An Efficient and Flexible Synthesis of Behavioral Patterns Specifications employing the GR(1) Style

Fernando Asteasuain[1]

Universidad Nacional de Avellaneda, Argentina
fasteasuain@undav.edu.ar

Abstract. The usage of the GR(1) subset of Linear Temporal Logic has been proposed as an efficient way of appying formal verification tools and techniques. Another important milestone for formal verification are behavioral specification patterns which ease the task of eliciting the expected behavior of software systems. Recently, attractive behavioral patterns specifications has been proposed in cutting edge domains such as robotic missions, but its codification in the GR(1) format has been pinpointed as a challenging process. In this work we propose the use of a flexible and powerful behavioral framework called FVS to specify and synthesize modern behavioral patterns in the GR(1) style to optimize the formal verification process. Formal and empirical validation of our initial results are also presented.

Keywords: Formal verification, GR(1) Synthesis, Behavioral Patterns

1 Introduction

Behavioral specifications patterns have played a significant role in the usage of formal validations techniques in software systems development [7, 15]. They allow complex behavior to be denoted employing simple and effective templates that are properly instantiated in a specific domain. Since their introduction [7] some extensions of the patterns has been proposed. For example, [9] introduces patterns for real-time software system. More recently, [15] presents a vast set of behavioral patterns for robotic missions behavioral specification.

One of the most widely known Achilles' heel of behavioral patterns application in the formal verification process has been the performance and computational complexity of the algorithms involved [6, 11]. To overcome this key issue work in [6] introduces the GR(1) fragment of Linear Temporal Logic (LTL). This particular strict subset of LTL exhibits polynomial execution times in the algorithms involved. GR(1) style mimics the behavior of an automaton defining a set of initial conditions for the initial state, a set of safety propositions over the current and next state and a set of justice constrains. Intuitively, GR(1) formula dictate that if the assumptions are satisfied by the environment the system has to satisfy the guarantees. The GR(1) fragment allows not only behavior specification but also behavioral synthesis [6]. Behavioral synthesis is an attractive

technique which assures consistency and compliance of systems implementation with the specification by construction. Once the behavior is specified, a controller satisfying all the conditions is automatically obtained employing advanced game theory algorithms [6]. A controller can be seen as an automaton that receives values from the environment and produces instructions to the system to achieve its goals. A controller is found if a winning strategy is obtained stating that no matter what the environment moves, the system will always fulfill its expected goals. GR(1) synthesis approach has been successfully applied in a plethora of domains such as reactive systems and robotics [8, 14].

In this work we propose to use the behavioral specification framework FVS (Feather Weight Visual Scenarios) [2, 3, 5] **to codify and synthesize behavioral specification patterns in the GR(1) format.** The original specification patterns [7] were codified under the GR(1) shape in [12]. However, a more declarative way of expressing behavior might ease the specification process [4, 10]. When introducing the robotic specification patterns the authors in [15] explicitly states that employing GR(1) would be extremely helpful but the error-prone process of specifying them in the GR(1) subset was a challenging task to overcome. We tackled these problems taking advantage of FVS powerful, flexible and expressive notation. All the robotic specification patterns proposed in [15] were specified and synthesized using FVS as the main formal mechanism underneath the verification process. In addition, we formally demonstrate that our GR(1) implementation is sound and complete with a efficient and polynomial execution time and we also conducted a performance evaluation in a interesting case of study.

The rest of this paper is structured as follows. Section 2 introduces preliminaries concepts including FVS, GR(1) and behavioral synthesis. Section 3 addresses how behavioral patterns are codified and synthesized with FVS whereas Section 4 states some general remarks about the GR(1) implementation in FVS. Finally, Sections 5 and 6 present related and future work and the final conclusions of this work.

2 Background

2.1 Feather Weight Visual Scenarios

In this subsection we will informally describe the standing features of FVS. The reader is referred to [2] for a formal characterization of the language. FVS is a graphical language based on scenarios. Scenarios are partial order of events, consisting of points, which are labeled with a logic formula expressing the possible events occurring at that point, and arrows connecting them. An arrow between two points of interest represent a total function between points establishing a **precedence relationship** between them. An arrow connecting a point A to a point B indicates that the A event occurs before the occurrence of point B. FVS also features a function to restrict behavior between points, which is called the **forbidden function**. This function allows to express that certain behavior must

not occur between points, which is a classical requirement for behavioral specification languages. The forbidden relationship is graphically shown as a label in the arrows connecting points. A simple FVS scenario is shown in Figure 1. The scenario exemplifies a behavior taken from the *ForkLift System* introduced in [13]. The system controls a forklift artefact which moves around stations lifting and dropping cargo. The scenario in Figure 1 shows the forklift visiting three locations namely l_1, l_2 and l_3 without stopping. Locations are visited in order. That is, first l_1 is visited, then l_2 and finally l_3. The bigger dot at the left wing in Figure 1 is a distinctive point in FVS to denote the beginning of the execution.

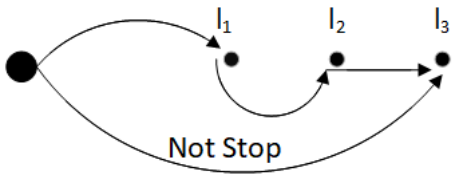

Fig. 1. A simple FVS rule for a robot visiting locations without stopping

We now introduce the concept of FVS rules, a core concept in the language. The intuition is that whenever a trace "matches" a given antecedent scenario, then it must also match at least one of the consequents. In other words, rules take the form of an implication: an antecedent scenario and one or more consequent scenarios. Graphically, the antecedent is shown in black, and consequents in grey. Since a rule can feature more than one consequent, elements which do not belong to the antecedent scenario are numbered to identify the consequent they belong to. An example is shown in Figure 2. This rule captures the following requirement for the forklift system: if location l_2 is visited after location l_1 then the system must either visit location l_3 (Consequent 1 in Figure 2) or it stops (Consequent 2 in Figure 2).

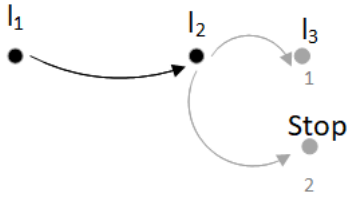

Fig. 2. An FVS rule example for the Forklift System

FVS can also handle the notion of what we denominate as "ghosts" events. Ghosts events introduce a new layer of abstraction given the possibility to reason about behavior that is not explicitly present in the system. For example, when locations l_1, l_2 and l_3 are visited the forklift may fulfill an objective thus entering an "accepting" state. This can be simply denoted with a $AcceptingState_{entered}$ ghost event. This is exemplified in Figure 3. Therefore, one way of employing ghosts events is to describe automata behavior as FVS rules, which is a particularly useful approach to denote behavior in GR(1) flavour. Semantics and further motivation of these type of events is found at [2, 5].

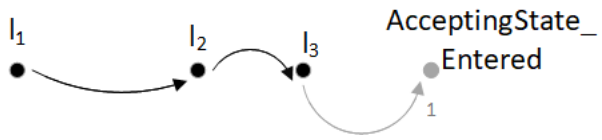

Fig. 3. An FVS rule using ghosts event to mimic automata behavior

2.2 GR(1) and Specification Patterns for Robotic Missions

One of the most successful and employed mechanisms to improve performance in the formal verification domain has been the General Reactivity of Rank 1, popularly known as GR(1), introduced in [6]. This strict subset of LTL has en efficient polynomial time symbolic synthesis algorithm [6]. Roughly speaking, this subset of LTL mimics the behavior of an automaton defining the following: a set of initial conditions for the initial state χ, a set of safety propositions over the current and next state ψ, and a set of justice constrains τ. Intuitively, if the assumptions are satisfied by the environment the system has to satisfy the guarantees. More formally, *Environment* $(\chi \wedge \psi \wedge \mathrm{G\ F}\ \tau) \Rightarrow$ *System*$(\chi \wedge \psi \wedge \mathrm{G\ F}\ \tau)$.

3 Synthesis and Specification of Behavioral Pattern in a GR(1) style with FVS

In this section we depict how specification patterns can be specified and synthesized under the GR(1) perspective using FVS as the underlying formal mechanism. A complete example including one of the original specification patterns [7] is shown in Section 3.1 whereas Section 3.2 illustrates the codification and synthesis of one of the Robotic specification patterns introduced in [15]. As it is full detailed in [5] these FVS specifications can be synthesized to automatically obtain a controller using tools such as GOAL [16].

3.1 Specifying Regular Patterns in a GR(1) style with FVS

As an example of how the original specification patterns [7] are GR(1) synthesized in FVS we showed in this section how the so called "Existence pattern with Between scope" is applied in the context of a the Forklift System case of study detailed in [12]. In Figure 4 this pattern is shown addressing the following requirement: the forklift has to to leave its pick-up station between lifting and dropping cargo. The instantiation of the pattern is as follows: the *(Not AtStation)* event must occur between the *Lift* event and the *Drop* event.

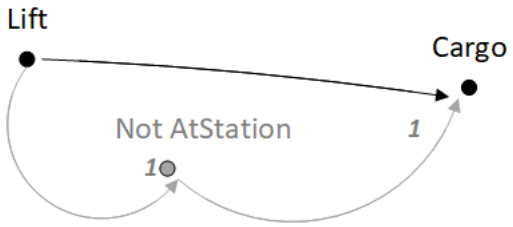

Fig. 4. Existence Pattern and Between Scope - Forklift System

Employing the tableau algorithm described in [2, 5] a Büchi automata can be automatically obtained from any FVS rule. By taking advantage of this algorithm a Büchi automata obtained from the Existence pattern in Figure 4 can be obtained. An sketch of this automata is shown in Figure 5. S_0 is the initial state. When the cargo is lifted, the automaton moves into state S_1. From this state, if the cargo is dropped without leaving the station, then the trap state is entered (S_2), which behaves as an error state since once entered the automata remains in this state for ever. On the other hand, if the forklift leaves the station (the Not AtStation event occurs), then the automata moves into S_0 again, since the rule is satisfied. Note that in other occurrences of events besides the mentioned ones in Figure 5, the automata stays in the same state (self-loops transitions).

The obtained automata in Figure 5 can be easily depicted in FVS rules following the GR(1) style employing *ghosts* events for each state (S_i), for entering or exiting each state ($S_i entered$ and $S_i exited$) and introducing rules for every transition. This is shown in Figure 6. The first rule simply states proper conditions for the initial state. The following three rules targets the required safety prepositions for the current and next state (the self-loops transitions are omitted for simplicity reasons). Finally, the rule at the bottom of Figure 6 pinpoints the justice constrains: the requirement being captured is hold infinitely often since either of both accepting states can always occur in the future taken *Any* event as reference (we use the *Any* event as a simplification representing the occurrence of any of the possible available events in the Forklift System).

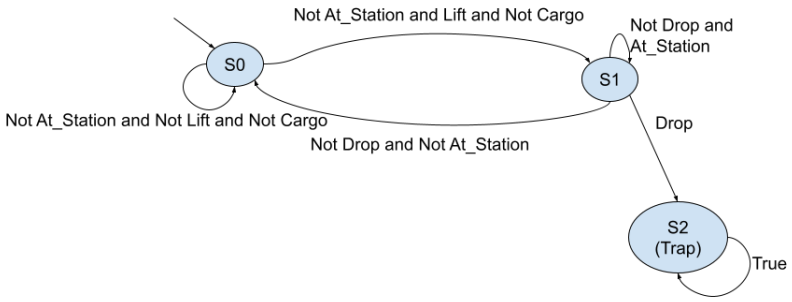

Fig. 5. A Büchi automaton for the Existence Pattern

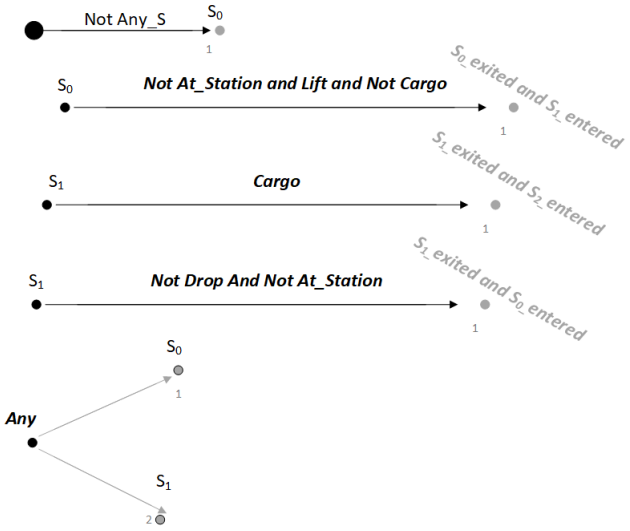

Fig. 6. FVS rules addressing the GR(1) style for the Existence Pattern

3.2 Specification Patterns for Robotic Missions in GR(1) flavour

We know illustrate how the specification patterns for robotic missions introduced in [15] can be denoted in a GR(1) style employing FVS as the behavioral language. We use as example one of the most representative patterns in the *Core Patterns* [12] category: the *Visit pattern*. This pattern capture the following requirement: the robot (or the system controlling any artefact being developed) must visit a set of locations in a unspecified order. Due to space considerations we showed here only this pattern, but the main concepts of the described technique can be straightforwardly extrapolated to all the others patterns. In our instantiation of the pattern we considered three locations: l_1,l_2, and l_3. The rule in Figure 7 shows an FVS rule for the mentioned pattern: the rule is satisfied if

and only if the three locations are visited, without imposing any restrictions in the order of the visits occurrence.

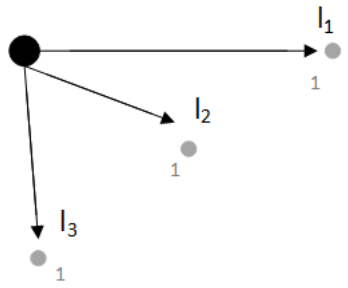

Fig. 7. An FVS rule modeling the Visit Specification pattern

The automaton that is obtained upon the FVS rule in figure 7 is shown in Figure 8. S_0 is the initial state, S_7 is the final and accepting state representing the system's state where all the locations were visited; either from state S_4 with the sequence: l_1,l_2 and l_3 or l_2,l_1 and l_3, or from state S_5 with the sequence: l_2,l_3 and l_1 or l_3,l_2 and l_1, or from state S_6 with the sequence: l_3,l_1 and l_2 or l_1,l_3 and l_1. From the accepting S_7 the automaton returns to the initial state due to a internal lambda transition.

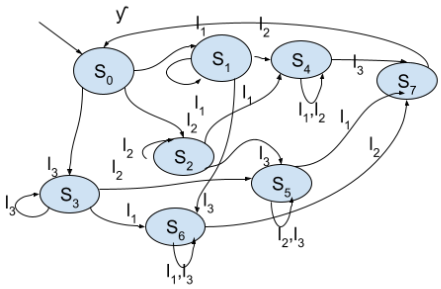

Fig. 8. A Büchi automaton for the Visit Pattern

Employing FVS mechanisms such as ghosts events this pattern can be straightforwardly encoded in the GR(1) flavour as is it seen in Figure 9. The first rule in Figure 9 targets conditions required in the initial state. Then, we modeled the required safety prepositions for the current and next state. Since they are very

similar, only rules for state S_0 are shown. These rules captures the initial behavior where the first location, either l_1 or l_2 or l_3 is visited and the automaton enters the S_1, S_2 or S_3 state respectively. Finally, the last rule at the bottom of Figure 6 addresses the justice constrains: the accepting state S_7 is visited infinitely often.

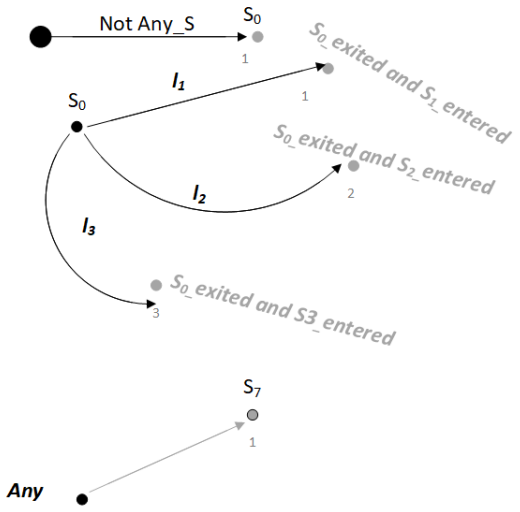

Fig. 9. FVS rules encoding in GR(1) the Visit Pattern

4 Evaluation of the GR(1) pattern synthesis with FVS

We evaluated the contributions pursued in this work under three important perspectives. First of all, we showed that our GR(1) translation is sound and complete, which constitutes a crucial milestone for any tool to be employed in formal verification. Secondly, we analyzed the performance and computational complexity costs of the FVS synthesis of the GR(1) version of the behavioral specification patterns. Finally, we contrasted final execution times against other prior version of the Forklift system implementation without GR(1) to measure the impact of the current FVS GR(1) implementation.

We can prove that the GR(1) behavioral specification pattern with FVS is sound and complete based on previous work. FVS tableau algorithm has been demonstrated to be sound and complete in [2]. FVS complete synthesis approach combining the GOAL tool [16] and the technique introduced in [12] was demonstrated sound and complete in [5]. Since in this work we rely on these mechanisms we can establish that the FVS implementation under the GR(1) format of the behavioral pattern is sound and complete.

Another important topic is to analyze the impact of the GR(1) FVS implementation considering computational complexity costs. In [12] this is measured analyzing the quantity of extra variables added needed to translate behavioral patterns into the GR(1) format. This quantity is particularly relevant since the computational complexity of synthesis algorithm rely on the size of the formula or automata to be analyzed. The results provided in [12] establishes that three extra variables for each pattern were added in the average case. In our case, as it is was shown in Section 3 we introduce extra variables as ghosts events to denote entering or exiting a given state and another one for each state. Therefore, the amount of extra cargo needed to model into the GR(1) style is exactly the same.

Finally, we aimed to evaluate the gain in performance with GR(1) with FVS. We consider several case of studies, but no execution times were reported in [12] or [15], the most related approaches with the contributions presented in this work. Nonetheless, we were able to replicate our experimentation with the Lego Forklift system used as benchmark in [12, 13]. We previously synthesized this case of study in [4]. We now revisited this system but employing the GR(1) version of the behavior. Comparing executions times controllers for the system were obtained in approximately twenty percent less of the overall execution time, which constitutes a significant impact (330 seconds without GR(1) against 264 seconds with the GR(1) flavour). We ran our experiments in a Bangho Inspiron5458, with a Dual Core i5-5200U and 8GB RAM memory.

Summarizing, FVS is able to denote and synthesize in the GR(1) format meaningful behavioral specification patterns, including all the specification patterns specially designed to address behavior in robotic missions [15]. Besides that, our approach is sound and complete and also its computational complexity is comparable to other known approaches. Our initial results of the performance impact are encouraging.

5 Related and Future Work

Work in [12] present a transcendental advance for behavioral synthesis and specification patterns. They efficiently codified all the regular or traditional specification patterns [7] and prove their results analyzing the Lego Forklift system [12, 13]. In [4] is was exemplified how FVS can be combined with this tool to provide a flexible and enriched behavioral notation for the synthesis of specification patterns. In this work we took a more efficient approach: we directly translated FVS automata into the GR(1) format taking advantage of FVS ghosts events and flexibility. In the same context, work in [15] introduces behavioral patterns for robotic missions. In this work we also provide an efficient and polynomial synthesis of these patterns employing GR(1). Future lines of work includes a more deep analysis of our GR(1) implementation, regarding computational complexity and executions times. Robotic specification patterns were explored with FVS in the satellite mission behavioral domain [1]. We are interested in continuing in this direction now including the GR(1) codification and synthesis of the mentioned patterns.

6 Conclusions

In this work we successfully codified in the GR(1) format meaningful behavioral specification patterns. This codification enables the possibility of obtaining synthesis controllers for medium and large software systems in polynomial time. We validated our results considering sound and completeness formal issues, computational complexity aspects of the involved algorithms as well as performance comparison against a prior implementation of a well known case of study, showing a considerable benefit in the final execution times. Although this line of research it is in early stages, the initial results presented in this paper are promisingly encouraging.

References

1. F. Asteasuain, J. Acuna, and M. Machuca. Easing the behavior specification for satellite missions. In *CACIC*, 2024.
2. F. Asteasuain and V. Braberman. Declaratively building behavior by means of scenario clauses. *Requirements Engineering*, pages Vol 22,239–274, 2017.
3. F. Asteasuain and L. R. Caldeira. A sound and correct formalism to specify, verify and synthesize behavior in big data systems. In *Argentine Congress of Computer Science*, pages 109–123. Springer, 2022.
4. F. Asteasuain, F. Calonge, and M. Dubinsky. Exploring specification pattern based behavioral synthesis with scenario clauses. In *CACIC*, 2018.
5. F. Asteasuain, F. Calonge, M. Dubinsky, and P. Gamboa. Open and branching behavioral synthesis with scenario clauses. *CLEI E-JOURNAL*, 24(3), 2021.
6. R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'Ar. Synthesis of reactive (1) designs. 2011.
7. M. Dwyer, M. Avrunin, and M. Corbett. Patterns in property specifications for finite-state verification. In *ICSE*, pages 411–420, 1999.
8. R. Ehlers and V. Raman. Slugs: Extensible gr (1) synthesis. In *International Conference on Computer Aided Verification*, pages 333–339. Springer, 2016.
9. S. Konrad and B. H. Cheng. Real-time specification patterns. In *Proceedings of the 27th international conference on Software engineering*, pages 372–381, 2005.
10. A. V. Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *RE*, 2001.
11. M. Maidi. The common fragment of ctl and ltl. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 643–652. IEEE, 2000.
12. S. Maoz and J. O. Ringert. Gr (1) synthesis for ltl specification patterns. In *FSE 2015*, pages 96–106, 2015.
13. S. Maoz and J. O. Ringert. Synthesizing a lego forklift controller in gr (1): A case study. *arXiv preprint arXiv:1602.01172*, 2016.
14. S. Maoz and J. O. Ringert. Spectra: a specification language for reactive systems. *Software and Systems Modeling*, 20(5):1553–1586, 2021.
15. C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger. Specification patterns for robotic missions. *TSE*, 47(10):2208–2224, 2019.
16. Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, K.-N. Wu, and W.-C. Chan. Goal: A graphical tool for manipulating büchi automata and temporal formulae. In *TACAS*, pages 466–471. Springer, 2007.