

# An Inference Model for Semantic Entailment in Natural Language

Rodrigo de Salvo Braz, Roxana Girju, Vasin Punyakanok  
Dan Roth, Mark Sammons

Department of Computer Science  
University of Illinois at Urbana-Champaign, Urbana IL 61801, USA

**Abstract.** *Semantic entailment* is the problem of determining if the meaning of a given sentence entails that of another. We present a principled approach to semantic entailment that builds on inducing re-representations of text snippets into a hierarchical knowledge representation along with an optimization-based inferential mechanism that makes use of it to prove semantic entailment. This paper provides details and analysis of the knowledge representation and knowledge resources issues encountered. We analyze our system’s behavior on the PASCAL text collection<sup>1</sup> and the PARC collection of question-answer pairs<sup>2</sup>. This is used to motivate and explain some of the design decisions in our hierarchical knowledge representation, that is centered around a predicate-argument type abstract representation of text.

## 1 Introduction

*Semantic entailment* is the task of determining, for example, that the sentence: “Walmart defended itself in court today against claims that its female employees were kept out of jobs in management because they are women” entails that “Walmart was sued for sexual discrimination”.

Determining whether the meaning of a given text snippet *entails* that of another or whether they have the same meaning is a fundamental problem in natural language understanding that requires the ability to abstract over the inherent syntactic and semantic variability in natural language [1]. This challenge is at the heart of many high level natural language processing tasks including Question Answering, Information Retrieval and Extraction, Machine Translation, and others that attempt to reason about and capture the meaning of linguistic expressions.

Research in natural language processing in the last few years has concentrated on developing resources that provide multiple levels of syntactic and semantic

<sup>1</sup> <http://www.pascal-network.org/Challenges/RTE/>

<sup>2</sup> The data is available by following the data link from <http://l2r.cs.uiuc.edu/~cogcomp/data.php>.

analysis, resolve context sensitive ambiguities, and identify relational structures and abstractions (from syntactic categories like POS tags to semantic categories such as named entities).

However, we believe that in order to move beyond this level and support fundamental tasks such as inferring semantic entailment between two text snippets, there needs to be a unified knowledge representation of the text that **(1)** provides a hierarchical encoding of the structural, relational and semantic properties of the given text, **(2)** is integrated with learning mechanisms that can be used to induce such information from raw text, and **(3)** is equipped with an inferential mechanism that can be used to support inferences over such representations.

Relying on general purpose knowledge representations — FOL, probabilistic or hybrids — along with their corresponding general purpose inference algorithms does not resolve the key issues of *what to represent* and *how to derive a sufficiently abstract representation* and, in addition, may lead to brittleness and complexity problems. On the other hand, relying only on somewhat immediate correspondences between question and candidate answers, such as shared words or shared named entities, has strong limitations. We avoid some of these problems by *inducing* an abstract representation of the text which does not attempt to represent the full meaning of text, but provides what could be seen as a *shallow semantic* representation; yet, it is significantly more expressive than extraction of straightforward phrase-level characteristics. We induce this into a description-logic based language that is more restricted than FOL yet is expressive enough to allow both easy incorporation of language and domain knowledge resources and strong inference mechanisms.

Unlike traditional approaches to inference in natural language [2–4] our approach (1) makes use of *machine learning* based resources in order to induce an abstract representation of the input data, as well as to support multiple inference stages and (2) models inference as an *optimization* process that provides robustness against inherent variability in natural language, inevitable noise in inducing the abstract representation, and missing information.

We present a principled computational approach to *semantic entailment* in natural language that addresses some of the key problems encountered in traditional approaches – knowledge acquisition and brittleness. The solution includes a hierarchical knowledge representation language into which we induce appropriate representations of the given text and required background knowledge. The other main element is a sound inferential mechanism that makes use of the induced representation to determine an extended notion of subsumption, using an optimization approach that supports abstracting over language variability and representation inaccuracies. Along with describing the key elements of our approach, we present a system that implements it, and an evaluation of this system on two corpora, PASCAL and PARC text collections.

## 1.1 General Description of Our Approach

Specifically, given two text snippets  $S$  (source) and  $T$  (target) where typically, but not necessarily,  $S$  consists of a short paragraph and  $T$ , a sentence, textual

semantic entailment is the problem of determining if  $S \models T$ , which we read as “ $S$  entails  $T$ ”. This informally means that *most people would agree that the meaning of  $S$  implies that of  $T$* . More formally, we say that  $S$  entails  $T$  when some representation of  $T$  can be “matched” (modulo some meaning-preserving transformations to be defined below) with some (or part of a) representation of  $S$ , at some level of granularity and abstraction. The approach consists of the following components:

A Description Logic based hierarchical knowledge representation, **EFDL** (Extended Feature Description Logic), [5], into which we re-represent the surface level text, augmented with induced syntactic and semantic parses and word and phrase level abstractions.

**A knowledge base** consisting of syntactic and semantic rewrite rules, written in EFDL.

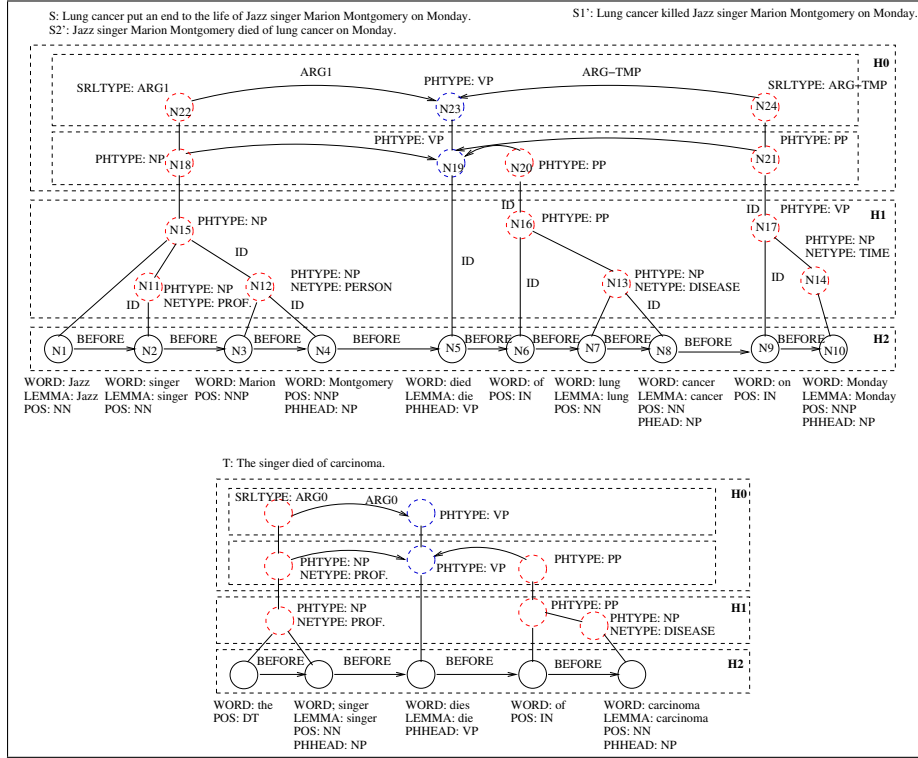
**An extended subsumption algorithm** which determines subsumption between EFDL expressions (representing text snippets or rewrite rules). “Extended” here means that the basic unification operator is extended to support several word level and phrase level abstractions.

First, a set of machine learning based resources are used to induce the representation for  $S$  and  $T$ . The entailment algorithm then proceeds in two phases: (1) it incrementally generates re-representations of the original representation of the source text  $S$  by augmenting it with heads of subsumed re-write rules, and (2) it makes use of an optimization based (extended) subsumption algorithm to check whether any of the alternative representations of the source entails the representation of the target  $T$ . The extended subsumption algorithm is used both in checking final entailment and in determining when and how to generate a re-representation in slightly different ways. Figure 1 provides a graphical example of the representation of two text snippets, along with a sketch of the extended subsumption approach to decide the entailment.

Along with the formal definition and justification developed here for our computational approach to semantic entailment, our knowledge representation and algorithmic method provide a novel solution that addresses some of the key issues the natural language research community needs to resolve in order to move forward towards higher level tasks of this sort. Namely, we provide ways to represent knowledge, either external or induced, at multiple levels of abstractions and granularity, and reason with it at the appropriate level. The evaluation of our approach is very encouraging and illustrates the significance of some of its key contributions, while also exhibiting the key areas where a significant progress is needed – that of the rewrite rule knowledge base.

## 2 Algorithmic Semantic Entailment

Let  $\mathcal{R}$  be a knowledge representation language with a well defined syntax and semantics over any domain  $\mathcal{D}$ . Specifically, we think of elements in  $\mathcal{R}$  as expressions in the language or, equivalently, as the set of interpretations that satisfy it



**Fig. 1.** Example of *Re-represented Source & Target* pairs as concept graphs. The original source sentence  $S$  generated several alternatives including  $S'_1$  and the sentence in the figure ( $S'_2$ ). Our algorithm was not able to determine entailment of the first alternative (as it fails to match in the extended subsumption phase), but it succeeded for  $S'_2$ . The dotted nodes represent phrase level abstractions.  $S'_2$  is generated in the first phase by applying the following chain of inference rules: #1 (genitives): “Z’s W  $\rightarrow$  W of Z”; #2: “X put end to Y’s life  $\rightarrow$  Y die of X”. In the extended subsumption, the system makes use of WordNet hypernymy relation (“lung cancer” IS-A “carcinoma”) and NP-subsumption rule (“Jazz singer Marion Montgomery” IS-A “singer”). The rectangles encode the hierarchical levels ( $H_0, H_1, H_2$ ) at which we applied the extended subsumption. Also note that entailment follows even though the structure corresponding to “on Monday” is not present in the target sentence, since “event happened on Monday” entails “event happened”.

[6]. Let  $r$  be a mapping from a set of text snippets  $\mathcal{T}$  to a set of expressions in  $\mathcal{R}$ . Denote the representations of two text snippets  $S, T$ , under this mapping by  $r_S, r_T$ , respectively. Note that we will use the word *expression* and *representation* interchangeably. Given the set of interpretations over  $\mathcal{D}$ , let  $M$  be a mapping from an expression in  $\mathcal{R}$  to the corresponding set of interpretations it satisfies. For expressions  $r_S, r_T$ , the images of  $S, T$  under  $\mathcal{R}$ , their model theoretic representations thus defined are denoted  $M(r_s), M(r_t)$ .

Conceptually, as in the traditional view of semantic entailment, this leads to a well defined notion of entailment, formally defined via the model theoretic view; traditionally, the algorithmic details are left to a *theorem prover* that uses the syntax of the representation language, and may also incorporate additional knowledge in its inference. We follow this view, and use a notion of *subsumption* between elements in  $\mathcal{R}$ , denoted  $u \sqsubseteq v$ , for  $u, v \in \mathcal{R}$ , that is formally defined via the model theoretic view — when  $M(u) \subseteq M(v)$ . Subsumption between representations provides an implicit way to represent entailment, where additional knowledge is conjoined with the source to “prove” the target.

However, the proof theoretic approach corresponding to this traditional view is unrealistic for natural language. Subsumption is based on *unification* and requires, in order to prove entailment, that the representation of  $T$  is entirely embedded in the representation of  $S$ . Natural languages allow for words to be replaced by synonyms, for modifier phrases to be dropped, etc., without affecting meaning. An extended notion of subsumption is therefore needed which captures sentence, phrase, and word-level abstractions.

Our algorithmic approach is thus designed to alleviate these difficulties in a proof theory that is too weak for natural language. Conceptually, a weak proof theory is overcome by entertaining multiple representations that are equivalent in meaning. We provide theoretical justification below, followed by the algorithmic implications.

We say that a representation  $r \in \mathcal{R}$  is *faithful* to  $S$  if  $r$  and  $r_S$  have the same model theoretic representation, i.e.,  $M(r) = M(r_S)$ . Informally, this means that  $r$  is the image under  $\mathcal{R}$  of a text snippet with the same meaning as  $S$ .

**Definition 1.** Let  $S, T$  be two text snippets with representations  $r_S, r_T$  in  $\mathcal{R}$ . We say that  $S \models T$  (read:  $S$  semantically entails  $T$ ) if there is a representation  $r \in \mathcal{R}$  that is faithful to  $S$  and that is subsumed by  $r_T$ .

Clearly, there is no practical way to exhaust the set of all those representations that are faithful to  $S$ . Instead, our approach searches a space of faithful representations, generated via a set of rewrite rules in our KB.

A *rewrite rule* is a pair  $(lhs, rhs)$  of expressions in  $\mathcal{R}$ , such that  $lhs \sqsubseteq rhs$ . Given a representation  $r_S$  of  $S$  and a rule  $(lhs, rhs)$  such that  $r_S \sqsubseteq lhs$ , the augmentation of  $r_S$  via  $(lhs, rhs)$  is the representation  $r'_S = r_S \wedge rhs$ .

**Claim:** The representation  $r'_S$  generated above is faithful to  $S$ .

To see this, note that as expressions in  $\mathcal{R}$ ,  $r'_S = r_S \wedge rhs$ , therefore  $M(r'_S) = M(r_S) \cap M(rhs)$ . However, since  $r_S \sqsubseteq lhs$ , and  $lhs \sqsubseteq rhs$ , then  $r_S \sqsubseteq rhs$  which implies that  $M(r_S) \subseteq M(rhs)$ . Consequently,  $M(r'_S) = M(r_S)$  and the new representation is faithful to  $S$ .

The claim gives rise to an algorithm, which suggests incrementally *augmenting* the original representation of  $S$  via the rewrite rules, and computing subsumption using the “weak” proof theory between the augmented representation and  $r_T$ . Informally, this claim means that while, in general, augmenting the representation of  $S$  with an expression  $rhs$  may restrict the number of interpretations the resulting expression has, in this case, since we only augment the

representation when the left hand side  $lhs$  subsumes  $r_S$ , we end up with a new representation that is in fact equivalent to  $r_S$ . Therefore, given a collection of rules  $\{(lhs \sqsubseteq rhs)\}$  we can chain their applications, and incrementally generate faithful representations of  $S$ . Consequently, this algorithm is a sound algorithm<sup>3</sup> for semantic entailment according to Def. 1, but it is not complete. Its success depends on the size and quality of the rule set<sup>4</sup> applied in the search.

Two important notes are in order. First, since rewrite rules typically “modify” a small part of a sentence representation (see Fig. 1), the augmented representation provides also a compact way to encode a large number of possible representations. Second, note that while the rule augmentation mechanism provides a justification for an algorithmic process, in practice, applying rewrite rules is somewhat more complicated. The key reason is that many rules have a large fan-out; that is, a large number of heads are possible for a given rule body. Examples include synonym rules, equivalent ways to represent names of people (e.g., John F. Kennedy and JFK), etc. We therefore implement the mechanism in two ways; one process which supports chaining well, in which we explicitly augment the representation with low fan-out rules (e.g., Passive-Active rules); and a second, appropriate to the large fan-out rules. In the latter, we abstain from augmenting the representation with the many possible heads but take those rules into account when comparing the augmented source with the target. For example, if a representation includes the expression “JFK/PER”, we do not augment it with all the many expressions equivalent to “JFK” but, when comparing it to a candidate in the target, such as “President Kennedy”, these equivalencies are taken into account. Semantically, this is equivalent to augmenting the representation. Instead of an explicit list of rules, the large fan-out rules are represented as a functional black box that can, in principle, contain any procedure for deciding comparisons. For this reason, this mechanism is called *functional subsumption*. The resulting algorithmic approach is therefore:

(1) After inducing a representation for  $S$  and  $T$ , the algorithm incrementally searches the rewrite rules in KB to find a rule with a body that subsumes the representation of  $S$ . In this case, the head of the rule is used to *augment* the representation of  $S$  and generate a new (equivalent) representation  $S'_i$  of  $S$ . KB consists of syntactic and semantic rewrite rules expressed at the word, syntactic and semantic categories, and phrase levels; the resulting new representations capture alternative ways of expressing the surface level text.

(2) Representation  $S'_i$ s are processed via the extended subsumption algorithm against the representation of  $T$ . The notion of extended subsumption captures, just like the rewrite rules, several sentence, phrase, and word-level abstractions. The extended subsumption process is also used when determining whether a rewrite rule applies.

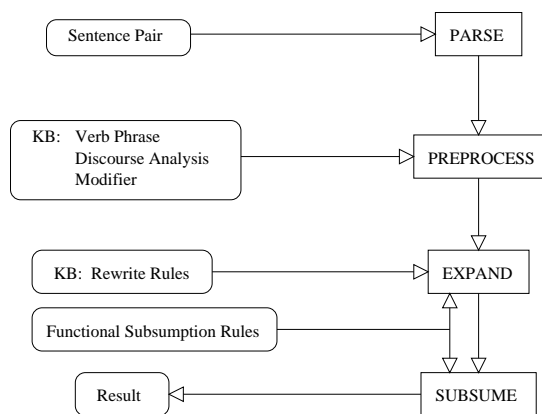
<sup>3</sup> Soundness depends on a “correct” induction of the representation of the text; we do not address this theoretically here.

<sup>4</sup> The power of this search procedure is in the rules.  $lhs$  and  $rhs$  might be very different at the surface level, yet, by satisfying model theoretic subsumption they provide expressivity to the re-representation in a way that facilitates the overall subsumption.

Rewrite rules and extended subsumption decisions take into account relational and structural information encoded in the hierarchical representation, which is discussed below. In both cases, decisions are quantified as input to an optimization algorithm that attempts to generate a “proof” that  $S$  entails  $T$ , and is discussed later in the paper.

### 3 High-level System Description

The full entailment system implements the algorithm described in section 2. This section provides a description of this system as well as some implementation-driven deviations from the abstract algorithm. Sentence pairs are annotated with various machine learning tools and parsed into the hierarchical representation described in section 4 (denoted by PARSE on the diagram in Fig. 2).



**Fig. 2.** System block diagram

The expansion described in Sec. 2 is subdivided into two stages: the first stage, labeled PREPROCESS on the accompanying diagram, applies a set of *semantic analysis* modules comprising rewrite rules that need only be applied once. Once these have been applied, the second sub-stage expands the source sentence only with rules from the Rewrite Rule knowledge base module. (The distinction between the rules in the semantic analysis modules and the rules in the Rewrite Rule knowledge base module is explained in Sec. 3.1.) The preprocessing module also uses heuristics to simplify complex predicate arguments.

The second expansion stage (EXPAND in the diagram) applies rewrite rules to the source sentence, and may chain these rules over successive iterations. (Expansion terminates either when no new rules can be applied without duplicating a previous application or after a fixed number of iterations).

After expansion, the system checks for subsumption (SUBSUME in the diagram) by comparing elements of the source and target sentences, generating

a cost solution for the comparison as described in Sec. 5, and comparing the minimum cost to a threshold. In both the expansion and subsumption stages, the extended subsumption knowledge base is used as necessary.

Changes to the system can be realized by adding or removing semantic analysis modules, by changing the subsumption algorithm, or by changing the weights on those variables in the optimization formulation corresponding to the relevant attribute/edge types in the representation and on the variables describing applications of different categories of rewrite rules.

### 3.1 Semantic Analysis Modules

The semantic analysis modules are essentially groupings of rewrite rules, separated from the more general rewrite rules because they deal with structural principles such as tense and auxiliary verb constructions rather than more specific paraphrasing. They behave in most respects like the general rewrite rules described in Sec. 5, except for the following distinctions:

1. They are applied to both S and T.
2. The Verb Processing module allows its rules to modify T by *rewriting* the verb construction in T instead of simply *expanding* the original T, as described in the Verb Processing Module section below.
3. These rules may not be chained indefinitely; either they add an attribute to a node which is then preserved through all subsequent rule applications, or they permanently simplify a multi-node structure to a simpler structure.

The system presently supports three semantic analysis modules: verb phrase compression, discourse analysis, and modifier analysis. Note that we use Discourse Analysis to denote semantic analysis of interaction between predicates (a refinement of the traditional denotation of relationships between sentences, as such interactions could occur between predicates in different, as well as within, sentences). The different semantic analysis modules depend on different levels of structure and the corresponding rules are therefore phrased at different levels of representation: verb phrase compression requires word order and part of speech; discourse analysis requires full parse information and part of speech, and modifier analysis requires full and shallow parse information and part of speech.

**3.1.1 The Verb Phrase module** The Verb Processing (VP) module rewrites certain verb phrases as a single verb with additional attributes. It uses word order and part of speech information to identify candidate patterns and, when the verbs in the construction in the sentence match a pattern in the VP module, the verb phrase is replaced by a single predicate node with additional attributes representing modality (MODALITY) and tense (TENSE). Simple past and future tenses are detected and represented, as are some conditional tenses.

The rules in the VP module are applied differently from those in other modules, allowing the representation of T to be changed by substituting the verb construction specified by the rule head for that specified by the rule body (rather

than simply expanding the representation of T by adding the head structure). This is permissible because the rules in this module have extremely high precision, so few or no errors are introduced by the alterations to T's structure.

The VP module presently recognizes modal constructions, tense constructions, and simple verb compounds of the form "VERB to VERB" (such as "*manage to enter*"). In each case, in the functional subsumption step of rule application, the first verb is compared to a list that maps verb lemmas to tenses and modifiers; for example, in the verb phrase "*has entered*", "*has*" is recognized as a tense auxiliary and results in the attribute "TENSE: past" being added to the second verb's node. The "*has*" node is then eliminated and the graph structure corrected.

**3.1.2 Discourse Analysis Module** The Discourse Analysis (DA) module detects the effects of an embedding predicate on the embedded predicate. It uses the full parse tree to identify likely candidate structures, then compares the embedding verb to a list mapping verbs to MODALITY. The main distinction that is presently supported is between "FACTUAL" and a set of values that distinguish various types of uncertainty. This allows different assumptions to be supported; for example, if we wish to assume that when something is said, it is taken as truth, we can treat the MODALITY value, "REPORTED", as entailing "FACTUAL" and vice versa. The module attaches the appropriate MODALITY attribute value to the embedded verb node; if this attribute is not matched during subsumption, subsumption fails.

**3.1.3 Modifier Module** The Modifier module allows comparison of noun modifiers such as "all", "some", "any", "no", etc. This module is important when two similar sentences differ in the generality of one or more arguments of otherwise identical predicates. For full effectiveness, this requires that the system determine whether the predicate in question is upward or downward monotonic, i.e. whether the source target entails more general or more specific cases. This problem is non-trivial and has not yet been resolved for this system.

Instead, we assume that predicates are upward monotonic. For example: A source sentence predicate with an argument modified by an adjective subsumes an identical target predicate with the same argument without the modifying adjective. This is non-reflexive, i.e. a target predicate with an argument having an adjectival modifier will not be subsumed by an identical source predicate with no adjectival modifier (or a modifier different in meaning).

## 3.2 Rewrite Rule Module

The Rewrite Rule module contains rules encoding paraphrase and logical rewrite information. Paraphrase rules encode valid substitutions for one verb (-phrase) with another (e.g. "*hawked*" may be replaced with "*sold*" given similar argument structure); logical rules encode predicates that may be inferred from existing predicates (e.g. if A sold B to C, then C bought B from A).

The Rewrite Rule module is implemented independently of the semantic analysis modules, and behaves as described in Sec. 2. Most rewrite rules require Semantic Role Labeling (SRL) information (Sec. 4), though some use only word order and the words. The Rewrite Rule knowledge base can be used independently of the other modules, but may benefit from them—e.g. VP module may compress a tense construction, allowing a rewrite rule to fire. To avoid the obvious problem of potentially generating grammatically incorrect sentences, the following restriction is imposed on rewrite rules that modify the node/edge structure of the sentence representation: A rewrite rule encoding predicate-level information must add a new node for the substituted predicate, connected to the relevant arguments of the existing predicate matching the rule body.

A sentence is not read as a sequence of words; rather, it is read as a collection of predicates. As such, it can be thought of as a set of trees whose roots (predicates) do not overlap, but whose leaves (arguments) may be shared by more than one predicate.

### 3.3 Variations of The Entailment System

By changing the weighting scheme of the cost function in the module that checks for subsumption, it is possible to restrict the kinds of information available to the system. For example, if only variables corresponding to words and SRL information are given non-zero weights, only rewrite rules using a subset of this information and the information itself will be represented in the final cost equation. Any distinct information source may be given its own weight, thus allowing different information sources to be given preference.

At present, these weights are found by brute force search; our current efforts are directed at learning these weights more efficiently.

For the experimental evaluation of the system, we also tried a hybrid system that used high-level semantic information and general rewrite rules, but which used a relatively basic word level approach (LLM; see Sec. 7) to compare the arguments of predicates.

## 4 Hierarchical Knowledge Representation

Our semantic entailment approach relies heavily on a hierarchical representation of natural language sentences, defined formally over a domain  $\mathcal{D} = \langle \mathcal{V}, \mathcal{A}, \mathcal{E} \rangle$  which consists of a set  $\mathcal{V}$  of typed elements, a set  $\mathcal{A}$  of attributes of elements, and a set  $\mathcal{E}$  of relations among elements. We use a Description-Logic inspired language, *Extended Feature Description Logic (EFDL)*, an extension of FDL [5]. As described there, expressions in the language have an equivalent representation as *concept graphs*, and we refer to the latter representation here for comprehensibility.

*Nodes* in the concept graph represent elements — words, (multiple levels of) phrases (including arguments to predicates), and predicates. *Attributes* of nodes represent properties of elements. Examples of attributes (they are explained in

more detail later) include {LEMMA, WORD, POS, PREDICATE\_VALUE, PHTYPE, PHHEAD, NETYPE, ARGTYPE, NEGATION, CONFIDENCE, TENSE}. The first three are word level, the next three are phrase level, NETYPE is the named entity of a phrase, ARGTYPE is the set of semantic arguments as defined in PropBank [7], NEGATION is a negation attribute, CONFIDENCE encodes a proposition’s modality (by tagging its main verb), and TENSE applies only to verbs. Only attributes with non-null values need to be specified.

*Relations* (roles) between two elements are represented by labeled edges between the corresponding nodes. Examples of roles (again, explained in more detail later) include: {BEFORE, ARG0, . . . ARG5}; BEFORE indicates the order between two individuals, and ARG0, . . . ARG5 represent the relations between a predicate (verb) and its argument.

Figure 1 shows a visual representation of a pair of sentences re-represented as concept graphs. Concept graphs are used both to describe instances (sentence representations) and rewrite rules. The expressivity of these differ - the body and head of rules are simple chain graphs, for inference complexity reasons. (Restricted expressivity is an important concept in Description Logics [8], from which we borrow several ideas and nomenclature.)

Concept graph representations are induced via state of the art machine learning based resources such as a part-of-speech tagger [9], a syntactic parser [10], a semantic parser [11, 12], a named entity recognizer <sup>5</sup>, and a name coreference system [13] with the additional tokenizer and lemmatizer derived from WordNet [14]. Rewrite rules were filtered from a large collection of paraphrase rules developed in [15]. These are inference rules that capture lexico-syntactic paraphrases, such as "X wrote Y" synonymous with "X is the author of Y".

The rules are compiled into our language. Moreover, a number of non-lexical rewrite rules were generated manually. Currently, our knowledge base consists of approximately 300 inference rules.

#### 4.1 Rule representation

A rule is a pair (*lhs*, *rhs*) of concept graphs (*lhs* is the rule’s *body*, while *rhs* is its *head*). These concept graphs are restricted in that they must be *paths*. This restricts the complexity of the inference algorithm while keeping them useful enough for our purposes. As a shorthand, more than one path can be described in *lhs*, but in this case the rule is implicitly treated as more than one rule, each with one path and the same *rhs*.

*lhs* describes a structure to match in the sentence concept graph, while *rhs* describes a new predicate (and related attributes and edges) to be added to the sentence concept graph in case there is a match. *rhs* can also describe attributes to add to one or more existing nodes without adding a new predicate, provided no new edges are introduced. These restrictions ensure that the data representation always remains, from the subsumption algorithm’s perspective, a set of overlapping trees.

<sup>5</sup> Named entity recognizer from Cognitive Computation Group, <http://l2r.cs.uiuc.edu/~cogcomp>

Variables can be used in *lhs* so that we can specify which entities have edges/attributes added by *rhs*. Rules thus allow modification of the original sentence; e.g. we encode DIRT [15] rules as predicate-argument structures and use them to allow (parts of) the original sentence to be re-represented via paraphrase, by linking existing arguments with new predicates.

## 5 Inference Model and Algorithm

This section describes how the extended subsumption process exploits the hierarchical knowledge representation and how inference is modeled as optimization.

### 5.1 Modeling Hierarchy & Unification Functions

An exact subsumption approach that requires the representation of  $T$  be entirely embedded in the representation of  $S'_i$  is unrealistic. Natural languages allow words to be replaced by synonyms, modifier phrases to be dropped, etc., without affecting meaning.

We define below our notion of extended subsumption, computed given two representations, which is designed to exploit the hierarchical representation and capture multiple levels of abstractions and granularity of properties represented at the sentence, phrase, and word-level.

Nodes in a concept graph are grouped into different hierarchical sets denoted by  $H = \{H_0, \dots, H_j\}$  where a lower value of  $j$  indicates higher hierarchical level (more important nodes). This hierarchical representation is derived from the underlying concept graph and plays an important role in the definitions below.

We say that  $S'_i$  entails  $T$  if  $T$  can be *unified into*  $S'_i$ . The significance of definitions below is that we define unification so that it takes into account both the hierarchical representation and multiple abstractions.

Let  $V(T)$ ,  $E(T)$ ,  $V(S'_i)$ , and  $E(S'_i)$  be the sets of nodes and edges in  $T$  and  $S'_i$ , respectively. Given a hierarchical set  $H$ , a *unification* is a 1-to-1 mapping  $U = (U_V, U_E)$  where  $U_V : V(T) \mapsto V(S'_i)$ , and  $U_E : E(T) \mapsto E(S'_i)$  satisfying:

1.  $\forall(x, y) \in U : x$  and  $y$  are in the same hierarchical level.

2.  $\forall(e, f) \in U_E : e$  and  $f$  must be unified accordingly. That is, for  $n_1, n_2, m_1$ , and  $m_2$  which are the sinks and the sources of  $e$  and  $f$  respectively,  $(n_1, m_1) \in U_V$  and  $(n_2, m_2) \in U_V$ .

Let  $\mathcal{U}(T, S'_i)$  denote the space of all unifications from  $T$  to  $S'_i$ . In our inference, we assume the existence of a unification function  $G$  that determines the cost of unifying pairs of nodes or edges.  $G$  may depend on language and domain knowledge, e.g. synonyms, name matching, and semantic relations. When two nodes or edges cannot be unified,  $G$  returns infinity. This leads to the definition of *unifiability*.

**Definition 2.** Given a hierarchical set  $H$ , a unification function  $G$ , and two concept graphs  $S'_i$  and  $T$ , we say that  $T$  is unifiable to  $S'_i$  if there exists a unification  $U$  from  $T$  to  $S'_i$  such that the cost of unification defined by

$$D(T, S'_i) = \min_{U \in \mathcal{U}(T, S'_i)} \sum_{H_j} \sum_{(x,y) \in U | x,y \in H_j} \lambda_j G(x, y)$$

is finite, where  $\lambda_j$  are some constants s.t. the cost of unifying nodes at higher levels dominates those of the lower levels.

Because top levels of the hierarchy dominate lower ones, nodes in both graphs are checked for subsumption in a top down manner. The levels and corresponding processes are:

**Hierarchy set  $H_0$**  corresponds to sentence-level nodes, represented by the verbs in the text. The inherent set of attributes is  $\{\text{PHTYPE}, \text{PREDICATE\_VALUE}, \text{LEMMA}\}$ . In order to capture the argument structure at sentence-level, each verb in  $S'_i$  and  $T$  has a set of edge attributes  $\{\text{ARG}_i, \text{PHTYPE}_i\}$ , where  $\text{ARG}_i$  and  $\text{PHTYPE}_i$  are the semantic role label and phrase type of each argument  $i$  of the verb considered [7].

For each verb in  $S'_i$  and  $T$ , check if they have the same attribute set and argument structure at two abstraction levels:

1. The semantic role level (SRL attributes). eg: ARG0 verb ARG1 : *[Contractors]/ARG0 build [houses]/ARG1 for \$100,000.*
2. The syntactic parse level (parse tree labels). Some arguments of the verb might not be captured by the semantic role labeler (SRL); we check their match at the syntactic parse level. eg: NP verb NP PP : *[Contractors]/NP build [houses]/NP [for \$100,000]/ PP.*

At this level, if all nodes are matched (modulo functional subsumption), the cost is 0, otherwise it is infinity.

**Hierarchy set  $H_1$**  corresponds to phrase-level nodes and represents the semantic and syntactic arguments of the  $H_0$  nodes (verbs). If the phrase-level nodes are recursive structures, all their constituent phrases are  $H_1$  nodes. For example, a complex noun phrase consists of various base-NPs. Base-NPs have edges to the words they contain.

The inference procedure recursively matches the corresponding  $H_1$  nodes in  $T$  and  $S'_i$  until it finds a pair whose constituents do not match. In this situation, a *Phrase-level Subsumption* algorithm is applied. The algorithm is based on subsumption rules that are applied in a strict order (as a decision list) and each rule is assigned a confidence factor.

The algorithm makes sure two  $H_1$  nodes have the same PHTYPE, but allows other attributes such as NETYPE to be optional. Each unmatched attribute results in a uniform cost.

**Hierarchy set  $H_2$**  corresponds to word-level nodes. The attributes used here are:  $\{\text{WORD}, \text{LEMMA}, \text{POS}\}$ . Unmatched attributes result in a uniform cost.

Figure 1 exemplifies the matching order between  $S'_i$  and  $T$  based on constraints imposed by the hierarchy.

## 5.2 Inference as Optimization

We solve the subsumption problem by formulating an equivalent Integer Linear Programming (ILP) problem<sup>6</sup>. An ILP problem involves a set of integer variables  $\{v_i\}$  and a set of linear equality or inequality constraints among them. Each variable  $v_i$  is associated with a cost  $c_i$ , and the problem is to find an assignment to the variables that satisfies the constraints and minimizes  $\sum_i c_i v_i$ .

To prove  $S \sqsubseteq T$ , we first start with the graph  $S$  (the initial graph). Then we extend  $S$  by adding the right hand sides of applicable rules. This is repeated up to a fixed number of rounds and results in an expanded graph  $S'_d$ . The formulation allows us to solve for the optimal unification from  $T$  to  $S'_d$  that minimizes the overall cost.

To formulate the problem this way, we need a set of variables that can represent different unifications from  $T$  to  $S'_d$ , and constraints to ensure the validity of the solution, i.e. that the unification does not violate any nonnegotiable property. We explain this below. For readability, we sometimes express constraints in a logic form that can be easily transformed to linear constraints.

**5.2.1 Representing Unification** We introduce Boolean variables  $u(n, m)$  for each pair of nodes  $n \in V(T)$  and  $m \in V(S'_d)$  in the same hierarchical level, and  $u(e, f)$  for each pair of edges  $e \in E(T)$  and  $f \in E(S'_d)$  in the same level.

To ensure that the assignment to the matching variables represents a valid unification from  $T$  and  $S'_d$ , we need two types of constraints. First, we ensure the unification preserves the node and edge structure. For each pair of edges  $e \in E(T)$  and  $f \in E(S'_d)$ , let  $n_i, n_j, m_k$ , and  $m_l$  be the sources and the sinks of  $e$  and  $f$  respectively. Then  $u(e, f) \Rightarrow u(n_i, m_k) \wedge u(n_j, m_l)$ . Finally, to ensure that the unification is a 1-to-1 mapping from  $T$  to  $S'_d$ ,  $\forall n_i \in V(T) \sum_{m_j \in S'_d} u(n_i, m_j) = 1$ , and  $\forall m_j \in V(S'_d) \sum_{n_i \in T} u(n_i, m_j) \leq 1$ .

**5.2.2 Finding A Minimal Cost Solution** We seek the unification with a minimum (and, of course, finite) cost:  $\sum_{H_j} \sum_{u(x,y)|x,y \in H_j} \lambda_j G(x, y) u(x, y)$ , where  $\lambda_j$  is the constant and  $G$  the cost of unification as we explained in the previous sections. The minimal subgraph  $S'_i$  of  $S'_d$  that  $T$  is unified to is also the minimal representation of  $S$  that incurs minimal unification cost.

## 6 Previous Work

Knowledge representation and reasoning techniques have been studied in NLP for a long time [2–4]. Most approaches relied on mapping to a canonical First

<sup>6</sup> Despite the fact that this optimization problem is NP hard, commercial packages have very good performance on sparse problems such as Xpress-MP by Dash Optimization, <http://www.dashoptimization.com>. See [16, 17] for details on modeling problems as ILP problems.

Order Logic representations with a general prover and without using acquired rich knowledge sources.

Significant development in NLP, specifically the ability to acquire knowledge and induce some level of abstract representation could, in principle, support more sophisticated and robust approaches. Nevertheless, most modern approaches developed so far are based on shallow representations of the text that capture lexico-syntactic relations based on dependency structures and are mostly built from grammatical functions in an extension to keyword-base matching [18]. Some systems make use of some semantic information, such as WordNet lexical chains [19], to slightly enrich the representation. Other have tried to learn various logical representations [20]. However, none of these approaches makes global use of a large number of resources as we do, or attempts to develop a flexible, hierarchical representation and an inference algorithm for it, as we present here.

Recently, as part of the PASCAL effort, more thought has been given to representation and inference with them. While most of the approaches presented there are still relatively shallow, some are more involved, including Glickman et. al [21] and Raina et. al [22]. The former provides an attempt to formalize the semantic entailment problem probabilistically, but the corresponding implementation focuses mostly on lexical level processing techniques. The latter is more similar to ours. Specifically, it is more related to one of our earlier attempt to model sentence equivalence via tree mapping [23]. Similar to our approach they use a large number of resources (such as a dependency parser, WordNet semantic information, and a PropBank semantic parser) to enrich a graph-based representation of source and target sentences. This information is then used to perform a number of lexico-syntactic and semantic transformations that would potentially lead to the source-target match. However, in contrast to our system, they do not attempt to formalize their approach.

## 7 Experimental Evaluation

We tested our approach on two different text collections, a set of question-answer pairs provided by the PARC AQUAINT team<sup>7</sup>, and on the PASCAL challenge data set<sup>8</sup>.

We first describe the performance of the system on the PARC data set, organizing the rewrite rules into separate components and examining the contribution of each. We then examine the system's performance on the PASCAL data set, and the need for different weights in the optimization function for this corpus. All numerical results represent the accuracy of the system when predictions are made for all examples in the corpus under consideration, i.e. for a recall of 100%.

### 7.1 Experiments Using the PARC Data Set

In this set of experiments, we tested our approach on a collection of question-answer pairs develop by Xerox PARC for a pilot evaluation of Knowledge-

<sup>7</sup> The data is available at <http://l2r.cs.uiuc.edu/~cogcomp>.

<sup>8</sup> <http://www.pascal-network.org/Challenges/RTE/>

Oriented Approaches to Question Answering under the ARDA-AQUAINT program. The PARC corpus consists of 76 Question-Answer pairs annotated as “true”, “false” or “unknown” (and an indication of the type of reasoning required to deduce the label). The question/answer pairs provided by PARC are designed to test different cases of linguistic entailment. The corpus concentrates on examples of strict and plausible linguistic (lexical and constructional) inferences and indicates whether it involves some degree of background world knowledge. The focus is on inferences that can be made purely on the basis of the meaning of words and phrases. The questions are straightforward and therefore easily rewritten (by hand) into statement form. One sentence pair involving modifiers was reordered to test modifier subsumption.

For evaluation reasons, we used only two labels in our experiments, “true” and “false”, corresponding to “S entails T” and “S does not entail T”. The “unknown” instances were classified as “false”. Of these 76 sentence pairs, 64 were perfectly tagged by our Semantic Role Labeler (SRL), and were used as a second noise-free test set to evaluate our system under “ideal” conditions.

This section illustrates the contributions of high-level (semantic) analysis modules to system performance and compares the full entailment system with the baseline LLM — lexical-level matching based on a bag-of-words representation with lemmatization and normalization. For each semantic analysis component we give one or more examples of sentence pairs affected by the new version/component. Finally, we present a summary of the performance of each system on the noise-free (perfect) and noisy (full) data sets.

The full system uses Semantic Role Labeling, full parse and shallow parse structure, which allows use of all semantic analysis modules. This version of the system, labeled “SRL+deep”, also uses Named Entity annotation from our Named Entity Recognizer (NER).

**7.1.1 LLM** The LLM system ignores a large set of stopwords — including some common verbs, such as “go” — which for certain positive sentence pairs allows entailment when the more sophisticated system requires a rewrite rule to map from the predicate in S to the predicate in T. For example: since the list of stopwords includes forms of “be”, the following sentence pair will be classified “true” by LLM, while the more sophisticated system requires a KB rule to link “visit” to “be (in)”:

S: *[The diplomat]/ARG1 visited [Iraq]/ARG1 [in September]/AM\_TMP*

T: *[The diplomat]/ARG1 was in [Iraq]/ARG2*

Of course, LLM is insensitive to small changes in wording. For the following sentence pair, LLM returns “true”, which is clearly incorrect:

S: *Legally, John could drive.*

T: *John drove.*

**7.1.2 SRL + Deep Structure** The entailment system without the high-level semantic modules correctly labels the following example, which is incorrectly labeled by the LLM system:

S: *No US congressman visited Iraq until the war.*

T: *Some US congressmen visited Iraq before the war.*

The entailment system includes the determiners “*no*” and “*some*” as modifiers of their respective entities; subsumption fails at the argument level because these modifiers don’t match. (Note: this does not require the MODIFIER module, as in this instance, the lexical tokens themselves are different.)

However, the new system also makes new mistakes:

S: *The room was full of women.*

T: *The room was full of intelligent women.*

The LLM system finds no match for “intelligent” in S, and so returns the correct answer, “false”. However, the SRL+deep structure system allows unbalanced T adjective modifiers, assuming that S must be more general than T, and allows subsumption.

*7.1.2.1 Verb Phrase (VP) Module* The Verb Processing module is beneficial when two similar sentences are distinguished by a modal construction.

In the example below, the VP recognizes the modal construction and adds the modifying attribute “MODALITY: potential” to the main verb node, “*drive*”:

S: *Legally, John could drive.*

T: *John drove.*

Subsumption in the entailment system then fails at the verb level, making entailment less likely.

This module also acts as an enabler for other resources (such as the Knowledge Base). This may result in a decrease in performance when those modules are not present, as it corrects T sentences that may have failed subsumption in the more restricted system because the auxiliary verb was not present in the corresponding S sentence:

S: *Bush said that Khan sold centrifuges to North Korea.*

T: *Centrifuges were sold to North Korea.*

The SRL+LLM system returns the correct answer, “false”, for this sentence pair, but for the wrong reason: SRL generates a separate predicate frame for “*were*” and for “*sold*” in T, and there is no matching verb for “*were*” in S.

When the VP module is added, the auxiliary construction in T is rewritten as a single verb with tense and modality attributes attached; the absence of the auxiliary verb means that SRL generates only a single predicate frame for “*sold*”. This matches its counterpart in S, and subsumption succeeds, as the qualifying effect of the verb “*said*” in S cannot be recognized without the deeper parse structure and the Discourse Analysis module.

On the PARC corpus, the net result of applying the VP module when the KB is not enabled is either no improvement or a decrease in performance, due to the specific mix of sentences. However, the importance of such a module to correctly identify positive examples becomes evident when the knowledge base is enabled, as the performance jumps significantly over that of the same system without VP enabled.

*7.1.2.2 Discourse Analysis (DA) Module* The following example highlights the importance of the way an embedded predicate is affected by the embedding predicate. In this example, the predicate “*Hanssen sold secrets to the Russians*” is embedded in the predicate “*The New York Times reported...*”.

S: *The New York Times reported that Hanssen sold FBI secrets to the Russians and could face the death penalty.*

T: *Hanssen sold FBI secrets to the Russians.*

Our system identifies the following verb frames in S and T:

S-A: [*The New York Times*]/ARG0 reported [*that Hanssen sold FBI secrets to the Russians...*]/ARG1

S-B: [*Hanssen*]/ARG0 sold [*FBI secrets*]/ARG1 to [*the Russians*]/ARG3

T-A: [*Hanssen*]/ARG0 sold [*FBI secrets*]/ARG1 to [*the Russians*]/ARG3

During preprocessing, our system detects the pattern “[VERB] that [VERB]”, and classifies the first verb as affecting the confidence of its embedded verb. The system marks the verb (predicate) “*sold*” in S with attribute and value “MODALITY: REPORTED”. Thus the subsumption check determines that entailment fails at the verb level, because by default, verbs are given the attribute and value “MODALITY: FACTUAL”, and the MODALITY values of the “*sold*” nodes in S and T do not match. This is in contrast to LLM and SRL+LLM, both of which return the answer “true”.

The next example demonstrates that the implementation of this embedding detection is robust enough to handle a subtly different sentence pair: in this case, the sentence structure “*Hanssen, who sold...*” indicates that the reader should understand that it is already proven (elsewhere) that Hanssen has sold secrets.

S: *The New York Times reported that Hanssen, who sold FBI secrets to the Russians, could face the death penalty.*

T: *Hanssen sold FBI secrets to the Russians.*

Our system identifies the following verb frames in S and T (using the full parse data provided by Collins’ parser to connect “*who*” to “*Hanssen*”):

S-A: [*The New York Times*]/ARG0 reported [*that Hanssen, who sold FBI secrets to the Russians...*]/ARG1

S-B: [*Hanssen*]/ARG0 sold [*FBI secrets*]/ARG1 to [*the Russians*]/ARG3

T-A: [*Hanssen*]/ARG0 sold [*FBI secrets*]/ARG1 to [*the Russians*]/ARG3

During preprocessing, the system does not detect an embedding of “*sold*” in “*reported*”, and so does not attach the attribute and value “MODALITY: REPORTED” to the verb “*sold*” in S. During the subsumption check, the “*sold*” verbs now match, as both are considered factual.

*7.1.2.3 Modifier (Mod) Module* In the experimental results summarized below, adding the Modifier module does not improve performance, because in all the PARC examples involving modifiers, a different modifier in S and T corresponds to a negative label.

However, the modifier module will correctly analyze the following sentence pair, which is a reordered pair from the PARC corpus:

S: *All soldiers were killed in the ambush.*

T: *Many soldiers were killed in the ambush.*

The default rule — non-identical argument modifiers cause subsumption to fail — is incorrect here, as S entails T. The Modifier module correctly identifies the entailment of “many” by “all”, and subsumption will succeed.

**7.1.3 Experimental Results: PARC** We present the results of the experiments on the PARC data set with two differently weighted inference formulations, which show that for the PARC data set, a formulation where weights on higher levels in the hierarchy described in Sec. 5 dominate performed better than a formulation favoring lower-level hierarchy elements.

Table 1 shows the overall performance of the system on the PARC data set for two sets of weights on the final inference formulation, one suited to the PARC data set and one suited to the PASCAL data set. In both cases the weights were determined empirically and the KB of rewrite rules was enabled.

**Table 1.** System’s performance on the PARC corpus with different optimization weighting schemes.

Corpus Version	Baseline (LLM)	Weighting Scheme	
		PARC scheme	PASCAL scheme
Perfect	59.38	88.21	57.81
Full	61.84	77.63	56.58

The breakdown of the contributions of different modules are presented in Table 2 and 3. We compare the baseline and the full entailment system (SRL+Deep). Table 2 presents the evaluation of the system with and without the KB inference rules, comparing the baseline and the full entailment system (SRL+Deep) when SRL is perfect, i.e. considering only examples on which our SRL tool gives correct annotation. Table 3 presents the evaluation when the entire dataset is used, including those examples on which the SRL tool makes mistakes.

**Table 2.** System’s performance obtained for the PARC question-answer pairs with perfect SRL. This corresponds to 64 question-answering pairs. N/A indicates that the module could not be used with the corresponding system configuration.

Module	without KB		with KB	
	LLM	SRL + Deep	LLM	SRL + Deep
Base	59.38	62.50	62.50	68.75
+ VP	N/A	62.50	N/A	75.00
+ DA	N/A	71.88	N/A	82.81
+ Mod	N/A	71.88	N/A	82.81

**Table 3.** System’s performance obtained for the PARC question-answer pairs on the full data set. N/A indicates that no knowledge information could be used.

Module	without KB		with KB	
	LLM	SRL+Deep	LLM	SRL+Deep
Base	61.84	61.84	64.47	67.11
+ VP	N/A	60.52	N/A	69.74
+ DA	N/A	68.42	N/A	77.63
+ Mod	N/A	68.42	N/A	77.63

The results obtained with perfect semantic argument structure (perfect SRL) are provided here to illustrate the advantages of the hierarchical approach, as noise introduced by SRL errors can obscure the effects of the different levels of the hierarchical representation/subsumption. The system behaves consistently, showing improvement as additional hierarchical structure and additional semantic analysis resources are added. These results validate the benefit of the hierarchical approach.

## 7.2 Experiments Using the PASCAL Data Set

In this experiment we tested our approach on a set of sentence pairs developed for the PASCAL challenge. As the system was designed to test for semantic entailment, the PASCAL test data set is well suited, being composed of 800 source — target sentence pairs, with a truth value indicating whether the source logically entails the target. The set is split into various tasks: CD (Comparable Documents), IE (Information Extraction), IR (Information Retrieval), MT (Machine Translation), PP (Prepositional Paraphrases), QA (Question Answering), and RC (Reading Comprehension). The typical sentence size varies from 11 (IR task) to 25 (MT task) words.

**7.2.1 Experimental Results: PASCAL** Table 4 shows the system’s performance. We first used the optimization function weighting scheme that was most successful for the PARC dataset (labeled “PARC scheme”); the resulting performance was poor. Error analysis revealed many mistakes at the predicate-argument level, probably due to the greater complexity of the PASCAL corpus. This led us to try other weighting schemes that emphasized lower-level information such as lexical tokens and word order; the most successful of these yielded the results labeled “PASCAL scheme”.

The system using the PASCAL weighting scheme does significantly better than the baseline LLM system, which shows evidence in some categories of negative correlation between matched keywords in the source and target sentences and entailment of T by S. Both do significantly better than the system using the PARC weighting scheme. Clearly, for the PASCAL corpus, higher level information (such as predicate-argument structure) is unhelpful to the system (reasons for this are suggested in the Error Analysis and Discussion sections below). How-

**Table 4.** System’s performance obtained for each experiment on the PASCAL corpus and its subtasks.

System	Overall	Task						
		CD	IE	IR	MT	PP	QA	RC
PARC Scheme	51.38	54.67	50.00	51.11	50.83	54.00	50.00	50.00
PASCAL Scheme	58.63	82.67	54.17	53.33	51.67	50.00	53.85	53.57
LLM	55.00	84.00	45.83	44.44	50.00	42.00	53.85	48.57

ever, the optimization formulation is robust enough to allow reasonably good performance even on this corpus.

**7.2.2 Error Analysis** Error analysis of the system’s output for the PASCAL test corpus revealed that the high-level semantic resources had a high error rate. These errors lead to missing or misleading information that is then included in the final subsumption step.

This section gives some examples of these errors, and where possible, some statistics relating to the frequency of their occurrence.

*7.2.2.1 Rewrite Rules* One major problem is the very incomplete coverage of our Knowledge Base of rewrite rules relating verb phrases. There are many entailment pairs in which very different verbs or verb phrases must be identified as having the same meaning to correctly determine subsumption, such as:

S: *Vanunu converted to Christianity while in prison, and has been [living]/MAINVERB in an anglican cathedral in Jerusalem since his release on April 21.*

T: *A convert to Christianity, Vanunu has [sequestered himself]/MAINVERB at a Jerusalem church since he was freed on April 21.*

Our system relies on our Knowledge Base of rewrite rules to allow modification of the source sentence and therefore allow it to match the corresponding part of the target sentence, and this incompleteness is a significant problem.

We are confident that the new VerbNet resources soon to be available will provide a good resource for more complete rewrite rules.

*7.2.2.2 Semantic Role Labeling* The high-level resources depend heavily on the Semantic Role Labeler correctly identifying predicates (verbs) and arguments. While the SRL’s performance on each element of a given verb frame is good – e.g. approaching 90ARG0 and ARG1 (similar to Subjects and Direct Objects) – it often makes mistakes on one or more elements in a given sentence. Moreover, when arguments are complex noun phrases (containing more than one base Noun Phrase), the system’s higher-level analysis modules use heuristics to find the main entity of the argument; these heuristics introduce additional errors.

Study of a 10% sample of the test corpus suggests that SRL or the supporting argument analysis module makes a ‘significant’ error (i.e., one that will interfere with subsumption) on about 58

The following examples indicate the types of errors SRL and its supporting module makes.

*Incorrect Verb:*

In the next example, the SRL tags 'articulate' in T as a verb, and generates a redundant verb frame.

S: *Clinton [is]/MAINVERB a very charismatic person.*

T: *Clinton [is]/MAINVERB [articulate]/MAINVERB.*

This adds incorrect predicate-argument information to the Target graph and interferes with subsumption.

*Missing Argument:*

In the following example, SRL correctly identifies all constituents of the verb frames for 'said' and 'wearing' in both S and T. However, SRL finds the subject of "carry" in T but not in S:

S: *Witnesses said the gunman was wearing gray pants and a tan jacket and was [carrying]/MAINVERB [a gray bag]/ARG1.*

T: *Witnesses said [the gunman]/ARG0 was wearing gray pants and a tan jacket COMMA and [carrying]/MAINVERB [a gray bag]/ARG1.*

This leads to high-level information being absent from S that is present in T, and interferes with subsumption.

*Incorrect Argument Type:*

In the next example, SRL mislabels a key argument of the main verb in T:

S: *[Satomi Mitarai]/ARG1 [died]/MAINVERB of blood loss.*

T: *[Satomi Mitarai]/AM\_EXT [bled]/MAINVERB to death.*

This leads to mismatched high-level information in S and T, interfering with subsumption.

*Incorrect Argument Simplification:*

In the next example, the preprocessing module that simplifies complex arguments oversimplifies a key argument for the verb frame '[threatens to] dismiss':

S: *[Israeli Prime Minister Ariel Sharon]/ARG0 threatened to [dismiss]/MAINVERB [cabinet ministers]/ARG1 who don 't support his plan to withdraw from the Gaza Strip.*

T: *[Israeli Prime Minister Ariel Sharon]/ARG0 threatened to [fire]/MAINVERB [cabinet opponents]/ARG1 of his Gaza withdrawal plan .*

This oversimplification results in mismatched arguments in S and T, and impedes subsumption.

**7.2.2.3 Verb Phrase Compression** The verb phrase compression module requires a good mapping from compressible verb phrase structures – such as auxiliary and modal verb structures, and structures where one verb modifies another (like “manages to enter”) – to their compressed counterparts.

As with the PARC corpus, this module does not have a marked effect in isolation; rather, it potentially enables better subsumption between verb phrases that differ mainly in structure. In the last example, for instance, the VP module replaces the structure “threaten to dismiss” with the verb “dismiss” plus an associated MODALITY attribute “POTENTIAL”.

**7.2.2.4 Discourse Analysis** The Discourse Analysis module detects embedding structures, and is useful when the embedding structure affects the truth value of

the embedded predicate. However, there are few such cases in the PASCAL corpus: for example, all “reported” structures are considered true (whereas PARC considers them “unknown”, which our system treats as “false”). For example:

S: *A spokeswoman said there were no more details available.*

T: *No further details were available.*

In the PARC corpus, this sentence pair would get the label “UNKNOWN”; in PASCAL, it is considered “TRUE”. As such, there is no advantage in activating the DA module over simply assuming “TRUE”, as the “reported” embedding is the predominant case, with few other embedding constructions.

*7.2.2.5 Modifier Analysis* The Modifier Analysis module works at the argument level, and is mainly of use when the two sentences being compared are similar. There are very few cases in the PASCAL corpus where such distinctions are important (i.e., when an argument subtype differs in otherwise identical predicate-argument pattern in S and T). Hence, this module has little effect.

The following examples show how the PASCAL weighted optimization model compares to the baseline LLM system.

#### *Example 1*

In this example, LLM matches the majority of words in T with words in S, and gives it the incorrect label “TRUE”. However, the weighted optimization function takes into account other low level information such as word order, and correctly identifies this example as a case where S does not entail T.

The weighting scheme for the PASCAL data set does not give special weight to elements such as numbers; this can result in false positives:

#### *Example 2*

S: *“Jennifer Hawkins is the 21-year-old beauty queen from Australia.”*

T: *“Jennifer Hawkins is Australia’s 20-year-old beauty queen.”*

Our system matches almost all the key words in T with those in S; as numbers do not carry more weight than other word elements, our system allows subsumption, resulting in a false positive. LLM makes the same error.

## **7.3 Discussion**

The results obtained for the different corpora, and the need for different weighting schemes for the optimization function used to resolve subsumption, indicate that the system can adapt to different corpora if the correct weights are learned. At present, these weights are not learned, but set by trial and error; devising appropriate learning mechanisms must be the next focus of research.

This section describes the way the optimization function is set up and how the weighting schemes differ for the two corpora.

**7.3.1 The Optimization Function** The hierarchical optimization function described in Sec. 5 is realized as three levels in these experiments:

1. Full predicate-argument information;
2. An intermediate level of information, using phrase level nodes in the concept graph representation;
3. A basic level of information using word-level information.

The function optimized in the inference step of the algorithm represents with weighted variables both the attributes and edges that encode this hierarchical information and the rules applied by the various knowledge modules.

The optimization problems formulated by the system for the two data sets are of the same form as they represent all three levels in each. However, we found empirically that different weights for some of the variables gave the best performance for the two data sets (meaning the weights on the variables associated with rule application and levels of representation).

While these weights can be learned, we got the best results from setting them by hand. For the PARC data set, weights were balanced between all three levels, while for the PASCAL data set, the weights for the third level dominated.

**7.3.2 The Need for Different Weighting Schemes** The need for the different weighting schemes is due to the greater complexity of the sentences in the PASCAL corpus, which have a number of relevant characteristics:

1. Many sentences are more complex in structure, with arguments of predicates widely separated from their predicates; this reduces SRL performance.
2. Many sentences have multiple predicates; particularly when T has multiple predicates, this increases the chance that an SRL error is made and therefore impedes entailment.
3. Many sentence pairs require verb phrase recognition and paraphrasing. Rules for making the appropriate substitution are hard to generate and apply, and SRL performance on recognizing verb phrase predicates is poorer than on single-verb predicates.
4. Many sentence pairs require complex “world knowledge”, i.e. reasoning resources that extend beyond paraphrasing and even simple predicate-level inference ( $A \text{ left } B \Rightarrow A \text{ was in/at } B$ ).

The error analysis detailed above indicates that the elements of our system that yield high-level information about predicates and arguments are not yet mature enough to handle many of the complexities of the PASCAL data set. It makes sense, then, that in the optimization formulation for the PASCAL data set, variables relating to the rules/resources from level 1 are not very significant, and those representing more general (low-level) features are more useful; in the formulation for the PARC data set, however, the variables relating to stricter, higher-level features/rule applications are more relevant (given that the sentences are typically very similar, and the amount of information to be gained from lower-level features is small).

## 7.4 Summary

The entailment system is flexible enough to handle two very different corpora by using different weighting schemes for the optimization function that resolves

subsumption. In effect, the system emphasizes lower-level information (such as lexical tokens and word order) for the harder PASCAL corpus, as the accuracy and coverage of higher-level information (such as the predicate-argument structure) is poor. For the simpler PARC corpus, higher level resources perform well and provide useful information; the weighting scheme for the optimization function for this corpus reflects this increased relevance.

Future work will involve improving the knowledge base using resources such as VerbNet, learning weights for the optimization functions via machine learning techniques, and improving the accuracy of key system resources. These should improve the value of the higher-level information and allow efficient optimization of the coefficients in the hierarchical optimization function, improving performance on harder corpora such as PASCAL.

## 8 Conclusions and Future Work

This paper presents a principled, integrated approach to *semantic entailment*. We developed an expressive knowledge representation that provides a hierarchical encoding of structural, relational and semantic properties of the text and populated it using a variety of machine learning based tools. An inferential mechanism over a knowledge representation that supports both abstractions and several levels of representations allows us to begin to address important issues in abstracting over the variability in natural language. Our preliminary evaluation is very encouraging, yet leaves a lot to hope for. Improving our resources and developing ways to augment the KB are some of the important steps we need to take. Beyond that, we intend to tune the inference algorithm by incorporating a better mechanism for choosing the appropriate level at which to require subsumption. Given the fact that we optimize a linear function, it is straight forward to learn the cost function. Moreover, this can be done in such a way that the decision list structure is maintained.

**Acknowledgments** We thank Dash Optimization for a free academic license to the Xpress-MP software. This work was supported by the Advanced Research and Development Activity (ARDA)s Advanced Question Answering for Intelligence (AQUAINT) program, NSF grant ITR-IIS-0085980 and ONRs TRECC and NCASSR programs.

## References

1. Dagan, I., Glickman, O.: Probabilistic textual entailment: Generic applied modeling of language variability. In: Learning Methods for Text Understanding and Mining, Grenoble, France (2004)
2. Schubert, L.K.: From english to logic: Context-free computation of 'conventional' logical translations. In Grosz, B.J., Sparck Jones, K., Webber, B.L., eds.: Natural Language Processing. Kaufmann, Los Altos, CA (1986)
3. Moore, R.C.: Problems in logical form. In Grosz, B.J., Sparck Jones, K., Webber, B.L., eds.: Natural Language Processing. Kaufmann, Los Altos, CA (1986)

4. Hobbs, J.R., Stickel, M., Martin, P., Edwards, D.: Interpretation as abduction. In: Proc. of the 26th Annual Meeting of the Association for Computational Linguistics (ACL). (1988) 95–103
5. Cumby, C.M., Roth, D.: Learning with feature description logics. In Matwin, S., Sammut, C., eds.: The 12th International Conference on Inductive Logic Programming (ILP-02), Springer (2003) 32–47 LNAI 2583.
6. Lloyd, J.W.: Foundations of Logic Programming. Springer (1987)
7. Kingsbury, P., Palmer, M., Marcus, M.: Adding semantic annotation to the Penn treebank. In: Proc. of the 2002 Human Language Technology conference (HLT), San Diego, CA (2002)
8. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: Description Logic Handbook. Cambridge (2003)
9. Even-Zohar, Y., Roth, D.: A sequential model for multi class classification. In: Proc. of the 2001 Conference on Empirical Methods for Natural Language Processing (EMNLP). (2001) 10–19
10. Collins, M.: Head-driven Statistical Models for Natural Language Parsing. PhD thesis, Computer Science Department, University of Pennsylvania, Philadelphia (1999)
11. Punyakanok, V., Roth, D., Yih, W., Zimak, D.: Semantic role labeling via integer linear programming inference. In: Proc. of the 20th International Conference on Computational Linguistics (COLING), Geneva, Switzerland (2004)
12. Punyakanok, V., Roth, D., Yih, W.: The necessity of syntactic parsing for semantic role labeling. In: Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI). (2005)
13. Li, X., Morie, P., Roth, D.: Identification and tracing of ambiguous names: Discriminative and generative approaches. In: Proc. of the 19th National Conference on Artificial Intelligence (AAAI). (2004)
14. Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press (1998)
15. Lin, D., Pantel, P.: DIRT: discovery of inference rules from text. In: Proc. of ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2001. (2001) 323–328
16. Roth, D., Yih, W.: A linear programming formulation for global inference in natural language tasks. In: Proceedings of CoNLL-2004. (2004) 1–8
17. Roth, D., Yih, W.: Integer linear programming inference for conditional random fields. In: Proceedings of the International Conference on Machine Learning (ICML). (2005)
18. Durme, B.V., Huang, Y., Kupsc, A., Nyberg, E.: Towards light semantic processing for question answering, HLT Workshop on Text Meaning (2003)
19. Moldovan, D., Clark, C., Harabagiu, S., Maiorano, S.: Cogex: A logic prover for question answering. In: Proc. of HLT-NAACL 2003. (2003)
20. Thompson, C., Mooney, R., Tang, L.: Learning to parse NL database queries into logical form. In: Workshop on Automata Induction, Grammatical Inference and Language Acquisition. (1997)
21. Glickman, O., Dagan, I., Koppel, M.: A probabilistic classification approach for lexical textual entailment. In: Proc. of AAAI 2005. (2005)
22. Raina, R., Ng, A., Manning, C.: Robust textual inference via learning and abductive reasoning. In: Proc. of AAAI 2005. (2005)
23. Punyakanok, V., Roth, D., Yih, W.: Natural language inference via dependency tree mapping: An application to question answering. Technical Report No. UIUCDCS-R-2004-2443, UIUC Computer Science Department (2004)