Structured English for Model Checking Specification

Stephan Flake*, Wolfgang Mueller*, Jürgen Ruf**

*C-LAB, Paderborn, Germany
**University of Tübingen, Tübingen, Germany

Abstract

Model checking has received wide acceptance as a valuable technique in the field of electronic design automation and is currently of growing interest in general systems design. Though its concepts and applications are well understood it often turns out that engineers have severe problems with the specification process and the underlying notation, i.e., formulation and understanding of specifications through means of temporal logic formulae. In this article, we present an approach for a natural language–oriented representation of temporal logic formulae by introducing patterns of structured English sentences for Clocked CTL (CCTL) specification. After outlining the basic patterns of the sentences we give their semantics by a translation to CCTL. A final example demonstrates their application.

1 Introduction

As computer–based systems design deals with more complex behavior formal verification becomes widely accepted as a complementary approach for system analysis. In electronic design automation, equivalence and model checking are already frequently applied. Whereas equivalence checking can be mostly controlled from means which are tailored to the user's individual notation such as RT-diagrams, model checking requires additional education for the specification of properties by means of temporal logic formulae like CTL. It often turns out that engineers even with mathematical education have severe problems to compose property specifications in terms of CTL–related languages. A couple of approaches tackle these problems by introducing temporal logic patterns for the representation of typical properties [8, 3, 1]. However, most of them are oriented towards a specific application domain like process modeling [6] and do not completely cover other applications. Additionally, they are mostly based on CTL or LTL, so that the user still has to deal with means which are syntactically oriented towards temporal logic formulae.

We introduce patterns for the specification of properties in structured English. Those patterns are based on frequently used specification patterns identified by [1, 2]. In contrast to other approaches our structured English oriented notation is not tailored to a specific application. Sentences are compiled by a user by selecting those patterns. Specifying a query by selecting fragments of an English sentence allow users more easily to formulate required properties for a given model which gives him/her the possibility to focus on the specification of the properties rather than on the specification process itself. Furthermore, compared to pure CTL–related formulae, the specification given in structured English can be easier communicated to a second

user. Each of our sentence corresponds to a CCTL (Clocked CTL) formula to which it is translated for the RAVEN model checker [10]. CCTL (Clocked CTL) is a variant of timed CTL based on I/O-interval structures introduced by Kropf and Ruf [9].

The remainder of this paper is structured as follows. The next section gives a brief overview of related works. Section 3 introduces CCTL (Clocked CTL) which is the temporal logic representation of the underlying system. Thereafter, we present syntax and semantics of our notation where the semantics is introduced by the translation to CCTL. After a short example this articles closes with a conclusion.

2 Related Works

Most related works address the identification of CTL and LTL specification patterns. Early attempts to categorize specifications led to taxonomies that coarsely distinguish between safety and liveness properties. The classification based on LTL formulae introduced by Manna and Pnueli [8] is more fine-grained. It is ordered by syntactical criteria of formulae so that it often does not match the way of thinking when compiling a specification. A first extensive systematical classification of specifications for finite-state verification was published in [1]. The classification is based on design patterns that were originally used to capture recurring solutions to design and coding problems [3]. That classification presents a semantically ordered hierarchy of property patterns and gives detailed information for each identified pattern with respect to the requirements for well-defined design patterns. All patterns are identified by an informal description of the intent, example mappings to different specification formalisms, their known application, and relationships to similar patterns. In a more recent publication, a slightly different pattern system is presented based on the investigation of more than 500 examples for property specifications [2].

The idea of patterns has been applied not only to *classify* but also to *construct* specifications for finite-state verification. The Testbed project [6] introduces a graphical user interface for business process modeling and provides a small set of templates for verification with the SPIN model checker [5]. These templates are also denoted as patterns though they refer to concrete specifications in contrast to the previously mentioned classification patterns. The Testbed patterns consist of a number of sentences that together form a question in structured natural English language. The resulting questions are relatively simple and therefore easy to understand, but it is still necessary to give additional outlines for sentences with ambiguous interpretations, e.g., the difference between 'an element of a set' and 'each element of a set'. Time conditions are not considered at present. In [2] it is noted that many informal requirements are specified as properties of segments of program execution. To our knowledge none of these approaches considers time intervals, neither on the more abstract level of pattern classification, nor on the concrete level of specification sentences.

In a more general approach for English language oriented specification, the PROSPER project aims at the specification through a natural English language subset [4]. The user specifies the property in an interactive dialog with the system. Syntactically correct English sentences are finally converted after parsing into CTL formulae as input to the SMV model checker. Major remaining challenges are the detection of ambiguities, handling of context-sensitive expressions, and consideration of domain-specific conventions in sentences.

Compared to pattern–based approaches we provide a richer set of specifications. In contrast to temporal logic formulae based approaches a sentence in structured English can by far easier be captured by non–experts than pure CTL or LTL. Compared to unstructured English the avail-

able structured English fragments give the novel user a better guidance through the allowable and non–allowable specifications with less iterations.

3 CCTL

CCTL (Clocked CTL) is a temporal logic in the context of the real-time system model checker RAVEN. In RAVEN, the model is specified in terms of so–called I/O-interval structures. I/O-interval structures are basically state transition systems with time annotations. A time annotation is a time constraint which specifies a [min,max]-time interval for a state transition [9]. CCTL is the underlying temporal logic for the specification of properties with additional timing analysis issues. This article only informally sketches the CCTL notation. For formal details the reader is referred to [10].

The CCTL syntax can be summarized as follows:

$$\begin{array}{rl} & a \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \mid \phi \leftrightarrow \phi \mid \phi \oplus \phi \\ \phi := & \mid \mathbf{EX}_{[x]}\phi \mid \mathbf{EF}_{[x,y]}\phi \mid \mathbf{EG}_{[x,y]}\phi \mid \mathbf{E}(\phi \underline{\mathbf{U}}_{[x,y]}\phi) \mid \mathbf{E}(\phi \mathbf{S}_{[x]}\phi) \mid \mathbf{E}(\phi \mathbf{C}_{[x]}\phi) \\ & \mid \mathbf{AX}_{[x]}\phi \mid \mathbf{AF}_{[x,y]}\phi \mid \mathbf{AG}_{[x,y]}\phi \mid \mathbf{A}(\phi \underline{\mathbf{U}}_{[x,y]}\phi) \mid \mathbf{A}(\phi \mathbf{S}_{[x]}\phi) \mid \mathbf{A}(\phi \mathbf{C}_{[x]}\phi) \end{array}$$

where a is an atomic proposition, and $x \in \mathbb{N}_0, y \in \mathbb{N}_0 \cup \{\infty\}$ are time bounds. The symbol ∞ is defined through: $\forall i \in \mathbb{N}_0 : i < \infty$. All interval operators can also be accompanied by a single time-bound only. In this case the lower bound is set to zero by default. If no interval is specified, the lower bound is implicitly set to zero and the upper bound is set to infinity. If the x-operator has no time bound, it is implicitly set to one.

The **A** resp. **E** are run quantifiers. A run is an infinite sequence of pairs of states and clock values (called a configuration) of the system. Every pair of states and clock values may be the start point of a run. The **A**-quantifier checks for the actual configuration if all runs satisfy the temporal operator. The **E**-quantifier checks if there exists at least one run starting in the actual configuration satisfying the temporal operator. Model checking examines the properties with respect to the initial states of the system. Table 1 gives an informal description of the temporal operators.

$\mathbf{X}_{[x]}\ \phi$	The configuration reachable in x time steps on the actual run satisfies the formula ϕ
$\mathbf{F}_{[x,y]}\ \phi$	A configuration satisfying ϕ is reachable within the interval $[x, y]$
$\mathbf{G}_{[x,y]}\ \phi$	All configurations within the interval $[x, y]$ satisfy ϕ
$\phi \mathbf{U}_{[x,y]} \psi$	A configuration satisfying ψ is reachable within the interval $[x, y]$ and all configurations before (on this run) satisfy ϕ
$\phi\ \mathbf{S}_{[x]}\ \psi$	The configurations from the beginning of the run up to the time $x-1$ satisfy ϕ, the configuration at time x satisfies ψ
$\phi\ \mathbf{C}_{[x]}\ \psi$	If the configurations from the beginning of the run up to the time $x-1$ satisfy ϕ, then the configuration at time x has to satisfy ψ

Table 1: Informal Description of the Temporal Operators

In addition to the previous specifications RAVEN supports three timing analysis algorithms: MIN(*start*, *target*), MAX(*start*, *target*), and STABLE(*region*). Typical queries concern the minimal and maximal answering time of a reactive system, as well as the guarantee of the

maximal stability of a state. Since this sort of specification is not discussed in the remainder of this article the reader is referred to [11] for more detailed information.

4 Structured English Sentences for Specification

We introduce a structured English oriented notation as a user front-end for the specification of Clocked CTL formulae for model checking. Natural English sentences are compiled from (i) lists (interpreted as a set or a sequence) of given states and (ii) a set of fragments in natural English. The user compiles a specification by selecting those lists and fragments and combines them to meaningful sentences of limited length and complexity. The fragments are so defined that they only combine to sentences which directly correspond to CCTL formulae. In order to provide a sufficient coverage in application the introduced fragments are derived from patterns which are identified as being frequently used in [1, 2].

The remainder of this section separately presents the syntactical and semantical aspects of those sentences.

4.1 Syntax

A specification sentence has a very simple basic structure with a scope and a specification where the latter can be given as a subsentence with condition and consequence. The sentences encapsulate *state specifications* which are lists of state identifiers. Due to the context those lists are interpreted as sets or sequences. Figure 1 gives the definition of the complete syntax in Extended BNF where terminals are given in capital letters.

In details, each specification sentence is composed of a main scope followed by '`that`' and a main specification (Rule 1). Main specifications are distinguished into sequence, set, and complex specifications (Rule 3). The first two are simple specifications which range over a sequence of states or single/all elements of a set of states. The latter one defines a relation between elements of two subspecifications S_1 and S_2 of form '`if` S_1 `...then ...`S_2 ' (Rule 15) where S_2 stands for a *consequence* with a scope and a specification (Rule 16) separated by '`that`'. The consequence specification can be a *sequence specification* or *set specifications* again. The sequence specification structure is the same as in the main specification. However, we introduce a small variation of the set specification in the context denoted as set subspecification (Rule 21). Since interval definitions are of no use in consequences when investigating the occurrence of single elements we do not allow their definition here (Rule 23: '`occurs`' without setScope).

4.2 Semantics

We define the semantics of sentences by their translation to CCTL formulae. Since the translation of sentence fragments is context sensitive it is not possible to simply define a self-contained formula in CCTL for each fragment of a sentence, although this might be possible for other temporal logics. So each derived sentence translates to a different formula.

Table 2 defines the mapping of the main scopes of main specifications where the Δ stands for one of the operators **F**, **G**, and **X**. The use of those operators depends on the occurrence modifier in the subsequent specifications.

Since it is not possible to enumerate all translations here we only give the mapping for the most important ones. Thus, the remainder of this section focuses on the mapping for the main

```
1:  sentence           ::= mainScope 'that' mainSpecification .
2:  mainScope          ::= 'It is' ['at all times'] ('true'|'possible'|'inevitable').
3:  mainSpecification  ::= sequenceSpec | setSpec | complexSpec .
4:  sequenceSpec       ::= seqType stateSpec seqScope seqOccurrence .
5:  seqType            ::= 'a weak sequence' | 'a strong sequence' .
6:  seqScope           ::= 'with time interval(s)''['INTEGER','(INTEGER|'infinity')']' |
                           'with arbitrary time interval(s)'                          |
                           'with interval(s) of exactly''['INTEGER']' 'time unit(s)'.
7:  seqOccurrence      ::= 'occurs' | 'does not occur' .
8:  setSpec            ::= (singleElements|allElements) setScope .
9:  singleElements     ::= 'an element of set' stateSpec simpleOccurrence .
10: simpleOccurrence   ::= ['always'|'never'] 'occurs' .
11: allElements        ::= 'all elements of set' stateSpec combinedOccurrence .
12: combinedOccurrence ::= ['always'|'never'] 'occur' combineModifier .
13: combineModifier    ::= 'simultaneously' | 'exclusively' .
14: setScope           ::= 'within the interval' '['INTEGER ',' (INTEGER|'infinity') ']' |
                           'within an arbitrary interval'                               |
                           'in exactly [' INTEGER '] time unit(s)' .
15: complexSpec        ::= 'if' consequenceCond 'then' consequence .
16: consequence        ::= consequenceScope 'that' consequenceSpec .
17: consequenceCond    ::= setSubSpec | sequenceSpec .
18: consequenceSpec    ::= setSubSpec | sequenceSpec .
19: consequenceScope   ::= followModifier 'within' '['INTEGER','INTEGER']' 'time unit(s)' |
                           followModifier 'at least after' '['INTEGER']' 'time unit(s)'   |
                           followModifier 'in exactly' '['INTEGER']' 'time unit(s)'       |
                           followModifier 'in' '['INTEGER']' 'time unit(s) at the latest' |
                           followModifier 'at some time later' .
20: followModifier     ::= 'it must happen' | 'it is possible' .
21: setSubSpec         ::= subSingleElements | subAllElements .
22: subSingleElements  ::= 'an element of set' stateSpec subSimpleOccurrence .
23: subSimpleOccurrence::= 'occurs' | ('always'|'never') 'occurs' setScope.
24: subAllElements     ::= 'all elements of set' stateSpec combinedOccurrence setScope.
25: stateSpec          ::= '{' STATE_IDENTIFIER idList '}' .
26: idList             ::= ',' STATE_IDENTIFIER idList | .
```

Figure 1: Syntax of Structured English Sentences

scope '`It is at all times possible that ...`' (resp. **AG E**Δ). Other specifications can then be easily derived by replacing **AG E**Δ. Note here that the mapping is different when the global scope changes to **AG** or **AG A**Δ. In those cases further occurrences of **E**-quantifiers have to be replaced by an **A**–quantifier in the CCTL formula. As some of the provided interval scopes are only special cases of the arbitrary interval $[x, y]$, we only outline CCTL formulae in which x and y are arbitrary numbers explicitly allowing $y = \infty$.

We start our investigation considering the substitution and mapping of sequence and set specifications. For sequences, we distinguish between two different types. Let $A = [a_1, \ldots, a_n]$ be a sequence of states. A *weak sequence* A allows arbitrary intermediate configurations between a_i and a_{i+1}, while a *strong sequence* A requires for all a_i that a_{i+1} is directly reachable from a_i. The given intervals refer to the number of allowed time steps (time units) between subsequent a_i. We present two of the possible specifications for weak sequences here from which the remaining weak sequence specifications can then be easily derived.

1. `It is at all times possible that a weak sequence` A `with time interval(s)` $[x, y]$ `occurs`

   $\Rightarrow$ **AG EF** $(a_1 \wedge \mathbf{EF}_{[x,y]}(a_2 \ldots \wedge \mathbf{EF}_{[x,y]}\ (a_n)\ldots))$

Main Scope	CCTL Operators
`It is true ...`	—
`It is possible ...`	$\mathbf{E}\Delta\ (\ldots)$
`It is inevitable ...`	$\mathbf{A}\Delta\ (\ldots)$
`It is at all times true ...`	$\mathbf{AG}\ (\ldots)$
`It is at all times possible ...`	$\mathbf{AG}\ \mathbf{E}\Delta\ (\ldots)$
`It is at all times inevitable ...`	$\mathbf{AG}\ \mathbf{A}\Delta\ (\ldots)$

Table 2: Mapping of Main Scopes to CCTL

2. `It is at all times possible that a weak sequence` A `with time interval(s)` $[x, y]$ `does not occur`

$$\Rightarrow \mathbf{AG}\ \mathbf{EG}\ (\ \neg a_1 \vee (a_1 \rightarrow \neg\mathbf{AF}_{[x,y]}(a_2)) \vee (a_1\ \wedge \mathbf{EF}_{[x,y]}(a_2\ \rightarrow \neg\mathbf{AF}_{[x,y]}a_3)) \vee \ldots (a_1\ \wedge \mathbf{EF}_{[x,y]}(a_2\ \wedge \mathbf{EF}_{[x,y]}(a_3\ \wedge \ldots \mathbf{EF}_{[x,y]}(a_{n-1} \rightarrow \neg\mathbf{AF}_{[x,y]}a_n)\ldots))))$$

When the interval scope with `exactly` $[x]$ `time unit(s)` is selected, all $\mathbf{EF}_{[x,y]}$ are replaced by $\mathbf{EX}_{[x]}$ in the above formulae. CCTL formulae for strong sequences are realized by applying the successor-operator and the strong until-operator which ensures that the subsequent configuration is reached.

3. `It is at all times possible that a strong sequence` A `with time interval(s)` $[x, y]$ `occurs`

$$\Rightarrow \mathbf{AG}\ \mathbf{EF}\ \mathbf{E}\ (a_1\ \underline{\mathbf{U}}_{[x,y]}(\mathbf{E}\ (a_2\ \underline{\mathbf{U}}_{[x,y]}(\mathbf{E}\ (a_3\ \underline{\mathbf{U}}_{[x,y]}\ \ldots a_n)\ldots))))$$

4. `It is at all times possible that a strong sequence` A `with time interval(s)` $[x, y]$ `does not occur`

$$\Rightarrow \mathbf{AG}\ \mathbf{EG}\ (\ \neg a_1 \vee (a_1 \rightarrow \neg\mathbf{A}(a_1\underline{\mathbf{U}}_{[x,y]}a_2)) \vee \mathbf{E}\ (a_1\ \underline{\mathbf{U}}_{[x,y]}\ (a_2 \rightarrow \neg\mathbf{A}(a_2\underline{\mathbf{U}}_{[x,y]}a_3))) \vee \ldots \mathbf{E}\ (a_1\ \underline{\mathbf{U}}_{[x,y]}\mathbf{E}\ (a_2\ \underline{\mathbf{U}}_{[x,y]}(\mathbf{E}\ (a_3\ \underline{\mathbf{U}}_{[x,y]} \ldots (a_{n-1} \rightarrow \neg\mathbf{A}(a_{n-1}\underline{\mathbf{U}}_{[x,y]}a_n))\ldots)))))$$

For set specifications, we also focus only on the most relevant combinations. Note here, that in some formulae the time scope is directly attached to the main scope operator. When all elements of a set A are considered, it must be additionally defined whether the elements occur simultaneously or exclusively.

5. `It is at all times possible that an element of set A occurs within the interval` $[x, y]$

$$\Rightarrow \mathbf{AG}\ \mathbf{EF}_{[x,y]}(a_1\ \vee\ \ldots\ \vee a_n)$$

6. `It is at all times possible that all elements of set A occur simultaneously within the interval` $[x, y]$

   $\Rightarrow \mathbf{AG}\ \mathbf{EF}_{[x,y]}(a_1\ \wedge\ \ldots\ \wedge a_n)$

7. `It is at all times possible that all elements of set A occur exclusively within the interval` $[x, y]$

$$\Rightarrow \mathbf{AG}\ \mathbf{EF}_{[x,y]}(\ \begin{array}{ll}(a_1 \rightarrow \neg(a_2\ \vee\ a_3\ \vee\ \ldots\ \vee\ a_n)) & \wedge\\ (a_2 \rightarrow \neg(a_1\ \vee\ a_2\ \vee\ \ldots\ \vee\ a_n)) & \wedge\\ \ldots & \\ (a_n \rightarrow \neg(a_1\ \vee\ a_2\ \vee\ \ldots\ \vee\ a_{n-1}))\ ) & \end{array}$$

Given the previous translations, it should be easily possible to derive the remaining translations of the other combinations. In order to extend the sequence scope to global occurrence, i.e., occurrence during the complete interval $[x, y]$, all **F**–operators from above have to be replaced by a **G**–operator. Correspondingly, in specifications defining the absence of configurations the **G**–operator has to be applied as follows:

8. `It is at all times possible that an element of set A never occurs within the interval` $[x, y]$

   $\Rightarrow \mathbf{AG}\ (\mathbf{EG}_{[x,y]}(\neg a_1)\ \vee \mathbf{EG}_{[x,y]}(\neg a_2)\ \ldots\ \vee \mathbf{EG}_{[x,y]}(\neg a_n))$

9. `It is at all times possible that all elements of set A never occur simultaneously within the interval` $[x, y]$

   $\Rightarrow \mathbf{AG}\ \mathbf{EG}_{[x,y]}(\neg(a_1\ \wedge\ a_2\ \wedge\ \ldots\ a_n))$

10. `It is at all times possible that all elements of set A never occur exclusively within the interval` $[x, y]$

$$\Rightarrow \mathbf{AG}\ \mathbf{EG}_{[x,y]}(\ \begin{array}{ll}\neg(a_1 \rightarrow \neg(a_2\ \vee\ a_3\ \vee\ \ldots\ \vee\ a_n)) & \vee\\ \neg(a_2 \rightarrow \neg(a_1\ \vee\ a_2\ \vee\ \ldots\ \vee\ a_n)) & \vee\\ \ldots & \\ \neg(a_n \rightarrow \neg(a_1\ \vee\ a_2\ \vee\ \ldots\ \vee\ a_{n-1}))\ ) & \end{array}$$

The previous paragraphs outlined mappings for simple sequence and set specifications. Although parts of those formulae can also be applied in consequence specifications, it is not valid to just concatenate two derived formulae. Instead, they have to be combined more meaningful into a third one. Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ be two sets or sequences of states. The following examples give possible combinations to compose them for a consequence specification:

11. `It is at all times true that if a strong sequence` A `with time interval(s)` $[x_1, y_1]$ `occurs then it must happen within` $[x_2, y_2]$ `time unit(s) that an element of set` B `occurs`

    $\Rightarrow \mathbf{AG}\ (\mathbf{A}(a_1\ \underline{\mathbf{U}}_{[x_1,y_1]}(\ \mathbf{A}(\ldots \underline{\mathbf{U}}_{[x_1,y_1]}(a_n\ \rightarrow \mathbf{AF}_{[x_2,y_2]}(b_1 \vee \ldots \vee b_m)\ldots))))$

12. `It is inevitable that if all elements of set` A `occur simultaneously within the interval` $[x_1, y_1]$ `then it is possible at some time later that a weak sequence` B `with time interval(s)` $[x_2, y_2]$ `occurs`

    $\Rightarrow \mathbf{AF}\ \mathbf{EF}_{[x_1,y_1]}((a_1 \wedge \ldots \wedge a_n) \rightarrow \mathbf{EF}_{[1,\infty]}(b_1 \wedge \mathbf{EF}_{[x_2,y_2]}(b_2 \ldots \wedge \mathbf{EF}_{[x_2,y_2]}\ (b_m)\ldots)))$

13. It is at all times true that if all elements of set A always occur simultaneously within the interval $[x_1, y_1]$ then it must happen in exactly $[z]$ time unit(s) that all elements of set B occur simultaneously within the interval $[x_2, y_2]$
$\Rightarrow \mathbf{AG}\ \mathbf{A}((a_1 \wedge \ldots \wedge a_n)\mathbf{C}_{[y_1-x_1]}(\mathbf{AX}_{[z]}\mathbf{AG}_{[x_2,y_2]}(b_1 \wedge \ldots \wedge b_m)))$

5 Example

We outline the previously introduced sentences the example of a simple sequential digital circuit. The behavior of gates and flip-flops is modeled as follows. For gates, impulses on the inputs which are shorter than the delay time are suppressed. Only if a signal remains stable for the minimum delay time, the gate changes at the output. For modeling flip-flops, we assume a setup and a hold time. If the input signals violate these timing constraints, the flipflop remains in its current state. As an example we choose the single pulser circuit [7] which is shown in Figure 2.

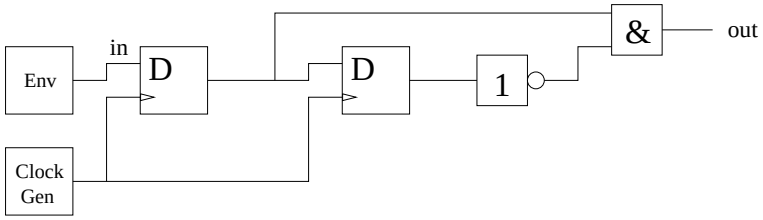

Figure 2: A Single Pulser

Figure 3 shows the behavior of the parts where initial states are marked bold. For the AND gate, we use edge sensitive presentation of the behavior: When the system starts in state *high* and the inputs fulfill $\neg a_1 \vee \neg a_2$ for δ time units then it changes to state *low*. If the inputs fulfill $a_1 \wedge a_2$ before δ time units are passed, the structure remains in state *high*. The clock generator is modeled by an I/O-interval structure with two states (*high* and *low*) which toggles between both states every cycle. The environment (Env) is either given by a user pressing the button (which should be single pulsed) or by a bouncing pulse. In both cases, I/O-interval structures as given in Figure 3 allow a very clear and compact modeling of timing constraints and communication behavior.

Based on this example, we specify that a raising edge on the input implies a raising edge on the output. Since we consider delayed gates, we assume that the input first remains low for 2 clock cycles and then keeps high for 2 clock cycles:

It is at all times true that if a strong sequence $\{\neg in, in\}$ with interval(s) of exactly $[2\delta_{clock}]$ time unit(s) occurs it must happen at some time later that a strong sequence $\{\neg out, out\}$ with interval(s) of exactly $[1]$ time unit(s) occurs.
$\Rightarrow \mathbf{AG}\ (\ \mathbf{A}(\neg in\ \mathbf{C}_{[2\delta_{clock}]}(in \rightarrow \mathbf{AF}(\neg out \wedge \mathbf{EX}_{[1]}out))))$

The second specification verifies that when the output becomes high, it remains high for one cycle period.

It is at all times true that if a strong sequence $\{\neg out, out\}$ with interval(s) of exactly $[1]$ time unit(s) occurs it must happen in exactly $[0]$ time unit(s) that an element of $\{out\}$ always occurs within the interval $[0, 2\delta_{clock}]$.
$\Rightarrow \mathbf{AG}\ (\ \mathbf{A}(\neg out\ \mathbf{C}_{[1]}(out \rightarrow \mathbf{AG}_{[0,2\delta_{clock}-1]}out)))$

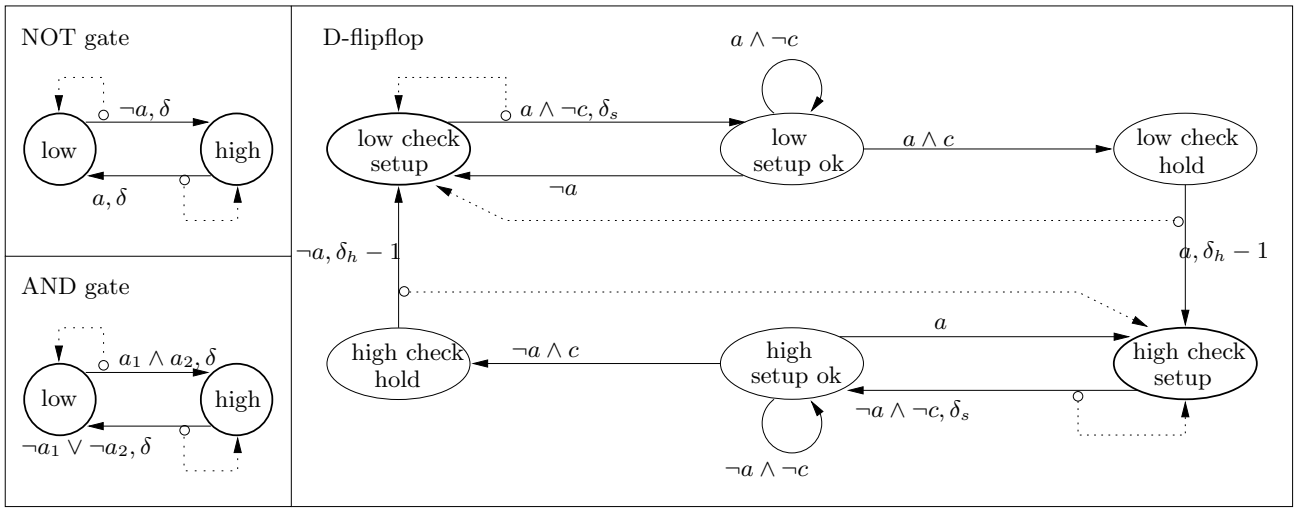

Figure 3: Gates with input a, a_1, a_2, clock c, delay time δ resp. setup δ_s and hold δ_h

6 Conclusion

We have introduced patterns for compiling structured natural English sentences for specification of real-time properties in model checking. The sentences are introduced by their syntax and semantics where the semantics is given by a translation to a CCTL formula. Due to page limitation we only introduced the translation of the most relevant structures. Note here, that the presented approach also covers the treatment of more advanced CCTL timing specification queries like STABLE, MIN, and MAX [11]. The specification of those properties follows the same patterns, e.g., '`How long does it take to get from` A `to all configurations of` B`?`'

Due to investigations given in [1] we think that the presently supported patterns are sufficient to cover the most typical cases for general systems design. However, circuit–specific verifications in final phases sometimes require CCTL formulae which are more complex than those which are currently covered by our notation. Such complex specifications would also result in much more complex English sentences. Though our approach can easily scale to complex structures we currently do not see how these sentences can be easily controlled and understood by a user. An advanced visual framework might help to control these complexity. But this clearly needs further investigations.

In our first experiments we found out that the meaning of some of our composed sentences still allow ambiguous interpretations. Consider the expression '`... all elements of set A occur exclusively ...`', for instance. One interpretation can be that *all elements have to occur and when they occur it has to be exclusive*. An alternative interpretation is that *if an element occurs the occurrence has to be exclusive*. In combination with the already considerably long sentences this effect can result in a significant drawback. We are currently investigating alternatives to overcome both drawbacks by either compiling shorter sentences and/or providing additional textual outlines of the intended semantics.

We are aware that the work presented here is only a first step towards a more advanced user interface which is supposed to be more acceptable for the general user with basic engineering and mathematical education. However, we think that this still requires significant future work since the presented approach has to be seamlessly integrated in an interactive user interface in order to become well applicable.

Acknowledgements

The work described herein is performed in the GRASP project (GRApical Specification and Real-Time Verification of Production Automation Systems) which is funded by the DFG under the Schwerpunktprogramm 'Integration von Techniken der Softwarespezifikation für ingenieurwissenschaftliche Anwendungen'. We also would like to thank the unknown reviewers for their valuable comments.

References

[1] Matthiew B. Dwyer, George S. Avrunin, and James C. Corbett. Property Specification Patterns for Finite-State Verification. In M. Ardis, editor, *Proceedings of the 2nd Workshop on Formal Methods in Software Practice, Clearwater Beach, Florida*, pages 7–15, March 1998.

[2] Matthiew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in Property Specifications for Finite-State Verification. In *21st International Conference on Software Engineering, Los Angeles, California*, May 1999.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reuseable Object-Oriented Software*. Addison-Wesley, 1994.

[4] Alexander Holt and Ewan Klein. A semantically-derived subset of English for hardware verification. In *37th Annual Meeting of the Association for Computational Linguistics: Proceedings of the Conference*, pages 451–456, University of Maryland, College Park, Maryland, USA, June 1999.

[5] Gerald J. Holzmann. The Model Checker SPIN. In *IEEE Transactions on Software Engineering, Vol. 23, No. 5*, May 1997.

[6] Wil Janssen, Radu Mateescu, Sjouke Mauw, Peter Fennema, and Petra van der Stappen. Model Checking for Managers. In Dennis Dams, Rob Gerth, Stefan Leue, and Mieke Massink, editors, *Theoretical and Practical Aspects of SPIN Model Checking, Proceedings of the 5th and 6th International SPIN Workshops, Trento, Italy, July 5, 1999, Toulouse, France, September 21 and 23, 1999*, number 1680 in Lecture Notes in Computer Science, pages 92–107, Heidelberg, September 1999. Springer.

[7] S. Johnson, P. Miner, and A. Camilleri. Studies of the Single Pulser in Various Reasoning Systems. In *International Conference on Theorem Provers in Circuit Design (TPCD), Bad Herrenalb, Germany*, number 901 in Lecture Notes in Computer Science, Heidelberg, 1995. Springer.

[8] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1991.

[9] Jürgen Ruf and Thomas Kropf. Symbolic Model Checking for a Discrete Clocked Temporal Logic with Intervals. In E. Cerny and D.K. Probst, editors, *Conference on Correct Hardware Design and Verification Methods (CHARME)*, pages 146–166, Montreal, Canada, October 1997. IFIP WG 10.5, Chapman and Hall.

[10] Jürgen Ruf and Thomas Kropf. Modeling and Checking Networks of Communicating Real-Time Systems. In *Correct Hardware Design and Verification Methods (CHARME 99)*, pages 265–279. IFIP WG 10.5, Springer, September 1999.

[11] Jürgen Ruf and Thomas Kropf. Analyzing Real-Time Systems. In *DATE 2000, to appear*. IEEE - Computer Society Press, March 2000.