Simple Interval Temporal Logic for Natural Language Assertion Descriptions

Reyadh Alluhaibi
The University of Manchester, UK
alluhair@cs.man.ac.uk

Abstract

SystemVerilog assertion (SVA) is widely used for verifying properties of hardware designs. This paper presents a new method of generating SVAs from natural language assertion descriptions. For capturing the temporal semantics in natural language descriptions, we develop a new logical form called *simple interval temporal logic* ($\mathcal{SITL}$) which can deal formally with temporal constructions such as temporal prepositions. Furthermore, $\mathcal{SITL}$ makes the transformations from natural language descriptions to SVAs possible. Thus, we build transformation rules to map our logic into SVAs. Our systematic experimental investigation on AXI bus protocol in ARM (2010) suggest that our method is applicable for generating SVAs from natural language descriptions.

Keywords: Temporal semantics; temporal prepositions; natural language descriptions; SystemVerilog assertions

1 Introduction

Formal verification is the process of checking whether a design fulfils certain requirements (properties). In the last decade, one major challenge in the area of formal verification has been to reduce the effort required to design hardware systems (Darringer, 1988; Milne, 1993). SVA is one of the formal verification tools used for verification finite state concurrent systems such as sequential circuit designs. SVA is a linear temporal logic which can express temporal behaviours of the system designs. In fact, SVA allows complex temporal relations between signals to be expressed in a concise and accurate way. However, capturing the required temporal behaviours of the design requires a precise temporal description in SVA.

Therefore, several attempts have been made to generate formal system requirements from natural language specifications in (Clarke et al., 1986; Holt, 1999; Grover et al., 2000; Harris, 2013). The motivations of these approaches are to reduce design time and errors and also to help us to have early identifications of incomplete and inconsistent specifications. Unfortunately, generating formal requirements from natural language specifications is still limited because most such approaches fail to capture certain temporal expressions commonly occurring in natural language specifications such as tenses, aspects, and temporal prepositions.

Of course, various linguists have attempted to provide formal semantics for certain temporal expressions in natural language (e.g. Reichenbach, 1947; Vendler, 1967; Dowty, 1972). Most approaches, (such as in Reichenbach (1947)), focus on tenses and are not readily applicable to hardware specifications, since, most of such specifications must be written in the present tense. Admittedly, the studies of aspectual classes (Vendler, 1967; Dowty, 1972) are useful when considering the temporal behaviours of natural language descriptions; however, in practice, these aspectual classes are not an essential requirements for generating SVAs.

In contradistinction to these linguistic studies, there is a logical form that can provide the formal semantics for temporal proposition phrases — *temporal preposition logic* ($\mathcal{TPL}$) which is one of the more recent interval logics as introduced in Pratt and Francez (2001). However, $\mathcal{TPL}$ is more expressive than SVA. Thus, this paper seeks to remedy these problems by finding a simpler way to translate natural language assertion descriptions featuring temporal expressions to SVAs directly. Our new logic $\mathcal{SITL}$

Proceedings of the 11th International Conference on Computational Semantics, pages 283–293,
London, UK, April 15-17 2015.

under some assumptions can be translated to SVA. $\mathcal{SITL}$ is a subset of $\mathcal{TPL}$. Although $\mathcal{TPL}$ is more expressive than $\mathcal{SITL}$, $\mathcal{SITL}$ can capture most temporal constructions that are described in natural language assertion descriptions. We hope in this paper to provide a method that is more precise and efficient in formal verifications than the previous attempts of Harris (2013).

The structure of the paper as follows. Section 2 introduces the basic definitions and notation of $\mathcal{TPL}$ and SVA. Section 3 introduces the syntax and semantics of $\mathcal{SITL}$ as well as its benefits of capturing temporal expressions. Section 4 shows a method for generating SVAs from natural language descriptions. The experimental methods and results are presented in Section 5. Section 6 reviews the techniques for generating formal specifications from natural language specifications and the effects of aspectual classes on the semantics of SVAs and Section 7 concludes the paper.

2 Preliminaries

2.1 Temporal Preposition Logic ($\mathcal{TPL}$)

In this section, we recall the semantics of $\mathcal{TPL}$ presented in Pratt-Hartmann (2005). $\mathcal{TPL}$ is a first-order language having variables which range over time-intervals and predicates corresponding to event-types and temporal order-relations. In this paper, the letters I, J with or without decorations, range over time intervals, where an interval is closed, bounded, non-empty subset of real numbers. Let us review the semantics of $\mathcal{TPL}$ using examples from natural language assertion descriptions. Consider the sentences:

(1) *Awid* is asserted

(2) *Awid* is asserted during every cycle

(3) *Awid* is asserted during every cycle until *Awvalid* goes high.

Sentence (1) states that, within the temporal context, there is an interval over which *Awid* is asserted. The meaning of (1) can be represented as follows:

(4) $\exists \mathrm{J}_0(\text{asserted(Awid, } \mathrm{J}_0) \wedge \mathrm{J}_0 \subseteq \mathrm{I})$.

Notice that the quantification of J_0 is limited to the temporal context which represented by the free variable I. Sentence (2) states that, within the temporal context I, every interval over which a cycle occurs includes an interval over which *Awid* is asserted. The meaning of (2) can be represented as follows:

(5) $\forall \mathrm{J}_1(\text{cycle}(\mathrm{J}_1) \wedge \mathrm{J}_1 \subseteq \mathrm{I} \rightarrow \exists \mathrm{J}_0(\text{asserted(Awid, } \mathrm{J}_0) \wedge \mathrm{J}_0 \subseteq \mathrm{J}_1))$.

Notice that *cycle* is considered in Pratt and Francez (2001) as a temporal noun which has an interval time such as *meeting*, *Monday*, and *1995*. Thus, the meaning of *cycle* should be as follows:

(6) $\lambda \mathrm{J} \lambda \mathrm{I}[\text{cycle(J)} \wedge \mathrm{J} \subseteq \mathrm{I}]$,

Intuitively, the word "cycle" picks out those intervals J over which a cycle occurs with some temporal context I as shown in Formula (5).

Some events take place not during, but before or after various time-intervals. Thus, we need to define some functions that can express these relationships.

Definition 2.1. Let *I* = [a,b] and *J* = [c,d] be intervals. If $\mathrm{a} < \mathrm{c} < \mathrm{d} < \mathrm{b}$, we let the terms init(*J*,*I*) and fin(*J*,*I*) denote the intervals [a,c] and [d,b], respectively, where init and fin are partial functions to denote the initial segment of *I* up to the beginning of *J*, and the final segment of *I* from the end of *J*, respectively. Finally, we denote the definite quantifier $\imath$ with the standard (Russellian) semantics $\imath(\psi, \psi')$.

Now, we can expresses sentence (3) in $\mathcal{TPL}$ as follows:

(7) $\imath J_2$(high(Awvalid, J_2) $\wedge$ $J_2 \subseteq$ I,
$\forall J_1$(cycle(J_1) $\wedge$ $J_1 \subseteq$ init(J_2,I) $\rightarrow$ $\exists J_0$(asserted(Awid,J_0) $\wedge$ $J_0 \subseteq J_1$))).

This formula states that, within the temporal context I, there is a unique interval J_2 such that *Awvalid* goes high at J_2 and every interval J_1 such that J_1 is a cycle and J_1 is contained init(J_2,I), which in turn includes an interval J_0 over which *Awid* is asserted.

In this section, we have given a flavour of the language $\mathcal{TPL}$. For a complete specification, we refer the reader to Pratt and Francez (2001).

2.2 SystemVerilog Assertions (SVA)

SVA is a subset of SystemVerilog which combines hardware descriptions and formal verifications. Assertions formally verify the correctness of the specifications. SVA has the ability to define sequential expressions with clear temporal relationships between them. These temporal relationships are expressed by one or more clocks. For example, assertion (8) checks that the signal "AWID" is high at every posedge clock. If the signal "AWID" is not high at any posedge clock, the assertion will fail.

(8) sequence s1;
@(posedge clk) AWID;
endsequence.

In SVA, the clock cycle delays are defined by a "##" sign and there are two types of delays possible, a single delay or a range of delays. Consider

(9) AWID ##2 AWVALID

(10) AWID ##[1:4] AWVALID.

Assertion (9) states that the signal "AWID" is true at the point of evaluation, and must remain true for next clock cycle before the single "AWVALID" will be true, while assertion (10) states that the signal "AWID" is true, and may remain true for up to 3 further clock cycles, directly after which the single "AWVALID" will be true. Note that we can specify an assertion with infinite repetition range such as in (11) where the $ symbol indicates that the signal "AWVALID" will eventually occur.

(11) AWID ##[1:$] AWVALID.

Furthermore, there are two types of implication in SVA: an overlapped implication and a non-overlapped implication. The overlapped implication is denoted by the symbol $|->$ while the non-overlapped implication is denoted by the symbol |=> as shows, respectively.

(12) AWID $|->$ AWVALID

(13) AWID |=> AWVALID.

Assertion (12) states that if the antecedent "AWID" holds, then the consequent expression "AWVALID" starts in the same clock cycle. On other hand, assertion (13) states that if the antecedent "AWID" holds, then the consequent expression "AWVALID" starts in the next clock cycle. Note that assertion (13) can be expressed differently using a "##" sign as shown below in (14) where ##1 means a delay of one clock cycle before the consequent expression "AWVALID" is started.

(14) AWID $|->$ $\#\#1$ AWVALID.

Moreover, SVA has sequential binary operators such as "and" and "or" operators which works with two sequences or boolean expressions. The "and" operator means that if both two sequences or boolean expressions are true, then the result of "and" operation is true. However, the resultant of the "or" operands is true whenever at least one of sequences boolean expressions is true. Consider

(15) AWID and AWVALID

(16) AWID or AWVALID.

Assertion (15) will be only true if both signals "AWID" and "AWVALID" are true; on other hand, the result of assertion (16) is true when either signal AWID or signal AWVALID is true.

All of the above-mentioned operators will be used for constructing transformation rules between $\mathcal{SITL}$ and SVA. For a more detailed account of SVA refer to Vijayaraghavan and Ramanathan (2006).

3 Simple Interval Temporal Logic ($\mathcal{SITL}$)

We define $\mathcal{SITL}$ as a subset of $\mathcal{TPL}$. Both logics work well for capturing temporal expressions in English. However, we choose $\mathcal{SITL}$ over $\mathcal{TPL}$ because it is more applicable for generating SVA.

A $\mathcal{SITL}$ language L is a triple (C,P,F) of sets of constant symbols, predicate symbols, and functional symbols. Note, each predicate symbol and functional symbol must be assigned to non-zero natural number which represents its arity. A term is a variable or a constant. Also, if f is a function symbol of arity n and t_1 , . ., t_n are terms, then f (t_1 , . . . , t_n) is a term.

$\mathcal{SITL}$-formula ψ is defined by the Backus-Naur Form production rule as follows.

ψ:= $\top$ | $\bot$ | $\neg\psi$ | $\psi \wedge \psi'$ | $\psi \vee \psi'$ | P(t_1 , . . . , t_n) | Q.TP(ψ, ψ').

Here $\top$, $\bot$, $\neg$, $\wedge$, and $\vee$ have the same meaning as in first-order logic, P is a predicate symbol of arity n and t_1 , . ., t_n are terms; TP is a binary predicate symbol that only corresponding to temporal prepositions such as *when*, *after*, *before*, and *until*. Note, TP is not a standard formulation of logic which means here if ψ and ψ' are formulas, then Q.TP(ψ, ψ') are formulas too where Q denotes any type of quantifiers (such as universal, existential or definite quantifier) and "TP(ψ, ψ')" is restricted by Q. For example, a formula like "$\imath$.when(ψ, ψ')" means that ψ and ψ' are bounded by the definite quantifier $\imath$.

3.1 Semantics

We define $\mathcal{SITL}$ based on $\mathcal{TPL}$ introduced by Pratt-Hartmann (2005) as follows:

(17) $[\imath.\text{when}(\psi, \psi')]^{\#}(\text{I}) = \imath\text{J}(\ [\psi]^{\#}(\text{J}) \wedge \text{J} \subseteq \text{I}\ , [\psi']^{\#}(\text{J}))$;

(18) $[\imath.\text{before}(\psi, \psi')]^{\#}(\text{I}) = \imath\text{J}(\ [\psi]^{\#}(\text{J}) \wedge \text{J} \subseteq \text{I}\ , [\psi']^{\#}(\text{fin(J,I)}))$;

(19) $[\imath.\text{after}(\psi, \psi')]^{\#}(\text{I}) = \imath\text{J}(\ [\psi]^{\#}(\text{J}) \wedge \text{J} \subseteq \text{I}\ , [\psi']^{\#}(\text{init(J,I)}))$;

(20) $[\ \imath.\text{until}(\psi, \psi')]^{\#}(\text{I}) = \imath\text{J}(\ [\psi]^{\#}(\text{J}) \wedge \text{J} \subseteq \text{I}\ , [\psi']^{\#}(\text{init(J,I)}))$;

(21) $[\imath.\text{until after}(\psi, \psi')]^{\#}(\text{I}) = \imath\text{J}(\ [\psi]^{\#}(\text{J}) \wedge \text{J} \subseteq \text{I}\ , [\psi']^{\#}(\text{init(J,fin(J,I))}))$;

(22) $[Q.\text{during}(\psi, \psi')]^{\#}(\text{I}) = Q\text{J}(\ [\psi]^{\#}(\text{J}) \wedge \text{J} \subseteq \text{I}\ , [\psi']^{\#}(\text{J}))$;

(23) $[\exists.\text{for}(\psi, \psi')]^{\#}(\text{I}) = \exists\text{J}(\ [\psi]^{\#}(\text{J}) \wedge \text{J} \subseteq \text{I} \wedge [\psi']^{\#}(\text{J}))$.

First of all, the temporal prepositions in (17-21) require their complements to be definitely quantised since these readings are suitable for many cases in natural language descriptions. Second, the temporal preposition *during* in (22) is located ψ' to be within the interval of ψ where the quantification to restrict its complement is not decided. Finally, the temporal preposition *for* in (23) enforces its complement to be existentially quantified. These quantification restrictions on temporal preposition phrases have been discussed in Pratt and Francez (2001), which drew attention to the fact that some temporal prepositions impose restrictions on the temporal quantification appearing in their complements.

Taking the prepositions *until* and *until after* into consideration as expressed in (20) and (21), it is worth noting that out interpretation are different from $\mathcal{TPL}$. In $\mathcal{TPL}$, the preposition *until* universally quantifies its modificands explicitly. We leave this quantification implicit for simplification purpose which as stated before facilitates the generation of SVA. $\mathcal{SITL}$ is therefore constructed to consider interpretation of temporal prepositions that are required for the generation of correct SVAs.

3.2 Interpretation of $\mathcal{SITL}$ in English

In this section, we provide an interpretation of $\mathcal{SITL}$ in English involving temporal constructions. The purpose of $\mathcal{SITL}$ is to have a subset of $\mathcal{TPL}$ that is closer in expressive power to common constructions encountered natural language specifications. Consider

(24) *Awid* must remain stable when *Awvalid* is asserted.

which can be expressed in $\mathcal{SITL}$ and $\mathcal{TPL}$, respectively, as follows:

(25) $\imath$.when(asserted(Awvalid), stable(Awid))

(26) $\imath \mathrm{J}_1(\ \mathrm{asserted(Awvalid, J}_1) \wedge \mathrm{J}_1 \subseteq \mathrm{I}, \exists \mathrm{J}_0(\ \mathrm{stable(Awid, J}_0) \wedge \mathrm{J}_0 \subseteq \mathrm{J}_1))$.

Both formulas have the same meaning, namely, that, within the temporal context I, there is a unique interval J over which *Awvalid* is true which includes an interval over which *Awid* is true. However, $\mathcal{SITL}$ formula (25) is less expressive than $\mathcal{TPL}$ formula (26) in which $\mathcal{SITL}$ can easily transform to SVA such as (27) for formula (25). The transformations from $\mathcal{SITL}$ to SVA will be discussed later.

(27) *AWVALID* |-> $stable(*AWID*).

More interpretations can be provided here with temporal prepositions such as *after* and *before*. Consider the following sentences

(28) *Awid* must be low after *Awvalid* goes high

(29) *Awid* must be low before *Awvalid* goes high.

Both sentences can be expressed in $\mathcal{SITL}$ as follows, respectively.

(30) $\imath$.after(high(Awvalid), low(Awid))

(31) $\imath$.before(high(Awvalid), low(Awid)).

Formula (30) states that when the signal Awvalid goes high, the signal Awid must be low in the next cycle. Formula (31) states that before the signal Awvalid goes high, the signal Awid must be low. Thus, $\mathcal{SITL}$ defines events and their temporal locations correctly which enable us to easily map $\mathcal{SITL}$ formula into SVA as shown in (32) and (33) for (30) and (31), respectively.

(32) *AWVALID* |-> ##1 !*AWID*

(33) !*AWID* |-> ##1 *AWVALID*.

Furthermore, $\mathcal{SITL}$ can interpret complex temporal expressions that occur in natural language assertion descriptions. Consider

(34) When *Awvalid* is asserted, Awid must remain low until *Awready* goes high.

Sentence (34) has a complicated meaning since it includes two temporal prepositions *when* and *until*. Thus, we need to interpret it based on the most natural reading which here means the semantics of *until* must be restricted by the semantics of *when* as follows.

(35) $\overbrace{\imath.\mathrm{when(asserted(Awvalid),}\ \overbrace{\imath.\mathrm{until(high(Awready),\ low(Awid))}}^{until})}^{When}$.

In order to explain our interpretation in $\mathcal{SITL}$, consider the following $\mathcal{TPL}$ formula

(36) $\imath J_2($ asserted(Awvalid, $J_2) \wedge J_2 \subseteq I$, $\imath J_1$(high(Awready, $J_1) \wedge J_1 \subseteq$fin($J_2$, I),
$\forall J_0(J_0 \subseteq$ init(J_1, fin(J_2, I)) $\rightarrow$ low(Awid, J_0)))),

which states that within the temporal context I, there is a unique interval J_1 such that *Awvalid* is asserted at J_1 and there is a unique interval J_0 such that J_0 is a subset of fin(J_1,I) and *Awready* is high at J_0 and *Awid* must be low during init(J_0,fin(J_1,I)). Notice, (36) is not constructed of using rules (17) and (20) in Section 3.1. The complexities of *when* in conjunction with *until* require us to define a special rule as shown in (37), where locates the temporal relation between them correctly.

(37) $[\imath.\text{when}(\psi, \imath.\text{until}(\psi', \psi''))]^{\#}(I) = \imath J_1([\psi]^{\#}(J_1) \wedge J_1 \subseteq I , \imath J_0([\psi']^{\#}(J_0) \wedge$
$J_0 \subseteq \text{fin}(J_1,I) , [\psi'']^{\#}(\text{init}(J_0,\text{fin}(J_1,I)))))$.

Even if we change the order of sentence (34) as in the following example, we can only have the interpretation (35), that in which the semantic of *when* has wider scope than the semantic of *until* because in our method we do not intend to embed temporal prepositions; therefore we will allow them in ordering which they appear.

(38) Awid must remain low until *Awready* goes high when *Awvalid* is asserted.

In the end, we have shown the interpretations of $\mathcal{SITL}$ in requirement examples that were taken from ARM (2010) and how precise $\mathcal{SITL}$ is for specifying these requirements including temporal constructions. Then, we have explained how $\mathcal{SITL}$ can eliminate the complexity which can be caused by temporal prepositions. In the next section, we will show how can be possible to translate $\mathcal{SITL}$ to SVA using transformations rules.

4 Generating SystemVerilog Assertions

In this section, we describe a method for generating SVA from natural language descriptions. Our method will be divided into two distinct steps. First, we extract $\mathcal{SITL}$ from a parse tree using semantic rules and then we generate SVA from $\mathcal{SITL}$ using transformation rules.

4.1 Semantic Rules for Extracting $\mathcal{SITL}$

We build our semantic rules based on combining typed logic with lambda abstraction. This method defined in Montague (1974), which has been used globally to build semantic representations for a fragment of English. First of all, we use a wide coverage parser that produces *Penn tree-bank* style parse trees. Then, we take parse trees from the adopted parser and we apply our semantic rules to extract $\mathcal{SITL}$ from the parse trees such as in Figure 1 for sentence (24).

Notice that in Figure 1, the modal auxiliary (MD), the verb in base form (VB), and the auxiliary verb (AUX) are ignored when followed by the "VP" or "ADJP" because such constructions lead to redundancy if we attempt to represent every category. More importantly, most existing parsers attach the "SBAR" category to the "VP" category instead of the "S" category, which causes difficulties when handling the scope of temporal quantifications. Therefore, we handle this issue by moving up the "SBAR" category as shown in Figure 1 to be combined with the "S_0" category to extract $\mathcal{SITL}$ precisely.

We have constructed 176 semantic rules to extract $\mathcal{SITL}$. Each terminal or non-terminal node has a semantic rule based on its part of speech tag. These rules are limited to our case study which might be extended in the future work. This step was made specifically to support our theory about how easy it can be to generate SVA from $\mathcal{SITL}$ than other logical forms as explained in Section 3.2.

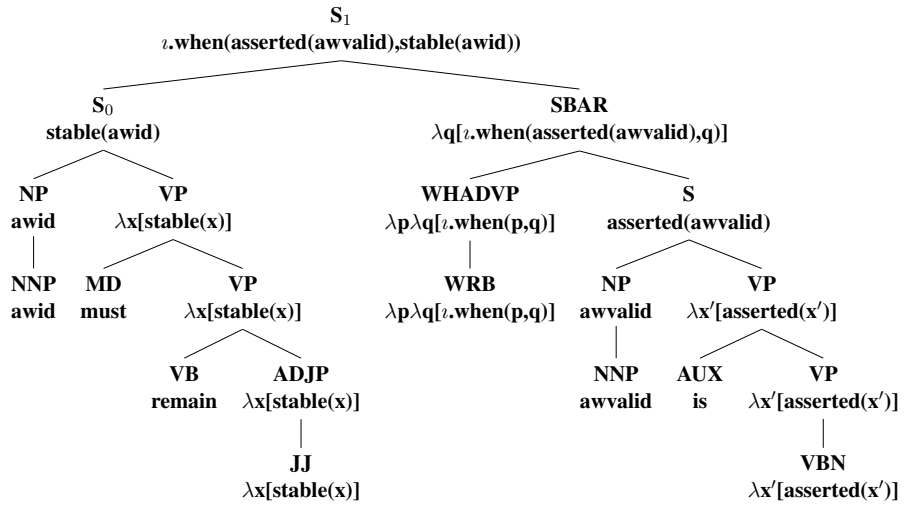

Figure 1: The annotated parse tree corresponds to semantic rules

4.2 Transforming $\mathcal{SITL}$ to SVA

This step is about generating SVA from $\mathcal{SITL}$ using transformation rules. These rules were constructed using all the possible interpretations of temporal expressions in SVA from ARM (2010) and Vijayaraghavan and Ramanathan (2006). The transformation rules take $\mathcal{SITL}$ as an input and produce its equivalent in SVA. Generating SVA from $\mathcal{SITL}$ will turn out to be straightforward because using $\mathcal{SITL}$ helps to reduce the difficulties of mapping natural language assertion descriptions to SVAs. The transformation rules are defined as follows:

1. $[\imath.\text{when}(\psi, \psi')]^{\#} = ([\psi]^{\#} \ |\text{-> } [\psi']^{\#})$;
2. $[\imath.\text{before}(\psi, \psi')]^{\#} = ([\psi']^{\#} \ |\text{-> } \#\#1\ [\psi]^{\#})$;
3. $[\imath.\text{after}(\psi, \psi')]^{\#} = ([\psi]^{\#} \ |\text{-> } \#\#1\ [\psi']^{\#})$;
4. $[\imath.\text{until}(\psi, \psi')]^{\#} = ([\psi']^{\#}[*0:\$] \ \#\#1 \ [\psi]^{\#})$;
5. $[\imath.\text{until after}(\psi, \psi')]^{\#} = ([\psi']^{\#}[*0:\$] \ \#\#0 \ [\psi]^{\#})$;
6. $[\exists.\text{for}(\mathbb{N}, \psi)]^{\#} = ([\psi]^{\#}[*\mathbb{N}]^{\#})$.

The transformation rule (1) maps ψ and ψ' into SVA as an overlap relation. In the transformation rule (2), ψ and ψ' maps into SVA where ψ' must occur a cycle before ψ is true, while the transformation rule (3) is inverse of (2). The transformation rule (4) maps ψ and ψ' into SVA where ψ' must occur until the cycle before ψ is true, while in the transformation rule (5), ψ' must remain true until ψ is completed. In addition to transformation rules (4) and (5), the temporal preposition $until$ frequently comes with another temporal prepositions such as $when$, $during$, and $after$. For example, assertion (39) is constructed by combining the transformation rule (3) with the transformation rule (4) which here means after ψ comes true, ψ'' must remain true until ψ' occurs.

$$\overbrace{([\psi]^{\#} \ |\text{-> } \#\#1 \ \overbrace{[\psi'']^{\#}[*0:\$] \ \#\#1 \ [\psi']^{\#}}^{until})}^{after}. \quad (39)$$

Finally, the transformation rule (6) is related to the preposition *for* which takes only numerical expressions as its complement such as one cycle, two cycles, and etc which denote number of repetitions. For example, sentence (40) indicates that after the signal "Awid" is true, Awvalid must be low for two consecutive cycles as shown in assertion (41).

(40) Awvalid is low for two cycles after Awid goes high.

(41) AWID |-> ##1 !AWVALID[*2].

Having discussed how to translate $\mathcal{SITL}$ into SVA involving temporal prepositions, there are other categories need to be translate into SVAs such verbs, adjectives, and noun phrases. Thus, we have collected the most common words in natural language descriptions from ARM (2010). Then, we assigned each word a link to a particular SVA's term. For example, the adjectives "stable" and "constant" are linked to a keyword "$stable". Table 1 shows some common words in $\mathcal{SITL}$ and their formal terms in SVA.

#	$SITL$	SVA
	verb	
7	has/have/use/require(ψ, ψ')	ψ |-> ψ'
8	¬exceed(ψ, ψ')	ψ <= ψ'
9	be/set(ψ, ψ')	(ψ==ψ')
10	asserted/permitted(ψ)	ψ
11	¬de-asserted/¬permitted(ψ)	!ψ
12	¬be(ψ, ψ')	(ψ!=ψ')
	Adjectives	
13	stable/constant(ψ)	$stable($\psi$)
14	high(ψ)	ψ
15	low(ψ)	!ψ
16	greater(ψ, ψ')	($\psi > \psi'$)
17	equal(ψ, ψ')	(ψ=ψ')
18	less(ψ, ψ')	($\psi < \psi'$)

#	$SITL$	SVA
	Conjunction and Disjunction	
19	$\psi \vee \psi'$	(ψ or ψ')
20	$\psi \wedge \psi'$	(ψ and ψ')
	Preposition	
21	on(ψ, ψ')	(ψ==ψ')
22	with(ψ, ψ')	(ψ==ψ')
	Noun phrases or signals	
23	write burst	(AWBURST==AXI_ABURST_INCR)
24	read burst	(ARBURST==AXI4PC_ABURST_INCR)
25	write transaction	AWBURST
26	read transaction	ARBURST
27	data_width parameter	DATA_WIDTH
28	awvalid	AWVALID
29	awid	AWID

Table 1: Transformation rules for some common words.

As shown in Table 1, it is necessary that we link each word with its SVA's term in our transformation rules; otherwise the output will not have the correct transformations.

What follows is a description of how our transformation rules can be an efficient method for generating SVAs. To illustrate how we perform our transformations, we execute it on example (25). The steps below show how SVA can be generated from $\mathcal{SITL}$ formula using our transformation rules.

$$[\imath.\text{when(asserted(Awvalid), stable(Awid))}]^{\#} = [\text{asserted(Awvalid)}]^{\#} \ |\text{-> } [\text{stable(Awid)}]^{\#} \text{ (rule 1);}$$
$$[\text{asserted(Awvalid)}]^{\#} \ |\text{-> } [\text{stable(Awid)}]^{\#} = [\text{Awvalid}]^{\#} \ |\text{-> } [\text{stable(Awid)}]^{\#} \text{ (rule 10);}$$
$$[\text{Awvalid}]^{\#} \ |\text{-> } [\text{stable(Awid)}]^{\#} = \text{AWVALID} \ |\text{-> } [\text{stable(Awid)}]^{\#} \text{ (rule 28);}$$
$$\text{AWVALID} \ |\text{-> } [\text{stable(Awid)}]^{\#} = \text{AWVALID} \ |\text{-> } \$\text{stable}([\text{Awid}]^{\#}) \text{ (rule 13);}$$
$$\text{AWVALID} \ |\text{-> } \$\text{stable}([\text{Awid}]^{\#}) = \text{AWVALID} \ |\text{-> } \$\text{stable(AWID) (rule 29).}$$

In summary, we described a method for translating $\mathcal{SITL}$ to SVA. This method enables us to translate most $\mathcal{SITL}$s into SVAs. Next section will show experimental results that demonstrate the performance of our method.

5 Experimental Methods and Results

To evaluate the ideas discussed in this paper empirically, we collected documents from ARM (2010) for verification of the AXI bus protocol. These documents contain a large number of SVAs specifying system requirements together with English comments explaining their meaning. We took the English comments and we parsed them using Charniak's parser (Charniak, 2000) to produce parse trees. Then, we extracted $\mathcal{SITL}$ from parse trees via semantic rules as explained in Section 4.1. Finally, we generated SVAs from $\mathcal{SITL}$ using our transformation rules that described in Section 4.2. The total number of such comments is 397 SVAs. Our program contains 83 transformation rules and approximately 2374 words with their logical forms in SVA. Most of these words are noun categories where approximately 65% of the words are automatically generated, and approximately 35% are manually written such as words (23-27) in Table 1.

In this paper, we compare our results with those obtained by ARM (2010). We consider our assertion results are true when they are identical to the originals or having the same meaning but different expressions since temporal behaviour sequences can be expressed in more than one way. As outcomes, we found our program successfully generated SVAs for 297 out of 397, or 74.81% of all assertions. Thus, our method can be useful for generating SVA which reduces design time and errors. A second point to make is that by observing the given results, the relation between the English comments and their equivalent meaning in SVA is not complex, but rather is straightforward mapping that only need a proper formal logic featuring temporal expressions such as $\mathcal{SITL}$.

Of course, our program does not work in every case. Consider

(42) A sequence of locked transactions must use a single ID.

where the expression "a sequence of locked transactions" corresponds to multiple signals which must use the "single ID" in a special order which can not be defined from a theoretical standpoint; unless if we use a practical approach which would not be sufficient here. Thus, our program failed to generate SVAs in three circumstances, (i) when it is difficult to determine the meaning of their lexical items in SVA as shown in (42), (ii) when sentences contain words that are not introduced in the transformation rules before running our system as explained in Section 4.2, and finally (iii) when the adopted parser gives a wrong parse tree in which case no result is computed. Note that the third limitation is due to the chosen parser which can be solved with a better parser.

6 Related Work

6.1 Specifications in Natural Languages

One of common ways to generate formal specifications from natural language specifications is to use natural language processing techniques which help engineers to express system requirements with unrestricted language such as in (Osborne and MacNish, 1996; Lamar, 2009). However, although natural language processing techniques provide very respectable accuracies for real-world texts by using part-of-speech taggers and syntactic parsers, these approaches may produce multiple syntactical parses which may not be plausible at the interpretation level to eliminate the ambiguous one.

Therefore, Grover et al. (2000) and Fuchs et al. (2008) use a controlled natural language (CNL) to deal with ambiguities of natural languages. CNL is a subset of natural language with a restricted syntax and semantics in order to reduce or eliminate ambiguity and complexity of natural languages. CNL based-tools have been used in many areas such as software and hardware specifications, specifications of legal contracts, and business rule specifications. For example, Fuchs et al. (2008) uses CNL to provide a knowledge representation language in several application domains and to translate CNL's texts to *discourse representation structures* and *first-order logic*. However, most of such approaches fail to cope with natural language specifications featuring temporal constructions.

In the case of expressing temporal constructions, Clarke et al. (1986) uses a controlled English tool to convert English specifications into computation tree logic (CTL) which is used as a logical representation for hardware verification. Although CTL is commonly used for specifying temporal properties of finite-state systems, most of these approaches are not expressive enough to capture the semantics of temporal prepositions in natural language descriptions.

A more practical approach for capturing SVAs from natural language descriptions is presented in Harris (2013). This approach uses an attribute grammar approach (Engelfriet, 1984) to generate SVA. However, it fails to discuss the issues of temporal constructions in natural language descriptions. This approach offers generally a suitable way to generate SVAs from natural language requirements. On other hand, our method aims to be more focused on specifying temporal expressions in natural language assertion descriptions and provides an efficient method for generating SVAs involving temporal behaviours.

6.2 Aspectual Class

Over the past decade researchers have investigated the effects of aspectual classes of verb phrases in natural language semantics. Vendler (1967) and Dowty (1972) have worked to build a taxonomy of temporal-event descriptions to provide better descriptions of how people describe events in our language. For example, Vendler (1967) has classified verbs into four aspectual classes – states, activities, achievements and accomplishments – in order to provide the way in which verbs can be viewed with respect to time. However, the effects of these aspectual classes on generating SVAs are limited because any temporal expression at verb phrase level is restricted by the semantics of temporal prepositions. Moreover, in practice, any SVA must be checked in every clock cycle regardless what its value in the previous clock cycle. Consider

(43) When Awid is low, Awvaild is high.

(44) When Awid goes low, Awvaild goes high.

Sentence (43) has a state verb in *when*'s complement, whereas sentence (44) has an event verb. The meanings of both sentences are different in literature. In sentence (43), "Awid is low" means that it is started before the current interval, while in sentence (44), "Awid goes low" means that it is started at the current interval. However, in SVA, both mean the same since both antecedent expressions will be checked at every posedge clock. Therefore, there is not going to be any gain from the studies of aspectual classes since state and event verbs are treated similarly in the domain of interest.

However, by examining the temporal semantics of some natural language descriptions, we found some surprising insights from the semantics of temporal prepositions with aspecual classes. For example, some temporal prepositions are constrained in respect of the event types (activities, achievements or accomplishments) they can take as arguments. Consider

(45) * Awid is low after Awvaild is high.

(46) Awid goes low after Awvaild goes high.

Sentence (45) is odd because *after* preposition resists to take a state verb in its complement. However, sentence (46) is a correct statement because *after* preposition can take an event verb in its complement. Note, these restrictions are also applied to *until*, *until after*, and *before* prepositions. On other hand, *when* or *while* does not have these restrictions in which both can have either a state or an event in their complements such as in (43) and (44). Finally, we can say aspectual classes can help us to provide legal grammatical constructions of various sentence forms and their interactions with different temporal prepositions. In this paper, we do not intend to take aspectual classes into consideration in our tool because our goal is to eliminate any unnecessary complexity for usability purposes.

7 Conclusion

We presented a method for translating natural language assertion descriptions into SVAs based on $\mathcal{SITL}$. We have constructed $\mathcal{SITL}$ using $\mathcal{TPL}$. We first showed some interpretations of $\mathcal{SITL}$ in English and then we presented transformation rules for mapping $\mathcal{SITL}$ to SVAs. We developed a small program for verifying our method on AXI bus protocol in ARM (2010). Our experimental results suggest that using $\mathcal{SITL}$ as a logical representation for capturing SVAs featuring temporal expressions can enable us to have more accurate and effective results than existing tools.

In the future, we plan to extend $\mathcal{SITL}$ to handle other temporal constructions such as prepositions specifying durations – e,g. *in*, *within*, or *throughout* – or prepositions specifying particular points in time – e,g. *by* or *since*. This extension will enhance the performance of our method by representing more temporal constructions, and generate their equivalent meaning in SVA.

References

ARM, A. (2010). Axi protocol specification (rev 2.0). *Available at http://www. arm. com*.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pp. 132–139. Association for Computational Linguistics.

Clarke, E. M., E. A. Emerson, and A. P. Sistla (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems, 8*(2), 244–263.

Darringer, J. A. (1988). The application of program verification techniques to hardware verification. In *Papers on Twenty-five years of electronic design automation*, pp. 373–379. ACM.

Dowty, D. R. (1972). *Studies in the logic of verb aspect and time reference in English*. Department of Linguistics, University of Texas at Austin.

Engelfriet, J. (1984). Attribute grammars: Attribute evaluation methods. *Methods and tools for compiler construction*, 103–138.

Fuchs, N. E., K. Kaljurand, and T. Kuhn (2008). Attempto Controlled English for knowledge representation. In *Reasoning Web*, pp. 104–124. Springer.

Grover, C., A. Holt, E. Klein, and M. Moens (2000). Designing a controlled language for interactive model checking. In *Proceedings of the Third International Workshop on Controlled Language Applications*, pp. 29–30.

Harris, I. G. (2013). Capturing assertions from natural language descriptions. In *Natural Language Analysis in Software Engineering (NaturaLiSE), 2013 1st International Workshop on*, pp. 17–24. IEEE.

Holt, A. (1999). Formal verification with natural language specifications: guidelines, experiments and lessons so far. *South African Computer Journal*, 253–257.

Lamar, C. (2009). Linguistic analysis of natural language engineering requirements.

Milne, G. J. (1993). *Formal specification and verification of digital systems*. McGraw-Hill, Inc.

Montague, R. (1974). Formal philosophy; selected papers of Richard Montague.

Osborne, M. and C. MacNish (1996). Processing natural language software requirement specifications. In *Requirements Engineering, 1996., Proceedings of the Second International Conference on*, pp. 229–236. IEEE.

Pratt, I. and N. Francez (2001). Temporal prepositions and temporal generalized quantifiers. *Linguistics and Philosophy 24*(2), 187–222.

Pratt-Hartmann, I. (2005). Temporal prepositions and their logic. *Artificial Intelligence 166*(1), 1–36.

Reichenbach, H. (1947). The tenses of verbs.

Vendler, Z. (1967). *Linguistics in Philosophy*. Cornell University Press.

Vijayaraghavan, S. and M. Ramanathan (2006). *A practical guide for SystemVerilog assertions*. Springer.