A Property Specification Tool for Generating Formal Specifications: Prospec 2.0
Irbis Gallegos\textsuperscript{1}, Omar Ochoa\textsuperscript{1}, Ann Gates\textsuperscript{2}, Steve Roach\textsuperscript{2}, Salamah Salamah\textsuperscript{3}, Corina Vela\textsuperscript{1}
\textsuperscript{1,2} Department of Computer Science, The University of Texas at El Paso, El Paso, TX 79968, USA
\textsuperscript{3} Department of Computer and Software Engineering, Embry Riddle Aeronautical University, Daytona Beach, FL 32114, USA
\textsuperscript{1,2}\{}ibrisg, omar, cyvela\}\{}@miners.utep.edu, \textsuperscript{2}\{}agates, sroach\}\{}@utep.edu, \textsuperscript{3}\{}Salamahs@erau.edu

Abstract

Numerous formal approaches to software assurance are available, including: runtime monitoring, model checking, and theorem proving. All of these approaches require formal specifications of behavioral properties to verify a software system. Creation of formal specifications is difficult, and previously, there has been inadequate tool support for this task. The Property Specification tool, Prospec, was developed to assist users in the creation of formal specifications. This paper describes Prospec 2.0, an improvement to the previous version, by addressing the results of a study conducted to assess the usability of the tool and by adding functionality that supports the validation process.

1. Introduction

Formal methods to support software assurance require the identification of behavioral properties of the software system, generation of formal specifications for the properties, validation of the specifications, and verification of the correctness of the system. The effectiveness of the assurance approach depends on the quality of the formal specifications, and a significant hurdle to the use of formal approaches is the development of correct formal specifications.

Typically, the person creating the formal specification must have a strong mathematical background and be aware of the subtleties of the specification language. For example, model checkers, such as SPIN \textsuperscript{1} and NuSMV \textsuperscript{2} use formal specifications written in Linear Temporal Logic (LTL) \textsuperscript{3}, which can be difficult to read, write, and validate. This problem is compounded if requirements must be specified in more than one formal language, which frequently is the case if more than one verification tool is used. The specifier must be aware of the differences in expressiveness of each of the target languages.

The Property Specification (Prospec) 1.0 tool was developed to address some of these challenges. Prospec uses the Specification Pattern System (SPS) \textsuperscript{4} and Composite Propositions (CP) \textsuperscript{5} to assist developers in the elicitation and specification of system properties.

Usability studies of Prospec have shown that it facilitates the elicitation, understanding, and specification of formal properties \textsuperscript{19}.

This paper describes Prospec 2.0. In particular, it describes the new features in Prospec that are aimed at improving the tool’s support for generating and validating formal property specifications.

2. Background

The Specification Pattern System (SPS) \textsuperscript{4} is a set of patterns used to assist in the formal specification of properties for finite-state verification tools. SPS patterns are high-level abstractions providing descriptions of common properties that hold on a sequence of conditions or events in a finite state model. SPS patterns characterize two behavioral aspects: the occurrence and the order of events or conditions.

Occurrence patterns are \textit{universality}, \textit{absence}, \textit{existence}, and \textit{bounded existence}. Order patterns are \textit{precedence}, \textit{response}, \textit{chain of precedence} and \textit{chain of response}. In SPS, a chain pattern defines a sequence of events or conditions. \textit{Chain-precedence} and \textit{chain-response} patterns permit specifying a sequence of events or conditions as a parameter of precedence or response patterns, respectively. SPS restricts the specification of sequences to precedence and response patterns.

In SPS, a pattern is bounded by the scope of computation over which the pattern applies. The beginning and end of the scope are specified by the conditions or events that define the left (L) and right (R) boundaries, respectively.

A study by Dwyer et. al. \textsuperscript{4} identified the \textit{response} pattern as the most commonly used pattern, followed by the \textit{universality} and \textit{absence} patterns. These three patterns accounted for 80\% of the 580 properties sampled in the study. Because of the frequency with which response properties occur, it is important to provide abstractions that support multiple propositions when specifying sequence of events or concurrent behavior. Because multiple propositions may occur in the cause and effect part of response properties, CPs can be used to assist in their specification and validation. By using CPs in either part of the response pattern (the cause or effect), it is possible to represent common behavior associated with
concurrent systems, such as synchronized join and fork, concurrency, non-determinism, and sequences.

Mondragon et al. [5] introduced Composite Propositions (CPs) to handle pattern and scope parameters that represent multiple conditions or events. The introduction of CPs supports the specification of concurrency, sequences, and non-consecutive sequential behavior on patterns and scope. Mondragon proposes a taxonomy with twelve classes of CPs. In this taxonomy, each class defines a detailed structure for either concurrent or sequential behavior based on the types of relations that exist among a set of propositions.

The original version of Prospec is an automated tool that guides a user in the development of formal specifications. It includes patterns and scopes, and it uses decision trees to assist users in the selection of appropriate patterns and scopes for a given property. Prospec 1.0 extends the capability of SPS by supporting sequences, and concurrency. Prospec to specify ordered sequences, non-deterministic pattern or scope that is comprised of multiple conditions or events. The use of CP classes allows practitioners using Prospec to specify ordered sequences, non-deterministic sequences, and concurrency. By using CPs, a practitioner is directed to clarify requirements, which leads to reduced ambiguity and incompleteness of property specifications.

Prospec uses guided questions to distinguish the types of scope or relations among multiple conditions or events. By answering a series of questions, the practitioner is lead to consider different aspects of the property. A type of scope or CP class is identified at the end of guidance. Prospec generates formal specifications in Future Interval Logic (FIL) [7] and the Meta-Event Definition Language (MEDL) [8].

3. PROSPEC 2.0

3.1 Prospec Revisions

A formal experiment evaluated the effects that the original Prospec and SPS have over the quality of the generated software property specifications with respect to completeness and correctness [19]. SPS supports the creation of specifications through a web site and manual substitution of propositions into templates. The following research hypothesis was supported: users who specify software properties using Prospec, identify, on the average, more correct patterns and scopes than users who specify software properties using the SPS web site. The subjects also provided comments for Prospec in the post-evaluation form. They suggested that Prospec:

- provide the capability to access all the properties defined in a given project;
- allow the capability to apply the negation operator to propositions;
- indicate the properties that contain a recorded assumption; and
- modify the physical position and labels for parameters $S$ and $P$ in the response and precedence patterns in the pattern screen.

These and other observations made when using Prospec motivated the creation of Prospec 2.0. The revised tool includes changes to the user interface and, more significantly, the tool is being revised to generate LTL specifications with support for validation of the specifications. To provide the ability to export properties into other software tools, Prospec 2.0 uses XML.

3.2 Linear Temporal Logic Generation

Salamah et al. [10, 17] showed that direct substitution of one or more parameters for a pattern or scope that includes CPs may result in a specification that does not meet the intent of the user. Consider the following example: “The delete button is enabled in the main window only if the user is logged in as administrator and the main window is invoked by selecting it from the Admin menu.” The property could be classified Existence(P) with Before R scope, and P is a classified as Eventual((p1,p2)\textsuperscript{1}) where p1 denotes “User logged in as an administrator,” p2 denotes “Main window is invoked,” and R denotes “Delete Button is enabled.” The SPS template for Existence(P) Before R is ($\diamond R$)→($!R \ U (P \ A !R)$) and the formula for Eventual\textsubscript{c} is $(p_1 \ X (p_2 \ U p_2))$. Direct substitution would yield:

$$\diamond R \rightarrow (!R \ U (P \ X (p_1 \ U p_2))).$$

This, however, would permit the delete button to be enabled between the time that the administrator logs in and the administrator invokes the main window.

To address this, Salamah [17] introduced general templates to support the generation of LTL formulas that use CPs. For example, consider the Response ($P, Q$) pattern with Global scope in which $P$ and $Q$ are Consecutive(p1, p2)\textsuperscript{j}, i.e., (p1 L X (p2)) and Parallel\textsubscript{c} (q1, q2)\textsuperscript{j}, i.e., (q1 L q2). The general template for the “Response- Global scope” is:

$$\Box (P^{\text{LTL}} \rightarrow (P^{\text{LTL}} \ & \ \textup{$\bigcirc$} Q^{\text{LTL}})).$$

where $P^{\text{LTL}}$ and $Q^{\text{LTL}}$ represent LTL formulas for the CP class and $\&$ is a special operator that ensures that $\textup{$\bigcirc$} Q^{\text{LTL}}$ is “anded” only with the last element of the sequence represented by $P^{\text{LTL}}$ [10, 17]. As a result the formula becomes:

$$\Box ((p_1 \ X \ p_2)) \rightarrow (p_1 \ X \ p_2 \ \& \ \textup{$\bigcirc$} (q_1 \ L \ q_2))).$$

\* The complete list of CP classes and their LTL descriptions is available in Mondragon et. al. [5].
In the previous example for enabling the delete button, the general template for the \texttt{Existence(P)} with \texttt{Before R} scope example described earlier is: \( !((!(P^{LTL}) \& \neg R^{LTL})) \cup R^{LTL}) \), where \& is a special operator that denotes that \(!R^{LTL}\) holds at all the states within the sequence represented by \(P^{LTL}\) [10, 17]. As a result, the correct formula is:

\[
!(((!(p_1 \& !R) \& X((p_2 \& !R))U (p_2 \& !R)))) U R)
\]

To support validation of generated formulas, each general formula has associated traces of computations that represent behaviors that are accepted or not accepted by the LTL formula being tested. The traces of computation can be used to test the formulas using a model checker [18, 22]. In addition to supporting the process of testing the LTL formulas, these templates provide visual representations to aid the user in understanding the meaning of the complex LTL formulas that are generated, helping those users who are not immersed in formal representations.

Salamah [17] used multiple techniques to verify the correctness of the general formulas: formal proofs, testing, and reviews. In order to support the formal proofs, the work included formal definitions of patterns and scopes that use CPs. This supports the ability to define similar properties in other specification languages.

A secondary effort in Salamah’s work involved modifying the original LTL formulas provided by Prospec for patterns and scopes whose parameters contained only single propositions and CP. The approach to simplify the formulas was to reduce the number of states in the Büchi automaton (BA) generated from an LTL formula. The size (number of states) of the automaton that results from the intersection of the BA generated by the LTL formula and that of the system model has as its upper bound the product of the number of states in each of the two. Work has been done by other researchers on the translation of LTL to BA to reduce the number of states in the resulting BA and to speed up the process of the BA generation [14, 15, 16]. It was possible to reduce the number of temporal operators, and as a result improve the efficiency of 17 out of 30 the formulas defined by Prospec [23].

### 3.3 Interface

Prospec 2.0 includes new features to support the specification of properties. The interface maintains the support and functionality of the original Prospec, i.e., the guiding screens for selecting scope and patterns remain the same. The main changes to the interface are related primarily to CP specifications and information presentation.

Figure 1. presents the main screen for Prospec 2.0. To the left in the main screen the \texttt{Property Browsing Tree} is shown. The larger frame on the right encloses screens (as opposed to the original implementation in which individual screens were distributed over the available screen space). The larger frame provides practitioners with a separate context for each property. The enclosing frame enables concurrent property specification as well as easy transition between these property specifications.

Since Prospec 2.0 supports concurrent property specifications, a \texttt{Property Browsing Tree} is available for accessing properties. The \texttt{Property Browsing Tree} allows practitioners to browse, traverse and quickly preview properties being specified. Also, the \texttt{Property Browsing Tree} allows editing of properties attributes such as scope, pattern, CPs, and propositions, as shown in Figure 1. Once a user selects a property attribute in the tree, the appropriate window will be opened allowing modification of the property attribute.

Another interface improvement is the \texttt{Visual Representation window}. This window will provide a visual representation of the specified property as a trace of computation that shows the scope and the specified property pattern. The visual representation in conjunction with the written description will be the base for the validation capabilities of Prospec 2.0.

![Prospec 2.0 Main Window](Image)

The \texttt{Property window} describes the basic property information such as the property name, the informal property description as provided by the client and any assumptions made about the property, as shown in Figure 2. Properties can be created by clicking on the \texttt{New} button, removed by clicking on the \texttt{Delete button} and stored into the viewing table by clicking on the \texttt{Save} button. Different Properties can be browsed and viewed by selecting them from the table of propositions. The \texttt{scope} section includes the \texttt{scope} type, assumptions made about the \texttt{scope}, and the left and right propositions. CP attributes include a \texttt{Type} to differentiate between events and conditions, a \texttt{CompositeProp} to identify the desired CP, and a \texttt{PropositionList} including the simple propositions to be used in the CPs. A proposition is described by a symbol and a description.

A new feature in Prospec 2.0 is a window that allows the user to view a summary of the property being
specified. The window shows the current state of the specification and the formal specification if defined. This window is embedded into the Property window, as shown in Figure 3.

Prospec 2.0 allows users to create, save, and print reports of the specifications. The reports are created either directly from the Prospec 2.0 application or from the XML files containing the metadata of the property specifications. The reports include the informal specification as provided in the description section of Prospec 2.0, the formal specification as generated by Prospec 2.0, information to construct the visual representation matching the generated formal specification, and the corresponding metadata such as logic used or version number.

4. Scenario

The following scenario illustrates the use of Prospec. Jill is a software engineer working on security issues in web services. She recognizes that the system must support the following requirement: “A message recipient shall reject messages containing invalid signatures, messages missing necessary claims, or messages whose claims have unacceptable values [20].” Since the project team will use a model checker to verify the algorithms, she needs an LTL specification.

Jill starts Prospec and creates the new property project SOAP Message Security and selects LTL as the logic to be used for the formal specification. She accesses the property browsing tree and double-clicks the property description attribute to open the property description screen. Using this screen, she names the property Message Recipient Protocol and provides the informal description.

Now Jill must identify the scope, the region of the program over which the property must hold. She accesses the property browsing tree and double-clicks the scope attribute to open the scope specification screen, which displays English descriptions of the five available scopes. After using the decision tree in the Guided Selection screen, Jill selects Global as the property scope as shown in Figure 3. Properties with Global scope must hold over all states of execution.

After identifying the scope, Jill selects a pattern. Selecting the pattern attribute of the property browsing tree opens the pattern specification screen. The five available patterns are described there, and Jill decides to use the Response \((T,P)\) pattern as shown in Figure 4. In this pattern, the conditions or events described by \(T\) are a response to the conditions or events described by \(P\). The letters \(T\) and \(P\) represent a proposition and a composite proposition (CPs), respectively. This choice is appropriate since Jill wants to ensure that there is a rejection whenever an unacceptable message is received. Prospec offers guided selection to assist a user in the selection of an appropriate pattern.

Once the pattern has been selected, Jill defines \(T\) and \(P\). To define \(P\), Jill creates the simple propositions invalid_sign (Message contains invalid signatures), miss_claims (Messages are missing necessary claims), and unacceptable_value (Messages have claims with unacceptable values). Because \(P\) is defined by three propositions, Prospec displays a message indicating that a CP must be defined. Once the CP screen is accessed as shown in Figure 5, Jill determines that the three simple propositions are conditions (propositions that hold in one or more consecutive states) and should be combined using \(\text{AtLeastOne}_C(G)\) to indicate that at least one of the propositions in the set \(G\) can trigger the condition. A decision tree in the Guided Selection screen for CPs can
be used to determine which CP to use. To define the \( T \) parameter, Jill creates a new proposition \( \text{reject} \) (\text{Message is rejected}), indicating that the message recipient rejects the incoming message. This completes the pattern definition.

**Figure 5. Prospec 2.0 Scope Window**

To create the LTL formula from the scope and pattern, Jill selects the view formula button on Prospec’s main screen. Prospec takes the scope, pattern, and CP and generates the LTL formula:

\[
\Box((\text{invalid\_sign} \lor \text{miss\_claim} \lor \text{unacceptable\_value}) \rightarrow \Diamond \text{reject})
\]

This formula specifies that during the execution of the program, whenever an unacceptable message is received by a message recipient, a rejection follows. In addition to the LTL formula, a set of sample execution traces is presented showing possible sequences of execution and the value of the formula for each sequence. Jill reviews these execution traces to ensure that the formula captures her intent.

An execution trace is a sequence of states read from left to right. Each space represents the propositions that are true in that state. If no proposition is true, a “-” is used. If more than one proposition is true, the propositions are enclosed in parentheses. Some of the execution traces for the SOAP Message Security property are shown in Table 1. For this example, symbol \( I \) stands for proposition \( \text{invalid\_sign} \), symbol \( M \) for proposition \( \text{miss\_claim} \), symbol \( U \) for proposition \( \text{unacceptable\_value} \), and symbol \( R \) for proposition \( \text{reject} \).

<table>
<thead>
<tr>
<th>Trace of Computation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(--I--R--)</td>
<td>Satisfied</td>
</tr>
<tr>
<td>(--RU-----)</td>
<td>Unsatisfied</td>
</tr>
<tr>
<td>(U--M-----)</td>
<td>Unsatisfied</td>
</tr>
<tr>
<td>((IM)------R)</td>
<td>Satisfied</td>
</tr>
</tbody>
</table>

**Table 1. SOAP Message Security Property Execution Traces Examples**

5. RELATED WORK

This section describes other tools used also for the elicitation and formal specification processes and how these efforts differ from Prospec 2.0.

5.1 Propel

The goal of Propel[12] is to help practitioners write and understand properties by providing templates that explicitly capture details as options for commonly-occurring property patterns based on SPS. The provided templates are represented using both disciplined natural language (DNL) and finite-state automata (FSA). The practitioner can view both representations simultaneously and select from which representation to elucidate the desired property.

The main difference between Prospec 2.0 and Propel is that Prospec 2.0 uses guided questions to distinguish the types of scope or relations among multiple conditions or events while Propel uses DNL. Also the pattern visual representations differ between the efforts, in Prospec 2.0 timelines are used while in Propel FSA are used.

5.2 Timeline Editor

Timeline Editor [13] allows the formalization of certain type of requirements. To formalize these requirements a series of events and required system responses are placed on a timeline. The tool converts the timeline specification automatically into a test automaton. The timeline specification can then be used directly by a logic model checker or a test-sequence generator.

As opposed to Prospec 2.0, Timeline Editor cannot capture group of events occurring in arbitrary order nor provide visual feedback for validation purposes. Prospec 2.0 allows practitioners to specify group of events occurring in arbitrary order by using CPs and SPS. Also, the traces of computation generated by Prospec 2.0 allow practitioners to validate that the specified properties match the practitioner’s intent.

5.3 SPIDER

SPIDER [21] generates specification properties using natural language representations. This process is based on a natural language grammar and specification pattern system to derive a natural language sentence. This sentence is then mapped to the temporal logic that can be analyzed formally by a tool such as SPIN. The structured language grammar supports translations of untimed and timed properties to multiple temporal logics.

There are three main differences between Prospec 2.0 and SPIDER. Prospec 2.0 offers support for composite
propositions, guided selection in the specification process, and property validation using traces of computations.

6. CONCLUSIONS

In this paper, we describe Prospec 2.0, an improvement to the property specification tool Prospec 1.0. We discuss the new features of Prospec 2.0 and describe how these changes enable practitioners to use Prospec 2.0 as both an automated formal property specification tool and as an automated formal property specification validation tool. The use of XML in Prospec 2.0 and its ability to be interoperable, it will be possible now to integrate the Prospec into the chain of tools that could provide the desired end-to-end automation for all aspects of software development.

7. ACKNOWLEDGMENTS

The authors would like to thank Milos Janek and Roberto Nevarez for their work towards this project. This work is partially supported through the CREST Cyber-ShARe Center funded by NSF grant number HRD-0734825.

8. REFERENCES