Natural Language Engineering **21** (2): 167–200. © Cambridge University Press 2013
doi:10.1017/S1351324913000351

Design and realization of a modular architecture for textual entailment

SEBASTIAN PADÓ[1], TAE-GIL NOH[2], ASHER STERN[3], RUI WANG[4] and ROBERTO ZANOLI[5]

[1]*Institute for Natural Language Processing, Stuttgart University, 70569 Stuttgart, Germany*
e-mail: pado@ims.uni-stuttgart.de
[2]*Institute of Computational Linguistics, Heidelberg University, 69120 Heidelberg, Germany*
e-mail: noh@cl.uni-heidelberg.de
[3]*Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel*
e-mail: astern7@gmail.com
[4]*German Research Center for Artificial Intelligence, 66123 Saarbrücken, Germany*
e-mail: rui.wang@dfki.de
[5]*Human Language Technology, Fondazione Bruno Kessler, 38123 Trento, Italy*
e-mail: zanoli@fbk.eu

(*Received 15 January 2013; revised 5 November 2013; accepted 6 November 2013; first published online 13 December 2013*)

Abstract

A key challenge at the core of many Natural Language Processing (NLP) tasks is the ability to determine which conclusions can be inferred from a given natural language text. This problem, called the *Recognition of Textual Entailment* (RTE), has initiated the development of a range of algorithms, methods, and technologies. Unfortunately, research on Textual Entailment (TE), like semantics research more generally, is fragmented into studies focussing on various aspects of semantics such as world knowledge, lexical and syntactic relations, or more specialized kinds of inference. This fragmentation has problematic practical consequences. Notably, interoperability among the existing RTE systems is poor, and reuse of resources and algorithms is mostly infeasible. This also makes systematic evaluations very difficult to carry out. Finally, textual entailment presents a wide array of approaches to potential end users with little guidance on which to pick. Our contribution to this situation is the novel EXCITEMENT architecture, which was developed to enable and encourage the consolidation of methods and resources in the textual entailment area. It decomposes RTE into components with strongly typed interfaces. We specify (a) a modular linguistic analysis pipeline and (b) a decomposition of the 'core' RTE methods into top-level algorithms and subcomponents. We identify four major subcomponent types, including knowledge bases and alignment methods. The architecture was developed with a focus on generality, supporting all major approaches to RTE and encouraging language independence. We illustrate the feasibility of the architecture by constructing mappings of major existing systems onto the architecture. The practical implementation of this architecture forms the EXCITEMENT open platform. It is a suite of textual entailment algorithms and components which contains the three systems named above, including linguistic-analysis pipelines for three languages (English, German, and Italian), and comprises a number of linguistic resources. By addressing the problems outlined above, the platform provides a comprehensive and flexible basis for research and experimentation in textual entailment and is available as open source software under the GNU General Public License.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

1 Introduction

Textual Entailment (in short, TE) is a common sense notion of inference, expressed as a relation between two natural language texts T (text) and H (hypothesis), defined as follows: *T entails H (T ⊨ H) if, typically, a human reading T would infer that H is most likely true* (Dagan, Glickman and Magnini 2005). Consider the following example, where T ⊨ H_1, but T ⊭ H_2.

(1) **T:** *That was the 1908 Tunguska event in Siberia, known as the Tunguska meteorite fall.*
H_1: *A shooting star fell in Russia in 1908.*
H_2: *Tunguska fell to Siberia in 1908.*

The Recognition of Textual Entailment (RTE) has found considerable interest in the Natural Language Processing (NLP) community as a unified paradigm that addresses the prevalent *fragmentation* in semantic processing due to its potential to cover the semantic processing requirements of varied applications. In contrast to syntax, where a substantial degree of consensus exists on most phenomena and their analysis, research in semantics covers a wide range of phenomena which are described with various theories. In fact, semantic processing used to be predominantly conceptualized as the task of producing *formal language representations* for which correct and complete inference mechanisms were known (cf. Meyers 2005). This makes it very difficult to compare approaches under comparable conditions. TE replaces the representation-centered notion of semantic processing by a task-based one that is agnostic to representations, focusing on the ability to decide the correctness of inferences among natural language texts. The vision of our line of research is a unified platform for semantic processing that is as easy to use as today's parsers for syntactic analysis.

Since it was first introduced, research on RTE explored various directions, ranging from shallow lexical-based systems to large and complex systems. Such systems typically use different linguistic analyses (e.g. part-of-speech (POS) tagging, syntactic parsing, semantic role labeling), and exploit various kinds of knowledge resources (e.g. WordNet (Fellbaum 1998), DIRT (Lin and Pantel 2002), or FrameNet (Baker, Fillmore and Lowe 1998)), resulting in considerable complexity.

As in other areas of NLP, a key issue in textual entailment research is to ensure that new research can build directly on, and extend, the state of the art. In many areas, this goal is met by the public availability of software that implements the state-of-the-art approaches (e.g. syntactic parsers, semantic role labelers, and machine translation systems). Arguably, as a complex task, this is equally crucial for RTE.

Unfortunately, while individual RTE systems were made publicly available, there are still considerable hurdles for new research due to two key issues. First, unlike in other areas (e.g. Statistical Machine Translation (SMT) and syntactic parsing), there is no unified paradigm in RTE: Many approaches have been proposed, but none of them has consistently outperformed all the others and has emerged as dominant. Thus, the availability of a particular RTE system can only help researchers who are interested in the approach used in that system, while research of a novel algorithm requires the development of a whole system from scratch. Second,

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

RTE systems rely on tightly integrated components such as linguistic analysis tools, knowledge resources, or processing modules for specific textual entailment phenomena (temporal entailment, arithmetic entailment, negation detection, etc.). While different systems often rely on the same or similar functionalities, their development is usually duplicated in almost every system, since such components are usually tailored to specific algorithms and therefore very difficult to reuse.

These observations lead us to the conclusion that research in the RTE area could be stimulated by having available a *suite* of textual inference components which can be combined as freely as possible into complete textual inference systems. Researchers with novel ideas could then 'mix and match' the components best suited to follow her/his idea, saving time and efforts of development.

Such a perspective on RTE has not only practical but also theoretical advantages as it simplifies evaluation. Currently, the lack of clean modularization of systems makes it very difficult to assess the influence of single components in isolation (e.g. specific knowledge resources or processing strategies). Correspondingly, studies have found it difficult to discover general patterns regarding the usefulness, e.g. of anaphora resolution (Mirkin, Dagan and Padó 2010) or of knowledge resources (Bentivogli *et al.* 2009). A shared RTE suite that can keep all components constant except the one under evaluation could greatly improve this situation.

Our contribution in this paper is such a component-based architecture for the recognition of textual entailment, the EXCITEMENT architecture. It follows ideas for sustainable software development for NLP (Curran 2003; Patrick and Cunningham 2003; Cohen and Carpenter 2008), and supports code and resource reuse. The interfaces of the components are designed to promote generality and openness: The architecture covers all major approaches to textual entailment, encourages language- and theory-independent processing, and was designed to allow for future extensions. The specification document is complete and freely available.[1]

The EXCITEMENT architecture is accompanied by the EXCITEMENT platform, that is, a reference implementation of the architecture with a set of components obtained by decomposing the three existing RTE systems into standalone components. This open source platform, released in summer 2013, covers major textual entailment recognition paradigms as well as linguistic analysis components and resources for English, German, and Italian,[2] making RTE more easily usable in NLP applications, enabling better evaluation, and encouraging sustainable system and resource development.

Plan of the paper. Section 2 sums up the state of the art in textual entailment and its shortcomings. Section 3 gives an overview of our architecture, with Sections 4 through 6 covering details. Section 7 shows how concrete existing systems can be

[1] URL: `http://excitement-project.eu/index.php/results`
[2] The platform is available at `http://hltfbk.github.io/Excitement-Open-Platform/`

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

mapped onto the architecture. Section 8 outlines the properties of the platform that we release on the basis of the architecture. Section 9 discusses related work, and Section 10 concludes the paper.

2 Recognition of Textual Entailment (RTE)

2.1 RTE as a general paradigm for semantic processing

As described above, textual entailment aims at capturing human intuitions about valid common sense reasoning patterns by defining a binary relation between two passages of text T (text) and H (hypothesis) that holds if readers of T 'would typically infer that H is most likely true'. Following this intuition, the role of textual entailment in applications is generally to deal with the *variability of language*, that is, the fact that the same state of affairs can be described in a large number of different ways. More specifically, almost all NLP tasks deal with pairs of texts (e.g. queries and answers, or answer candidates and gold answers) whose semantic relationship must be determined in the face of variability. Compare Example (1) above, where T and H_1 are related while T and H_2 are not. At the same time, T and H_1 have a relatively small lexical overlap, a phenomenon named the 'lexical chasm' (Berger *et al.* 2000). This indicates that the exclusive use of shallow methods can be insufficient, and motivates the use of the more general approach of textual entailment.

Many applications broadly related to search, such as question answering, can employ textual entailment as a *verification* step. For efficiency reasons, such search tasks generally adopt a retrieval phase that gathers a small set of plausible candidates from large data sets using shallow methods. Textual entailment can then be used to perform a deeper analysis that verifies or rejects these candidates. In question answering, answer candidates are tested for whether they entail the search query reformulated as a statement (Harabagiu and Hickl 2006; Peñas *et al.* 2008). In information extraction, instance candidates can be compared against templates (Romano *et al.* 2006). A second group of applications use textual entailment as a *scoring* method that quantifies to what extent one text implies another. Examples include Intelligent Tutoring (Nielsen, Ward and Martin 2009), where good student answers should entail gold standard answers, and Machine Translation (Padó *et al.* 2009), where system output and manual translation should entail each other. Finally, textual entailment can be used for (hierarchically) *structuring* sets of texts, for example in multi-document summarization (Harabagiu, Hickl and Lacatusu 2007) and information visualization (Berant, Dagan and Goldberger 2012).

2.2 The state of the art in RTE

A wide variety of NLP methods have been applied to RTE.[3] Almost all systems first preprocess T and H (e.g. by lemmatization, POS tagging, or parsing) to abstract

[3] See Androutsopoulos and Malakasiotis (2010) and Sammons, Vydiswaran and Roth (2012) for recent overviews.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

away from surface variation. The subsequent processing strategies fall into three broad classes: alignment-based, transformation-based, and formal reasoning-based RTE. Note that many implemented systems are hybrid, combining elements of several strategies.

Alignment-based RTE constructs an explicit or implicit alignment between the linguistic entities of T and H (e.g. words, phrases, dependency nodes, or semantic structure). Entailment is decided on the basis of alignment quality, generally by assessing the validity of each 'local' inference in an aligned T-H pair of entities. Often different aspects of quality are combined to arrive at a final decision using supervised learning methods (Monz and de Rijke 2001; MacCartney *et al.* 2006; Zanzotto, Pennacchiotti and Moschitti 2009).

Transformation-based RTE aims at validating the entailment relation $T \models H$ by finding a 'proof', i.e. a sequence $(T, T_1, \ldots, T_n)$, such that $T_n = H$ (Bar-Haim, Szpektor and Glickman 2005; Harmeling 2009). The individual transformation steps $T_i \models T_{i+1}$ are generally motivated by lexical or lexico-syntactic inference rules such as *'Y was purchased by X'* $\models$ *'X paid for Y'*.

Formal reasoning-based RTE pursues a more traditional approach to inference by constructing meaning representations for T and H in a formal language with formal reasoning procedures, such as predicate or description logics (Tatu and Moldovan 2005; Bos and Markert 2006; Bobrow *et al.* 2007).

The second distinction can be drawn between knowledge-lean algorithms (Harmeling 2009) and algorithms that rely on linguistic and world knowledge to decide the validity of inferences. The latter class accounts for the vast majority of approaches. The use of linguistic knowledge (such as lexical inference rules) actually cuts across the three broad classes outlined above: In alignment-based RTE, such rules can be used to score matches between entities; in transformation-based RTE, these can be used as proof steps; and in formal reasoning (logics)-based RTE, these can be formulated as background axioms. Such knowledge is generally drawn from knowledge resources, such as WordNet (Fellbaum 1998) or DIRT (Lin and Pantel 2002), but knowledge gaps remain a key obstacle (Mirkin, Dagan and Shnarch 2009).

2.3 Problems of the state of the art

As we have sketched above, no canonical algorithm, nor even a canonical strategy, has emerged for textual entailment. This situation mirrors the state of semantics as complex area of research that still lacks unified processing approaches that are available for other fields like syntactic analysis or machine translation.

At the same time, the development of a system for recognizing textual entailment represents a substantial development effort. Only in the last two or three years, first publicly available systems for RTE, such as BIUTEE (Stern and

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

Dagan 2011) and EDITS (Negri *et al.* 2009), have appeared. These systems are however still implementations of particular RTE strategies and not generic platforms. In contrast, in the field of machine translation, researchers can download MOSES (Koehn *et al.* 2007), an open-source toolkit for machine translation, and extend a ready-to-use competitive platform with their own ideas.

The absence of a sufficiently generic system of this type for textual entailment has a number of problematic consequences, both for researchers interested in textual entailment itself and those who want to apply RTE technology to applications.

(1) *Reusability of algorithms and resources.* Researchers who plan to work on textual entailment have a choice: They can either develop a new system from scratch, which involves substantial effort, or use available systems like BIUTEE or EDITS as a basis for further development. If they choose the first option, they need to develop the complete system. Concerning the second option, existing systems are not generally designed to be modular or reusable since there is no standard for packaging RTE methods. The same holds for resources: It is well-established that for most approaches, knowledge resources play a central role in the recognition of textual entailment. Given the absence of a widely used standard for such resources, new systems typically develop their own resources from scratch since they cannot reuse existing resources of other systems. This raises a high bar for new entrants in the field of textual entailment.

(2) *Generality*. If researchers choose the second avenue outlined above, i.e. to use the existing RTE systems, they encounter another problem. Since these systems were not developed as generic platforms, users may be limited by the design decisions taken by its developers, e.g. the overall strategy chosen for RTE (cf. Section 2.2). Only research along the lines of the existing systems can be performed without investing a major development effort. Given the absence of a unified theory of textual inference compared with, for instance, SMT, this stifles the development of novel approaches in the area.

(3) *Systematic evaluation.* Progress in processing-oriented research areas like textual entailment hinges on advances in the understanding of the impact of various factors on output quality. Ideally, experiments would vary factors systematically while keeping the overall system setup constant. However, due to the specialized nature of the publicly available RTE systems mentioned above, comparisons across systems have turned out to be extremely difficult. Some of the RTE workshops (e.g. RTE 2 and 4) have attempted to analyze the system results in terms of knowledge resources utilized, but were not able to establish firm patterns. Thus, the present situation lacks a clear understanding of the systematic influence of design decisions in RTE systems.

(4) *Multilinguality*. To our knowledge, EDITS is the only publicly available system which has been developed with multilinguality in mind; it has been applied to both English and Italian. Applying other systems to new languages involves, in addition to replacing the relevant preprocessing tools, a thorough inspection of the code for details that may be language-specific. This problem seriously

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

impacts the uptake of entailment technology for more languages: after ten years, most textual entailment technology has been applied only to English and Italian, with pilot studies for Spanish (Castillo 2010) and German (Wang and Neumann 2008b). More recently, there has been interest in cross-lingual textual entailment (Mehdad, Negri and Federico 2010, 2011; Negri *et al.* 2011). Tasks on cross-lingual textual entailment for content synchronization have been carried out at SemEval in 2012[4] (Negri *et al.* 2012) and 2013.[5]

(5) *Integration in applications.* Integration of RTE technology into NLP applications faces the challenge that existing applications typically have a preexisting processing pipeline that they build on. Integrating an existing RTE system with its own pipeline into the application typically entails that the data are preprocessed again within the RTE system, which is inelegant and may even be impossible for time-critical applications. Consequently, although many NLP tasks have been phrased in terms of entailment (cf. Section 2.1), usage of textual entailment in actual applications is still rare.

3 EXCITEMENT: a modular architecture for RTE

Our goal is to address the shortcomings described in the previous section. Our main observation is that RTE systems, although significantly differ from each other, tend to share many resources as well as underlying structure. Thus, while there is currently no perspective for a completely generic Moses-like RTE system, we provide a specification for an architecture, accompanied by an implemented suite that provides most of the required components for new RTE systems. We believe that this contribution has the potential to substantially decrease the effort for the development of novel RTE systems and the evaluation and analysis of the existing systems.

Our general approach follows the idea of *modularization*: We decompose the recognition of textual entailment into a number of components. The resulting EXCITEMENT architecture provides clear definitions and interfaces for these components. Its overall structure is shown in Figure 1: We first decompose RTE into two parts, one for linguistic analysis (the Linguistic Analysis Pipeline, LAP), and one for entailment computation (the Entailment Core, EC). The LAP is essentially a series of annotation components that provide linguistic annotation on various layers for the input. The EC then performs the actual entailment recognition based on the preprocessed text. The separation ensures (a) that the algorithms in the EC only rely on linguistic analyses in well-defined ways (for details, see Section 4.2); and (b) that the LAP and the EC can be run independent of each other in practice (e.g. to preprocess all data beforehand).

Furthermore, we decompose both LAP and EC into components that are as independent of one another as possible. The top-level object in the EC is the Entailment Decision Algorithm (EDA): it computes an entailment decision for a given Text/Hypothesis pair, and can use components that provide standardized

[4] http://www.cs.york.ac.uk/semeval-2012/task8/
[5] http://www.cs.york.ac.uk/semeval-2013/task8/

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

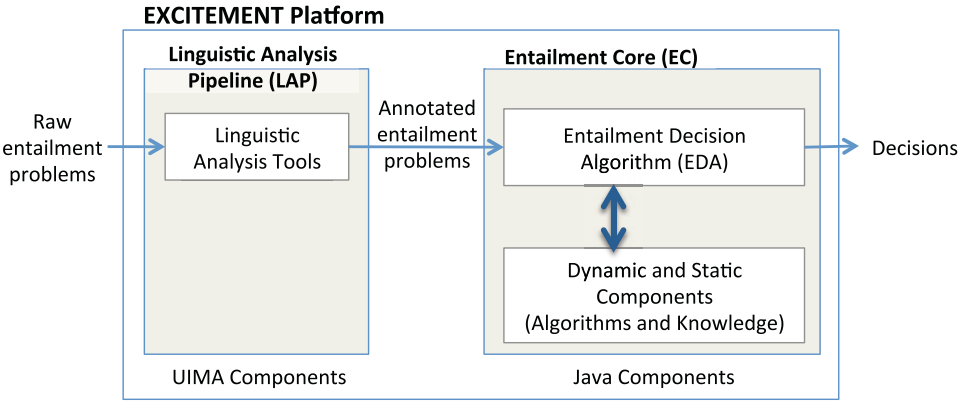


Fig. 1. (Colour online) EXCITEMENT architecture overview.

algorithms or knowledge resources. The EDA is accompanied by a set of standard component types described below. This establishment of EDAs and a pool of components are crucial for code reusability.

The design of such an architecture requires a strong focus on procedures for interoperability and good behavior of components. These procedures differ between LAP and EC. For the LAP, inputs and outputs of components are defined by IBM's Unstructured Information Management Architecture (UIMA) (Ferrucci and Lally 2004) and its type system. In contrast, the EC components are normal Java modules, and they are defined by a set of Java interfaces and Java data structures, accompanied by policies and recommendations on the implementation of the data structures.[6]

While the separation into LAP and EC is present to some degree in many RTE systems, our study is, to our knowledge, the first one to propose a generic decomposition of entailment computation in terms of generic interfaces. We now provide details on the LAP (Section 4), EDAs (Section 5), and the components (Section 6).

4 Linguistic analysis pipeline (LAP)

The desiderata of the architecture (cf. Section 3) directly translate into requirements for the LAP. Generality requires that the LAP handles input data and processing components for arbitrary languages and (as far as reasonable) linguistic formalisms. For extensibility, the LAP must be able to accommodate new processing components and represent their output. The LAP must also support metadata: In a modular entailment platform, the EC must be able to determine whether the input meets the requirements of the currently activated EDA and components (e.g. it is impossible

[6] We chose to specify the hard constraints as Java interfaces, since the Java type system meets our needs, is largely operating system-independent, and much work in NLP (including several RTE systems) is realized in Java.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

to apply a tree edit distance-based classifier if the input is not parsed). Finally, the LAP should be transparent and easy to configure and extend.

4.1 Unstructured information management architecture (UIMA)

We assessed several possible choices for data representation in the LAP. The column-based CoNLL 2009 file format (Hajič *et al.* 2009) offers a lightweight, general representation, but lacks in extensibility: new layers must be defined *ad hoc*, variable annotation (such as semantic roles) leads to variable numbers of columns, and there is no declarative support for metadata. More expressive representations, such as LAF and GrAF (Ide and Suderman 2007), overcome these problems, but are only in the process of being standardized.

We decided to adopt UIMA (Ferrucci and Lally 2004), a general architecture for analysis components that is supported by the Apache Foundation and has healthy communities in both academic and commercial domain. UIMA provides a unified way of constructing linguistic pipelines from analysis components, which are wrapped into the so-called UIMA components.

We concentrate on UIMA's support for storing components' analysis results in the so-called Common Analysis Structures (CAS).[7] A CAS starts out containing only plain text. As it passes through analysis components, additional annotation layers are stacked onto the text. The input and output of every component are standardized by a type system. CAS also supports the storage of metadata such as language flags. Components can check for the presence of analysis layers to detect incompatibilities (e.g. a parsing component can check that its input is POS-tagged).

The role of the CAS type system in defining the admissible annotations can be understood in analogy to POS tag inventories, but is more powerful in that it defines not only the inventory but also the structures of annotation. Fortunately, there are existing CAS type systems for the NLP domain. We adopted the DKPro type system (Gurevych *et al.* 2007), which was designed with language independence in mind. It provides types for morphological information, POS tags, dependency structure, named entities, coreference, and semantic roles.

We have made two major customizations to use UIMA CAS for textual entailment: one extension is the definition of new type systems for entailment task, and the other one is the representation of entailment problems as single CASes.

First, we needed to represent entailment problems (in the easiest case, a text-hypothesis pair) in terms of UIMA CASes. In general text analysis use cases, one UIMA CAS corresponds to one document; this suggests representing both text and hypothesis as separate CASes. However, we decided to store complete entailment problems as individual CASes. This approach has two major advantages: (i) It enables us to obtain a uniform representation for 'simple' entailment problems (one text and one hypothesis) and 'complex' entailment problems (one text and multiple hypotheses or *vice versa*, as in the 'RTE search' task (Bentivogli *et al.* 2009)). (ii)

[7] UIMA also supports the combination of components into pipelines through standardized runtime behavior. We do not use this aspect of UIMA at the moment.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

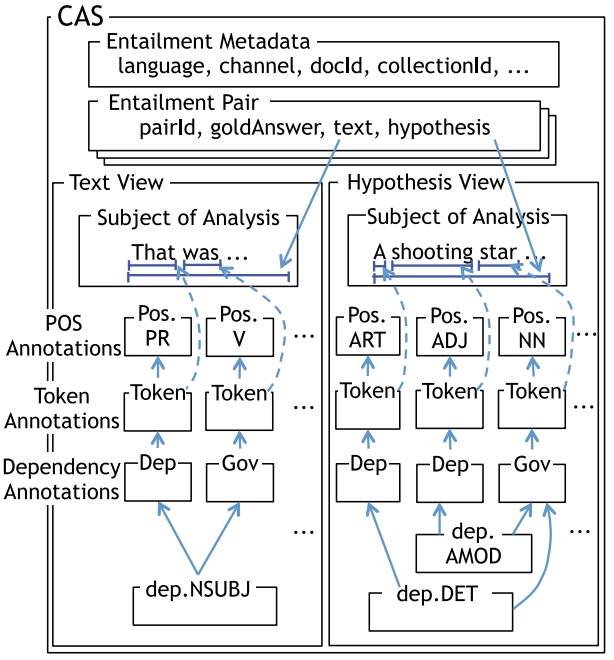


Fig. 2. (Colour online) CAS representation of a text-hypothesis pair.

Joint storage in the same CAS enables us to store cross-annotations between texts and hypotheses, notably alignments (cf. Section 2.2).

Second, we needed to extend the DKPro type system with the types necessary to define textual entailment-specific annotation. This involved types for marking stretches of text as texts and hypotheses, respectively, as well as storing correspondence information between texts and hypotheses, pair IDs, gold labels, and some metadata such as language. We also added types for linguistic annotation that are not exclusively entailment-specific, but have found use in RTE and were not covered yet by DKPro. This included annotation for polarity, the reference of temporal expressions, and word and phrase alignments between text and hypothesis.

Figure 2 shows a pictorial example of the CAS data structure for the pair (T, H_1) from the example in Section 1. It contains the two text fragments ('subject of analysis' in UIMA terms) and their annotations (here POS and dependencies), as well as global data such as metadata proper (e.g. language) and properties of the entailment pair (e.g. the gold-standard answer). The example shows one T-H pair, but an entailment problem (one CAS) can have multiple pairs with multiple texts or hypotheses.

LAP versus EC. We only adopted UIMA for LAP, and not for EC, whose components (EDAs, algorithms, and knowledge components) are 'normal' Java components (cf. Figure 1). The reason is that UIMA's strength is defining document-based annotation pipelines: starting out from a 'raw' document, components are called sequentially and each one adds its output as a new annotation layer to the document. In contrast, UIMA cannot deal well with situations where there is no

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

underlying document that is enriched, or where there is a more flexible flow of information. This is the case in the EC.

As an example, consider the interaction between an EDA and a lexical knowledge component such as WordNet (cf. Section 6.3 for details). In the typical use case, the input to the lexical knowledge component is a single word (for example, *dog*), and the output is the set of words lexically entailed by the input word (*mammal, animal, object, …*). The first problem is that each component in a UIMA pipeline must be called a fixed number of times. This order and number of times are defined at the pipeline's building time, which is not appropriate for dynamic situations such as lexical lookup. The second problem is the UIMA CAS format itself. Being optimized for storing documents, it does not provide fast (ideally constant-time) queries and is therefore ill-suited for knowledge storage and access.

Consequently, we have decided to define the EC side of the architecture with Java interfaces and Java data types. These types are described in the following sections.

4.2 Interoperability: achievements and limits

Our adoption of a UIMA-based LAP architecture achieves our goal of decoupling the LAP from the EC on a technical level: the UIMA CASes and their type system serve as the only way to exchange information (cf. Figure 1). However, this situation still leaves room for two broad classes of incompatibility.

First, 'hard' incompatibilities arise when dependencies are violated, for example when analysis components for different languages are combined, or when the EC requires some annotation that was not produced by the LAP. We catch such cases by requiring the EC to check for the presence and consistency of all required layers.

The second, more difficult, class consists of 'soft' incompatibilities below the level of standardization provided by the type system. These can be broadly characterized as differences in 'annotation style'. For example, consider a constituent parser that produces flat NP analyses. If it is replaced by another parser which assumes hierarchical analysis of NPs, the difference cannot be detected at the level of the UIMA type system. However, it can clearly degrade the functionality of downstream components, in particular for linguistically informed components which identify particular linguistic structures, e.g. to apply inference rules.

One solution to this problem would be to prescribe one canonical 'annotation style' for each analysis level, for example, Stanford dependencies for parsing. This idea however runs counter to our goal of an open platform. Without automatic methods to detect soft incompatibility, our solution is a combination of documentation and policy. On the policy level, we strongly recommend that EDA developers keep the core EDAs as generic as possible, placing annotation's style-specific parts into knowledge components (cf. Section 5). On the documentation level, all EDAs and linguistic knowledge components are to be annotated with information about the analysis tools used for their creation. The price we pay for the openness is that the change of a major LAP component may entail adapting linguistically informed EC components and resources. Nevertheless, we believe that a pluralistic approach is preferable, and support it with a clear methodology and best practice guidelines.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

5 Entailment core (EC)

Various studies have investigated the role of different types of knowledge in textual entailment, such as morphological, syntactic, semantic, and world knowledge (Clark *et al.* 2007; Mirkin *et al.* 2009). This line of research has also driven the development of various systems, including EDITS and TIE (Wang and Neumann 2008a; Cabrio and Magnini 2011). However, these systems are still implementations of individual strategies, rather than generic architectures designed to accommodate a wide range of entailment components.

Our architecture makes a concrete proposal for the modularization of general RTE approaches. The EC consists of an EDA and zero or more subordinate components. EDAs are the top-level algorithms for RTE, realized as Java interfaces. Their core functionality is covered by one method that takes annotated input in the form of a UIMA CAS, and returns a decision (see below). Components wrap reusable functionality either in the form of algorithms (e.g. alignment computation, feature computation, etc.) or knowledge resources and are described by a set of Java interfaces. The main aspects of specifying the EC interface are covered in the following sections; the definition of component types follows in Section 6.

5.1 Decomposition: EDAs versus components

Existing RTE systems can be trivially wrapped as a whole into an EDA. However, our goal is to support the decomposition of existing systems into single EDAs plus a range of subordinate components in order to encourage developers to focus on the reusability of their RTE algorithms and resources. Since there is no unique correct way of decomposition, what is necessary are guidelines that describe as precisely as possible how to split up a given system into EDAs and components.

Our guiding principle is that EDAs should be as minimal and ideally as language-independent as possible, while the following three criteria identify functionalities that should be put into components: (a) functionality that is strongly dependent on particular linguistic structures (cf. Section 4.2); (b) substantial amounts of declarative knowledge (i.e. resources); (c) functionality that is useful for more than one class of EDAs. Following this idea, we have developed a set of four component interfaces (i.e. types) which cover the main classes of reusable functionalities that we identified, discussed in detail in Section 6. Developers are encouraged to use those components. They can add their own types if required, but since this involves additional overhead, we hope that this setup encourages a degree of reusability.

5.2 Result types

At first glance, the result of the EDA should obviously be a binary variable (ENTAILMENT versus NON-ENTAILMENT), as adopted by the first RTE workshops. However, the task was subsequently extended in various ways. (de Marneffe, Rafferty and Manning 2008) consider a three-way decision that includes CONTRADICTION. Androutsopoulos and Malakasiotis's (2010) survey argues that techniques for PARAPHRASE acquisition share many techniques with RTE. (MacCartney and Manning

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

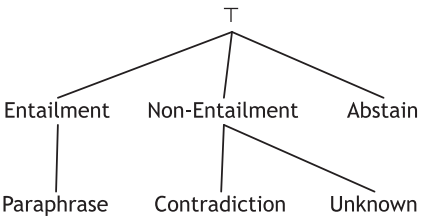


Fig. 3. EDA result type hierarchy.

2007) and (Wang 2011) also consider more comprehensive sets of semantic relations between texts.

In order to keep the architecture general, but focused on textual entailment, we decided to model the decision type as follows. We define six labels in a hierarchically structured `Enum` as shown in Figure 3. EDAs can use the upper level for a more coarse-grained task, or can selectively include the lower level for finer-grained distinctions.[8] ABSTAIN gives EDAs an option to signal missing coverage. Instead of simply collapsing this type into UNKNOWN, we would like to separate the cases which are *truly* underspecified from the cases for which the system (or single EDA) cannot make a decision. As also expressed by (Bergmair 2009), this can be helpful when considering ensembles of RTE systems. If necessary, this type can be extended at the cost of additional effort by the developer.

5.3 Efficiency and scalability

The use of textual entailment in practical NLP applications requires very fast or even real-time processing, which is non-trivial for many algorithms. This is especially true for tasks involving large amount of text, like the search task or the summarization task of RTE-6 and 7. We deal with efficiency in two ways. One is the clear separation and serialization of preprocessing, and the other is the support for parallelization.

The separation between LAP and EC helps to reduce the amount of duplicated preprocessing (linguistic annotation). With the CAS serialization format, it is possible to analyze the input data just once while running multiple ECs on the data.

Parallelization can also provide a way to speed up processing. The platform supports two different levels of parallelization. The first level is parallel processing of multiple H/T pairs, which can be assumed to be independent. This capability is integrated via an optional method in the EDA interface that accepts a set of H/T pairs and is implemented by EDAs that are able to concurrently process several H/T pairs.

The second level is parallelization of computations for a single H/T pair within the EDA. While calls to several components can be trivially parallelized with multiple threads, multiple calls to the same component present a bottleneck. This situation arises frequently in the case of knowledge components, e.g. when lexical similarities for a large number of potential word alignments are queried. We therefore require all

[8] EDAs make this choice by declaring their result type.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

components to declare whether they are thread-safe or not. If they are, the EDA can call them from different threads similar to the calling of different components; if they are not, the EDA can still choose to initialize several instances of the component, memory permitting.

6 EC components

Our current architecture defines four component types for EC, each specified as a Java interface: two algorithmic components (for scoring and annotation), and two knowledge components (for lexical knowledge and lexical–syntactic knowledge).

This set of component types attempts to strike a balance: it contains sufficiently few types to be general, and sufficiently many for these types to offer specific methods. We do not see the current inventory as complete, though: For example, the two knowledge component types allow for a range of queries, while the two 'algorithmic' components are more rudimentary with one central method each. This reflects the more mature state of standardization for knowledge resources compared with RTE algorithms. Thus, we expect that further algorithmic developments will motivate the introduction of novel component types.[9]

6.1 Scoring components

Scoring components accept an H/T pair as input, and return a vector of scores for that H/T pair. Technically, the central method of this type gets a Java object mirroring a CAS object (a JCas object) representing the complete H/T pair and returns a vector filled with arbitrary numeric values. Note that a JCas object holds prior annotations, and the scoring component is based on those annotations (for example, tree edit distance scoring relies on parse tree annotations etc). We use JCas as the exchange type to ensure that the scoring components are independent of the internal data structures used in the EDA under the assumption that the LAP-produced CAS objects are available in the EDA.

The semantics of the components' return values depend on the scoring component. In fact, the interface can be used to extract arbitrary features over the H/T pair, such as modality and polarity match. This functionality follows the criticism by MacCartney *et al.* (2006), who have argued that distance on its own is not a good correlate of entailment probability. Components that return such feature vectors can be combined with very simple classifier-based EDAs to build straightforward RTE systems.

Another use of this component is for distance calculation, where the intuition is that the probability of an entailment relation between T and H is related to the distance between them. The platform standardizes the distance calculation modules with the distance component, which is an extended version of the scoring

[9] Developers can already use components that do not correspond to any of the established component types at the expense of higher implementation effort.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

component that can accommodate normalized and unnormalized distance/similarity values along with the scoring vector.

6.2 Annotation components

The second type of component consists of components that add annotation to the text, the hypothesis, or the text/hypothesis pair. Again, the input type for this component is a JCas object to give the component access to the complete T/H pair. However, this time the component also returns the JCas object, extended with additional information.

An example of this type is a component that produces word or phrase alignments between the text and hypothesis. Such alignments form an important part of many RTE algorithms and can, or even should, be separated cleanly from the actual entailment decision (MacCartney *et al.* 2006). Other examples are annotations of modality or polarity information (Nairn, Condoravdi and Karttunen 2006).

Note that this definition introduces a certain overlap between the LAP and annotation components, since both can produce annotations. We feel that this situation creates a useful degree of choice: In any evaluation experiment that assesses the impact of some preprocessing step on RTE, there will be a 'static' annotation that remains constant during the experiment. This annotation is best produced and serialized in the LAP. In contrast, the 'dynamic' annotation that changes with the experimental manipulation can be produced on demand through an annotation component. Also note that cross-annotations like alignment annotations are not natural for LAP annotators, where an annotator is supposed to enrich one subject of analysis.

6.3 Lexical knowledge components

Lexical knowledge describes semantic relationships between words. This type of knowledge is useful for alignment-based approaches to RTE (to distinguish likely from unlikely alignments) as well as for transformation-based approaches (where it provides entailment rules at the lexical level).

We assume that this knowledge can be expressed as directed rules built from two *(word, POS)* pairs, a Left-Hand Side (LHS) and a Right-Hand Side (RHS). Two rules instances relevant for the example from Section 1 are $(Siberia, N) \rightarrow (Russia, N)$ and $(shooting_star, N) \rightarrow (meteorite, N)$. Its interface consists of three queries, which are as follows: (a) list all RHS for a given LHS; (b) list all LHS for a given RHS; and (c) check the existence of an entailment relation when both LHS and RHS are given. The meaning of a rule is usually that the LHS entails the RHS ($LHS \models RHS$).

This interface can wrap all major lexical knowledge sources currently used in RTE. This includes manually constructed ontologies (e.g. WordNet and VerbOcean (Chklovski and Pantel 2004)), encyclopedic resources (e.g. Wikipedia) as well as

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

automatically constructed thesauri and distributional similarity resources.[10] The interface does not impose any constraints on the storage of lexical knowledge. That is, for knowledge sources for which Application Programmer's Interfaces (APIs) are already available (such as WordNet), integration into our platform involves only translation of our interface methods into queries of that API.

The signature of the component is defined by a set of Java interfaces. All lexical knowledge components support the basic interface that provides the above-mentioned querying capability of 'entailing words' from LHS to RHS. In addition to that, a lexical knowledge component can choose to support a additional interface that provides resource-specific relations (for example, this additional interface enables WordNet-based implementation to support WordNet-specific queries, or VerbOcean-based implementation to support querying on relations of VerbOcean etc). Both interfaces return a list of lexical rules.

6.4 Syntactic knowledge components

Syntactic knowledge resources capture entailment relationships between syntactic and lexical–syntactic structures. Such rules involve variables that stand for arbitrary material. For example, the structure 'fall of X' entails 'X falls' (cf. example in Section 1); similarly, 'X sells Y to Z' entails 'Z buys Y from X'. Such relationships are captured insufficiently by purely lexical rules like $(sell, V) \rightarrow (buy, V)$.

Following Bar-Haim et al. (2007), we represent lexical–syntactic knowledge as directed rules. Each rule consists of two dependency trees, which may contain variables as nodes (such as X, Y, and Z in the examples) plus a mapping from the variables of the LHS tree to the variables of the RHS tree.[11]

An illustration of a lexical–syntactic rule is given in Figure 4. The left-hand side (shown at the top) is a tree-fragment which represents the syntactic structure of 'X sells Y to Z', while the right hand side (in the bottom) represents 'Z buys Y from X'. The variable names form the mapping, i.e. X in LHS is mapped to X in RHS, Y to Y and Z to Z.

The dependencies in the rules' dependency trees (the LHS and RHS) are Stanford Dependencies (de Marneffe and Manning 2008), which capture relations that are relevant for semantic tasks (e.g. subject, object, types of modifiers, types of new clauses, etc.). We use a small list of POS tags, derived from Penn Treebank POS list. This list was generated by mapping several tags that represent similar POS into a 'canonical' tag. For example, VBD, VBG, VBN, VBP, and VBZ are all mapped into 'Verb'. This process was similarly done for nouns, adjectives, etc.

A *rule application* is the operation of replacing the rule's LHS by its RHS in a given parse tree, resulting in a new parse tree. For example, when the rule of Figure 4 is applied on the (parse tree that corresponds to the) sentence 'It happened after

[10] The use of the latter type of knowledge requires some decision criterion that defines entailment in terms of similarity.

[11] Variables of the LHS may also map to null, when material of the LHS must be present but is deleted in the inference step.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

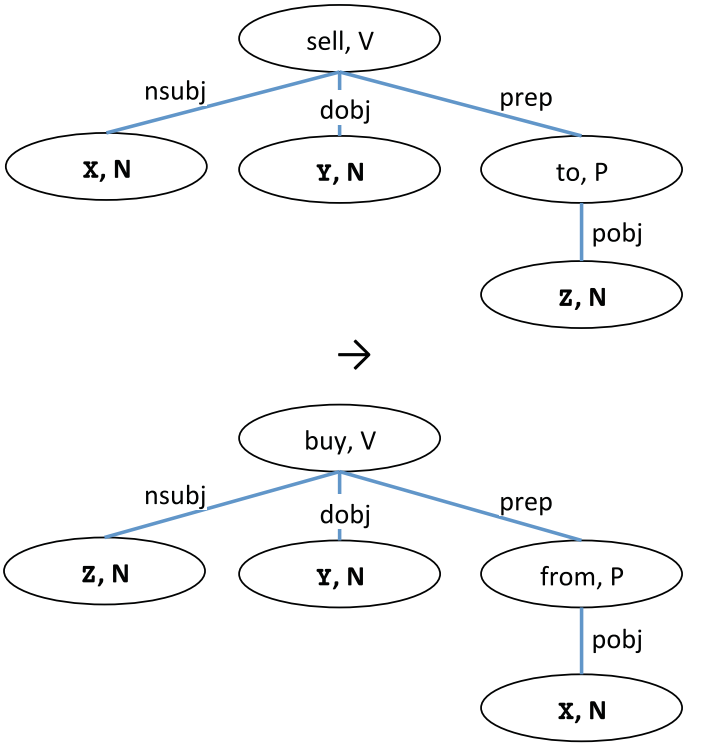


Fig. 4. (Colour online) Example of a lexical–syntactic rule: 'X sells Y to Z' → 'Z buys Y from X' (LHS above, RHS below). Mapping indicated by variable names.

Harry sold the car to John', a new sentence is generated: 'It happened after John bought the car from Harry'. This application is illustrated in Figure 5.

Syntactic rule formalization can represent knowledge about entailment relationships between predicate–argument structures, as well as about purely syntactic phenomena, like passive–active equivalence of the 'X Verb Y ↔ Y is Verb[ed] by X' type. Therefore, it can be used to represent a large variety of resources with (lexical–) syntactic knowledge, such as DIRT (Lin and Pantel 2002), REVERB (Berant 2012), FrameNet-based entailment rules (Ben Aharon, Szpektor and Dagan 2010), and many others. Transformation-based systems exploit such knowledge directly by using instantiated rules as proof steps. Alignment-based systems use it to identify semantic equivalence between subtrees of T and H, establishing phrase alignments.

Additional complexity compared with the lexical knowledge resource type arises from the more complex linguistic structures that make up the rules, namely parse trees with variables. These make a sophisticated query mechanism necessary. At the lexical level, we only have a linear number of words which we can query to retrieve all inference rules for a given input. At the syntactic level, however, the number of subtrees of a given parse tree is exponential in the input length. Our solution is to define a method in the component interface that works on the parse tree of the text

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

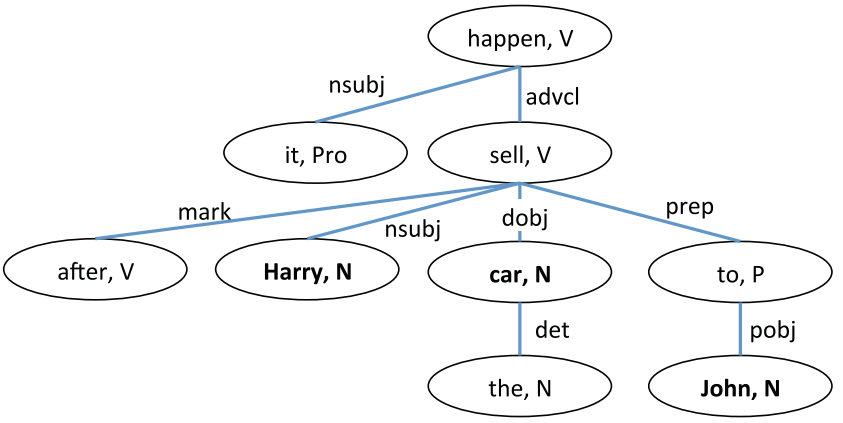


Fig. 5. (Colour online) Application of the lexical–syntactic rule from Figure 4 (above: before; below: after). Variable instantiation: X = Harry, Y = car, Z = John.

and returns all matching rules from the current knowledge resource. The efficient identification of these rules, which may differ based on details of the knowledge resource, is left to the resource developer.

7 System mapping

In this section, we show the practical usability of our architecture. We illustrate how three existing RTE systems can be mapped concretely onto our architecture: BIUTEE, a transformation-based system; EDITS, an alignment-based system; and TIE, a multi-stage classification system, which represent different approaches to RTE. We also discuss mapping for formal reasoning-based systems.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

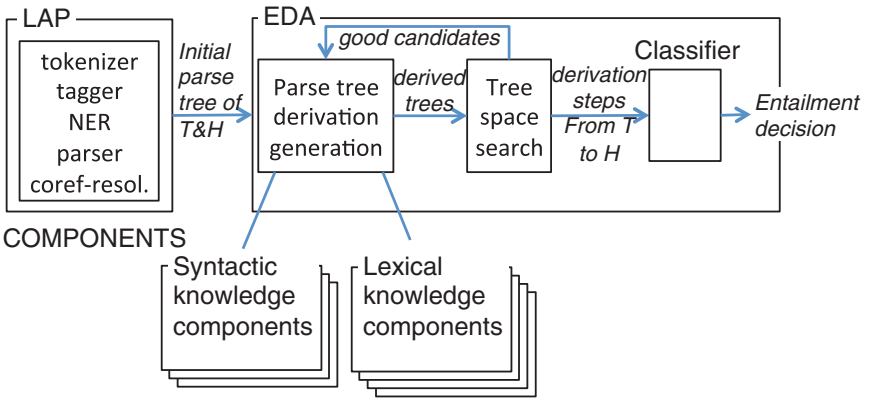


Fig. 6. (Colour online) System decomposition for the transformation-based BIUTEE system.

7.1 BIUTEE

BIUTEE (Stern and Dagan 2011) is a system for recognizing textual entailment based on the transformation-based approach outlined in Section 2.2 developed at Bar-Ilan University (BIU), Israel.[12] The system derives the Hypothesis (H) from the Text (T) with a sequence of rewrite steps. Since BIUTEE represents H and T as dependency trees, these rewrite steps can either involve just lexical knowledge (node rewrites) or lexico-syntactic knowledge (subtree rewrites). Additional operations (e.g. the insertion of syntactic structure or modification of POS types) ensure the existence for a derivation for any T-H pair. The likelihood that a derivation preserves entailment is finally calculated by a confidence model.

Figure 6 shows the decomposition of BIUTEE in terms of the EXCITEMENT architecture. The LAP provides all required annotations, including parse trees. The EDA encapsulates the 'top level' of the algorithm. It consists of three parts: a generation part, where rewrite steps are performed, i.e. entailed trees are generated; a search part, which selects good candidates among the generated derivations; and a classifier. The generation part relies on entailment rules which are stored modularly in (lexico-)syntactic knowledge components and lexical knowledge components. The search part evaluates the set of derived trees and selects good ones. This evaluation is performed based on two criteria: one is the reliability of derivation rules, and the other is the distance between the derived tree T' and H. The search ends when a derivation from T to H is found, and the final entailment decision is taken based on the confidence of the derivation.

Even though the BIUTEE EDA could be decomposed in even more parts (e.g. the search strategies or the classifier could be stored in separate components), this would incur a considerably higher re-engineering effort. The decomposition shown in Figure 6 represents a pragmatic trade-off that involves a feasible porting effort while meeting the main goals formulated in Section 5.1: First, we have identified three

[12] BIUTEE 2.4.1 is downloadable from `http://www.cs.biu.ac.il/nlp/downloads/biutee/`

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

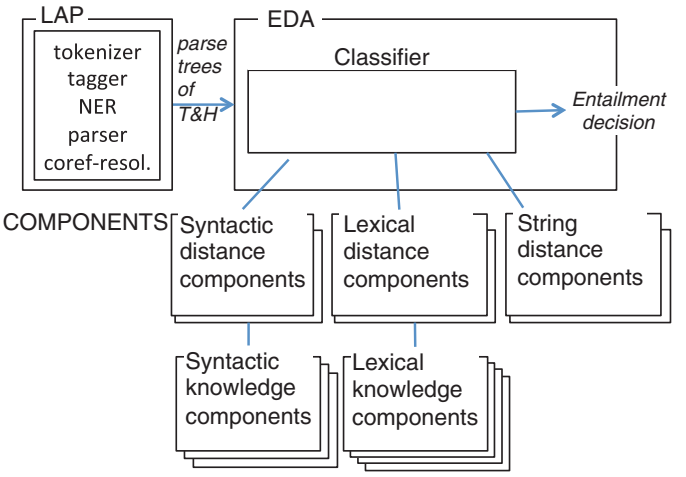


Fig. 7. (Colour online) System decomposition for the alignment-based EDITS system.

modular parts of the BIUTEE system that are stored separately and can be reused in other systems. Second, the BIUTEE EDA becomes largely parser-independent and even language-independent since all annotation style-dependent processing is encapsulated in components.

7.2 EDITS

EDITS (Negri *et al.* 2009) is an alignment-based RTE system developed at Fondazione Bruno Kessler (FBK), Italy.[13] It builds on the assumption that 'the probability of an entailment relation between a given T-H pair is inversely proportional to the (edit) distance between T and H' (Negri *et al.* 2009). Thus, entailment recognition in EDITS can be thought of as a sequence of two steps. First, the distance between H and T is determined. EDITS offers a range of distance measures, including string and tree edit distance. Second, these distance measures serve as features in a binary classifier which decides entailment.

Figure 7 shows how EDITS can be decomposed in terms of our architecture. The figure shows that EDITS can reuse the BIUTEE analysis pipeline as LAP. The EDA is minimal: it does not even contain RTE-specific functionality. Rather, it embodies a minimal supervised machine learning setup – it consists of a feature extraction step (which calls various components that measure the distance between T and H) and a binary classifier using these features. This EDA is clearly parser- and language-independent.[14]

The decomposition of EDITS is fairly uncontroversial. Nevertheless, EDITS illustrates a hierarchical structure that is typical for alignment-based approaches: The EDA does not use linguistic knowledge components directly, but uses distance

[13] EDITS 2.1 is downloadable from `http://edits.fbk.eu`

[14] Of course, the classifier must be retrained for a new parser, as would the BIUTEE confidence model.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

scoring components, which in turn take advantage of linguistic knowledge. For example, token edit distance can take lexical knowledge into account to reduce the distance for H-T pairs where an H word is entailed by a T word aligned to it. Similarly, syntactic tree edit distance can be enriched with paraphrase knowledge.

7.3 TIE

Textual Inference Engine (or TIE) is an RTE system that has been developed at the German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany[15] (Wang and Neumann 2008a; Wang and Zhang 2009; Wang 2011) and was not publicly released before. TIE was designed to cover a more general set of semantic relations (cf. Section 5.2). It decomposes 'classical' two-class RTE into a two-step task. The first step consists of a set of three independent classification problems corresponding to the presence of three semantic relations: TIE determines (a) whether T and H are related; (b) whether they are mutually consistent; and (c) whether there is an inherent directionality. The second step decides entailment based on the results of the first step, taking into account not only the classifier results but also their confidence values. This strategy has achieved decent performance in RTE challenges (e.g. second place in RTE-5). The multi-step classification setup of TIE is also used by a number of other RTE research prototypes.

The overall architecture of TIE is similar to EDITS, consisting of linguistic analysis components, feature extraction components, and classifier(s). During the linguistic analysis step, the system builds a joint syntactic and semantic dependency graph representation. In the feature extraction step, the system computes various features, including simple bag-of-words similarity scores as well as dependency path comparisons. In the final stage, these features are fed into three classifiers to measure relatedness, consistency, and directionality for each T-H pair, and then fed into the second classifier.

Figure 8 shows how the architecture of TIE can be mapped onto the EXCITEMENT architecture. Again, the EDA consists mainly of machine learning machinery, although this time it involves a total of four classifiers. Note that TIE can cover more classes than just entailment versus non-entailment and thus takes advantages of the flexible result type described in Section 5.2.

The linguistic knowledge is located in the various scoring components. Two main extensions over EDITS are features that target specifically matches and mismatches (a) between named entities and dates, and (b) on the level of semantic roles. These features are shown in the figures as the 'NE scoring component' and 'semantic scoring component', which can be reused by other systems.

The most challenging part of mapping TIE onto the EXCITEMENT architecture is the decomposition of the (two-stage) EDA. Ideally, each classifier would be encapsulated to a separate EDA in the spirit of a completely general multi-level classification setup. However, the three first-stage classifiers are not EDAs in their

[15] `http://www.dfki.de/lt/`

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

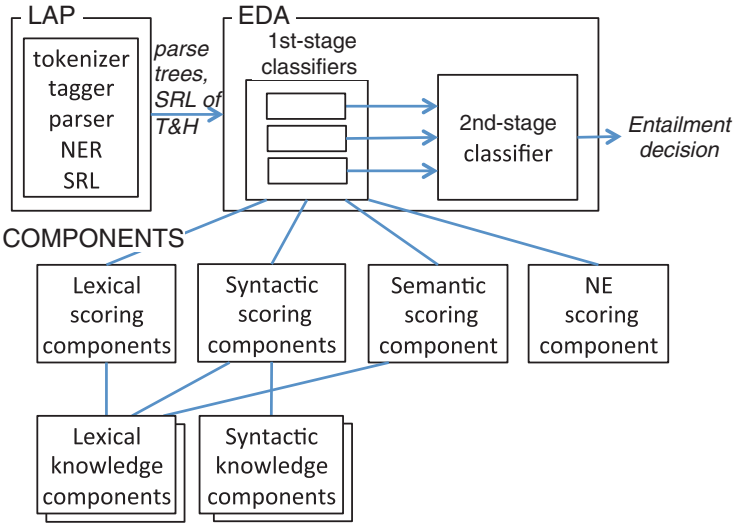


Fig. 8. (Colour online) System decomposition for the multi-stage classification system TIE.

own right since they do not provide complete entailment decisions. For this reason, and to keep the architecture concise, we combine both stages into one EDA.

7.4 Formal reasoning-based textual entailment

The three systems whose mapping we have discussed above represent the majority of current data-driven approaches. However, as mentioned in Section 2.2, some systems approach textual entailment from another angle by translating the input into some kind of formal representation amenable to formal reasoning methods (Bos and Markert 2005; Tatu and Moldovan 2005; Bobrow *et al.* 2007). Such a system is currently not included in the EXCITEMENT Open Platform (EOP), and we have not looked at individual systems of this type at the same level of detail as the three systems that we have included.

Nevertheless, we believe that the mapping of such systems onto the EXCITEMENT architecture can take place along roughly the same lines, as shown in Figure 9. The necessary extensions should all be possible within the EXCITEMENT architecture. The main reasoning part maps most straightforwardly onto an EDA as illustrated in the figure. A hybrid system that uses both reasoning and machine learning methods such as (Bos and Markert 2005) might however decide to use a statistical EDA and put the reasoner into an algorithmic component.

The most obvious defining property of other reasoning-based approaches is the complete transformation of the natural language input into a formal representation. This raises the question of where the transformation will take place: in the LAP or in the EDA. To promote reusability, we would encourage placing this translation in the LAP, which requires extending the UIMA CAS type system to accommodate the formal representation, be it some sort of description logics, predicate logics, or something else.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

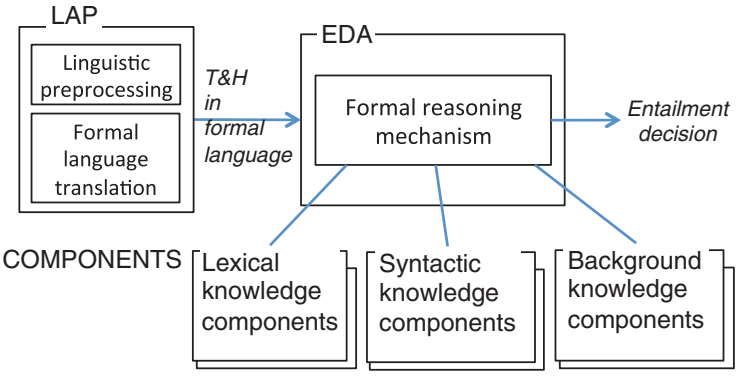


Fig. 9. (Colour online) System decomposition for a formal reasoning TE system.

The second major question is: what types of knowledge the reasoning process requires. The analysis by (Clark *et al.* 2007) indicates that the lexical and lexico-syntactic knowledge components that form part of EXCITEMENT can cover a significant portion (paraphrases, synonyms, etc.), assuming that a conversion process can be found that translates the output of these components into a formal representation (e.g. a pair of synonymous words into a universal meaning postulate). Another knowledge type that Clark *et al.* discuss is generic world knowledge such as the so-called *core theories* that express 'general, fundamental knowledge (e.g. space, time, plans, goals)'. This type of knowledge raises two challenges: (a) we assume that the existing query interfaces of the lexical and lexical–syntactic components are insufficient; and (b) the data must be exchanged at the level of the formal representation language (or its UIMA type). Therefore, we believe that general world knowledge is best represented as a novel knowledge component type.

8 The EXCITEMENT platform

To prove the practical usefulness of the EXCITEMENT architecture, we have developed the EXCITEMENT platform, an open source suite of RTE modules based on the architecture. Based on the mappings described in the previous section, the initial version of the platform includes the functionality of three existing systems: BIUTEE, EDITS, and TIE, covering a substantial part of the space of RTE algorithms.[16]

We hope that it can make a contribution similar to MOSES (Koehn *et al.* 2007) in the machine translation field by making RTE more easily usable in NLP applications, enabling better evaluation, and encouraging sustainable system and resource development within an open-source community. In machine translation, the development of the MOSES platform has led to a research boost in the community.

[16] The platform is released under the GPL v3.0 license, and its source code and all resources are freely available. See `http://hltfbk.github.io/Excitement-Open-Platform/`

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

We believe that the services that EXCITEMENT provides (enabling code exchange, sharing knowledge, and systematizing evaluation) can do the same for semantic processing.

8.1 Development and scope of the platform

The first release of the EOP comprises a suite of EDAs, EC components, and LAP components for three languages (English, German, and Italian). The following list gives the most important components.

LAP: The platform includes one LAP for each language that covers tokenization, POS tagging, lemmatization, and dependency parsing. The English LAP includes a pipeline based on the Stanford NLP tools (Toutanova and Manning 2000; Finkel, Grenager and Manning 2005), an easy-first parser (Goldberg and Elhadad 2010), and the Ark coreference resolver (Haghighi and Klein 2009). Another pipeline that is available for English and German is based on TreeTagger (Schmid 1994) and MaltParser (Nivre and Nilsson 2005). The Italian LAP is based on the TextPro package (Pianta, Girardi and Zanoli 2008).

EDAs: The platform covers the three systems whose mappings were outlined in the previous section: a transformation-based EDA (BIUTEE), a single-level classification EDA (EDITS), and a multi-level classification EDA (TIE). These EDAs are language-independent, providing a set of generic basic entailment algorithms.

Algorithmic components: The platform comes with a set of EC components. It includes distance components for various linguistic levels, components that extract match and mismatch features, and annotator components, e.g. for predicate truth (Lotan, Stern and Dagan 2013). Even though these components were generally implemented for one specific system, we have reviewed the code so that as many components as possible are language-independent (e.g. edit distance), and applicable to multiple EDAs.

Knowledge components: The platform includes about two dozen lexical and syntactic resources for three languages. Much of these resources are inherited from migrating RTE systems, but now use the same set of interfaces that makes them accessible in a uniform fashion. Since there is still a clear predominance of English resources, the platform includes lexical and syntactic knowledge mining tools to bootstrap resources for other languages or specific domains from corpora.

Data sets: Finally, the platform includes a number of RTE data sets for all three languages. For English, it covers the well-known RTE workshop test data sets. RTE-3 is also present in translated versions for both Italian and German. Furthermore, we include two task-specific datasets, such as the monolingual English data set prepared for the SemEval-2012 Content Synchronization task (Negri *et al.* 2012), and a German Web search task dataset (Zeller and Padó 2013).

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

This first release of the platform is able to demonstrate our original goals:

- The CAS type systems and the component interfaces form a fundamental layer of the platform that is shared by all three languages and all components. It can be reused by any future components and languages.
- For each language, we have prepared a single linguistic annotation pipeline. This LAP is reused by all EDAs for that language. For example, all three EDAs for English (EDITS, TIE, and BIUTEE) now share one LAP, whereas each had its own preprocessing toolchain. This separation of LAP from EC worked well with the proposed CAS types. We did not encounter major problems in converting the preprocessing to the UIMA CAS framework.
- The EDAs are also usable for multiple languages now. For example, we are now able to apply the EDITS EDA to German text, which was not possible earlier.
- Finally, the generic access methods to knowledge resources provided through the component interfaces mean knowledge resources can now be used by any EDAs. This enables EDAs to use resources that were not used before with virtually no efforts. For example, the adaptation of EDITS to access WordNet through our lexical knowledge interface automatically means that it can use GermaNet when applied to German. With the platform, this would have required dedicated development effort.

8.2 Pilot evaluation

The main aim of the first release is to reduce the future effort of building and improving systems for textual entailment. Thus, the first release is conservative regarding quality and efficiency: The EDA algorithms remain the same as before, and no explicit improvements were made regarding scalability.

Regarding quality, the migrated EDAs should therefore mirror the performance of individual systems. As for efficiency, the integration into the EOP involved some re-implementation as well as some glue code, and the consequences of these changes are not *a priori* clear.

This section compares three English EDAs (EDITS, BIUTEE, and TIE), as integrated in the platform, with standalone versions of the systems. We evaluate their accuracy and runtime on the test section of the English RTE-3 dataset consisting of 800 T-H pairs. We do not use published numbers of the standalone systems, but configured them so as to correspond as closely as possible to the EDA versions.

Accuracy. The results in Table 1 show that accuracy differences exist between the standalone and platform versions, but they are fairly small. There is almost no change for EDITS (+0.1%). There are small drops in performance in the EDA version of BIUTEE (–0.4%) and TIE (–0.8%). These are presumably due to the refactoring steps that took place during migration and inevitably involved the removal of some representation- and language-specific processing. However, when we assessed the significance of these differences with a bootstrap re-sampling test (Efron and Tibshirani 1993), we found that none of the differences were significant at

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

Table 1. *Comparison of accuracy (%) between standalone systems and EXCITEMENT platform versions using specified resources on the English RTE-3 dataset. All differences are not significant at $p = 0.05$*

System	Lexical level	Syntactic level	Standalone	In EOP
EDITS 2.1	VerbOcean, WordNet	Not used	63.0	63.1
BIUTEE	WordNet, CatVar, VerbOcean	Normalization and paraphrasing rules	65.4	65.0
TIE	WordNet	Syntactic features	66.1	65.3

Table 2. *Comparison of runtimes (in ms; excluding startup time) between standalone systems and EXCITEMENT platform versions using specified resources on the English RTE-3 dataset*

System	Lexical level	Syntactic level	Standalone	In EOP
EDITS 2.1	VerbOcean, WordNet	Not used	160	210
BIUTEE	WordNet, CatVar, VerbOcean	Normalization and paraphrasing rules	357.000	492.000
TIE	WordNet	Syntactic features	3.500	3.048

$p = 0.05$. We see this as a promising result: We were able to generalize and migrate our systems into the EXCITEMENT platform without a statistically significant loss in performance, and we hope that the same could be done for other systems.

Efficiency. Table 2 shows the runtimes of the three EDAs, excluding startup time (i.e. the one-time overhead necessary to read resources and input files). The runtimes show a much less uniform picture than the accuracies. For two systems, runtimes have increased (BIUTEE: +38%, EDITS: +31%), while one system has become faster (TIE: –13%). In the case of BIUTEE, where the absolute increase is by far the highest, the reason is that standalone BIUTEE used a POS tag set different from the EOP standard. The developers decided to convert tags dynamically. The resulting performance hit could be avoided by converting all resources to the EOP standard tagsets. More generally, the runtimes are dominated by the overall differences among EDAs, which amount to orders of magnitude. They reflect differences between the processing mechanisms of the EDAs – string edit distance in the case of EDITS versus large-scale lexical–syntactic processing in the case of BIUTEE. In comparison, the impact of importing the EDAs into the EOP is relatively small.

8.3 Platform usage and use cases

User interface. The components provided by our suite of components are combined into actual 'inference engines' through configuration files which contain information

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

to initialize and instantiate a complete inference engine. The user interacts with the platform in the following three steps: (1) write or select a configuration and instantiate the engine; (2) obtain a model by training on new data or loading a previously trained model; and (3) process textual entailment instances.

The platform ships with a set of predefined configuration files covering the shipped components. Each configuration is accompanied by a documentation that describes its characteristics, including efficiency and accuracy. In addition, each separate EDA and component come with documentation on how to configure their functionality.

Use cases. We have two main use cases for the platform in mind. The first one is as a general, self-contained textual entailment engine. This usage is suitable for industrial users, or research users who want to use an already established inference engine as a part of their language processing systems. In this use case, the main benefit of the EXCITEMENT platform is that integration of one platform allows the end user to compare the performance of various RTE systems and configurations in their application. For this use case, the platform can be used as an Apache Maven[17] Java module. This enables off-the-shelf usage without knowledge of project internal dependencies.

The second use case is a platform for research in inference. For this group, the most important aspects of the platform are the ability to extend and modify the code base. The EXCITEMENT platform development adopted several policies and principles to make this as easy as possible. Notably, all EXCITEMENT component types introduced in Section 6 support the common behaviors defined in the component type and expose the configurable internal values directly in their class constructor, among other things. Also, the components and their interfaces are documented extensively so that they can be exchanged modularly. Finally, the platform comes with a set of utilities that support development, such as knowledge extraction facilities.

Expectations. We cannot guarantee that the EXCITEMENT platform will find widespread adoption in the textual entailment field. Nevertheless, we believe that it has a good chance of playing a significant role. This judgment is based on the 'investment versus payback' trade-off: researchers will adopt the tools that allow them to reach their research goals with the minimal effort necessary. In this respect, the EXCITEMENT platform is an attractive offer: a clean code base for multilingual research that covers several major approaches, can be easily extended and modified, and is accompanied by a layer of research utilities. Naturally, this trade-off differs for each researcher. EXCITEMENT should be interesting for groups who want to enter the field of textual entailment and for whom it can radically lower the initial effort necessary (e.g. developing textual entailment systems for new languages). Meanwhile, established groups with self-developed textual entailment systems naturally have a much smaller incentive to migrate. However, such groups might still be interested

[17] `http://maven.apache.org`

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

in contributing smaller components (e.g. knowledge resources) from their systems to EXCITEMENT to make them available to a broad community. Conversely, if researchers find it helpful to integrate EXCITEMENT components into their own systems, the architecture would still provide a benefit to the community as a standard for exchangeable components, rather than as a complete platform.

9 Related work

The focus of our work is to provide a general, open architecture for RTE that supports the implementation within a common platform of a wide range of approaches developed (Androutsopoulos and Malakasiotis 2010; Dagan, Roth and Zanzotto 2012). We believe that our considerations are also relevant for the recently proposed task of determining Short Text Textual Similarity (Agirre *et al.* 2012). This task is similar to, but distinct from, textual entailment in that it quantifies the degree of 'graded semantic equivalence' between sentence pairs.

While we are the first to propose a general, strategy-independent decomposition of entailment computation, there is a considerable body of work on standardizing preprocessing (LAP in our model), although generally with a different focus. General Architecture for Text Engineering (GATE) (Cunningham 2002; Cunningham *et al.* 2011) is a text processing platform that decouples workflows into components that communicate via APIs, but concentrates on the annotation and processing of complete corpora with the primary goal of information extraction and indexing. The Whiteboard and the Heart of Gold architectures (Crysmann *et al.* 2002; Callmeier *et al.* 2004; Schäfer 2007) were specifically developed to combine diverse (e.g. shallow and deep) language processing components with a focus on the integration of results from different components and the resolution of potential conflicts. This is a complementary goal to our goal; conflict resolution can be integrated into our architecture via an additional component. (Hinrichs, Zastrow and Hinrichs 2010) present an architecture for multilingual language analysis specialized to web services without ties to a specific application. We do not currently touch the topic of distributed processing, but components are free to distribute computation internally.

The work that is most close to our work in spirit is (Clarke *et al.* 2012), whose goal is the development of a flexible, distributed, and easy-to-use processing pipeline. Given a different set of priorities than ours – inclusion of components in different programming languages, distribution across servers, and minimization of user effort – they reject UIMA in favor of a newly developed architecture based on the Apache thrift library,[18] while we feel that RTE can profit more from UIMA's greater maturity and strongly typed representation.

10 Conclusions and outlook

In this paper, we have described EXCITEMENT, a general, open architecture for textual entailment. Our goal is to overcome the fragmentation in the textual

[18] `http://thrift.apache.org`

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

entailment community that results from the fact that virtually all existing RTE systems are prototypes specialized on one particular RTE strategy and are tied to particular preprocessing toolchains.

Our first contribution, the specification of this platform, is located primarily on the level of sustainable software engineering for NLP. Its core is the decomposition of both linguistic analysis pipeline and entailment computation into a set of components with well-defined interfaces. We adopt UIMA for the linguistic analysis side, and have proposed a novel modularization of entailment computation into top-level EDAs and four major types of components. Our primary desiderata were *generality* with regard to RTE strategies, languages, and linguistic analysis pipelines, *openness* for future extensions, and *reusability* of code and resources.[19]

We found that the twin goals of openness and reusability often conflict in practice, since openness is best served by underspecifying interfaces. Underspecification, however, also tends to result in weak standardization, and thus problematic reusability. We have generally tried to identify reasonably detailed interfaces while allowing developers' future extensions at additional implementation cost (cf. Result Types, Section 5.2, and Component Types, Section 6).

Our second contribution is the open source platform for textual entailment on the basis of our specification. By including code from two of the most mature, available RTE systems, EDITS and BIUTEE, the EOP forms an RTE suite that provides EDAs for two main RTE strategies (alignment-based and transformation-based), as well as providing a range of entailment knowledge resources. It supports RTE for English, German, and Italian out of the box. The EOP, which is available as an open source software under the GNU General Public License, should substantially lower the barrier for developing new ideas in the field of textual entailment, applying RTE to new languages, and curating existing resources. We intend to build a community of developers around the EOP to ensure its long-term viability.

Acknowledgments

This work was supported by the EC-funded project EXCITEMENT (FP7 ICT-287923). We would like to thank Alberto Lavelli for his comments and suggestions.

References

Agirre, E., Cer, D., Diab, M., and Gonzalez-Agirre, A. 2012. SemEval-2012 task 6: a pilot on semantic textual similarity. In *Proceedings of the International Workshop on Semantic Evaluation*, Montréal, Canada, pp. 385–93.

Androutsopoulos, I., and Malakasiotis, P. 2010. A survey of paraphrasing and textual entailment methods. *Journal of Artificial Intelligence Research* **38**: 135–87.

[19] The full specification, which is available from `http://www.excitement-project.eu/index.php/results` covers further aspects (shared training facilities, handling of multi-Text or multi-Hypothesis setups in 'Search' tasks, configuration, etc.).

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

Baker, C. F., Fillmore, C. J., and Lowe, J. B. 1998. The Berkeley FrameNet project. In *Proceedings of the Joint International Conference on Computational Linguistics and Annual Meeting of the Association for Computational Linguistics*, Montréal, QC, pp. 86–90.

Bar-Haim, R., Dagan, I., Greental, I., and Shnarch, E. 2007. Semantic inference at the lexical-syntactic level. In *Proceedings of the Annual Meeting of the American Association for Artificial Intelligence*, Vancouver, BC, pp. 871–6.

Bar-Haim, R., Szpektor, I., and Glickman, O. 2005. Definition and analysis of intermediate entailment levels. In *Proceedings of the ACL-PASCAL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*, Ann Arbor, MI, pp. 55–60.

Ben Aharon, R., Szpektor, I., and Dagan, I. 2010. Generating entailment rules from FrameNet. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Uppsala, Sweden, pp. 241–6.

Bentivogli, L., Magnini, B., Dagan, I., Trang Dang, H., and Giampiccolo, D. 2009. The fifth PASCAL recognising textual entailment challenge. In *Proceedings of the TAC Workshop on Textual Entailment*, Gaithersburg, MD.

Berant, J. 2012. *Global Learning of Textual Entailment Graphs*. PhD thesis, Tel Aviv University, Israel.

Berant, J., Dagan, I., and Goldberger, J. 2012. Learning entailment relations by global graph structure optimization. *Computational Linguistics* **38**(1): 73–111.

Berger, A., Caruana, R., Cohn, D., Freitag, D., and Mittal, V. 2000. Bridging the lexical chasm: statistical approaches to answer-finding. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Athens, Greece, pp. 192–9.

Bergmair, R. 2009. A proposal on evaluation measures for RTE. In *Proceedings of the Workshop on Applied Textual Inference*, Singapore, pp. 10–17.

Bobrow, D., Condoravdi, C., Crouch, R., Paiva, V. De, Karttunen, L., King, T., Nairn, R., Price, C., and Zaenen, A. 2007. Precision-focused textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, Prague, Czech Republic, pp. 16–21.

Bos, J., and Markert, K. 2005. Recognising textual entailment with logical inference. In *Proceedings of the Joint Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, Vancouver, BC, pp. 628–635.

Bos, J., and Markert, K. 2006. When logical inference helps determining textual entailment (and when it doesn't). In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, Venice, Italy, pp. 98–103.

Cabrio, E., and Magnini, B. 2011. Towards component-based textual entailment. In *Proceedings of the International Conference on Computational Semantics*, Oxford, UK, pp. 320–4.

Callmeier, U., Eisele, A., Schäfer, U., and Siegel, M. 2004. The DeepThought core architecture framework. In *Proceedings of the International Conference on Language Resources and Evaluation*, Lisbon, Portugal, pp. 1205–8.

Castillo, J. 2010. A machine learning approach for recognizing textual entailment in Spanish. In *Proceedings of the NAACL/HLT Young Investigators Workshop on Computational Approaches to Languages of the Americas*, Los Angeles, CA, pp. 62–7.

Chklovski, T., and Pantel, P. 2004. VerbOcean: mining the web for fine-grained semantic verb relations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain, pp. 33–40.

Clark, P., Harrison, P., Thompson, J., Murray, W., Hobbs, J., and Fellbaum, C. 2007. On the role of lexical and world knowledge in RTE3. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, Prague, pp. 54–9.

Clarke, J., Srikumar, V., Sammons, M., and Roth, D. 2012. An NLP curator (or: How I learned to stop worrying and love NLP pipelines). In *Proceedings of the International Conference on Language Resources and Evaluation*, Istanbul, Turkey, pp. 3276–83.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

Cohen, K. Bretonnel, and Carpenter, B. (eds.) 2008. *Proceedings of the ACL Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*. Stroudsburg, PA: Association for Computational Linguistics.

Crysmann, B., Frank, A., Kiefer, B., Müller, S., Neumann, G., Piskorski, J., Schäfer, U., Siegel, M., Uszkoreit, H., Xu, F., Becker, M., and Krieger, H.-U. 2002. An integrated architecture for shallow and deep processing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA, pp. 441–8.

Cunningham, H. 2002. GATE, a general architecture for text engineering. *Computers and the Humanities* **36**(2): 223–54.

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Aswani, N., Roberts, I., Gorrell, G., Funk, A., Roberts, A., Damljanovic, D., Heitz, T., Greenwood, M. A., Saggion, H., Petrak, J., Li, Y., and Peters, W. 2011. *Text Processing with GATE (Version 6)*. Sheffield, UK: University of Sheffield.

Curran, J. 2003. Blueprint for a high-performance NLP infrastructure. In *Proceedings of the HLT-NAACL Workshop on Software Engineering and Architecture of Language Technology Systems*, Berkeley, CA, pp. 39–44.

Dagan, I., Glickman, O., and Magnini, B. 2005. The PASCAL recognising textual entailment challenge. In *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*, Southampton, UK, pp. 177–90.

Dagan, I., Roth, D., and Zanzotto, F. M. 2012. *Recognizing Textual Entailment: Models and Applications*. Synthesis Lectures on Human Language Technologies number 17. New York: Morgan & Claypool.

de Marneffe, M.-C., and Manning, C. D. 2008. The Stanford typed dependencies representation. In *Proceedings of the COLING Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, Manchester, UK, pp. 1–8.

de Marneffe, M.-C., Rafferty, A. N., and Manning, C. D. 2008. Finding contradictions in text. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Columbus, OH, pp. 1039–47.

Efron, B., and Tibshirani, R. J. 1993. *An Introduction to the Bootstrap*. New York: Chapman and Hall.

Fellbaum, C. (ed.) 1998. *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press.

Ferrucci, D., and Lally, A. 2004. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering* **10**(3–4): 327–48.

Finkel, J. R., Grenager, T., and Manning, C. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*, Ann Arbor, MI, pp. 363–70.

Goldberg, Y., and Elhadad, M. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Proceedings of the Annual Conference of the North American Chapter of the ACL*, Los Angeles, CA, pp. 742–50.

Gurevych, I., Mühlhäuser, M., Müller, C., Steimle, J., Weimer, M., and Zesch, T. 2007. Darmstadt knowledge processing repository based on UIMA. In *Proceedings of the First Workshop on Unstructured Information Management Architecture at the Conference of the Society for Computational Linguistics and Language Technology*, Tübingen, Germany.

Haghighi, A., and Klein, D. 2009. Simple coreference resolution with rich syntactic and semantic features. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Singapore, pp. 1152–61.

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martì, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štepánek, J., Stranák, P., Surdeanu, M., Xue, N., and Zhang, Y. 2009. The CoNLL-2009 shared task: syntactic and semantic dependencies in multiple languages. In *Proceedings of the Conference of Natural Language Learning*, Boulder, CO, pp. 1–18.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

Harabagiu, S., and Hickl, A. 2006. Methods for using textual entailment in open-domain question answering. In *Proceedings of the International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia, pp. 905–12.

Harabagiu, S., Hickl, A., and Lacatusu, F. 2007. Satisfying information needs with multi-document summaries. *Information Processing and Management* **43**(6):1619–42.

Harmeling, S. 2009. Inferring textual entailment with a probabilistically sound calculus. *Journal of Natural Language Engineering* **15**(4): 459–77.

Hinrichs, M., Zastrow, T., and Hinrichs, E. 2010. WebLicht: web-based LRT services in a distributed eScience infrastructure. In *Proceedings of the International Conference on Language Resources and Evaluation*, Valletta, Malta, pp. 489–93.

Ide, N., and Suderman, K. 2007. GrAF: a graph-based format for linguistic annotations. In *Proceedings of the ACL Linguistic Annotation Workshop*, Prague, Czech Republic, pp. 1–8.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. 2007. Moses: open source toolkit for statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Prague, Czech Republic, pp. 177–80.

Lin, D., and Pantel, P. 2002. Discovery of inference rules for question answering. *Journal of Natural Language Engineering* **7**(4): 343–60.

Lotan, A., Stern, A., and Dagan, I. 2013. TruthTeller: annotating predicate truth. In *Proceedings of the Annual Meeting of the North American Chapter of the ACL*, Atlanta, GA, pp. 752–7.

MacCartney, B., Grenager, T., de Marneffe, M.-C., Cer, D., and Manning, C. D. 2006. Learning to recognize features of valid textual entailments. In *Proceedings of the Human Language Technology Conference of the NAACL*, New York City, NY, pp. 41–8.

MacCartney, B., and Manning, C. D. 2007, June. Natural logic for textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, Prague, Czech Republic, pp. 193–200.

Mehdad, Y., Negri, M., and Federico, M. 2010. Towards cross-lingual textual entailment. In *Proceedings of the Annual Conference of the North American Chapter of the ACL*, Los Angeles, CA, pp. 321–4.

Mehdad, Y., Negri, M., and Federico, M. 2011. Using bilingual parallel corpora for cross-lingual textual entailment. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Portland, OR, pp. 1336–45.

Meyers, A. (edr.) 2005. *Proceedings of the Workshop on Frontiers in Corpus Annotations II: Pie in the Sky*. Stroudsburg, PA: Association for Computational Linguistics.

Mirkin, S., Dagan, I., and Padó, S. 2010. Assessing the role of discourse references in entailment inference. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Uppsala, Sweden, pp. 1209–19.

Mirkin, S., Dagan, I., and Shnarch, E. 2009. Evaluating the inferential utility of lexical-semantic resources. In *Proceedings of the Conference of the European Chapter of the ACL*, Athens, Greece, pp. 558–66.

Monz, C., and de Rijke, M. 2001. Light-weight entailment checking for computational semantics. In *Proceedings of the Conference on Inference in Computational Semantics*, Siena, Italy, pp. 59–72.

Nairn, R., Condoravdi, C., and Karttunen, L. 2006. Computing relative polarity for textual inference. In *Proceedings of the Conference on Inference in Computational Semantics*, Buxton, UK, pp. 67–76.

Negri, M., Bentivogli, L., Mehdad, Y., Giampiccolo, D., and Marchetti, A. 2011. Divide and conquer: crowdsourcing the creation of cross-lingual textual entailment corpora.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Edinburgh, Scotland, UK, pp. 670–9.

Negri, M., Kouylekov, M., Magnini, B., Mehdad, Y., and Cabrio, E. 2009. Towards extensible textual entailment engines: the EDITS package. In *Proceeding of the Conference of the Italian Association for Artificial Intelligence*, Reggio Emilia, Italy, pp. 314–23.

Negri, M., Marchetti, A., Mehdad, Y., Bentivogli, L., and Giampiccolo, D. 2012. SemEval-2012 task 8: cross-lingual textual entailment for content synchronization. In *The Joint Conference on Lexical and Computational Semantics and Sixth International Workshop on Semantic Evaluation*, Montréal, Canada, pp. 399–407.

Nielsen, R. D., Ward, W., and Martin, J. H. 2009. Recognizing entailment in intelligent tutoring systems. *Journal of Natural Language Engineering* **15**(4): 479–501.

Nivre, J., and Nilsson, J. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, Ann Arbor, MI, pp. 99–106.

Padó, S., Cer, D., Galley, M., Jurafsky, D., and Manning, C. D. 2009. Measuring machine translation quality as semantic equivalence: a metric based on entailment features. *Machine Translation*, **23**(2–3): 181–93.

Patrick, J., and Cunningham, H. (eds.) 2003. *Proceedings of the HLT-NAACL Workshop on Software Engineering and Architecture of Language Technology Systems*. Stroudsburg, PA: Association for Computational Linguistics.

Peñas, A., Rodrigo, Á., Sama, V., and Verdejo, F. 2008. Testing the reasoning for question answering validation. *Journal of Logic and Computation* **18**: 459–474.

Pianta, E., Girardi, C., and Zanoli, R. 2008. The TextPro tool suite. In *Proceedings of the International Conference on Language Resources and Evaluation*, Marrakech, Morocco, pp. 2603-7.

Romano, L., Kouylekov, M., Szpektor, I., Dagan, I., and Lavelli, A. 2006. Investigating a generic paraphrase-based approach for relation extraction. In *Proceedings of the Conference of the European Chapter of the ACL*, Trento, Italy, pp. 401–8.

Sammons, M., Vydiswaran, V., and Roth, D. 2012. Recognizing textual entailment. In Daniel M. Bikel and I. Zitouni (eds.), *Multilingual Natural Language Applications: From Theory to Practice*. Englewood Cliffs, NJ: Prentice Hall, pp. 209–258.

Schäfer, U. 2007. *Integrating Deep and Shallow Natural Language Processing Components – Representations and Hybrid Architectures*. PhD thesis, Saarland University, Saarbrücken, Germany.

Schmid, H. 1994. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing*, Manchester, UK, pp. 44–9.

Stern, A., and Dagan, I. 2011. A confidence model for syntactically motivated entailment proofs. In *Proceedings of the Conference on Recent Advances in Natural Language Processing*, Borovets, Bulgaria, pp. 455–62.

Tatu, M., and Moldovan, D. 2005. A semantic approach to recognizing textual entailment. In *Proceedings of the Joint Conference on Human Language Technology and Conference on Empirical Methods in Natural Language Processing*, Vancouver, BC, pp. 371–8.

Toutanova, K., and Manning, C. D. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the Conference on Empirical methods in Natural Language Processing*, Hong Kong, pp. 63–70.

Wang, R. 2011. *Intrinsic and Extrinsic Approaches to Recognizing Textual Entailment*. PhD. thesis, Saarland University, Saarbrücken, Germany.

Wang, R., and Neumann, G. 2008a. An accuracy-oriented divide-and-conquer strategy for recognizing textual entailment. In *Proceedings of the TAC Workshop on Textual Entailment*, Gaithersburg, MD.

Wang, R., and Neumann, G. 2008b. Information synthesis for answer validation. In *Working Notes for the CLEF 2008 Workshop*, Aarhus, Denmark, pp. 742–5.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press

Wang, R., and Zhang, Y. 2009. Recognizing textual relatedness with predicate-argument structures. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, Singapore, pp. 784–92.

Zanzotto, F. M., Pennacchiotti, M., and Moschitti, A. 2009. A machine learning approach to textual entailment recognition. *Journal of Natural Language Engineering* **15**(4): 551–82.

Zeller, B. D., and Padó, S. 2013. A search task dataset for German textual entailment. In *Proceedings of the International Conference on Computational Semantics*, Potsdam, Germany, pp. 288–99.

https://doi.org/10.1017/S1351324913000351 Published online by Cambridge University Press