Case Definitions

Test cases have the role of selecting or manipulating entity profiles characterised by certain properties of interest. As exemplified in (Cohen et al., 2010), the equivalence relations captured in such test cases may focus on length-based properties, lexical composition, lexical variation, etc. In principle, we can classify test cases into two categories: simple and composite. Simple test cases have a non-parametric form and analyse a particular property of entity profiles – e.g., if they contain punctuation. Composite test cases consist of a series of simple test cases concentrated on a single property, but which can be parametrized. For example, the process of verifying the existence of a given stop word (e.g., “of”, “by”, “from”) in an entity profile is independent of the actual stop word. Hence, a test case targeting treatment of terms containing stop words can take the stop word as a parameter. We consider this type of test case to be composite.

Our framework supports both types of test cases. In general, a Test Case includes high-level metadata (i.e., an identifier and a name, to improve human readability) and the set of properties that can be configured – as per the listing below.

```
public interface ITestCase {

  public String getId();

  public String getName();

  public List<String> getAcceptedProperties();

  public void addEntity(IEntityProfile profile);
}

public interface ISimpleTestCase extends ITestCase {

  public void runTestCases(Properties properties);

  public List<ITestCaseResult> retrieveTestCases();
}
```

The properties supported by the Test Case might include the number of entries to be generated in the test suite for this test case (this would apply to both test case types), or parameter values (which would be particular to a composite test case), e.g., the set of stop words to be analysed. The runtime values for these properties are transferred to the test case via a TestCaseDefinition, or in a programmatic manner, subject to the deployment settings.

Running a Test Case involves three steps: (i) populating the Test Case with Entity Profiles, (ii) generating Test Case Results according to the specified properties values, and (iii) retrieving the Test Case Results. The last two steps are dependent on the test category, as shown in the definition of the Simple Test Case interface in the listing above.

The results are currently provided as a set of objects that contain the resulting `textual grounding` (to be used as input in validation), the `original lexical representation` and the `identifier` of the associated entity. For example, let’s consider a lexical variation Test Case applied to the Gene Ontology[3] (Gene Ontology Consortium, 2000) concept GO:0070170 (*regulation of tooth mineralization*). The process result consists of:

- `textual grounding`: *regulated* tooth mineralization
- `original lexical representation`: regulation of tooth mineralization
- `concept identifier`: GO:0070170

Currently, the framework contains three simple test cases:

- Contains Arabic numeral – generates candidates that contain isolated Arabic numerals (e.g., 1, 2, ...)
- Contains Roman numeral – generates candidates that contain isolated Roman numerals (e.g., I, IX, ...)
- Contains punctuation – generates candidates that contain punctuation tokens

and two parametric composite test cases

[3]The Gene Ontology is an ontology capturing concepts related to gene function and biological processes.

- Contains stop word – generates candidates that contain user-specified stop words (e.g., OF, FROM, BY, ...)

- Length – generates candidates that have lexical groundings with a length in tokens equal to the list of user-specified lengths.

All test cases generate results in a randomised manner. That is, except for the core test case functionality, no particular heuristics or rules are used when selecting the resulting concepts.

3.3 Test Suite Factory

The Test Suite Factory connects the Input Wrapper to the existing Test Case implementations. Its role is to generate sets of Test Cases – a Test Suite – according to a provided definition on a given input. The implementation of the Test Suite Factory allows it to be used both in a continuous pipeline manner, as well as in a batch process. In the pipeline setting, the factory accepts dynamic creation and alteration of Test Suite definitions, while in the batch process setting the definitions need to be provided via a simple configuration file. Subject to the deployment setting, the resulting Test Suite can be used directly in evaluation experiments, or serialised for offline processing.

There are a few technical aspects that are worth mentioning in the context of the Test Suite Factory. The current implementation forces a generic Test Case to ingest one Entity Profile at-a-time (provided by the Input Wrapper Entity Profile iterator) – see the Test Case interface definition in the listing above. The actual processing of this Entity Profile is then delegated to the specific Test Case implementation (independently of its category). The rationale behind this decision was to maintain the memory footprint of the Input Wrapper at a reasonable level. This enables, for example, the processing of the 110MB SNOMED-CT clinical vocabulary (in its tabulated format, containing 398K concepts and 1.19M descriptions) on a standard machine without the need of a large memory allocation. Yet in order to take advantage of a multi-core architecture, where this is available, the test suite generation process introduces two parallelisation points. A first parallelisation point is created when iterating over the Entity Profiles, with each Entity Profile being provided at the same time to all instantiated Test Cases. The second parallelisation point is delegated to the Test Case implementation, which may take advantage of it when generating the Test Case Results.

4 Use of the generated Test Suite for evaluation

The framework we have developed provides the critical scaffolding for designing and creating Test Suites. It can be applied for concept recognition evaluation using the following workflow:

1. Given an ontology of interest, define the desired Input Wrapper specification – see the example discussed above;

2. Specify a desired Test Suite definition – using existing Test Cases and/or implementing additional ones;

3. Generate Test Case Results (via the Test Case Factory) and serialise them on disk.

To allow for an easy and versatile creation of Test Suite definitions, the Test Case Factory is able to instantiate Test Suites based on a configuration file that specifies the list of Test Cases and the properties to be used at runtime. Below we list an example of such a configuration file using all existing Test Case implementations (introduced above). Each Test Case is specified using its unique identifier, followed by a set of values for the properties it requires. The number of entries to be generated (by both simple and composite test cases) is specified via the `NO_ENTRIES` property. In addition, the composite Test Case Contains-STOP requires the actual stop words to be analysed (here TO, FROM and OF). A similar configuration could be provided also to the Length composite Test Case. The current implementation provides, however, the option of generating tests on all available lengths represented in the input terminology (ontology or corpus annotations), as shown in the listing below.

```
testcase[0].id=Contains-Arabic
testcase[0].property[NO_ENTRIES]=6

testcase[1].id=Contains-Arabic
testcase[1].property[NO_ENTRIES]=6

testcase[2].id=Contains-Punctuation
testcase[2].property[NO_ENTRIES]=4

testcase[3].id=Contains-STOP
testcase[3].set[TO].property[STOP_WORD]=to
testcase[3].set[TO].property[NO_ENTRIES]=10
testcase[3].set[FROM].property[STOP_WORD]=from
testcase[3].set[FROM].property[NO_ENTRIES]=6
testcase[3].set[OF].property[STOP_WORD]=of
testcase[3].set[OF].property[NO_ENTRIES]=5

testcase[4].id=Length
testcase[4].set[ALL].property[LENGTH]=ALL
```

An excerpt from the application of this Test Suite to the Gene Ontology is listed below.

```
#Contains-Arabic

T-helper 1 cell differentiation | GO:0045063
RNA cap 4 binding | GO:0000342

#Contains-Roman

transcription from RNA polymerase
   III type 2 promoter | GO:0001009
mitochondrial respiratory chain complex
   III assembly | GO:0034551

#Contains-Punctuation

21U-RNA binding | GO:0034583
6-deoxy-6-sulfofructose-1-phosphate
   aldolase activity | GO:0061595

#Contains-STOP-OF

establishment of neuroblast polarity | GO:0045200
regulation of tooth mineralization | GO:0070170

#Contains-STOP-TO

response to cortisone | GO:0051413
glutamate catabolic process to 4-hydroxybutyrate |
 GO:0036241

#Contains-STOP-FROM

calcitriol biosynthetic process from calciol |
 GO:0036378
positive regulation of exit from mitosis | GO:0031536

#Length-1

costamere | GO:0043034
amicyanin | GO:0009488
plasmodesma | GO:0009506

#Length-2

spermidine transport | GO:0015848
lobed nucleus | GO:0098537

#Length-3

energy transducer activity | GO:0031992
sinoatrial valve morphogenesis | GO:0003185
```

A specific concept recognition system evaluation process can ingest this serialisation, parse it into strings and annotations (identifier labels), and apply the concept recognition system directly to the test suite. Standard evaluation metrics (e.g., Precision, Recall, F-Score) can be computed directly on this data. Furthermore, by taking advantage of the intrinsic structure of the test suite, with individual test case strings grouped together, the Test Suite can be used to compute evaluation metrics per-category basis. This provides a more informative view of the strengths and weaknesses of the system under scrutiny, on the basis of the test cases. Coupled with a standardised error analysis framework, the test suite can be used to create comparative overviews across multiple concept recognition systems.

5 Discussion

5.1 Towards an end-to-end test suite-based evaluation system

The current implementation focuses on the test suite generation framework. We assume that a test suite generated with the framework will be used in a separate evaluation process, as described in Section 4.

Future developments of the framework will include an integrated evaluation pipeline, which will realise the required steps, notably parsing of the test suite, submission of each test string in turn to a concept recognition system, tracking of matches (TP/FP/FN), and category-based calculation of quantitative evaluation metrics.

Moreover, for ontology-based concept recognition, we intend to provide a library of generated Test Suites using ontologies from the BioPortal (Whetzel et al., 2011), in addition to a series of baselines, created using off-the-shelf systems, such as the NCBO Annotator (Jonquet et al., 2009) or ConceptMapper (Tanenblatt et al., 2010).

5.2 Generation of term variants

The sample test cases implemented to date address particular characteristics of concept terms. They involve matching of existing ontology terms and synonyms in the input source to these characteristics, and result in the organisation and grouping of those terms according to those characteristics. However, test cases can also be defined that manipulate terms in controlled ways to produce term variants for testing. This allows testing of the robustness of concept recognition in the face of particular types of changes to the input.

Several such changes were explored in the Gene Ontology test suite of (Cohen et al., 2010), including generation of plural variants of singular terms, and manipulation of word order of a multi-word term (which could either be expected to be tolerated by a concept recognition system, or an explicit error case that should be avoided).

Variants might be generated in which words of a multi-word term are separated, e.g. with a particular type of intervening text. An adjective might be inserted in a noun phrase (e.g., *regulation of exit from mitosis* → *regulation of **rapid** exit from mitosis*), or a quantifier added (e.g., *ensheathment of neurons* → *ensheathment of **some** neurons*).

Alternative syntactic realisations such as nominalisations or adjectival forms (e.g., *nucleus* →

nuclear), or linguistic alternations (e.g., *regulation of X $\rightarrow$ X regulation*) can be generated. Semantic variation can also be captured, such as substitution of a phrase within a synonym, e.g., *positive regulation $\rightarrow$ up-regulation* (as a substring of a longer term). Similarly, variants that involve abstraction or manipulation of terms with other terms embedded within them (i.e., recursive structure) can be generated to measure structural impacts (Verspoor et al., 2009; Ogren et al., 2005).

To the extent that such changes are systematic and generalisable, they can be represented programmatically and used to generate test cases within the test suite. This is planned for the next phase of system development.

5.3 Sentential contexts

The current framework focuses on generating test suites that consist of target vocabulary terms, or controlled variants of those terms. However, it has been previously pointed out that the performance of a concept recognition system may be dependent on the complexity of linguistic environment in which a concept is mentioned, rather than (or in combination with) the characteristics of the concept term itself (Cohen et al., 2004). Indeed, many methods for named entity recognition depend on the availability of meaningful (or at least syntactically correct) linguistic contexts in which term mentions occur; conditional random field models that are trained on naturally occurring data, for instance, are explicitly defined to make use of sentential context in their models.

Therefore, we aim to provide Test Case definitions that enable systematic specification of sentential contexts for the terms of the vocabulary source. This can be achieved with a Composite Test Case which combines Test Cases for concepts, with a set of sentential contexts (themselves varying according to controlled characteristics).

6 Conclusions

We have introduced a framework for automated creation of test suites for concept recognition systems. While prior work on test suites has either produced a static test suite for a particular NLP task (e.g., grammar engineering), or provided data aimed at generating specific types of test cases (Cohen et al., 2004), we have produced a software implementation that directly supports the specification of test cases, and generation of the test suite according to those test cases for a provided input terminology. The input can be extracted directly from a structured vocabulary resource such as an ontology, or inferred from annotations over a natural language corpus.

Test suites provide a powerful tool for error analysis. Following software engineering methodology, the organisation of data into explicitly defined classes provides insight into *how* a system succeeds or fails, rather than *how often*. An analysis of the performance of a concept recognition system in these terms is complementary to the standard evaluation metrics. While assessment of precision, recall, and F-score over naturalistic data clearly remains the most suitable strategy for gauging overall performance of the system, a test suite provides a more granular assessment corresponding to potential error categories.

Our initial implementation contains only a limited number of existing test case definitions. However, the framework is flexible and new test cases appropriate to particular sets of concepts, and particular corpus characteristics, can easily be added. We invite the community to contribute test cases to the framework.

Acknowledgments

Tudor Groza gracefully acknowledges the funding received from the Australian Research Council (ARC) via the Discovery Early Career Researcher Award (DECRA) [DE120100508].

References

Alan R. Aronson and Francois-Michel Lang. 2010. An overview of MetaMap: historical perspective and recent advances. *Journal of the American Medical Informatics Association*, 17:229–236.

Emily M. Bender, Dan Flickinger, and Stephan Oepen. 2002. The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In John Carroll, Nelleke Oostdijk, and Richard Sutcliffe, editors, *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, Taipei, Taiwan.

Emily M. Bender, Dan Flickinger, and Stephan Oepen. 2008. Grammar engineering for linguistic hypothesis testing. In Nicholas Gaylord, Stephen Hilderbrand, Heeyoung Lyu, Alexis Palmer, and Elias Ponvert, editors, *Texas Linguistics Society 10: Computational Linguistics for Less-Studied Languages.* CSLI Publications.

Emily M. Bender, Scott Drellishak, Antske Fokkens, Laurie Poulson, and Safiyyah Saleem. 2010. Grammar customization. *Research on Language & Computation*, 8(1):23–72. 10.1007/s11168-010-9070-1.

David Campos, Sergio Matos, and Jose Luis Oliveira. 2013. A modular framework for biomedical concept recognition. *BMC Bioinformatics*, 14:281.

K. Bretonnel Cohen, Lorraine Tanabe, Shuhei Kinoshita, and Lawrence Hunter. 2004. A resource for constructing customized test suites for molecular biology entity identification systems. In *HLT-NAACL 2004 Workshop: BioLINK 2004, Linking Biological Literature, Ontologies and Databases*, pages 1–8. Association for Computational Linguistics.

K. Bretonnel Cohen, William A. Baumgartner, Jr., and Lawrence Hunter. 2008. Software testing and the naturally occurring data assumption in natural language processing. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, SETQA-NLP '08, pages 23–30, Stroudsburg, PA, USA. Association for Computational Linguistics.

K. Bretonnel Cohen, Christophe Roeder, William A. Baumgartner Jr., Lawrence E. Hunter, and Karin Verspoor. 2010. Test suite design for biomedical ontology concept recognition systems. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May. European Language Resources Association (ELRA).

Christopher Funk, William Baumgartner, Benjamin Garcia, et al. 2014. Large-scale biomedical concept recognition: an evaluation of current automatic annotators and their parameters. *BMC Bioinformatics*, 15(1):59.

Gene Ontology Consortium. 2000. Gene Ontology: tool for the unification of biology. *Nat Genet*, 25(1):25–29.

Matthew Horridge and Sean Bechhofer. 2011. The OWL API: A Java API for OWL Ontologies. *Semantic Web Journal*, 2(1):11–21.

Clement Jonquet, Nigam H Shah, and Mark A Musen. 2009. The open biomedical annotator. *Summit on translational bioinformatics*, 2009:56–60.

Yoshinobu Kano, William A. Baumgartner, Luke McCrohon, et al. 2009. U-compare. *Bioinformatics*, 25(15):1997–1998.

Y. Kano, J. Bjorne, F. Ginter, T. Salakoski, et al. 2011. U-compare bio-event meta-service: compatible bionlp event extraction services. *BMC bioinformatics*, 12(1):481.

K. Jonathan Kummerfeld and Dan Klein. 2013. Error-driven analysis of challenges in coreference resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 265–277. Association for Computational Linguistics.

K. Jonathan Kummerfeld, David Hall, R. James Curran, and Dan Klein. 2012. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1048–1059. Association for Computational Linguistics.

Zhiyong Lu, Hung-Yu Kao, Chih-Hsuan Wei, et al. 2011. The gene normalization task in BioCreative III. *BMC Bioinformatics*, 12(Suppl 8):S2.

Glenford J. Myers. 1979. *The Art of Software Testing*. John Wiley & Sons, Inc.

S. Oepen, K. Netter, and J. Klein. 1998. TSNLP - test suites for natural language processing. In John Nerbonne, editor, *Linguistic Databases*, chapter 2, pages 13–36. CSLI Publications.

Stephan Oepen. 1999. Competence and Performance Laboratory. User and Reference Manual. Technical report, Computational Linguistics, Saarland University.

P Ogren, K Cohen, and L Hunter. 2005. Implications of compositionality in the Gene Ontology for its curation and usage. In *Pacific Symposium on Biocomputing*, pages 174–185.

Dietrich Rebholz-Schuhmann et al. 2008. Text processing through Web services: calling Whatizit. *Bioinformatics*, 24(2):296–298.

Veselin Stoyanov, Nathan Gilbert, Claire Cardie, and Ellen Riloff. 2009. Conundrums in noun phrase coreference resolution: Making sense of the state-of-the-art. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 656–664. Association for Computational Linguistics.

Michael Tanenblatt, Anni Coden, and Igor Sominsky. 2010. The ConceptMapper Approach to Named Entity Recognition. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May. European Language Resources Association (ELRA).

K. Verspoor, D. Dvorkin, K.B. Cohen, and L. Hunter. 2009. Ontology quality assurance through analysis of term transformations. *Bioinformatics*, 25(12):i77.

Patricia L Whetzel, Natalya F Noy, Nigam H Shah, Paul R Alexander, Csongor Nyulas, Tania Tudorache, and Mark A Musen. 2011. BioPortal: enhanced functionality via new Web services from the National Center for Biomedical Ontology to access and use ontologies in software applications. *Nucleic Acids Research*, 39(Web Server issue):W541–5, July.