Proceedings of the 13th IFAC Symposium on
Information Control Problems in Manufacturing
Moscow, Russia, June 3-5, 2009

Pattern-based Analysis of Automated Production Systems

José Creissac Campos * **José Machado** **

* *Departamento de Informática/CCTC, Universidade do Minho, Braga, Portugal, (e-mail: jose.campos@di.uminho.pt).*
** *Departamento de Engenharia Mecânica, Universidade do Minho, Guimarães, Portugal, (e-mail: jmachado@dem.uminho.pt)*

Abstract: As formal verification tools gain popularity, the problem arises of making them more accessible to engineers. A correct understanding of the logics in which properties are expressed is needed in order to guarantee that properties correctly encode the intent of the verification process. Writing appropriate properties, in a logic suitable for verification, is a skilful process. Errors in this step of the process can create serious problems since a false sense of security if gained with the analysis. However, when compared to the effort put into developing and applying modelling languages, little attention has been devoted to the process of writing properties that accurately capture verification requirements. This paper illustrates how a collection of property patterns, and its tool support, can help in simplifying the process of generating logical formulae from informally expressed requirements.

Keywords: Formal verification; Safety analysis; Discrete systems; Modelling; Error criteria.

1. INTRODUCTION

Formal verification is the process of proving the correctness of a system (or a model thereof) with respect to some formally expressed property. Model-checking [Bérard et al., 1999] is becoming an established technique for the formal verification of Discrete Event Systems (DES) automation. A finite state system can be represented by a labelled state transition graph, where labels of a state are the values of atomic propositions in that state (for example the values of the latches). Properties about the system are expressed as formulae in temporal logic. Model-checking consists of traversing the graph of the transition system, verifying that it satisfies the formula representing the property, i.e., that the system is a "model" of the property.

Temporal logic formulae enable expressing properties of the behaviour of the system. For instance, some properties of the controller model, like safety and liveness properties, related to the internal states of the controller program [Bornot et al., 2000]. Writing these formulae is a two step process:

(1) the relevant properties of the system must be identified;
(2) for each property, the correct encoding in the logic of the verification tool must be found.

Step 1 is domain dependent, and largely relies on knowledge about the specific system being designed/verified and what its properties should be. Step 2 is a technical step. A correct understanding of the model, the requirement, and the logic in which properties are expressed is needed in order to guarantee that the property being tested correctly encodes the intent of the testing process. This is a non-trivial step. As illustrated by Dwyer et al. [1998] and Campos et al. [2008], instances can be found in the literature where the logical formulae used for verification do not correspond to what was intended. This is a serious problem since a false sense of security if gained with the analysis.

In order to ease the process, strategies can be applied such as breaking down a property in smaller parts, or using observer automata to express the behaviour one wants to verify. The former case begs the question of how to compose the results of the smaller verification steps. In the latter case, the complexity of the model is increased. This has the potential to make the verification of large systems less feasible.

By studying and identifying the properties used for the verification of DES automation, it becomes possible to systematize the writing of such properties in an automatic way. That is the goal of this work.

In [Campos et al., 2008] a study about the type of properties that are typically verified of industrial controllers using formal analysis techniques is presented. The results of that study were systematised in a collection of patterns to help analysis. A tool was also developed to support the approach.

The current paper develops the pattern collection further, and illustrates how it can be used in a concrete example. It also briefly discusses how the developed tool can assist in the process.

2. PROPERTY SPECIFICATION PATTERNS

The term "design pattern" was first introduced by Gamma et al. [1995] as a means of capturing and transmitting experts' knowledge in the field of object-oriented design. A

pattern is not simply a mechanism to classify some artefact (be it a object-oriented design, or a property specification) into a category. A pattern's goal is to capture proven solutions to known problems, and to demonstrate how they can be used in practice to solve the same or similar problems in new situations.

With the above in mind, Dwyer et al. [1998] propose a system of property specification patterns. They carried out an extensive review of published property specifications, and identified recurring patterns which they organised into an hierarchy.

For each pattern, a description that includes the pattern's intent, examples and known uses, relationships to other patterns, and mappings to different logics is provided. Additionally, the patterns can be tailored with scopes: they can be applied to the whole of the model's behaviour, or be restricted to work between specified conditions.

Patterns do not provide concrete solutions. Instead, they provide templates that must be tailored for specific purposes. Hence, when attempting to apply the patterns to a specific area two issues arise:

- The patterns should capture the relevant knowledge for the specific area being considered.
- The manner in which the properties are formulated should be adequate to the logics and modelling approaches used in the area of interest.

In order to answer these two questions a study of property specifications in the area of automation control was carried out.

The objective of the study was similar to that carried out by Dwyer et al. [1998]: to collect properties used in the literature, and look for possible patterns. The justification to perform a new instance of a study of this type was two fold: on the one hand, the original study already had a number of years, begging the question of whether new patterns had arisen; on the other hand, there was interest in a particular application of the verification technology, and in determining whether domain specific patterns were being used in that context.

A total of 6 main case studies was analysed, resulting in over 70 property specifications (see [Campos et al., 2008] for details). These properties were then aggregated into classes according to their syntactical structure, and the type of application. These classes then originated patterns for which LTL (Linear Temporal Logic) and CTL (Computational Tree Logic) formulae were provided.

Given the particular modelling approach used in the field, a considerable number of properties used special variables to restrict the analysis to stable states. This concept of "stability" is related with states of the plant model that may not correspond to true states of the plant behaviour. The concept of "stability" is well explained by Machado et al. [2006]. While this concept can at first seem a detail to be dealt with during pattern instantiation, it has in fact a great impact on the structure of the properties. The introduction of such variables requires considerable knowledge of the logics being used if it is to be done properly.

3. A REVISED PATTERN COLLECTION

Since the first version, introduced in Campos et al. [2008], the pattern collection has been subject to a number of updates. A new pattern has been added (*Liveness*), one pattern has been sub-divided (*Response*), and (most importantly) the notion of scope has been introduced.

This section describes the revised version of the pattern collection proposed in Campos et al. [2008]. Due to space contraints, the presentation of the patterns has been simplified for this paper: only the *after* and *after/until* scopes are presented (the *after* scope is used to express the pattern holds after some condition; the *after/until* scope is used to express the pattern holds between two conditions); scopes are presented for the base formulation only; CTL formulations of the patterns are used preferably (where such formulations are not available, LTL is used); alternative formulations to the use of the weak until operator are not included; only one example per pattern is presented.

Throughout the descriptions P and Q will be used to denote variables that need instantiation. St and Sp will be used for scoping the patterns. The *stable* variable defines the stable states.

3.1 Possibility Pattern

In many situations it is relevant to verify whether some event or system state is possible. This type of requirement is captured by the *Possibility Pattern*. This pattern was found to be one of the three most common patterns in the literature.

Property Pattern: *Possibility*
Intent: To express that some event or state (P) is always possible throughout the execution of the system. Note that it does not require that the state or event actually happens in a specific execution of the model, only that it is possible that it will.
Basic Formulation (CTL) ***Globally***: AG EF P ***After*** S_t: AG($S_t \rightarrow$ AG EF P) ***After*** S_t ***until*** S_p: AG($(S_t \wedge \neg S_p) \rightarrow$ A[E[$\neg S_p$ U $(P \wedge \neg S_p)$] W S_p])
Stable Formulation (CTL) ***Globally***: AG EF ($stable \wedge P$)
Examples: Rossi [2003] uses this pattern to express the absence of dead code. The author writes a family of properties $AG\ EF(etat = pre_i)$ where *etat* is a variable capturing the current state of the system and pre_i the possible execution steps. What each properties says is that a particular execution step is always possible.

3.2 Fairness Pattern

In some situations it is not enough to express that some event or state is possible, it must be possible consistently through out the behaviour of the system. This property is called *Fairness*. This pattern, despite not being one of the most used, was used to express relevant properties.

Property Pattern: *Fairness*
Intent: To express that some event or state (P) is repeatedly possible throughout the execution of the system. Unlike the possibility pattern, this pattern does require that the state or event actually happens in the execution of the model.
Basic Formulation (LTL) ***Globally***: G F P ***After*** S_t: G($S_t \rightarrow$ G F P) ***After*** S_t ***until*** S_p: G($(S_t \wedge \neg S_p) \rightarrow$ [F $(P \wedge \neg S_p)$ W S_p])
Stable Formulation (LTL) ***Globally***: G F ($stable \wedge P$)
Examples: Rossi [2003] uses this pattern to express dead lock freedom $G\ F\ fdc$ where *fdc* represents the end of the processing cycle. Note however, that a behaviour which satisfies the above property is that where the system does not leave the *fdc* condition.

3.3 Absence Pattern

In many cases it is relevant to verify that undesirable situations cannot occur. This can be captures by the *Absence Pattern*. This was one of the most common patterns.

Property Pattern: *Absence*
Intent: To express that some event or state P is not present throughout the execution of the system.
Basic Formulation (CTL) ***Globally***: AG ($\neg P$) ***After*** S_t: AG($S_t \rightarrow$ AG $\neg P$) ***After*** S_t ***until*** S_p: AG($(S_t \wedge \neg S_p) \rightarrow$ A[$\neg P$ W S_p])
Stable Formulation (CTL) ***Globally***: AG $\neg$($stable \wedge P$)
Examples: Yang et al. [2001b] use this pattern repeatedly to express both that a tank should not become empty, and that it should not overflow $AG\neg(Lev = 0) \wedge AG\neg(Lev = 6)$ where *Lev* represents the Level of the tank.

3.4 Universality Pattern

Guaranteeing that some condition is true in all states of the system is also a common requirement. This is captured by the *Universality Pattern*.

Property Pattern: *Universality*
Intent: To express that some event or state condition P occurs in every state of the execution of the system. This pattern is in effect the opposite of the absence pattern.
Basic Formulation (CTL) ***Globally***: AG P ***After*** S_t: AG($S_t \rightarrow$AG P) ***After St until Sp***: AG($(S_t \wedge \neg S_p) \rightarrow$A[$P$ W S_p])
Stable Formulation (CTL) ***Globally***: AG ($stable \rightarrow P$)
Examples: Yang et al. [2001a] use this pattern to express that the temperature of a reactor always stays inside a desirable range $AG(reactor.TREA > 0 \wedge reactor.TREA < 6)$ where $reactor.TREA$ is the reactor's temperature.

3.5 Eventual/Immediate Response Patterns

In some situations there might be the need to verify causal relations between two states or events. One possibility is one state/event leading to another. This is captured by the *Response Patterns*. Patterns are provided for both the case when the response does not need to be immediate, and for the immediate case.

Property Pattern: *Eventual Response*
Intent: To express that some event or state P will always lead, at some point in the future, to another event or state Q.
Basic Formulation (CTL) ***Globally***: AG ($P \rightarrow$ AF Q) ***After*** S_t: A[$\neg S_t$ W ($S_t \wedge$ AG($P \rightarrow$AF Q))] ***After*** S_t ***until*** S_p: AG($(S_t \wedge \neg S_p) \rightarrow$ A[($P \rightarrow$ A[$\neg S_p$ U ($Q \wedge \neg S_p$)]) W S_p])
Stable Formulation (CTL) ***Globally***: AG (($P \wedge stable$) $\rightarrow$ AF ($stable \wedge Q$))
Examples: Yang et al. [2001b] use this pattern to express that a pump should not carry on working when the level of a tank is running low $AG(Lev < 2 \rightarrow AF(\neg m2 \wedge \neg vB \wedge \neg v4))$ where *Lev* represents the level of the tank, *m2* is the state of the pump, and *vB* and *v4* are valves' states.

Property Pattern: *Immediate Response*
Intent: To express that some event or state P will always immediately lead to another event or state Q.
Basic Formulation (CTL) ***Globally***: AG ($P \rightarrow$ AX Q) ***After*** S_t: AG($S_t \rightarrow$ AG ($P \rightarrow$ AX Q)) ***After*** S_t ***until*** S_p: AG($(S_t \wedge \neg S_p) \rightarrow$ A[($P \rightarrow$ AX ($Q \wedge \neg S_p$)) W S_p])
Stable Formulation (CTL) ***Globally***: AG (($P \wedge stable$) $\rightarrow$ A[$\neg stable$ U ($stable \wedge Q$)])
Examples: Bornot et al. [2000] use this pattern in the formula $AG((active \wedge s6) \rightarrow AX(x \wedge y \rightarrow s7 \wedge \neg s8))$ to express that a specific condition on the state of the system ($active \wedge s6$) immediately leads to a transition, and not to another ($x \wedge y \rightarrow s7 \wedge \neg s8$).

3.6 Precedence Pattern

Another possible causal relation is that some state/event must always precede some other state/event. This is captured by the *Precedence Pattern*.

Property Pattern: *Precedence*
Intent: To express that some event or state Q must occur before some other event or state P. Conceptually this pattern is the opposite of the response pattern.
Basic Formulation (CTL) ***Globally***: A[$\neg Q$ W P] ***After*** S_t: A[$\neg S_t$ W ($S_t \wedge$ A[$\neg Q$ W P])] ***After*** S_t ***until*** S_p: AG($(S_t \wedge \neg S_p) \rightarrow$A[$\neg Q$ W ($P \vee S_p$)])
Stable Formulation (CTL) ***Globally***: A[$\neg$ ($stable \wedge Q$) W ($stable \wedge P$)]
continues...

...continued
Examples: Rossi [2003] uses the LTL encoding of this pattern in the property $$G(\neg dp\_conveyor\_motor \ W \ (\neg dp\_drill\_motor))$$ to express that a drill should always be stopped ($\neg dp\_drill\_motor$) before a conveyor belt is started ($dp\_conveyor\_motor$).

3.7 Liveness Pattern

This pattern was not present in the original proposal in Campos et al. [2008]. Further analysis, however, revealed the relevance of considering situations where some state or event must be possible after another state or event (as opposed to the response patterns which make it mandatory). This type of property is captured by the *Liveness Pattern*.

Property Pattern: ***Liveness***
Intent: To express that some event or state Q can occur after some other event or state P.
Basic Formulation (CTL) ***Globally***: AG($P \rightarrow$ EF Q) ***After*** S_t: A[$\neg S_t$ W ($S_t \wedge$ AG($P \rightarrow$EF Q))] ***After*** S_t ***until*** S_p: AG(($S_t \wedge \neg S_p$) $\rightarrow$ A[($P \rightarrow$ E[$\neg S_p$ U ($Q \wedge \neg S_p$)]) W S_p])
Stable Formulation (CTL) ***Globally***: AG(($P \wedge stable$) $\rightarrow$ EF ($Q \wedge stable$))
Examples: Machado et al. [2006] use this pattern in the formula $$AG(X1 \rightarrow EF \neg X1)$$ to express deadlock freedom (by saying that the sate of $X1$ can always change).

4. TOOL SUPPORT

As already discussed, expressing properties in a formal logic can be a complex task. While the patterns above can be a useful tool in dealing with this complexity, the manual process of selecting and instantiating a pattern is error prone, and such errors can be hard to detect. This is particularly the case when complex formula are at stake.

In order to address this, a tool has been developed to help pattern instantiation. The tool (Properties Editor – see figure 1) is based on the notion of property patterns described above. A list of property patterns and help for instantiating those patterns is provided.

The patterns include all the information in the original patterns, such as intention and known uses. Additionally the tools allows for the definition of the scope for the property. This allows the user to browse the patterns in order to select the one most adequate to the type of property of interest.

As illustrated in figure 1 the tool supports different collections of patterns. In the current case, both the patterns in [Dwyer et al., 1998] (Dwyer), and the patterns introduced above (SCAPS) are being made available. To this end, a DTD (Document Type Definition) for the description of patterns has been developed, and support for reading pattern collections expressed in XML (eXtensible Markup Language), in accordance to that DTD, integrated into the tool.

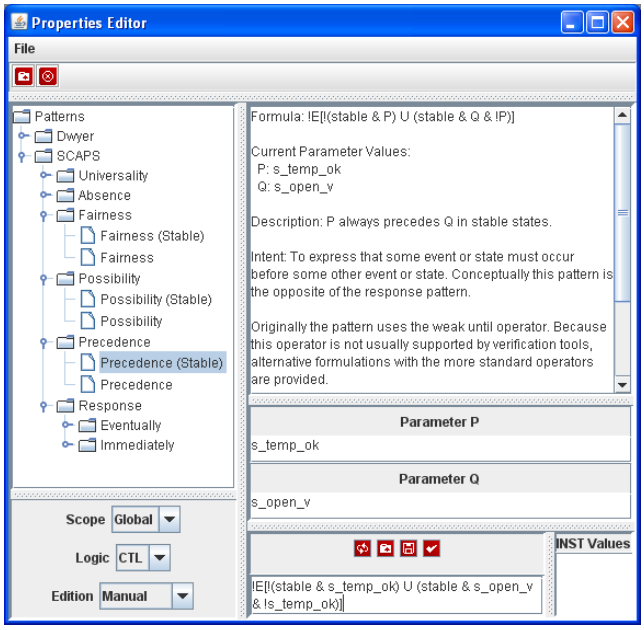

Fig. 1. The patterns tool

5. AN EXAMPLE

This section uses an example, taken from [Machado, 2006] to illustrate how the pattern collection can be useful in the analysis of Discrete Event Systems.

5.1 The example system

The system chosen for this case study lies in the well-known category of "pick-and-place" systems (see Figure 2). Its function is to take parts, fed by gravity into three feed chutes, for placement in a single unloading chute. Sensors $pp1$, $pp2$ and $pp3$ indicate the presence of a part in one of the feed chutes, while sensor $pp0$ signals the presence of a part in the unloading chute.

The device that enables picking and placing a part is composed of a group of three pneumatic cylinders plus a vacuum suction cup system. The vertical cylinder (VC) places the suction cup in contact with a part. Longitudinal cylinders $L1C$ and $L2C$ are arranged in series to allow positioning the vertical cylinder VC in front of the four chutes ($L2C$ stroke is twice as long as $L1C$ stroke). The four positions reached are thereby detected by position sensors $s0$, $s1$, $s2$ and $s3$. The depression in the suction cup is obtained by virtue of a venturi device and detected by a *vacuum* sensor.

The vertical cylinder is controlled by a monostable electro-valve (order $VCGD$ – Vertical Cylinder Go Down), and its positions of end of stroke are detected by sensors *vcu* (vertical cylinder up) and *vcd* (vertical cylinder down). The horizontal cylinders $L1C$ and $L2C$ are controlled by bistable electro-valves and the control orders of the corresponding electro-valve are $L1CGO$ ($L1C$ Go Out), and $L1CGI$ ($L1C$ Go In). By analogy, the orders $L2CGO$ and $L2CGI$ are the orders sent from the controller to the electro-valve of the cylinder $L2C$ for, respectively, the moving forward and moving back of the cylinder $L2C$ piston rod. For the picking-up of the parts, the order $VENTURI$ is sent from the controller to the associated

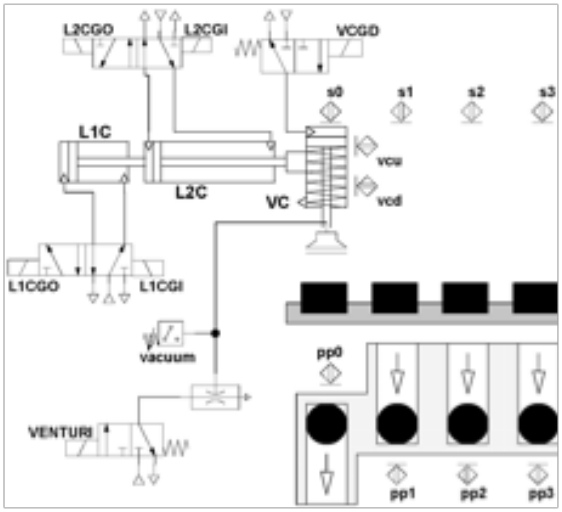

Fig. 2. Plant of the case study pick-and-place system.

electro-valve, and the aspiration is detected by the sensor *vacuum*.

The system was formally modelled to allow for formal verification. Describing the model is out of the scope of this paper (see [Machado, 2006] for a description). Here the interest is in expressing the properties. For the discussion that follows it is enough to know the meaning of the variables described next.

Concerning variables that represent plant model states:

- V_P2: VC is in the deployment movement;
- V_P5: $L1C$ is in the retracted position;
- V_P6: $L1C$ is in the deployment movement;
- V_P7: $L1C$ is in the deployed position;
- V_P8: $L1C$ is in the retraction movement;
- V_P9: $L2C$ is in the retracted position;
- V_P10: $L2C$ is in the deployment movement;
- V_P11: $L2C$ is in the deployed position;
- V_P12: $L2C$ is in the retraction movement.

Concerning variables that represent controller model states, a family of variables Xi represents the internal state of the controller during the evolution of the system.

5.2 Desired System Behaviour

Informally the desired system behavior can be described by the following properties:

- PR_1.i: The controller never commands a horizontal cylinder i in two directions at the same time;
- PR_2: If the controller commands the vertical cylinder to go down, then it must not command any movement to the horizontal cylinders;
- PR_3: The controller commands horizontal cylinders only while sensor *vcu* is on;
- PR_4: After the part is picked up, in the "pick-up position", it must not be dropped down until the suction cup reaches the "place position".
- PR_5: The horizontal cylinders move only while the sensor *vcu* is on;
- PR_6.i: The controller model must not have deadlock;
- PR_7.i: When a part is detected by sensor *ppi*, then in the future, the corresponding horizontal cylinder(s) will be deploying;
- PR_8.i: When a part is detected by sensor *ppi*, then in the future, it will be picked;
- PR_9: While the vertical cylinder is moving down, all the other cylinders stay in deployed or retracted position.

5.3 Property Formalization

Machado [2006] used Computation Tree Logic (CTL) [Bérard et al., 1999] for expressing most of the properties. In the case of property PR_4 an observer automata was used, due to the complexity of the behaviour that was being expressed. In what follows it will be shown how the patterns collection can ease the process of property formalization. Appropriate patterns will be selected for each property, and its parameters instantiated with appropriate expressions. The generated formulae are presented in Table 1 below.

Property PR_1.i can be seen as wanting to guarantee that the system will never reach specific undesirable states. Looking at the pattern collection, this can be expressed using the *Absence pattern* (section 3.3) using the *Globally* scope. It is now enough to define what the undesirable states for each cylinder are, and instantiate P in the pattern with them. For $L1C$, P becomes $L1CGO \wedge L1CGI$. For $L2C$, p becomes $L2CGO \wedge L2CGI$. Instantiating the pattern originates the first two formulae in Table 1.

Four of the eight remaining properties correspond to conditions that should always hold. This is true of properties PR_2, PR_3, PR_5, and PR_9. In this case, once each condition is defined, the *Universality pattern* (section 3.4) with the *Globally* scope can be used. For PR_2, P becomes $VCGD \rightarrow \neg(L1CGI \vee L1CGO \vee L2CGI \vee L2CGO)$ (i.e. movement in the vertical cylinder means no movement in the horizontal ones). Applying the pattern originates the third formula in the table. The same is done for the other properties.

Property PR_4 refers to a property that must always be true between two specific instants. The goal is to guarantee that the piece never drops down between the pick-up and place positions. Again, this can be expressed using the *Universality pattern*, but now with the "*After...Until...*" scope. P becomes *vacuum* (the piece never drops down), S_t (the pick-up positions) is replaced by (s1∨s2∨s3)∧vcd, and S_p (the place position) by s0 ∧vcd. Instantiating the pattern originates the fifth formula in the table.

Notice that the stable version of the "*After...Until...*" scope was used. Notice also that, for simplicity, in the table the weak until operator is used. The tool does not use weak operators since these are typically not supported by model checkers. Hence, the generated formula becomes:

$$\begin{aligned}&\mathrm{AG}((\mathrm{stable} \wedge (\mathrm{s1} \vee \mathrm{s2} \vee \mathrm{s3}) \wedge \mathrm{vcd} \wedge \neg(\mathrm{s0} \wedge \mathrm{vcd})) \rightarrow \\ &\neg\mathrm{E}[\neg(\mathrm{stable} \wedge \mathrm{s0} \wedge \mathrm{vcd})\mathrm{U}(\neg(\mathrm{stable} \rightarrow \mathrm{vacuum}) \\ &\wedge\neg(\mathrm{stable} \wedge \mathrm{s0} \wedge \mathrm{vcd}))])\end{aligned}$$

As noted above, in [Machado, 2006] the option was made not to try writing the formula, and an Observer Automata

was used instead. Using patterns, generating this complex property was no more difficult than generating the property for simpler cases.

In property PR_6 the goal is to ensure that the system state can always evolve. This can be defined using the *Liveness pattern* (section 3.7) for each of the Xi internal state variables. Defining P as $X1$ and Q as $\neg X1$, originates the seventh formula in the table. The same process is applied to each variable ($X1$ to $X38$). In fact, the tool supports the simultaneous generation of the 38 needed formulae in one step (for simplicity only the first one is presented). Note that, following [Machado, 2006], the choice was made not to consider stable states only.

Properties PR_7.i and P_8.i are similar. Both are instances of the *Eventual Response pattern* (section 3.5). For sensor $pp1$, P is replaced with $pp1$ and Q with V_P6 (the corresponding cylinder) to get the ninth property in the table, which corresponds to PR_7.1. For PR_8.1, P is replaced with $pp1$ and Q with $s1 \wedge vcd \wedge vacuum$ (i.e. vertical cylinder is down at the right position, and there is vacuum in the suction cup). For the remaining cases the process is the same.

Table 1. Results from applying the patterns

Property	CTL formalization
PR_1.1	AG ¬(stable ∧ L1CGO ∧ L1CGI)
PR_1.2	AG ¬(stable ∧ L2CGO ∧ L2CGI)
PR_2	AG (stable →(VCGD → ¬(L1CGI ∨ L1CGO ∨ L2CGI ∨ L2CGO)))
PR_3	AG (stable → ((L1CGI ∨ L1CGO ∨ L2CGI ∨ L2CGO) → vcu))
PR_4	AG((stable∧(s1∨s2∨s3)∧vcd∧¬ (s0 ∧vcd))→ A[(stable→vacuum) W (stable∧ s0 ∧vcd)])
PR_5	AG (stable → ((V_P6 ∨ V_P8 ∨ V_P10 ∨ V_P12) → vcu))
PR_6.1	AG (X1 → EF ¬X1)
PR_7.1	AG ((stable ∧ pp1) → EF (stable ∧ V_P6))
PR_7.2	AG ((stable ∧ pp2) → EF (stable ∧ V_P10))
PR_7.3	AG ((stable ∧ pp3) → EF (stable ∧ V_P6 ∧ V_P10))
PR_8.1	AG ((stable ∧ pp1) → EF (stable ∧ s1 ∧ vcd ∧ vacuum))
PR_8.2	AG ((stable ∧ pp2)→ EF (stable ∧ s2 ∧ vcd ∧ vacuum))
PR_8.3	AG ((stable ∧ pp3) → EF (stable ∧ s3 ∧ vcd ∧ vacuum))
PR_9	AG (stable → (V_P2 → ((V_P5 ∧ V_P9) ∨ (V_P5 ∧ V_P11) ∨ (V_P7 ∧ V_P9) ∨ (V_P7 ∧ V_P11))))

6. CONCLUSIONS AND FUTURE WORK

In recent years, several approaches to applying formal verification techniques on automation systems dependability have been proposed. As verification tools gain popularity, the problem arises of making them more accessible to engineers.

This paper has looked at the issue of supporting the expression of property specifications. A collection of patterns has been put forward as an aid to expressing relevant properties of a system's behaviour, together with a tool to help in using the pattern collection.

The applicability of the pattern collection was demonstrated with an example. Comparing our results with those of Machado [2006], it can be seen that equivalent CTL formulae were obtained for all properties (except PR_4). However, this was achieved without the need to resort to temporal logic expertise. The only requirements were an understanding of the problem domain and basic propositional logic knowledge. The temporal logic aspects were captured by the patterns. In the special case of PR_4, the approach was able to generate a temporal formula (again, using propositional logic to instantiate the pattern) while originally there was the need to resort to an observer automaton.

Future work is two fold. One one hand, continuing to collect data from the verification literature, in order to further validate and perfect the current set of patterns. On the other hand, there is the need continue applying the tool to a number of case studies in order to better assess its value in the verification process.

REFERENCES

B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification: Model-Checking techniques and tools*. Springer, 1999.

S. Bornot, R. Huuck, B. Lukoschus, and Y. Lakhnech. Verification of sequential function charts using SMV. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, volume V, pages 2987–2993. CSREA Press, June 2000.

J. C. Campos, J. Machado, and E. Seabra. Property patterns for the formal verification of automated production systems. In *Proceedings of the 17th IFAC World Congress*, pages 5107–5112. IFAC, 2008.

M.B. Dwyer, G.S. Avrunin, and J.C. Corbett. Patterns in property specification for finite-state verification. In B. Boehm, D. Garlan, and J. Kramer, editors, *21st Intern. Conf. on Software Engineering (ICSE'98)*, pages 411–420. IEEE Computer Society Press, 1998.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1995.

J. Machado, B. Denis, and J.-J. Lesage. A generic approach to build plant models for DES verification purposes. In *8th International Workshop On Discrete Event Systems (WODES'06)*, pages 407–412, July 2006.

J.M. Machado. *Influence de la prise en compte d'un modèle de processus en vérification formelle des Systèmes à Evénements Discrets*. PhD thesis, Escola de Engenharia, Universidade do Minho, 2006.

O. Rossi. *Validation formelle de programmes ladder pour automates programmables industriels*. PhD thesis, École Normale Supérieure de Cachan, France, June 2003.

S.H. Yang, O. Stursberg, P.W.H. Chung, and S. Kowalewski. Automatic safety analysis of computer-controlled plants. *Computers and Chemical Engineering*, 25:913–922, 2001a.

S.H. Yang, L.S. Tan, and C.H. He. Automatic verification of safety interlock systems for industrial processes. *Journal of Loss Prevention in the Process Industries*, 14: 379–386, 2001b.