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

Reut Tsarfaty
Weizmann Institute
Rehovot, Israel

Ilia Pogrebezky
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 the 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 can be directly translated 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 show empirically that the discourse-based model consistently outperforms the sentence-based model when 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 system (software or hardware) 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 system 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 translated into *live sequence charts* (LSC) (Damm and Harel, 2001; Harel and Marelly, 2003), a formal language for specifying the dynamic behavior of reactive systems. However, the grammar that underlies this language fragment is highly ambiguous, and all disambiguation has to be conducted manually by a human agent. Indeed, it is commonly accepted that the more natural a controlled language fragment 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 — one that *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 learning signals for statistical 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 the entire document's context.

The key idea we promote here is that discourse context provides substantial disambiguating information for sentence analysis. We suggest a novel

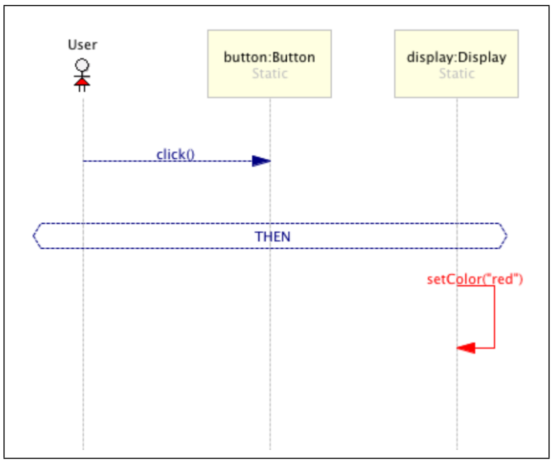

Figure 1: An LSC scenario: "When the user clicks the button, the display color must change to red."

model for integrated sentence-level and discourse-level processing, in a joint generative probabilistic framework. The input for the requirements elicitation task is given in a simplified, yet highly ambiguous, fragment of English, as specified in Gordon and Harel (2009). The output, in contrast, is a sequence of unambiguous and well-formed *live sequence charts* (LSC) (Damm and Harel, 2001; Harel and Marelly, 2003) describing the dynamic behavior of the system, tied to a single shared code-base called a *system model* (SM).

Our solution takes the form of a *hidden markov model* (HMM) 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 a single, shared, domain. Using efficient viterbi decoding, we search for the best sequence of domain snapshots that has most likely generated the entire requirements document. We empirically show that such an integrated model consistently outperforms a sentence-based model learned from the same set of data.

The remainder of this document is organized as follows. In Section 2 we describe the task, and spell out our formal assumptions concerning the input and the output. In Section 3 we present our target semantic representation and a specially tailored notion of *grounding* for anchoring the requirements in a code-base. In Section 4 we develop our sentence-based and discourse-based models, and in Section 5 we evaluate the models on various case studies. In Section 6 we discuss applications and future extensions, and in Section 7 we summarize and conclude.

2 Parsing Requirements Elicitation Documents: Task Description

There is an inherent discrepancy between the input and the output of the software engineering process. The input, system requirements, is specified in a natural, informal, language. The output, the system, is ultimately implemented in a formal unambiguous programming language. Can we automatically recover such 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 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, which are closed and unambiguous, this fragment of English employs an open-ended 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 and Marelly, 2003).[2] Using LSC diagrams for software modelling enjoys the advantages of being easily learnable (Harel and Gordon, 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).

[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).

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*, connecting a sender to a receiver. Messages may refer to other entities (or properties of entities) as arguments. Time in LSCs proceeds 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, "execute"). The LSC language also encompasses conditions and control structures, and it allows defining requirements in terms of the 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 *system model* (SM) is a code-base hierarchy containing the classes USER, BUTTON, DISPLAY, the method BUTTON.CLICK() and the property DISPLAY.COLOR.

3 Formal Settings

In the text-to-code generation task, we aim to implement a prediction function $f : \mathcal{D} \rightarrow \mathcal{M}$, such that $D \in \mathcal{D}$ is a piece of discourse consisting of 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; we denote this by $M \triangleright \mathbf{sem}(d_1, ..., d_n)$. We now define the objects D, M, and describe how to construct the semantic interpretation function $(\mathbf{sem}(.))$. We then spell out the notion of grounding $(\triangleright)$.

Surface Structures: Let Σ be a finite lexicon and let $\mathcal{L}_{req} \subseteq \Sigma^*$ be a language 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 the 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: $u = r_1 \circ ... \circ r_k$; $\forall i : r_i \in \mathcal{R}$. We call such a sequence a *derivation* of u. These derivations may be graphically depicted as parse trees, where the utterance u defines the sequence of tree terminals in the leaves.

We define $\mathcal{T}_{req}$ to be the set of trees strongly generated by G, and utilize an auxiliary yield function $yield : \mathcal{T}_{req} \rightarrow \mathcal{L}_{req}$ returning the leaves of the given tree $t \in .\mathcal{L}_{req}$. Different parse-trees can generate the same utterance, so the task of analyzing the 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 $d \in \mathcal{L}_{req}$ is a diagrammatic structure called a *live sequence chart* (LSC). The LSC formal definition we provide here is based on the appendix of Harel and Marelly (2003), but rephrased in set-theoretic, event-based, terms. We defined this alternative formalization in order to make LSCs compatible with Neo-Davidsonian, event-based, semantic theories. As a result, this form of LSC formalization is well-suited for representing the semantics of natural language utterances.

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 refer to a state, where a logical expression is being evaluated over a set of property:value pairs. We call such an event a *condition*, and specify the set of possible conditions as follows:

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

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

[3]The LSC language also 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 listed as one of the properties of the relevant lifeline.

Specifically, $e = \langle exp, \{l : p : v\}_{i=0}^{k}, temp, exe\rangle$ is a condition to be evaluated, where $l_i \in L, p_i \in P, v_i \in V, temp \in \{hot, cold\}$ and $exe \in \{executed, monitored\}$ are as specified above. The condition $exp \in Exp$ is a first-order logic formula using the usual operators $(\vee, \wedge, \rightarrow, \neg, \exists, \forall)$. 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 more complex events by combining partially ordered sets of events with control structures.

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

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

N is a set of non-negative integers, E_{cond} is a set of conditions 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 $c = \langle E, <\rangle$ is a partially ordered set of events such that

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

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. For example, if our target programming language is an Object-Oriented programming language such as Java, then the code-base will include the objects, the methods and the properties that are minimally required for executing the scenario that is represented by the LSC. We refer to this code-base as a *system model* (henceforth, SM), and define it as follows.

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. We further define the 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 $l \in L_m$ that implements the method $a \in A_m$, the entity $l \in L_m$ that contains the property $p \in P_m$, and the entity property $\langle l, p\rangle \in L_m \times P_m$ that assumes that value $v \in V_m$, respectively. A *system model* (SM) 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 permits the execution of a given LSC c.

Definition 3(a): (Grounding) Let $\mathcal{M}$ be the set of system models and let $\mathcal{C}$ be the set of LSC charts. We say that m *grounds* $c = \langle E, <\rangle$, and write $m \triangleright c$, if $\forall e \in E : m \triangleright e$, 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 $e \in E_{cond}$ 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 the semantics of a single LSC can be grounded in a single SM. In the real world, however, a requirements document typically contains multiple different requirements, but it is interpreted as a complete whole. The desired SM is then one that represents a single domain shared by all the specified requirements. Let us then assume a document $\mathbf{d} = d_1, ..., d_n$ containing n requirements, where $\forall i : d_i \in \mathcal{L}_{req}$, and let $\sqcup$ be a unification operation that returns the formal unification of two SMs if such exists, and an empty SM otherwise. We define a discourse interpretation function $\mathbf{sem}(\mathbf{d})$ that returns a single SM for the entire document, where different mentions across sentences may share the same reference. The discourse interpretation function $\mathbf{sem}$ can be as simple as unifying all individual SMs for d_i, and asserting that all elements that have the same name in different SMs refer to a single element in the overall SM. Or, 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. We can now define the grounding of an entire requirements document.

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

In this work we assume that $\mathbf{sem}(\mathbf{d})$ is a simple discourse interpretation function, where entities, methods, properties, etc. that are referred to using the same name in different local SMs refer to a single element in the overall code-base. This simple assumption already carries a substantial amount of disambiguating information concerning individual requirements. For example, assume that we have seen a "click" method over a "button" object in sentence i. This 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 the accurate semantic analysis of individual requirements.

4 Probabilistic Modeling

In this section we set out to explicitly model the requirement's *context*, formally captured as a document-level SM, in order to support the accurate disambiguation of the requirements' *content*. We first specify our probabilistic *content* model, a sentence-level model which is based on a probabilistic grammar augmented with compositional semantic rules. We then specify our probabilistic *context* model, a document-level sequence model that takes into account the content as well as the relation between SMs at different time points.

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 formally, given a discourse $D = d_1...d_n$ we think of each d_i as having been generated by a *probabilistic context-free grammar* (PCFG) 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:

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

Because of the context-freeness assumption, it holds 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 $d \in \mathcal{L}_{req}$, i.e., $\sum_{d\in\mathcal{L}_{req}} \sum_{t\in\mathcal{T}_{req}, yield(t)=d} P(t) = 1$.

We assume a function $sem : \mathcal{T} \rightarrow \mathcal{C}$ mapping syntactic parse trees to semantic constructs in the LSC language. 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. Here, 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 (events or parts of events),[4] with $\hat{\circ}$ merging the resulting sets of events. Our sentence-based compositional semantics is summarized as:

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

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

[4]Here, it suffices to say that sem maps edges in the syntax tree to functions in the API of an existing LSC editor. For example: $sem(NP \rightarrow DET\ NN) = fCreateObject(DET.sem, NN.sem)$. We specify the function sem in the supplementary materials. The code of sem is available as part of *PlayGo* (`www.playgo.co`).

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 $\forall i : m_i \triangleright sem(syn(d_i))$ and $\sqcup$ unifies different mentions of the same string to a single element.

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 provided with an ambiguous natural language evidence, an elicited discourse D, 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.

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

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

Both M and D are structured objects with complex internal structure. In order to assign probabilities to objects involving such complex structures it is customary to break them 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 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 transition.

4.3 Training and Decoding

Our model is in essence a Hidden Markov Model in which states capture SM snapshots, 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 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 data.

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

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, while manually providing disambiguating information. 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 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 n_i 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 of a requirement given an SM snapshot which grounds the semantics of the description. A single SM may result from different syntactic derivations. We calculate this probability by marginalizing over 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 $\hat{P} : \mathcal{M} \times \mathcal{M} \to [0, 1]$ and, for all m_i, m_j,: $\sum_{m_j} \hat{P}(m_i|m_j) = 1$. (For efficiency reasons, we consider $\mathcal{M}$ to be a restricted universe that is considered be the decoder, as specified shortly.)

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.

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.

Decoding 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, retrieved via a CKY chart parser. 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 $N \times n$ SM snapshots, reflecting the N most-likely analyses of n sentences, as provided by the probabilistic syntactic model. (As shall be seen shortly, even with this restricted[5] universe approximating $\mathcal{M}$, our discourse-based model provides substantial improvements over a sentence-based model).

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 discourse-based probabilistic model.

[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 the SM analyses of the sentences observed in the document only. This estimation could also be addressed via, e.g., sampling methods.

The overall complexity decoding a document with n sentences of which max length is l, using a grammar G of size $|G|$ and a fixed N, is given by:

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

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 a CKY chart (Younger, 1967). (ii) In $O(l^2 \times N^2 \times n)$ we create the universe $\mathcal{M}$ based on the N best trees of the n requirements, and calculate $N \times N$ transitions. (iii) In $O((N \times n)^2 \times n) = O(N^2 \times n^3)$ we decode the $n \times N$ grid using Viterbi (1967) decoding.

5 Experiments

Goal. We set out to evaluate the accuracy of a semantic parser for requirements documents, in the two modes of analysis presented above. Our evaluation methodology is as standardly assumed in machine learning and NLP: given a set of annotated examples — that is, given a set of requirements documents, where each requirement is annotated with its correct LSC representation and each document is associated with a complete SM — we partition this set into a *training set* and a *test set* that are disjoint. We train our statistical model on the examples in the training set and automatically analyze the requirements in the test set. We then compare the predicted semantic analyses of the test set with the human-annotated (henceforth, *gold*) semantic analyses of this test set, and empirically quantify our prediction accuracy.

Metrics. Our semantic LSC objects have the form of a tree (reflecting the sequence of nested events in our scenarios). Therefore, we can use standard tree evaluation metrics, such as ParseEval (Black et al., 1992), to evaluate the accuracy of the prediction. Overall, we define three metrics to evaluate the accuracy of the LSC trees:

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

In the case of SM trees, as opposed to the LSC trees, we cannot assume identity of the yield between the gold and parse trees for the same sentence,[6] so we cannot use ParseEval. Therefore, we implement a distance-based metrics in the spirit of Tsarfaty et al. (2012). Then, to evaluate the accuracy of the SM, we use two kinds of scores:

SM-TED: TED is the normalized edit distance between the predicted and gold SM trees, subtracted from a unity.
SM-EM: EM is 1 if the predicted SM is an exact match with the gold SM, 0 otherwise.

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

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.

Data. We have a small seed of correctly annotated requirements-specification case studies that describe simple reactive systems in the LSC language. Each document contains a sequence of requirements, each of which is annotated with the correct LSC diagram. The entire program is grounded in a java implementation. As training data, 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 quite 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 this, we create a synthetic set of examples using the grammar of Gordon and Harel (2009) in generation mode, and randomly generate trees $t \in \mathcal{T}_{req}$.

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

[6]This is because the LSC trees are predicted bottom up and the SM trees are predicted top-down.

Results. Table 3 presents the results for parsing the *Phone* document, our development set, with the sentence-based model, varying the training data. We see that despite the small size of the seed, adding it to our set if synthetics examples substantially improves results over a model trained on synthetic examples only.

In our next experiment, we provide empirical upper-bounds and lower-bounds for the discourse-based model. Table 4 presents the results of the discourse-based model for $N > 1$ on the *Phone* example. *Gen-Only* presents the results of the discourse-based model with a PCFG learned from synthetic trees only, incorporating transitions obeying the **max-overlap** assumption. Already here, we see a mild improvement for $N > 1$ relative to the $N = 1$ results, indicating that even a weak signal such as the overlap between neighboring sentences already improves sentence disambiguation in context. We next present the results of an *Oracle* experiment, where every requirement is assigned the highest scoring tree in terms of LSC-F1 with respect to the gold tree, keeping the same transitions. Again we see that results improve with N, indicating that the syntactic model alone does not provide optimal disambiguation. These results provides an upper bound on the parser performance for each N. *Gen+Seed* presents results of the discourse-based model where the PCFG interpolates the seed set and the synthetic train set, with **max-overlap** transitions. Here, we see larger improvements over the synthetic-only PCFG. That is, modeling grammaticality of individual sentences improves the interpretation of the document.

Next we compare the performance for different implementations of the $gap(m_i, m_j)$ function. 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). In Table 5, the trend from the previous experiment persists. Notably, the **hybrid** model provides a larger error reduction than its components used separately, indicating that in order to capture discourse context we may need to balance possibly conflicting factors. In **no emissions** we rely solely on the probability of state transitions, and again increasing N leads to improvement. This result confirms that *context* is indispensable for sentence interpretation — even when probabilities for the sentence's semantics (*content*) are entirely absent.

System	N=2	4	8	16	32	64	128
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
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

Table 4: Discourse-Based Modeling: Accuracy results on the *Phone* dev set. The Oracle selects the highest scoring LSC tree among the N-candidates, providing an upper bound on accuracy. Gen-Only selects the most probable tree, relying on synthetic examples only, providing a lower bound.

We finally perform a cross-fold experiment in which we leave one document out as a test set and take the rest as our seed. The results are provided in Table 6. The discourse-based model outperforms the sentence-based model $N = 1$ in all cases. Moreover, the drop in $N = 128$ for *Phone* seems incidental to this set, and the other cases level off beforehand. 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.

6 Applications and Discussion

The statistical models we present here are applied in the context of *PlayGo*,[7] a comprehensive tool for behavioral, scenario-based, programming. PlayGo now provides two modes of *playing-in* natural language requirements: interactive play-in, where a user manually disambiguates the analyses of the requirements (Gordon and Harel, 2009), and statistical play-in, where disambiguation decisions are taken using our discourse-based model.

The fragment of English we use is very expressive. It covers not only entities and predicates, but also temporal and aspectual information, modalities, and program flow. Beyond that, we assume an open-ended lexicon. Overall, we are not

[7] `www.playgo.co`.

Transitions	N=2	4	8	16	32	64	128
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 5: Discourse-Based modeling: Experiments on the *Phone* development set. Estimation procedure for transition probabilities. All experiments use the Gen+Seed emission probablities.

only translating English sentences into executable LSCs — we provide a fully generative model for translating a complete document (text) into a complete system model (code).

This text-to-code problem may be thought of as a machine translation (MT) problem, where one aims to translate sentences in English to the formal language of LSCs. However, standard statistical MT techniques rely on the assumption that textual requirements and code are aligned at a sentence level. Creating a formal model that aligns text and code on a sentence-by-sentence basis is precisely our technical contribution in Section 3.

To our knowledge, modeling syntax and discourse processing via a fully joint generative model, where a discourse level HMM is interleaved with PCFG sentence-based emissions, is novel. By plugging in different models for $p(d|m)$, different languages may be parsed. This method may further be utilized for relating content and context in other tasks: parsing and document-level NER, parsing and document-level IE, etc. To do so, one only needs to redefine the PCFG (emissions) and state-overlap (transition) parameters, as appropriate for their data.[8]

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 6: Cross-Fold Validation for N=1..128. Seed+Generated emissions, **Hybrid** transitions.

7 Conclusion

The requirements understanding task presents an exciting challenge for CL/NLP. We ought to automatically discover 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 contributions of this paper are three-fold: we formalize the text-to-code prediction task, propose a semantic representation with well-defined grounding, and empirically evaluate models for this prediction. We show consistent improvement of discourse-based over sentence-based models, in all 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.

Acknowledgements

We thank Shahar Maoz, Rami Marelly, Yoav Goldberg and three anonymous reviewers for their insightful comments on an earlier draft. This research was supported by an Advanced Research Grant to D. Harel from the European Research Council (ERC) under the European Community‘s Seventh Framework Programme (FP7/2007-2013), and by a grant to D. Harel from the Israel Science Foundation (ISF).

[8]Our code, annotated data, four case studies, and the LSC visual editor are available via `http://wiki.weizmann.ac.il/playgo/index.php/Download_PlayGo`.

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 *Proceedings of 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 *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.

N. Eitan, M. Gordon, D. Harel, A. Marron, and G. 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. 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 LSCs 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. Springer-Verlag.

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, *Proceedings of 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.