Semantic Parsing Using Content and Context: A Case Study from Requirements Elicitation

Reut Tsarfaty
Weizmann Institute
Rehovot, Israel

Eli Pogrebetzky
Interdisciplinary Center
Herzliya, Israel

Guy Weiss
Weizmann Institute
Rehovot, Israel

Yaarit Natan
Weizmann Institute
Rehovot, Israel

Smadar Szekely
Weizmann Institute
Rehovot, Israel

David Harel
Weizmann Institute
Rehovot, Israel

Abstract

We present a model for automatic semantic analysis of requirements elicitation documents. Our target semantic representation employs *live sequence charts*, a multi-modal visual language for scenario-based programming, which has a direct translation into executable code. The architecture we propose integrates sentence-level and discourse-level processing in a generative probabilistic framework for the analysis and disambiguation of individual sentences in context. We empirically show that the discourse-based model consistently outperforms a sentence-based model on constructing a system that reflects all the static (entities, properties) and dynamic (behavioral scenarios) requirements in the document.

1 Introduction

Requirements elicitation is a process whereby a system analyst gathers information from a stakeholder about a desired software to be implemented. The knowledge collected by the analyst may be *static*, referring to the conceptual model (the entities, their properties, the possible values) or *dynamic*, referring to the behavior that the system should follow (who does what to whom, when, how, etc). A stakeholder interested in the software typically has a specific (static and dynamic) domain in mind, but he or she cannot necessarily prescribe any formal models or code artifacts. The term *requirements elicitation* we use here refers to a piece of discourse in natural language by means of which a stakeholder communicates their desiderata to the system analyst.

The role of a system analyst is to understand the different requirements and transform them into formal constructs (formal diagrams or executable code). Moreover, the analyst needs to consolidate the different pieces of information to uncover a single shared domain. Studies in software engineering aim to develop intuitive symbolic systems with which human agents can encode requirements that would then be unambiguously translated into a formal model (Fuchs and Schwitter, 1995; Bryant and Lee, 2002). More recently, Gordon and Harel (2009) defined a natural fragment of English that can be used for specifying requirements, which can be effectively parsed and interpreted using a chart parser. However, the grammar that underlies this fragment is highly ambiguous and all disambiguation needed is conducted by a human agent by hand. In general, it is repeatedly shown that the more natural the formal system is, the harder it is to develop an unambiguous translation mechanism (Kuhn, 2014).

In this paper we accept the ambiguity of requirements descriptions as a premise, and aim to answer the following question: can we automatically recover a formal representation of the complete system, which *best* reflects the human-perceived interpretation of the entire document? Recent advances in natural language processing, with an eye to semantic parsing (Zettlemoyer and Collins, 2005; Liang et al., 2011; Artzi and Zettlemoyer, 2013; Liang and Potts, 2014), use different formalisms and various kinds of signals for semantic parsing. In particular, the model of Lei et al. (2013) induces input parsers from format descriptions. However, rarely do these models take into account an entire document's context.

The key idea we promote here is that discourse context provides substantial disambiguating information for sentences' analysis. We suggest a novel model for integrated sentence-level and discourse-level processing in a joint generative probabilistic model, from which we can recover a representation of the specified system. Our input is given in the simplified, yet highly ambiguous, fragment

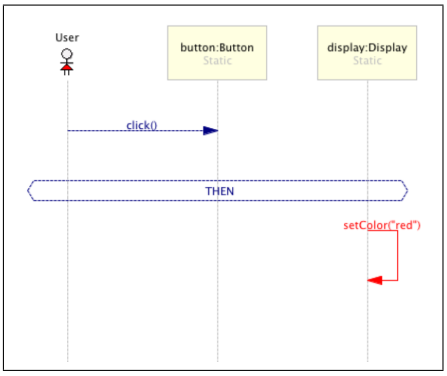

Figure 1: An LSC scenario: "When the user clicks the button, the display color must change to red."

of English of Gordon and Harel (2009). The output, in contrast, is an unambiguous and well-formed sequence of formal constructs that represent the dynamic behavior of the system, called *live sequence charts* (LSC) (Damm and Harel, 2001; Harel and Marelly, 2003), tied to a single shared code-base ontology, called a *system model* (SM). Our solution then takes the form of a *hidden markov model* where emission probabilities reflect the grammaticality and interpretability of textual requirements via a *probabilistic grammar*, and transition probabilities model the overlap between SM snapshots of the shared domain. Using efficient viterbi decoding, we search for the best sequence of domain snapshots that has most likely generated the document. We empirically show that such an integrated model consistently outperforms a sentence-based model learned from the same data.

The remainder of this document is organized as follows. In Section 2 we describe the task, and spell out our assumptions concerning the input and the output. In Section 3 we present our target semantic represention, and a specially tailored notion of *grounding* requirements in a code-base. In Section 4 we develop our sentence-based and discourse-based models, and in Section 5 we compare the models on various case studies. Finally, in Section 6 we summarize and conclude.

2 Parsing Requirements Elicitation Documents: Task Description

There is an inherent discrepancy between the input and the output of software engineering processes. The input, system requirements, is usually specified in a natural, informal, language. The output, the system, is ultimately implemented in a formal unambiguous programming language. Can we automatically recover a formal representation of a complete system from a set of requirements? In this work we explore this challenge empirically.

The Input. We assume a *scenario-based programming* paradigm (a.k.a *behavioral programming* (BP) (Harel et al., 2012)) in which system development is seen as a process whereby humans describe the expected behavior of the system by means of "short-stories", formally called *scenarios* (Harel, 2001). We further assume that a given requirements document describes exactly one system, and that each sentence in the document describes a single, possibly complex, scenario. The requirements we aim to parse are given in a simplified form of English (specifically, the English fragment described in Gordon and Harel (2009)). Contrary to strictly formal specification languages, this simplified variant of English allows for an open domain lexicon and exhibits extensive syntactic and semantic ambiguity.[1]

The Output. Our target semantic representation employs *live sequence charts* (LSC), a diagrammatic formal language for scenario-based programming (Damm and Harel, 2001). Formally, LSCs are an extension of the well-known UML message sequence diagrams (Harel and Maoz, 2006) and they have a direct translation into executable code (Harel et al., 2007).[2] Using LSC diagrams for software modelling enjoys the advantages of being easily learnable (Harel and Gordon-Kiwkowitz, 2009), intuitively interpretable (Eitan et al., 2011) and straightforwardly amenable to execution (Harel et al., 2002) and verification (Harel et al., 2013). The LSC language is particularly suited for representing natural language requirements since its basic formal constructs, *scenarios*, nicely align with *events*, the primitive objects of Neo-Davidsonian Semantics (Parsons, 1990).

Live Sequence Charts and Code Artifacts. A live sequence chart (LSC) is a diagram that describes a possible or necessary run of a specified system. In a single LSC diagram, entities are represented as vertical lines called *lifelines* and interactions between entities are represented using horizontal arrows between lifelines called *messages*,

[1] Formally, this variant may be viewed as a CNL of degree P2 E3 N4 S4 with properties C,F,W,A (Kuhn, 2014, pp 6-12).

[2] It can be shown that the execution semantics of the LSC language is embedded in a fragment of a branching temporal logic called CTL* (Kugler et al., 2005).

connecting a sender to a receiver. Messages may refer to other entities (or properties of entities) as arguments. Time in LSCs precedes from top to bottom, imposing a partial order on the execution of messages. LSC messages can be *hot* (red, "must happen") or *cold* (blue, "may happen"). A message may have an execution status, which designates it as *monitored* (dashed arrow, "wait for") or *executed* (full arrow, "do"). The LSC language also encompasses conditions and control structures, and it further allows defining requirements in terms of negation of charts. Figure 1 illustrates the LSC for the scenario "When the user clicks the button, the display color must change to red." The respective SM would be a code-base hierarchy containing the classes User, Button, Display, with method Button.click and a property Display.color.

3 Formal Settings

In the text-to-code generation task we aim to implement a function $f : \mathcal{D} \rightarrow \mathcal{M}$ where $D \in \mathcal{D}$ is a piece of discourse consisting an ordered set of requirements $D = d_1, d_2 ... d_n$, and $f(D) = M \in \mathcal{M}$ is a code-base hierarchy that grounds the semantic interpretation of D, i.e., $M \triangleright \mathbf{sem}(d_1, ..., d_n)$.

Both D, M are complex objects, that we now define formally. Next we describe the semantic interpretation function ($\mathbf{sem}(.)$) and we then spell out the definition of grounding ($\triangleright$).

Surface Structures: Let Σ be a finite lexicon and let $\mathcal{L}_{req} \subseteq \Sigma^*$ be a language fragment for specifying requirements. We assume the sentences in $\mathcal{L}_{req}$ have been generated by a context free grammar $G = \langle \mathcal{N}, \Sigma, S \in \mathcal{N}, \mathcal{R} \rangle$ where $\mathcal{N}$ is a set of non-terminals, Σ is the aforementioned lexicon, $S \in \mathcal{N}$ is a start symbol and $\mathcal{R}$ is a set of context-free rules $\{A \rightarrow \alpha | A \in \mathcal{N}, \alpha \in (\mathcal{N} \cup \Sigma)^*\}$. For each utterance $u \in \mathcal{L}_{req}$ we can find a sequential application of rules that generates it $u = r_1 \circ ... \circ r_k$; $\forall i : r_i \in \mathcal{R}$. These derivations are graphically depicted as parse trees, with u defining the sequence of tree terminals in the leaves. We define $\mathcal{T}_{req}$ to be the set of trees strongly generated by G, and an auxiliary yield function $yield : \mathcal{T}_{req} \rightarrow \mathcal{L}_{req}$ returning the leaves of a tree. Different parse-trees can generate the same utterance, so the task of analysing the surface structure of an utterance $u \in \mathcal{L}_{req}$ is modeled via a function $syn : \mathcal{L}_{req} \rightarrow \mathcal{T}_{req}$ that returns the correct, human-perceived, parse of u.

Semantic Structures: Our target semantic representation of a requirement $d \in \mathcal{L}_{req}$, denoted here $sem(d)$ is a diagrammatic structure called a *live sequence chart* (LSC), an event-based formal representation for specifying requirements.

Let us assume that L is a dictionary of entities (lifelines), A is a dictionary of actions, P is a dictionary of attribute names and V a dictionary of attribute values. The set of simple events in the LSC formal system is defined as follows:

$$E_{active} \subset L \times A \times L \times (L \times P \times V)^*$$

$$\times \{hot, cold\} \times \{executed, monitored\}$$

where $e = \langle l_1, a, l_2, \{l_i : p_i : v_i\}_{i=3}^k, temp, exe \rangle$ and $l_i \in L, a \in A, p_i \in P, temp \in \{hot, cold\}$, $exe \in \{executed, monitored\}$. The event e is called a *message* in which an action a is carried over from a sender l_1 to a receiver l_2.[3] The set $\{l_i : p_i : v_i\}_{i=3}^k$ depicts a set of attribute:value pairs provided as arguments to action a. The temperature $temp$ marks the modality of the action (may, must), and the status exe distinguishes actions to be taken from actions to be waited for.

An event e can also be a stative event, called a *condition*, in which a logical expression is being evaluated over a set of property:value pairs:

$$E_{stative} \subset Exp \times (L \times P \times V)^*$$

$$\times \{hot, cold\} \times \{executed, monitored\}$$

Specifically, $e = \langle exp, \{l : p : v\}_{i=0}^k, temp, exe \rangle$ is a stative event to be evaluated, where $l_i \in L, p_i \in P, v_i \in V, temp \in \{hot, cold\}$, $exe \in \{executed, monitored\}$. The condition $exp \in Exp$ is a first-order logic formula using the usual operators ($\vee, \wedge, \rightarrow, \neg$). The set $\{l : p : v\}_{i=0}^k$ depicts a (possibly empty) set of attribute:value pairs that participates as predicates in exp. Executing a condition, that is, evaluating the logical expression specified by exp, also has a modality (may/must) and an execution status (performed/waited for).

The LSC language further allows us to define arbitrarily complex events by combining partially ordered sets of events with control structures.

$$E_{complex} \subset N \times E_{stative} \times$$

$$\{\langle E_c, < \rangle | \langle E_c, < \rangle \textit{ is a poset } \}$$

[3] The LSC language distinguishes static lifelines from dynamically bound lifelines. For brevity, we omit this from the formal description of events, and simply assert that it may be one of the properties of the relevant lifeline.

N is a set of non-negative integers, $E_{stative}$ is a set of stative events as described above, and each element $\langle E_c, < \rangle$ is a partially ordered set of events. This structure allows us to derive three kinds of control structures:

- $e = \langle \#, \emptyset, \langle E, < \rangle \rangle$ is a loop in which $\langle E, < \rangle$ is executed $\#$ times.
- $e = \langle 0, cond, \langle E, < \rangle \rangle$ is a conditioned execution. If $cond$ holds, $\langle E, < \rangle$ is executed.
- $e = \langle \#, \{cond\}_{i=1}^{\#}, \{\langle E_c, < \rangle\}_{i=1}^{\#} \rangle$ is a switch: in case i, if the condition i holds, $\langle E_c, < \rangle_i$ is executed.

Definition 1 (LSC) An LSC is a partially ordered set of events $c = \langle E, < \rangle$ where

$$\forall e \in E : e \in E_{active} \vee e \in E_{stative} \vee e \in E_{complex}$$

Grounded Semantics: The information represented in the LSC provides the recipe for a rigorous construction of the code-base that will implement the program. This code-base is said to *ground* the semantic representation. In this work we suggest to explicitly model this code-base as the document's context. If our target programming language is an OO language such as Java, then the code-base will include the entities, methods, and properties that are minimally required for executing the LSCs. We refer to this code-base using as system model, or in short SM, as defined below.

Definition 2: (SM) Let L_m be a set of implemented objects, A_m a set of implemented methods, P_m a set of arguments and V_m argument values. Additionally we define auxiliary functions $methods : A_m \rightarrow L_m$, $props : P_m \rightarrow L_m$ and $values : V_m \rightarrow L_m \times P_m$ for identifying the entity which contains the implementation of the method, the entity that contains the property, and the entity attribute that bears that value, respectively. A system model is a tuple m representing the implemented architecture.

$$m = \langle L_m, A_m, P_m, V_m, methods, props, values \rangle$$

Analogously to *interpretation* functions in logic and natural language semantics we assume here an *implementation* function denoted $[[.]]$ which maps each formal entity in the LSC semantic representation to its instantiation in the code-base. Using this function we define a notion of grounding that captures the fact that a certain code-base satisfies the requirements of an LSC c.

Definition 3(a): (Grounding) Let $\mathcal{M}$ be the set of system models and let $\mathcal{C}$ be the set of definable LSC charts. We say that m grounds $c = \langle E, < \rangle$ and write $m \triangleright c$ iff $\forall e \in E : m \triangleright e$. Where:

- if $e \in E_{active}$ then:
  $m \triangleright e \Leftrightarrow$
  $$[[l_1]], [[l_2]] \in L \;\&\; [[a]] \in methods([[l_2]]) \;\&\; \forall i : \langle l : p : v \rangle_i \Rightarrow [[l]] \in L_m \& [[p]] \in props[[[l]] \& v \in values([[l]], [[p]])$$
- if $e \in E_{stative}$ then:
  $m \triangleright e \Leftrightarrow$
  $\forall i : \langle l : p : v \rangle_i \Rightarrow [[l]] \in L_m \& [[p]] \in props[[[l]] \& v \in values([[l]], [[p]])$
- if $e = \langle \#, e_s, \langle E_c, < \rangle \in E_{complex}$ then:
  $m \triangleright e \Leftrightarrow m \triangleright e_s \;\&\; \forall e' \in E_c : m \triangleright e'$

We have thus far defined how an SM grounds the semantics of an LSC. In the real-world, however, a requirements document is interpreted as a complete whole, conveying a single shared domain that satisfies all dynamic requirements. Let us assume a requirements document containing n requirements $\mathbf{d} = d_1, d_2, ..., d_n$; $d_i \in \mathcal{L}_{req}$. We assume that $\mathbf{sem}(\mathbf{d})$ is a discourse interpretation function that returns a single semantic representation for the document where identical elements across sentences share the same reference. We finally assume that $\sqcup$ is a unification operation, returning the formal unification of two SMs if such exists, and an empty SM otherwise. We can now define grounding of a sequence of requirements.

Definition 3(b): (Grounding) Let $\mathbf{d}$ be a requirements document and let $M = \langle \mathbf{m}, \sqcup \rangle$ be a sequence of models and a unification operation. We say that $M \triangleright \mathbf{sem}(\mathbf{d})$ iff $\forall i : m_i \triangleright sem(d_i)$ and $((m_1 \sqcup m_2) .... \sqcup m_n) \triangleright \mathbf{sem}(d_1, ..., d_n)$.

The discourse semantic interpretation provided by $\mathbf{sem}$ can be as simple as asserting that all elements that have the same string name refer to the same element (entity, action, etc), and it can be as complex as taking into account synonyms ("*clicks* the button" and "*presses* the button"), anaphora ("when the user clicks the button, *it* changes colour"), binding ("when the user clicks *any* button, *this* button is highlighted"), and so on. In this work, we assume a simple discourse interpretation function where entities, methods, properties and values that are referred to using the same string refer to the same elements in the code-base.

This simple assumption already holds a substantial amount of disambiguating information concerning individual requirements. For example, assuming we have seen a "click" method over a "button" object in sentence i, it may help us disambiguate future attachment ambiguity, favoring structures where a "button" is attached to "click" over other attachment alternatives. Our goal is then to model discourse-level context for supporting accurate analysis of individual requirements.

4 Probabilistic Modeling

4.1 Sentence-Based Modeling

The task of our sentence-based model is to learn a function that maps each requirement sentence to its correct LSC diagram and SM snapshot. In a nutshell, we do this via a (partially lexicalized) probabilistic context-free grammar augmented with a semantic interpretation function.

More specifically, given a discourse $D = d_1...d_n$ we think of each d_i as having been generated by a CFG G. The syntactic analysis of d_i may be ambiguous, so we first implement a syntactic analysis function $syn : \mathcal{L}_{req} \rightarrow \mathcal{T}_{req}$ using a probabilistic model that selects the most likely syntax tree t of each d individually. We can simplify $syn(d)$, with d constant with respect to the maximization:

$$\begin{aligned} syn(d) &= argmax_{t \in \mathcal{T}_{req}} P(t|d) \\ &= argmax_{t \in \mathcal{T}_{req}} \frac{P(t,d)}{p(d)} \\ &= argmax_{t \in \mathcal{T}_{req}} P(t,d) \\ &= argmax_{t \in \{t | t \in \mathcal{T}_{req}, yield(t)=d\}} P(t) \end{aligned}$$

We define a probability distribution over trees $t \in \mathcal{T}_{req}$ by augmenting G with a function $P : \mathcal{R} \rightarrow [0,1]$ (for all rules of G it holds that $\sum_\alpha P(A \rightarrow \alpha) = 1$). Because of context-freeness we get that $P(t) = \prod_{r \in der(t)} P(r)$, where $der(t)$ returns the rules that derive t. The resulting probability distribution $P : \mathcal{T}_{req} \rightarrow [0,1]$ defines a probabilistic language model over all requirements in $\mathcal{L}_{req}$.

Syntactic parse trees are complex entities, assigning structures to the flat sequences of words. The principle of compositionality asserts that the meaning of a complex syntactic entity is a function of the meaning of its parts and their mode of combination. We assume a function $sem : \mathcal{T} \rightarrow \mathcal{C}$ mapping trees to semantic constructs in the LSC language. The semantics of a tree $t \in \mathcal{T}_{req}$ is derived compositionally from the interpretation of the rules in the grammar G. We overload the sem notation to define $sem : \mathcal{R} \rightarrow \mathcal{C}$ as a function assigning rules to LSC constructs,[4] with $\hat{\circ}$ merging the resulting sets of events. Our sentence-based compositional semantics is summarized as:

$$sem(u) = sem(syn(u)) = sem(r_1 \circ ... \circ r_n) =$$

$$sem(r_1)\hat{\circ}...\hat{\circ}sem(r_n) = c_1\hat{\circ}...\hat{\circ}c_n = c$$

For a single chart c, one can easily construct an implementation for every entity, action and property in the chart. Then, by design, we get an SM m such that $m \triangleright c$. To construct the SM of the entire discourse in the sentence-based model we simply return $f(d_1, ..., d_n) = \sqcup_{i=1}^{n} m_i$ where $\forall i : m_i \triangleright sem(syn(d_i))$.

4.2 Discourse-Based Modeling

We assume a given document $D \in \mathcal{D}$ and aim to find the most probable system model $M \in \mathcal{M}$ that satisfies the requirements. We assume that M reflects a single domain that the stakeholders have in mind, and we are given ambiguous natural language evidence as elicited discourse in which they convey it. We instantiate this view as a *noisy channel* model (Shannon, 1948), which provides the foundation for many NLP applications, such as speech recognition (Bahl et al., 1983) and machine translation (Brown et al., 1993).

According to the noisy channel model, when a signal is received it does not uniquely identify the message being sent. A probabilistic model is then used to decode the original message. In our case, the signal is the discourse and the message is the overall system model. In formal terms, we want to find a model M that maximises the following:

$$f(D) = argmax_{M \in \mathcal{M}} P(M|D)$$

We can simplify further using Bayes law, where D is constant with respect to the maximisation.

$$\begin{aligned} f(D) &= argmax_{M \in \mathcal{M}} P(M|D) \\ &= argmax_{M \in \mathcal{M}} \frac{P(D|M) \times P(M)}{P(D)} \\ &= argmax_{M \in \mathcal{M}} P(D|M) \times P(M) \end{aligned}$$

We would thus like to estimate two types of probability distributions, $P(M)$ over the source and $P(D|M)$ over the channel.

[4] Here, it suffices to say that sem maps edges in the syntax tree to functions in the API of an LSC editor. For example: $sem(NP \rightarrow DET\ NN) = fCreateObject(DET.sem, NN.sem)$. We specify the function sem completely in the supplementary materials.

Both M and D are structured objects with complex internal structure. In order to assign distribution to events involving such complex structures it is customary to break the complex events down into simpler, more basic events. We know that $D = d_1, d_2, ..., d_n$ is composed of n individual sentences, each representing a certain aspect of the model M. We assume a sequence of *snapshots* of M that correspond to the timestamps $1...n$, that is: $m_1, m_2, ..., m_n \in \mathcal{M}$ where $\forall i : m_i \triangleright sem(d_i)$. The complete SM is given by the union of the different snapshots reflected in different requirements, i.e., $M = \bigsqcup_i m_i$. We then rephrase:

$$\begin{aligned} P(M) &= P(m_1, ..., m_n) \\ P(D|M) &= P(d_1, ..., d_n|m_1, ..., m_n) \end{aligned}$$

These events may be seen as points in a high dimensional space. In actuality, they are too complex and would be too hard to estimate directly. We then define two independence assumptions. First, we assume that a system model snapshot at time i depends only on a fixed k previous snapshots (a stationary distribution). Secondly, we assume that each sentence i depends only on the SM snapshot at time i. We now get:

$$\begin{aligned} P(m_1...m_n) &\approx \textstyle\prod_i P(m_i|m_{i-1}...m_{i-k}) \\ P(d_1...d_n|m_1...m_n) &\approx \textstyle\prod_i P(d_i|m_i) \end{aligned}$$

Furthermore, assuming bi-gram transitions, our objective function is now represented as follows:

$$f(D) = argmax_{M \in \mathcal{M}} \prod_{i=1}^{n} P(m_i|m_{i-1}) P(d_i|m_i)$$

Note that m_0 may be empty if the system is implemented from scratch, and non-empty if the requirements assume an existing code-base, which makes $p(m_1|m_0)$ a non-trivial starting point.

4.3 Training and Inference

Our model is in essence a Hidden Markov Model in which state-transition probabilities model transitions between SM snapshots and emission probabilities model the verbal description of each state. To implement this, we need to implement two different algorithms. A decoding algorithm that searches through all possible state sequences, and a training algorithm that can automatically learn the values of the, still rather complex, parameters $P(m_i|m_{i-1}), P(d_i|m_i)$ from corpora.

$$f(D) = \underbrace{argmax_{M \in \mathcal{M}}}_{decoding} \prod_{i=1}^{n} \underbrace{P(m_i|m_{i-1}) P(d_i|m_i)}_{training}$$

Training: We assume a supervised training setting in which we are given a set of examples annotated by a human expert. For instance, these can be requirements an analyst has formulated and encoded using an LSC editor, providing disambiguating information by hand.

We are provided with a set of pairs $\{D_i, M_i\}_{i=1}^{n}$ containing n documents, where each of the pairs in $i = 1..n$ is represented as a tuple-set $\{d_{ij}, t_{ij}, c_{ij}, m_{ij}\}_{j=1}^{n_i}$. For all i, j it holds that $t_{ij} = syn(d_{ij})$, $c_{ij} = sem(t_{ij})$, and $m_{ij} \triangleright sem(syn(d_{ij}))$. The union of the SM snapshots yields the entire model $\sqcup_j m_{i_j} = M_i$, that satisfies the set of requirements $M_i \triangleright \mathbf{sem}(d_{i1}, ..., d_{in_i})$.

(i) Emission Parameters Our emission parameters $P(d_i|m_i)$ represent the probability of a verbal description given an SM snapshot which grounds the semantics of the sentence. A single SM may result from different syntactic derivations of a sentence. We calculate this probability by marginalizing out the syntactic trees that are grounded in the same SM snapshot.

$$\frac{P(d, m)}{P(m)} = \frac{\sum_{t \in \{t | yield(t) = d, m \triangleright sem(t)\}} P(t)}{\sum_{t \in \{t | t \in \mathcal{T}_{req}, m \triangleright sem(t)\}} P(t)}$$

The probability of $P(t)$ is estimated using a treebank PCFG (Charniak, 1996) based on all pairs $\langle d_{ij}, t_{ij} \rangle$ in the annotated corpus. We estimate rule probabilities using maximum-likelihood estimates and use simple smoothing for unknown lexical items using rare-words distributions.

(ii) Transition Parameters Our transition parameters $P(m_i|m_{i-1})$ represent the amount of overlap between the SM snapshots. We look at the current and the previous system model, and aim to estimate how likely the current SM is, given the previous one. There are different assumptions that may underly this probability distribution, reflecting different principles of human communication. We first define a generic estimator as follows:

$$\hat{P}(m_i|m_j) = \frac{gap(m_i, m_j)}{\sum_{m_j} gap(m_i, m_j)}$$

where $gap(.)$ quantifies the information sharing between SM snapshots. Regardless of the implementation of gap, it can be easily shown that $\hat{P}$ is a conditional probability distribution where $\hat{P} : \mathcal{M} \times \mathcal{M} \rightarrow [0, 1]$ and, for all $m_i, m_j, :$ $\sum_{m_j} \hat{P}(m_i|m_j) = 1$. For efficiency, $\mathcal{M}$ is the restricted universe considered via inference, below.

Transition:	$gap(m_{curr}, m_{prev})$
max-overlap	$\frac{\lvert set(m_{curr}) \cap set(m_{prev}) \rvert}{\lvert set(m_{curr}) \rvert}$
max-expansion	$1 - \frac{\lvert set(m_{curr}) \cap set(m_{prev}) \rvert}{\lvert set(m_{prev}) \cup set(m_{curr}) \rvert}$
min-distance	$1 - \frac{ted(m_{prev}, m_{curr})}{\lvert set(m_{prev}) \rvert + \lvert set(m_{curr}) \rvert}$

Table 1: Quantifying the gap between snapshots. $set(m_i)$ is a set of nodes marked by path to root.

We define different gap implementations, reflecting different assumptions about the discourse. Our first assumption here is that different SM snapshots refer to the same conceptual world, so there should be a large overlap between them. We call this the **max-overlap** assumption. A second assumption is that in collaborative communication a new requirement will only be stated if it provides new information, akin to Grice (1975). This is the **max-expansion** assumption. An additional assumption prefers "easy" transitions over "hard" ones, this is the **min-distance** assumption. The different gap calculations are listed in Table 1.

Inference An input document contains n requirements. Our decoding algorithm considers the N-best syntactic analyses for each requirement. At each time step $i = 1...n$ we assume N states representing the semantics of the N best syntax trees. Thus, setting $N = 1$ is equal to a sentence-based model in which for each sentence we simply select the most likely tree according to a probabilistic grammar, and construct a semantic representation for it.

For each document of length n, we assume that our entire universe of system models $\mathcal{M}$ is composed of $N \times n$ snapshots reflecting $N \times n$ most likely analyses of sentences as provided by the probabilistic syntactic model. (As shall be seen shortly, even with this simplifying assumption concerning the size of our universe $\mathcal{M}$, our discourse-based model provides substantial improvements over a sentence-based model).[5]

Our discourse based model is an HMM where each requirement is an observed signal, and each i=1..N is a state representing the SM that grounds the i_th best tree. Because of the Markov independence assumption our setup satisfies the *optimal subproblem* and *overlapping problem* properties and we can use efficient viterbi decoding to exhaustively search through the different state sequences, and find the most probable sequence that has generated the sequence of requirements according to our probabilistic model.

The overall complexity of the decoder for a requirements document with n sentences of which max length is l, a grammar G of size $|G|$, and a constant N is given by the following expression:

$$O(n^3 \times N^2 + n \times l^3 \times |G|^3 + l^2 \times n \times N)$$

We can break this expression down as follows: (i) In $O(n \times l^3 \times |G|^3)$ we generate N best trees for each one of the n requirements, using CKY (Younger, 1967). (ii) In $O(l^2 \times N \times n)$ we create the universe $\mathcal{M}$ based on the N best trees of the n requirements, and (iii) In $O((N \times n)^2 \times n) = O(N^2 \times n^3)$ we decode using Viterbi (1967).

5 Experiments

Goal. We set out to evaluate the accuracy of representing and grounding the semantics of requirements documents in the two new modes of analysis. Our evaluation methodology is as standardly assumed in machine learning and in NLP. Given a set of annotated examples, we divide them into disjoint training set and test set. We train our statistical model on a set of training documents and predict the semantic analysis of the test requirements documents. We then compare the predicted structures to the gold semantic analyses of the test documents in order to empirically quantify our prediction error.

Metrics. Our semantic LSC objects in the system are formally synset trees. Therefore, we can use standard tree evaluation metrics, such as the ParseEval scores (Black et al., 1992). Overall, we evaluate the accuracy of the LSC representations using three metrics:

POS: the percentage of part-of-speech tags predicted correctly.
LSC-F1: the harmonic means of precision and recall on the tree.
LSC-EM: 1 if the predicted tree is an exact match to the gold tree, 0 otherwise.

In the case of SM trees, as opposed to the LSC trees, we can not assume identity of the yield between the gold and parse trees so we cannot use

[5] This restriction is akin to pseudo-likelihood estimation, as in Arnold and Strauss (1991). In pseudo-likelihood estimation, instead of normalizing over the entire set of elements, one uses a subset that reflects only the possible outcomes. Here, instead of summing SM probabilities over all possible sentences in the language, we sum up alternative SM analyses of the observed sentences in the document only. This estimation could also be addressed via, e.g., sampling methods.

System	#Scenarios	avg sentence length
Phone	21	24.33
WristWatch	15	29.8
Chess	18	15.83
Baby Monitor	14	20
Total	68	22.395

Table 2: Seed Gold-Annotated Requirements

ParseEval. Therefore, we implement a distance-based metrics in the spirit of Tsarfaty et al. (2012). Then, to evaluate the accuracy of the SM representation we use two kinds of scores:

SM-F1: the normalized edit distance between the gold and predicted SM trees, subtracted from a unity.

SM-EM: 1 if the predicted SM is an exact match with the gold SM, and 0 otherwise.

Data. We have a small seed of correctly annotated requirements-specification case studies that describe simple reactive systems in the LSC language. Each document is annotated with the correct LSC sequence and is grounded in a java implementation. We use the case studies provided by Gordon and Harel (2009). Table 2 lists the case studies and basic statistics concerning these data.

As our annotated seed is quitte small, it is hard to generalize from it to unseen examples. In particular, we are not guaranteed to have observed all possible structures that are theoretically permitted by the assumed grammar. To cope with that, we create a synthesis set of examples using the grammar of Gordon and Harel (2009) in generation mode, and randomly generate trees $t \in \mathcal{T}_{req}$. We assume a grammar with a fixed functional lexicon and an open set of nouns, verbs and adjectives.

The grammar G we use to generate the synthetic examples clearly *over-generates*. That is, it creates many trees that do not have a sound interpretation. In fact, according to our semantic interpretation function, only 3000 our of 10000 generated examples have a sound semantic interpretation grounded in an SM. Nonetheless, these data allow us to smooth the syntactic distributions that are observed from the seed alone.

Results. Our first experiment aims to evaluate the sentence-based model using our small seed and large set of synthetic examples. Table 3 presents our results for parsing the *Phone* document, our development set. We see that despite the small size of our seed, adding this seed to our training of emission parameters substantially improves results over a model trained on synthetic examples.

Table 4 presents the results of the discourse-based model for $N > 1$ on the *Phone* example. We first present the results of an Oracle experiment, wherein for every requirement we select the highest scoring tree in terms of the LSC-F1. The initial ranking is provided by a PCFG learned from our seed set interpolated with a PCFG learned from the synthetic examples (99/1). This experiment provides an upper bound on the results we can get for each N value. Next we present the results of our discourse based model with PCFG learned from only synthetic trees, but incorporate transitions obeying the **max-overlap** assumption. Already here, we see mild improvement for $N > 1$ relative to the $N = 1$ results, indicating that even a weak signal such as overlap between the domains of neighboring sentences already improves sentence disambiguation in context. The third part of the table lists the results of the discourse-based model where the PCFG interpolates the seed as well as synthetic-set like before. The transitions are estimated to reflect the **max-overlap** assumptions. Here, we see substantial improvements over the synthetic-only PCFG. That is, modeling grammaticality of individual requirements helps interpreting the document.

In our next experiment we compare different implementations of $gap(m_i, m_j)$. We estimate probability distributions that reflect each of the assumptions we discussed, and add an additional method called **hybrid**, in which we interpolate the **max-expansion** and **max-overlap** estimates (equal weights) — aiming to capture two assumptions about discourse at the same time. In Table 6 the trend from the previous experiment persists. Notably, the hybrid model provides a larger error reduction than its components used separately, indicating that to capture discourse context we may need to balance possibly conflicting factors.

We finally perform a cross-fold experiment in which we leave one document out as a test and make the rest our seed. The results are provided in Table 5. The discourse-based model outperforms the sentence based model $N = 1$ in all cases. Moreover, the drop in $N = 128$ for *Phone* seems to be incidental to this set, and the other cases level off before that. Despite our small seed, the persistent improvement on all metrics is consistent with our hypothesis that modeling the interpretation process within the discourse has substantial benefits for automatic understanding of the text.

N=1	POS	LSC-F1	LSC-EM	SM-TED	SM-EM
Gen-Only	85.52	84.40	9.52	84.25	9.52
Gen+Seed	**91.59**	**88.05**	**14.29**	**85.17**	**14.29**

Table 3: Sentence-Based modeling: Accuracy results on the *Phone* development set.

System	N=2	4	8	16	32	64	128
Oracle							
POS	91.98	93.54	94.91	95.30	96.09	96.67	**96.87**
LSC-F1	88.73	91.33	93.19	94.39	95.11	95.91	**96.70**
LSC-EM	23.81	42.86	61.90	61.90	66.67	76.19	**76.19**
SM-TED	86.54	91.28	94.28	94.88	96.24	97.51	**98.80**
SM-EM	23.81	42.86	66.67	71.43	**76.19**	76.19	76.19
Gen-Only							
POS	85.52	86.30	87.67	88.45	**88.85**	88.85	88.85
LSC-F1	84.40	85.35	86.31	87.51	88.81	89.30	**89.51**
LSC-EM	9.52	9.52	**14.29**	14.29	14.29	14.29	14.29
SM-TED	84.25	85.94	89.14	91.90	92.81	**93.31**	92.70
SM-EM	9.52	19.05	33.33	33.33	33.33	**38.10**	33.33
Gen+Seed							
POS	91.78	92.95	93.54	93.35	94.32	**94.52**	93.93
LSC-F1	88.11	90.18	91.00	90.99	91.81	**92.09**	91.73
LSC-EM	19.05	38.10	**42.86**	42.86	42.86	42.86	42.86
SM-TED	85.49	90.78	93.59	93.02	94.81	**95.69**	93.76
SM-EM	19.05	38.10	**52.38**	52.38	52.38	52.38	52.38

Table 4: Discourse-Based Modeling: Accuracy results on the *Phone* dev set. The Oracle experiment selects the highest scoring LSC tree among the N-candidates, providing an accuracy upper bound. Gen-Only selects the most probable tree, relying on synthetic examples only, providing a lower bound. Gen+Seed adds an annotated seed for an improved acquired probabilistic grammar, providing a strong baseline for the task.

Data Set	N=1	32	64	128
Baby Monitor				
POS	94.29	96.07	96.07	96.07
LSC-F1	91.50	94.96	94.96	94.96
LSC-EM	14.29	21.43	21.43	21.43
SM-TED	88.63	91.11	91.11	91.11
SM-EM	28.57	50.00	50.00	50.00
Chess				
POS	92.63	93.68	93.68	93.68
LSC-F1	95.79	96.16	96.16	96.16
LSC-EM	5.56	11.11	11.11	11.11
SM-TED	94.90	97.10	97.10	97.10
SM-EM	61.11	66.67	66.67	66.67
Phone				
POS	91.59	94.72	94.91	94.32
LSC-F1	88.05	92.15	92.42	92.07
LSC-EM	14.29	47.62	47.62	47.62
SM-TED	85.17	94.87	95.75	93.83
SM-EM	14.29	57.14	57.14	57.14
WristWatch				
POS	34.23	34.45	34.45	34.45
LSC-F1	50.06	51.05	51.05	51.05
LSC-EM	26.67	26.67	26.67	26.67
SM-TED	71.15	72.73	72.73	72.73
SM-EM	26.67	33.33	33.33	33.33

Table 5: Cross-Fold Validation for N=1, 32, 64, 128. For the Seed+Generated trained system, with the **Hybrid** transition parameters.

Transitions	N=2	4	8	16	32	64	128
No Transitions							
POS	91.78	92.56	93.35	93.15	94.32	**94.52**	93.93
LSC-F1	88.11	89.49	90.67	90.66	91.81	**92.09**	91.73
LSC-EM	19.05	38.10	**42.86**	42.86	42.86	42.86	42.86
SM-TED	85.49	90.76	93.68	93.11	94.81	**95.69**	93.76
SM-EM	19.05	38.10	**52.38**	52.38	52.38	52.38	52.38
Min Dist							
POS	91.98	92.76	93.54	93.35	94.32	**94.52**	93.93
LSC-F1	88.39	89.77	91.00	90.99	91.81	**92.09**	91.73
LSC-EM	23.81	42.86	**47.62**	47.62	47.62	47.62	47.62
SM-TED	86.54	91.71	94.38	93.81	95.57	**96.43**	94.53
SM-EM	23.81	42.86	**57.14**	57.14	57.14	57.14	57.14
Max Overlap							
POS	91.78	92.95	93.54	93.35	94.32	**94.52**	93.93
LSC-F1	88.11	90.18	91.00	90.99	91.81	**92.09**	91.73
LSC-EM	19.05	38.10	**42.86**	42.86	42.86	42.86	42.86
SM-TED	85.49	90.78	93.59	93.02	94.81	**95.69**	93.76
SM-EM	19.05	38.10	**52.38**	52.38	52.38	52.38	52.38
Max Expand							
POS	91.98	92.76	93.74	93.54	94.32	**94.52**	93.93
LSC-F1	88.39	89.71	91.00	90.99	91.68	**91.96**	91.60
LSC-EM	23.81	42.86	**47.62**	47.62	47.62	47.62	47.62
SM-TED	86.54	91.93	93.75	93.18	94.79	**95.66**	93.75
SM-EM	23.81	42.86	**57.14**	57.14	57.14	57.14	57.14
Hybrid							
POS	91.78	92.95	93.93	93.74	94.72	**94.91**	94.32
LSC-F1	88.11	90.18	91.34	91.33	92.15	**92.42**	92.07
LSC-EM	19.05	38.10	**47.62**	47.62	47.62	47.62	47.62
SM-TED	85.49	90.78	93.66	93.09	94.87	**95.75**	93.83
SM-EM	19.05	38.10	**57.14**	57.14	57.14	57.14	57.14
No Emissions							
POS	91.78	91.98	92.37	92.37	92.17	92.76	**93.15**
LSC-F1	88.11	88.79	89.12	89.12	89.39	89.67	**89.89**
LSC-EM	19.05	19.05	**23.81**	23.81	23.81	23.81	23.81
SM-TED	85.49	85.74	85.82	85.82	85.87	86.85	**86.92**
SM-EM	19.05	19.05	**23.81**	23.81	23.81	23.81	23.81

Table 6: Discourse-Based modeling: Experiments on the *Phone* development set. Estimation procedure for transition probabilities. All experiments use the Gen+Seed emission probablities.

6 Conclusion

The requirements understanding task presents an exciting challenge to CL/NLP. We ought to automatically recover the entities in the discourse, the actions they take, conditions, temporal constraints, and execution modalities. Furthermore, it requires us to extract a single ontology that satisfies all individual requirements. The contribution of this paper is three-fold: we formalize the text-to-code prediction task, propose a semantic representation with well-defined grounding into code, and empirically evaluate models on the discourse-to-system prediction. We show consistent improvement of discourse-based over sentence-based models, in several case studies. In the future we intend to extend this model for interpreting requirements in un-restricted or less-restricted English, endowed with a more sophisticated discourse interpretation function.[6]

[6] All of our code, data, annotated case studies, and a visual editor for annotating LSCs, will be made publicly available.

References

B. C. Arnold and D. Strauss. 1991. Pseudolikelihood Estimation: Some Examples. *Sankhyā: The Indian Journal of Statistics, Series B (1960-2002)*, 53(2).

Y. Artzi and L. Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *TACL*, 1:49–62.

L. R. Bahl, F. Jelinek, and R. L. Mercer. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 5(2):179–190.

E. Black, J. D. Lafferty, and S. Roukos. 1992. Development and evaluation of a broad-coverage probabilistic grammar of english-language computer manuals. In *ACL*, pages 185–192.

P. F. Brown, V. J. Della Pietra, S. A. Della Pietra, and R. L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Comput. Linguist.*, 19(2):263–311, June.

B. Bryant and B.-S. Lee. 2002. Two-level grammar as an object-oriented requirements specification language. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9 - Volume 9*, HICSS '02, pages 280–, Washington, DC, USA. IEEE Computer Society.

E. Charniak. 1996. Tree-bank grammars. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1031–1036.

W. Damm and D. Harel. 2001. Lscs: Breathing life into message sequence charts. *Form. Methods Syst. Des.*, 19(1):45–80, July.

Nir Eitan, Michal Gordon, David Harel, Assaf Marron, and Gera Weiss. 2011. On visualization and comprehension of scenario-based programs. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ICPC '11, pages 189–192, Washington, DC, USA. IEEE Computer Society.

N. E. Fuchs and R. Schwitter. 1995. Attempto: Controlled natural language for requirements specifications. In Markus P. J. Fromherz, Marc Kirschenbaum, and Anthony J. Kusalik, editors, *LPE*.

M. Gordon and D. Harel. 2009. Generating executable scenarios from natural language. In *Proceedings of the 10th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing '09, pages 456–467, Berlin, Heidelberg. Springer-Verlag.

H. P. Grice. 1975. Logic and conversation. In P. Cole and J. L. Morgan, editors, *Syntax and Semantics: Vol. 3: Speech Acts*, pages 41–58. Academic Press, San Diego, CA.

D. Harel and M. Gordon-Kiwkowitz. 2009. On teaching visual formalisms. *IEEE Softw.*, 26(3):87–95, May.

D. Harel and S. Maoz. 2006. Assert and negate revisited: Modal semantics for uml sequence diagrams. In *Proceedings of the 2006 International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools*, SCESM '06, pages 13–20, New York, NY, USA. ACM.

D. Harel and R. Marelly. 2003. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

D. Harel, H. Kugler, R. Marelly, and A. Pnueli. 2002. Smart play-out of behavioral requirements. In *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design*, FMCAD '02, pages 378–398, London, UK, UK. Springer-Verlag.

D. Harel, A. Kleinbort, and S. Maoz. 2007. S2a: A compiler for multi-modal uml sequence diagrams. In *In Proc. Fundamental Approaches to Software Engineering (FASE'07), volume 4422 of LNCS*, pages 121–124. Springer.

D. Harel, A. Marron, and G. Weiss. 2012. Behavioral programming. *Commun. ACM*, 55(7):90–100, July.

D. Harel, A. Kantor, G. Katz, A. Marron, L. Mizrahi, and G. Weiss. 2013. On composing and proving the correctness of reactive behavior. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–10, Sept.

D. Harel. 2001. From play-in scenarios to code: An achievable dream. *Computer*, 34(1):53–60, January.

H. Kugler, D. Harel, A. Pnueli, Y. Lu, and Y. Bontemps. 2005. Temporal logic for scenario-based specifications. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'05, pages 445–460, Berlin, Heidelberg. Springer-Verlag.

T. Kuhn. 2014. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170.

T. Lei, F. Long, R. Barzilay, and M. C. Rinard. 2013. From natural language specifications to program input parsers. In *ACL (1)*, pages 1294–1303.

P. Liang and C. Potts. 2014. Bringing machine learning and compositional semantics together. *Annual Reviews of Linguistics (submitted)*, 0.

P. Liang, M. I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, pages 590–599.

T. Parsons. 1990. *Events in the Semantics of English: A study in subatomic semantics*. MIT Press, Cambridge, MA.

C. Shannon. 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, July, October.

R. Tsarfaty, J. Nivre, and E. Andersson. 2012. Cross-framework evaluation for statistical parsing. In W. Daelemans, M. Lapata, and L. Màrquez, editors, *EACL*, pages 44–54. The Association for Computer Linguistics.

A. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theor.*

D. H. Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and Control*, 10(2):189–208.

L. S. Zettlemoyer and M. Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, pages 658–666. AUAI Press.